BGPmon: A real-time, scalable, extensible monitoring system

He Yan
Colorado State University
yanhe@cs.colostate.edu

Ricardo Oliveira
UCLA
rveloso@cs.ucla.edu

Kevin Burnett
Colorado State University
burnet@cs.colostate.edu

Dave Matthews
Colorado State University
dvmtthws@cs.colostate.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

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

Abstract

This paper presents a new system, called BGPmon, for monitoring the Border Gateway Protocol (BGP). BGP is the routing protocol for the global Internet. Monitoring BGP is important for both operations and research; a number of public and private BGP monitors are deployed and widely used. These 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. BGPmon uses a publish/subscribe overlay network to provide real-time access to vast numbers of peers and clients. All routing events are consolidated into a single XML stream. XML allows us to add additional features such as labeling updates to allow easy identification of useful data by clients. Clients subscribe to BGPmon and receive the XML stream, performing tasks such as archiving, filtering, or real-time data analysis. BGPmon enables scalable real-time monitoring data distribution by allowing monitors to peer with each other and form an overlay network to provide new services and features without modifying the monitors. We illustrate the effectiveness of the BGPmon data using the Cyclops route monitoring system.

1. Introduction

Understanding global routing is critically important for current Internet research and operational network security. The existing Internet uses BGP[3] 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[15], routing convergence problems have been identified and various solutions have been proposed[13, 8], the research community is actively working on understanding the impact of routing policies[11, 10], and efforts on next generation designs[14] have been motivated by problems experienced in the current system and are often evaluated using data drawn from the operational Internet. The BGP monitoring data supports a wide range of efforts ranging from understanding the Internet topology to building more accurate simulations for network protocols.

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[7] and RIPE RIS[6], 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 numbers 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.

This paper presents the design and implementation of a next generation BGP monitoring system. We propose a mesh of interconnected data collectors and data brokers that

978-0-7695-3568-5/09 $25.00 © 2009 IEEE
DOI 10.1109/CATCH.2009.28

operate using a publish/subscribe model. Our approach extends the scalable event driven architecture[17] to meet the requirements of BGP monitoring. Interested clients receive an event stream in real-time or may read historical event streams from archival sources. The event streams provide both incremental BGP update messages and periodic routing table snapshots. We use XML to provide extensibility, integration with common tools, and to allow local data annotations.

Using data from BGPmon, one can solve a wide range of critical problems. As an example we present the Cyclops system[9] (*http://cyclops.cs.ucla.edu*). Cyclops was designed to be a generic framework that would compare the intended behavior of the network with the observed behavior. The intended behavior can either be explicitly entered by the user or statistically inferred. Network operators are automatically alerted (e.g. by email or sms) anytime there is a change in the network that deviates from the expected behavior. In particular, one could use the underlying BGPmon data to detect and immediately report BGP prefix hijacking events. Tools such as Cyclops can benefit greatly from BGPmon since it can easily scale up the number of BGP feeds; adding locations where RouteViews and RIPE do not have a presence. Furthermore, BGPmon eliminates the delay involved in receiving the feeds since they can be provided in real time.

The paper is organized as follows. Section 2 reviews the current state of BGP route monitoring and section 3 introduces our new approach. Section 4 describes how our design "scales up" to meet both peer and client demands while section 5 shows how our approach "scales out" using a publish/subscribe overlay network. Section 6 shows some early results from deployment and section 7 shows how Cyclops makes use of BGPmon data. Finally, Section 8 concludes the paper.

2. Background

Before introducing our new BGP Monitoring system, we will first review the basic concepts used by public data collection sites such as Oregon RouteViews[7] and RIPE RIS[6]. 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. 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 Multi-threaded Routing Toolkit (MRT)[4] format and then 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[2].

In addition to providing update logs, monitors also provide snapshots of the resulting BGP routing table, referred to as RIBs. The collector builds a RIB table by applying the standard BGP protocol rules. 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. As updates are received from a peer, the collector updates the routing table for that peer and periodically writes it 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.

Currently, RouteViews provides update files that are roughly 15 minutes in duration and provides routing table snapshots roughly every 2 hours. This 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. At best, today's RouteViews system only allows hijack alert systems to report hijacks that occurred many minutes ago.

In addition to providing data in real-time, an ideal BGP monitoring system would scale to dramatically increase the number of peers providing data. Given data from more locations, BGP analysis systems and tools could potentially provide better answers. For example, a BGP prefix hijack may only be visible in a small portion of the network and ideally one would like to have a monitor present in that same portion of the network. Thus our goal is not only to make the data available in real-time, but also to dramatically increase the volume of data available.

