BGPmon v7: A Scalable Real-time BGP Monitor

Catherine Olschanowsky
Colorado State University
cathie@cs.colostate.edu

Dan Massey
Colorado State University
massey@cs.colostate.edu

***Abstract*— BGP is the global networking protocol for the Internet today. It is a critical component in the Internet architecture, but despite its importance has several known weaknesses and vulnerabilities. These vulnerabilities, when exploited (accidentally or maliciously), manifest as reachability problems in the Internet. Large-scale events in recent history include the accidental global hijack of YouTube and the disruption of Internet access in Australia in 2012.**

BGP monitoring is necessary to gaining a better understanding of BGP operations in order to better secure the Internet and prevent both accidental and malicious hijacks. BGPmon improved the current state of the art in BGP monitoring by providing real-time access to BGP data in an extensible format. This paper describes the next version of BGPmon, v7. This version expands on the capabilities of BGPmon by including handling for slow clients, providing a separate routing table data stream, and more fully integrating with Routeviews.

Index TermsBGP, Networking, Monitoring

I. Introduction

The basic operations of the Internet depend on underlying routing infrastructure. This infrastructure comprises a mesh of routers that direct all traffic in the Internet. The routers can be seen as sign posts that direct messages from one point to another in order to reach their final destination. The information to set up the sign posts initiate from the final destination and propagate through the entire Internet. The propagation of the routing information through the Internet is achieved using routing protocols.

The existing Internet uses BGP [2] as its global routing protocol, but research challenges related to BGP are well known. Security remains an open challenge and is the subject of active research [11], routing convergence problems have been identified and various solutions have been proposed [9], [6], and the research community is actively working on understanding the impact of routing policies [8], [7]. Efforts on next generation designs [10] have been motivated by problems experienced in the current system and are often evaluated using data drawn from the operational Internet.

Several events in resent history illustrate the vulnerability of BGP. For example, Pakistan used BGP routes in an attempt to control access to YouTube. Unfortunately, an error in the Pakistani approach ended up hijacking the YouTube service for users throughout the globe. These types of routing errors are fairly common. In a more recent example, a configuration error at an ISP in Australia disrupted connectivity throughout Australia in February of 2012.

Events such as the YouTube hijack and large-scale route leaks are well known primarily because they were unintentional errors that had global impact. Smaller scale events can still have devastating impact on the affected sites, but are hard to diagnose. More fundamentally, these were unintentional errors, not stealthy attacks. If operator errors can hijack large portions of the Internet simply by accident, consider what might be accomplished by a stealthy attacker whose aim is to target specific sites.

To truly understand and properly analyze the global routing system, one needs to collect BGP data from a wide range of sites with different geographical locations and different types (tiers) of ISPs. Fortunately global routing monitoring projects, such as Oregon RouteViews [5] and RIPE RIS [4], have been providing this essential data to both the operations and research communities. Google Scholar lists hundreds of papers whose results are based on these monitoring resources. Results on route damping, route convergence, routing policies, Internet topologies, routing security, routing protocol design, and so forth have all benefited from this data. Clearly, these monitoring projects are very useful.

However, experience over the years has also shown a number of major limitations in the current BGP data collection process. An ideal monitoring system would scale to a vast number of peer routers and provide BGP data in real-time to an even larger number of clients. For example, one might like to add operational routers from different geographic locations and lower tier ISPs. At the same time, real-time access would enable all interested parties to analyze the data to detect events such as fiber cuts, prefix hijacks, and so forth. The monitoring system should also reflect the fact that BGP is still evolving and the system should be easily extended to handle new BGP extensions, such as the expansion to four byte AS numbers, new security measures, and any number of current or future extensions to the protocol.

BGPmon was developed to overcome the above mentioned challenges with BGP monitoring. V6 was the first publicly available, production level release of BGPmon. BGPmon has been actively deployed and provides Routeviews data and data from direct peers through a single real-time data interface. However, through the development and use of BGPmon v6 it became clear that further improvements are necessary. This paper describes BGPmon v7, which improves upon the previous release.

BGPmon v7 continues to provide the advantages created by v6 (live-data access, scalability, an extensible data format, and a simplified code base) while also addressing additional challenges (handling slow clients, providing a separate routing

table data stream, and integration with Routeviews).

The rest of this paper is organized as follows: section II presents the current state of BGP monitoring, section III introduces BGPmon and section IV presents a set of tools available to aid in the utilization BGPmon data.

II. Background

BGPmon has been developed in coordination with and as a replacement for the existing Routeviews project. Routeviews collects BGP data and archives it at the University of Oregon. BGPmon was initiated to improve upon the Routeviews system by simplifying the code base, providing real-time data, providing scalability, and offering an extensible data format. Each of these goals was achieved by the v6 release of BGPmon [14]. The latest release (v7) provides further improvements to the BGP monitoring infrastructure.

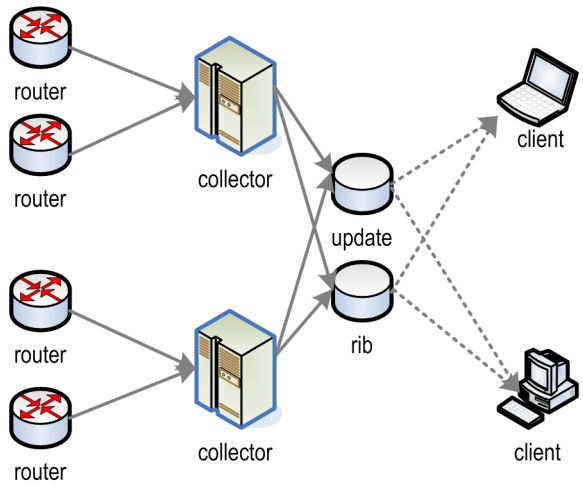

Fig. 1. Routeviews Infrastructure

Before introducing the improvements to the BGPmon system, we will first review the basic concepts used by public data collection sites such as Oregon RouteViews [5] and RIPE RIS [4]. The objective of these sites is to provide interested researchers and operators with access to the updates sent by routers at various ISPs and to also provide periodic snapshots of the corresponding BGP routing tables. To accomplish this, the current monitoring system negotiates BGP peering agreements with ISPs and deploys one or more collectors to obtain the BGP data. To the ISP routers being monitored, a collector is simply another BGP peer router. As shown in Figure 1, the collector receives and logs the BGP messages received from the ISP router being monitored.

The heart of the system is the data collectors. A collector may be a simple unix machine running an open source routing toolkit. The collector simply writes all received updates to a file in MRT format (a binary representation originally developed for the Multi-threaded Routing Toolkit) [3] and the file is made publicly available. Applications can read the MRT formatted file directly or first convert the binary format to text using tools such as bdpdump[1].

In addition to providing update logs, monitors also provide snapshots of the resulting BGP routing table, referred to as RIBs (Routing Information Base). The collector maintains an internal RIB table for each peer by applying standard BGP protocol rules as updates are received. A BGP update may add a route to the RIB table, remove a route from the RIB table, or modify an existing route. Whenever an update is received, the RIB table is modified accordingly. Each update received triggers the appropriate RIB table modifications and a write to disk in MRT format.

RIB files provide a snapshot of the routing tables over a very short interval while the udpates provide a stream of changes that occur between the rib file snapshots. Together, the RIB and update files provide the ability to rebuild the state of the routes at a particular time and replay subsequent changes to the routing infrastructure for analysis. RouteViews provides update files that are roughly 15 minutes in duration and provides routing table snapshots approximately every 2 hours.

As mentioned previously the v6 [14] release of BGPmon improved BGP monitoring in four primary ways:

1) **Live-data Access** BGPmon provides a live data stream through a well-known port.
2) **Scalability** Each BGPmon collector can peer directly with a large number of peers and chain together to provide a single data access point.
3) **Extensible Data Format** BGPmon provides the data using an easily parsed XML format.
4) **Simplification of Code base** BGPmon implements only the software necessary to establish a peering relationship and receive BGP.

Immediate access to *live-data* is necessary for prefix hijack detection. Access to data that is fifteen minutes old is sufficient for analysis of past events, but real-time monitoring of BGP activity requires update files be available in seconds. For example, current BGP prefix hijack alert systems would like to detect a potential route hijack within a few seconds. BGPmon provides access to the live-data through a single interface. All BGP messages received are applied to internal RIB tables and reformatted into XML to be sent to interested clients.

Access to the live-data required the development of a more *scalable* system. Routeviews has many collectors (13 at this time), each of which have peering sessions established with up to 22 peers. Each collector handles archiving its own data separately and the data must be accessed separately. BGPmon aims to provide a single data access point for all of the data from all of the collectors.

BGPmon collectors may be chained together in order to provide the needed scalability. A BGPmon instance does not process any messages received from a chain, but rather forwards them through its own output port, essentially consolidating the data, without incurring the overhead of RIB table maintenance.

BGPmon v6 provides the BGP data using an XML format. This format allows for easy parsing and it has the benefit of being human-readable. Aside from easy parsing XML was

chosen because of its extensibility. For instance, BGPmon can easily annotate the BGP data with labeling information without affecting downstream clients unnecessarily.

Existing monitors typically collect data using a full implementation of a BGP router. In contrast, BGPmon eliminates the unnecessary functions of route selection and data forwarding to focus solely on the monitoring function, effectively simplifying the code base.

BGPmon v7 represents further improvements to the BGP monitoring infrastructure.

1) **Handle slow clients**
2) **Separate RIB stream**
3) **Integration with Routeviews**

Each of these improvements is discussed in further detail, along with implementation details in the following section.

III. BGPmon Architecture

The BGPmon architecture is based on the Staged Event-Driven Architecture (SEDA) [13] approach. The SEDA approach enables high levels of concurrency in server applications. The application, in this case BGPmon, is broken down into stages and a queue is utilized between each stage in order to absorb data bursts and enable load-shedding should it become necessary. Each stage of the application is handled in its own thread or set of threads. In addition to the performance advantages the SEDA approach encourages clean modular code design.

Figure 2 shows BGPmon broken down into a series of four stages.

1) Handling the connection and receipt of data from either a peer, an Routeviews collector, or another BGPmon.
2) RIB maintenance and Labeling
3) Building the XML
4) Handling connections and sending data to clients.

Between each of the stages are queues. The queues each hold distinct units of data, usually corresponding to individual BGP messages, but sometimes internal status messages. All of the queues, with the exception of the update and RIB queues, hold data in an internal BGPmon format called BMF (BGPmon Message Format).

Stage 1 comprises three possible entry points for data: direct peers, existing Routeviews collectors, or a BGPmon chain. Data is collected from direct peers by BGPmon emulating a router. BGPmon creates a thread for each peer router. Upon receipt, BGP messages (Open, Update, Notification, Keepalive, Route-refresh) are translated into BMF and placed in the peer queue to create a single, consolidated stream of peer events.

The peer thread manages the connection with the router, detecting loss of connection and automatically initiating recovery. In addition, all changes in the BGP finite state machine (FSM) are placed in the peer queue. The peer thread attempts to emulate the connection behavior of routers; it is able to negotiate capabilities and authenticate using MD5.

The Routeviews collectors have been modified to send each BGP message received to BGPmon in MRT format as well as writing the messages to disk. Due to the fact that a Routeviews collector may be directly peering with many routers a much larger volume of data is expected to be handled by these connections. For the same reason, there is the potential for very large bursts of data to come in through these connections. In order to handle the data volume and absorb large bursts the Routeviews connections are handled by a pair of MRT threads.

The first MRT thread is created when a Routeviews collector attempts to connect to a BGPmon instance. This thread is responsible for receiving messages as quickly as possible; it performs no processing, instead simply writes to the in-memory MRT backlog as quickly as possible. The second MRT thread is responsible for reading the data from the backlog, parsing it into BMF and placing it on the MRT queue.

In the case of a burst of data the MRT backlog can be expanded (up to a limit) to absorb the burst. Using this approach it is possible for the live-stream data to be delayed. This is unavoidable, and during such bursts the primary goal is to avoid data loss. There is only a single MRT queue in the system. Multiple Routeviews connection can be maintained and each will write to the same MRT queue.

The third possible entry point for data coming into a BGPmon instance is through a chain. During chaining a BGPmon is configured to connect to another BGPmon instance and forward all of the data received. In the case of chaining, no RIB table is maintained locally. The data bypasses the labeling thread and is inserted directly to the update queue to be forwarded to clients.

A single BGPmon instance can be configured to manage multiple peering connections, Routeviews connections and chains simultaneously.

Stage 2 is the internal RIB table maintenance and labeling thread. Along with applying the incoming BGP messages to the RIB table the labeling thread adds meta data to the message. A label thread processes the events from the peer and MRT queues and maintains a RIB table which contains unprocessed routing information advertised by peers. This thread determines label information based on the state of the RIB tables, and places the event and corresponding label in the label queue. The labels identify announcements, withdrawals, new updates, duplicate updates, same path, and different path to aid filtering and analysis [12]. Since the RIB tables are the major memory constraint for the system, labeling is an optional feature and when turned off, the memory used by BGPmon drastically decreases.

Stage 3 is the XML thread. The XML thread processes events in the label queue, converts them to XML then places them into the Update or RIB queue. Events from another BGPmon instance can be aggregated into the Update or RIB queue to form BGPmon meshes.

BGPmon can provide a real-time event stream to a large number of clients. XML was chosen as the message format for the event stream because it is extendable, for both clients and servers, and also readable by both applications and humans. Also, XML allows BGPBrokers to route, filter, and aggregate

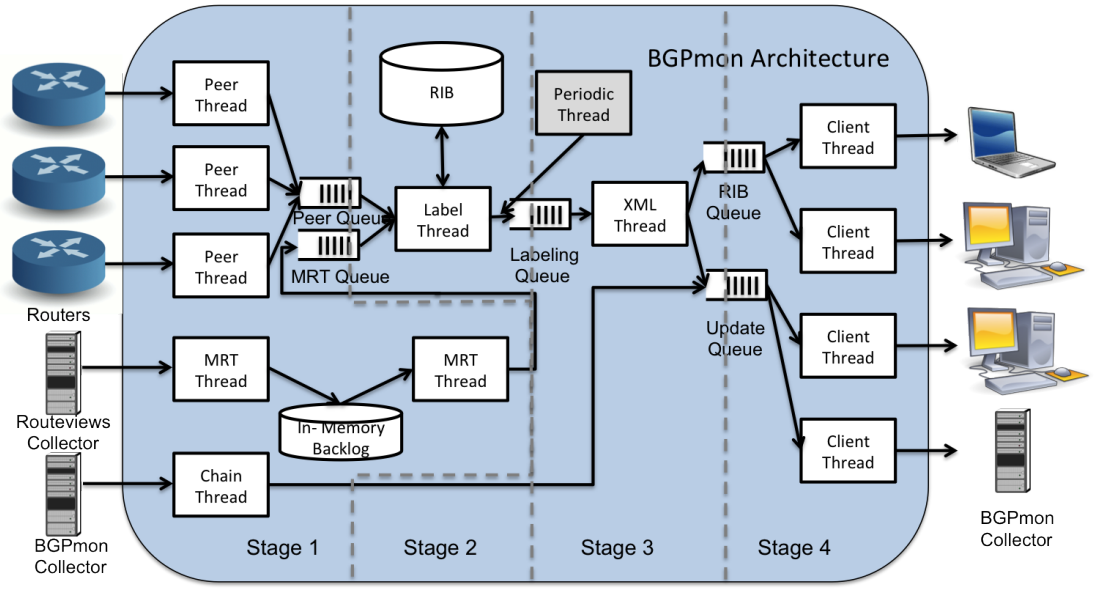

Fig. 2. BGPmon architecture.

events using XPath[1] queries.

Stage 4 disseminates the data to interested clients. BGPmon listens for interested clients to connect. Each client connection is managed by its own thread that becomes a reader to the update or RIB queue. Data is not deleted from the queues until every client thread has read it. This ensures that every client receives all of the data. However, it also creates the situation that a single slow reader can cause the queue to become full and necessitate load shedding from the queue. This situation is remedied by the queue management described in the next section.

In addition to the four stages of the architecture there is an additional out-of-band thread, the periodic thread. The periodic thread triggers time-controlled events such as status messages and initiating sending route table summaries.

A. Stream controller

As Figure 2 shows, in BGPmon the event stream flows from peers to clients through three queues. The main difference between our design and SEDA is that all the queues in BGPmon have a fixed length. So the key challenge in the design of BGPmon is to prevent fixed-length queues being overwhelmed while supporting many peers and clients with different writing/reading rates. For example, large spikes in the event stream will overwhelm the queues when the entire routing table is received. This occurs any time that the session is reset. The worst case will happen when BGPmon starts and all peers send their routing tables simultaneously. Route refreshes and other major changes in topology can also cause significant spikes. A similar situation can also occur when a slower client is unable to process events in a timely fashion.

In BGPmon, data is added to the queue by writers and removed only after all readers have accessed the data. The writers, which are typically BGP routers, send data according to BGP protocol standards. The number of messages sent by writers is primarily a function of the number of route changes seen by a peer. At the same time, readers read data from the queue at varying speeds due to bandwidth or processing constraints. BGPmon employs two mechanisms that are designed to handle the varying read and write speeds of the clients and peers.

Pacing writers: The queue paces the writers according to the average reading rate across all readers. When a queue length exceeds a configurable threshold, pacing is enabled until the queue length drops below a second threshold. For example if the queue length is larger than the pacing enable threshold, pacing will be enabled. When the system is in pacing mode it will ensure that no writer can be starved of resources and may limit the speed at which particular writers add to the queue.

For example, suppose there are 4 readers and 2 writers and on average each reader can read 8 messages per second. Ideal pacing should limit the writing rate of each writer to 8/2 = 4 in order to avoid overwhelming the queue. Our approach ensures that each writer can add 4 messages per second and limits writers to 4 messages only if the queue size is growing too rapidly.

In BGP terms, this may mean BGP updates are read at a slower rate if the queues are filling too fast. In turn, this will apply back pressure on the TCP connection between the peer and BGPmon. If BGPmon does not read data then the TCP buffers fill and eventually new data cannot be sent which causes the peer router to delay updates. Ultimately, if the delay is too long then the peer may terminate the connection with BGPmon. For example, the peer will close the connection if keepalive messages cannot be exchanged at a sufficient rate. Our objective is not to permanently limit the connection, but rather to survive bursts of data. We do this by using a large

[1]XPath is a standard method for querying XML documents

queue and by pacing how fast the peers write.

Skipping data for slow readers: In the case of a very slow reader, the queue length may continue to grow despite our attempt to pace writers to the average reader. Ultimately, if writers add data faster than the slowest reader can consume it, any queue must eventually fill up. In this case, the readers are simply too slow and our system detects the slowest reader, sends it a notification that messages are being skipped, and catches that reader up to the position in the queue where the *ideal reader* is. The ideal reader is on that reads messages at the same rate they are written. The rate is calculated as a moving average of the write speeds.

For example, suppose the queue is almost full and again there are 4 readers and 2 writers. Suppose also that 3 of the 4 readers can read 8 messages per second but one of them can only read 2 message per second. Data is only removed from the queue after all readers have accessed the data so only 2 messages are removed from the queue per second. As a result, the average reading rate across the 4 readers is 6.5 messages per second. In this case, even if pacing is turned on and each writer is limited to write 6.5/2 = 3.25 messages per second, the queue will still be overwhelmed because of the slow reader. When the queue nears capacity, the slowest reader (2 messages per second) is fast forwarded to the position of the ideal reader.

This fast forwarding has two important effects. First, the queue size drops immediately. At least one item in the queue is present only because the slowest reader has yet to read that item. When the slowest reader is fast-forwarded, the oldest queue item is deleted, freeing at least one spot in the queue. Second, other readers remain unaffected by this fast forwarding. The advantage of this over dropping the client is that the overhead of reestablishing the connection is avoided.

IV. BGPmon Tools

In addition to BGPmon v7 a set of tools has been simultaneously released. A set of Perl modules has been developed in order to provide requested processing of the BGPmon stream as well as provide an easy starting point to creating custom Perl modules. To this end, several Perl packages have been written and are available as open source downloads on the Comprehensive Perl Archive Network (CPAN).

1) BGPmon-core
2) BGPmon-Archive
3) BGPmon-Analytics-DB

A. Core Perl Modules

The core Perl modules provide the basic functionality required to develop BGPmon clients. The Module BGPmon::Fetch is an interface for fetching BGPmon data. There are three current implementations of this interface: Client, File, and Archive. The Client implementation connects to a live BGPmon instance in order to ease processing live-data. The File implementation opens a saved file containing BGPmon data and the Archive implementation can step through archived BGPmon data.

The BGPmon::Fetch interface is used to access all three implementations; this makes it possible to write a single processing script without regard to if the data is live or archived. This distinction is useful, because processing techniques can be used to analyze archived data where known Internet events are known to be located or process live-data in order to detect current events.

In addition to the basic data fetching interface the core package includes modules for translating BGPmon messages, filtering the messages and logging activity. The BGPmon::Translator modules allow messages to be translated from BGPmon XML to a bgpdump-like format, or to an internal perl hash that can be used for further processing.

The BGPmon::Filter modules are responsible for filtering a BGPmon XML stream based on prefix or AS number. This is useful functionality for reducing the size of the stream to be processed. Figure 3 shows an example of how the Perl modules can be used to filter a stream and find the prefixes and coresponding as paths that come from a specific autonomous system.

B. Archiver

The archiver is a script that can take the BGPmon XML messages and organize them into files based on peer IP address. The archiver script rolls the files based at a parameterized interval. This means that each file is given a name that includes a starting time and only messages that are received after that time and within the given interval are written to that file before a new file is created.

The data is archived in the BGPmon XML format as well as in the bgpdump-like format provided through the Translator modules. The bgpdump format is a pipe separated text format. It is provided in order to provide backward compatablity for tools written to make use of Routeviews archives processed by bgpdump.

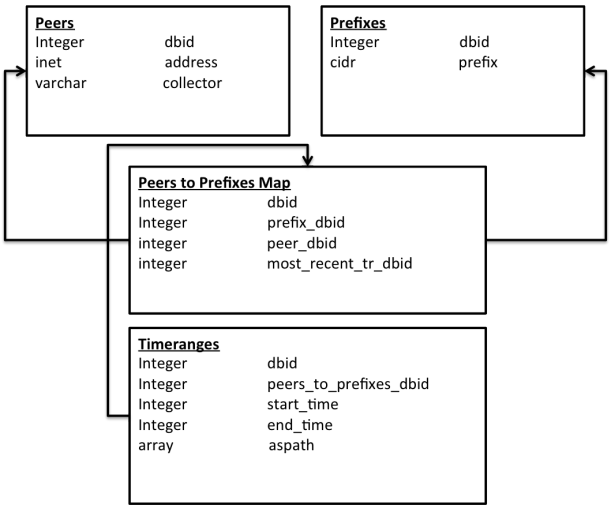

Fig. 4. BGPmon Analytics DB Schema

```
use BGPmon::Fetch;
use BGPmon::Filter;
use BGPmon::Translator;

connect_bgpdata();

my $xml_msg = read_xml_message();
while($xml_msg){
    # throw out everything except routes from our desired ASN
    if (ComesFrom($desiredASN)){
        # extract a print the prefix and path
        my $time = get_time($xml_msg);
        my $prefix = get_prefix($xml_msg);
        my $path= get_aspath($xml_msg);
        print "At time $time,  there is a route $prefix with path $path\n";
    }
   $xml_msg = read_xml_message();
}
close_connection();
```

Fig. 3. The Data Access Script for Somewhat Sophisticated User Who Wants to Examine Route Announcement and Changes for A Particular Origin

C. Analytics Database

One of the common uses of BGP data is to track changes over time as they relate to a single specific prefix or net. In order to do this a history related to said prefix must be maintained. The BGPmon analytics database provides a mechanism for saving the history.

The analytics DB package provides a script that can connect to a live BGPmon feed or one created using a standalone filter. The data is received and inserted into the database. The current implementation uses the postgresql database engine. The data can be queried directly using psql or using the included scripts for generating reports.

The database schema (see Fig. 4 has been designed specifically to allow the history of a specific prefix to be viewed. The information on the prefix is associated with the prefix as well as with the peer it was observed from. This enables the comparison of prefix path information from several vantage points.

V. Conclusion

This paper presents the latest version of BGPmon, a next generation BGP monitoring system. The main contributions of this version are slow client detection and handling, separation of update and RIB dump data streams, and integration with Routeviews. The integration with Routeviews required the addition of an in-memory backlog for each Routeviews connection in order to deal with the high data rate and natural burstyness of the data.

The BGPmon tools are also presented. These Perl modules can ease the development of direct BGPmon clients and are available for download on CPAN.

References

[1] bgptools. http://nms.lcs.mit.edu/software/bgp/bgptools/.

[2] A border gateway protocol 4 (bgp-4). http://www.ietf.org/rfc/rfc4271.txt.

[3] Mrt routing information export format. http://www.ietf.org/internet-drafts/draft-ietf-grow-mrt-07.txt.

[4] Ripe (rseaux ip europens) routing information service. http://www.ripe.net/projects/ris/.

[5] University of oregon route views project. http://www.routeviews.org/.

[6] A. Bremler-Barr, Y. Afek, and S. Schwarz. Improved bgp convergence via ghost flushing. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, 2003.

[7] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of autonomy for the expressiveness of policy routing. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 25–36, New York, NY, USA, 2005. ACM.

[8] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, 2001.

[9] Dan Pei, Matt Azuma, Dan Massey, and Lixia Zhang. Bgp-rcn: improving bgp convergence through root cause notification. *Comput. Netw. ISDN Syst.*, 48(2):175–194, 2005.

[10] Lakshminarayanan Subramanian, Matthew Caesar, Cheng Tien Ee, Mark Handley, Morley Mao, Scott Shenker, and Ion Stoica. Hlp: a next generation inter-domain routing protocol. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 13–24, New York, NY, USA, 2005. ACM.

[11] Lakshminarayanan Subramanian, Volker Roth, Ion Stoica, Scott Shenker, and Randy H. Katz. Listen and whisper: security mechanisms for bgp. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[12] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S.F. Wu, and L. Zhang. Observation and analysis of BGP behavior under stress. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 183–195, 2002.

[13] Matt Welsh, David Culler, and Eric Brewer. Seda: an architecture for well-conditioned, scalable internet services. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 230–243, New York, NY, USA, 2001. ACM.

[14] He Yan, Ricardo Oliveira, Kevin Burnett, Dave Matthews, Lixia Zhang, and Dan Massey. Bgpmon: A real-time, scalable, extensible monitoring system. In *Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, 2009.