Finally, such an ideal system could attract a large number of new applications. The data is public and should be available to any interested researcher or operator. In many cases, the data collected by RouteViews can serve as one input to monitoring systems throughout the network.

In summary, the current system is useful but it would be useful to make the data available in real-time while simultaneously increasing the amount of data collected and dramatically increasing the number of locations obtaining the data. All this should occur without lose of data fidelity.

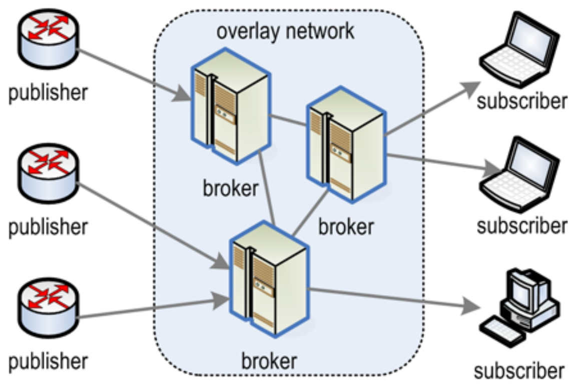


Figure 1. Physical view of publish/subscribe overlay network.

3. A New Monitoring System

Open source routing software that is currently used as a collector typically implements a full routing protocol, including receiving routes, applying policies, setting forwarding states, and announcing routes to peers. Applying polices, setting forwarding states, and announcing routes to peers involve considerable complexity, but none of these actions are needed be collector. A collector simply needs to receive and log routes. Our new collector design focuses on a narrow set of data collection functions. By focusing on the collection functionality and eliminating unnecessary tasks, the new collector is able to *scale up* and support more peer routers while making the data available in real-time to a potentially vast number of clients.

To support hundreds of peer routers and thousands of clients, one would like to *scale out* across multiple systems by adding more collectors and distributing the services. At the same time, a client should see a single monitoring service and be unaware that the implementation of the service may be done through multiple collectors.

Our approach is based on publish/subscribe overlay networks that consist of brokers, publishers, and subscribers as shown in Figure 1. The brokers form the overlay network, allowing publishers to send event streams to the overlay network and allowing subscribers to receive event streams from the overlay network. Publishers and subscribers interact only with brokers, not with each other, allowing the overlay network to insulate publishers and subscribers from each other. The brokers manage the distribution of the event streams based on the client subscriptions, identifying the best path from the publisher to the subscriber. Services, such as filtering, aggregation, and querying, are typically performed by the brokers on behalf of the subscribers.

BGPmon publishes an event stream containing data while applications subscribe to these streams based on attributes of the data in the streams. These streams will include all BGP messages (open, close, update, notification, keepalive, route-refresh) as well as state changes in the BGP Finite State Machine (FSM). An application may subscribe to all events or only a subset of events based on peer, autonomous sytems, events type, or other information contained in the data. To improve fault tolerance, multiple brokers may monitor the same or different peers in an AS, yet appear to a client application as a single subscription. This allows critical applications to continue to receive event streams in the case of a failure of a peer, monitor, or broker.

Our system implementation begins with BGPmon, a simple monitoring system now available that incorporates all three functions: publish, broker, and subscribe. The second stage is BGPbroker which separates these functions to support Internet scale and additional services.

4. Scaling Up: BGPmon

The BGPmon architecture shown in Figure 2 reflects a real-time monitor capable of scaling up and out. The design makes use of threading to provide real-time support and scales up the number of peers and clients supported by the monitor. BGPmon uses a lightweight thread for each peer and client connection. In addition, there are threads for chains to other instances of BGPmon, and some internal functions, such as labelling and XML conversion. The use of threads takes advantage of the trend towards multicore processors.

BGPmon creates a peer thread for each peer router and places all BGP messages (Open, Update, Notification, Keepalive, Route-refresh) in the peer queue to create a single, consolidated stream of events. The peer thread detects loss of connection and automatically initiates recovery of the connection. In addition, all changes in the BGP FSM are placed in the peer queue. The peer thread uses MD5 authentication for the router connection if configured.

A label thread processes the events from the peer queue and maintains a RIBIN table which contains unprocessed routing information advertised by peers. This thread determines label information based on the state of the RIBIN 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 [16]. Since the RIBIN 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.

Finally, the monitor thread periodically issues status information and injects route tables into the event stream. Route tables are obtained directly from the peer router by requesting a route refresh. The RIBIN table can be used to simulate a route refresh if the peer does not support it.

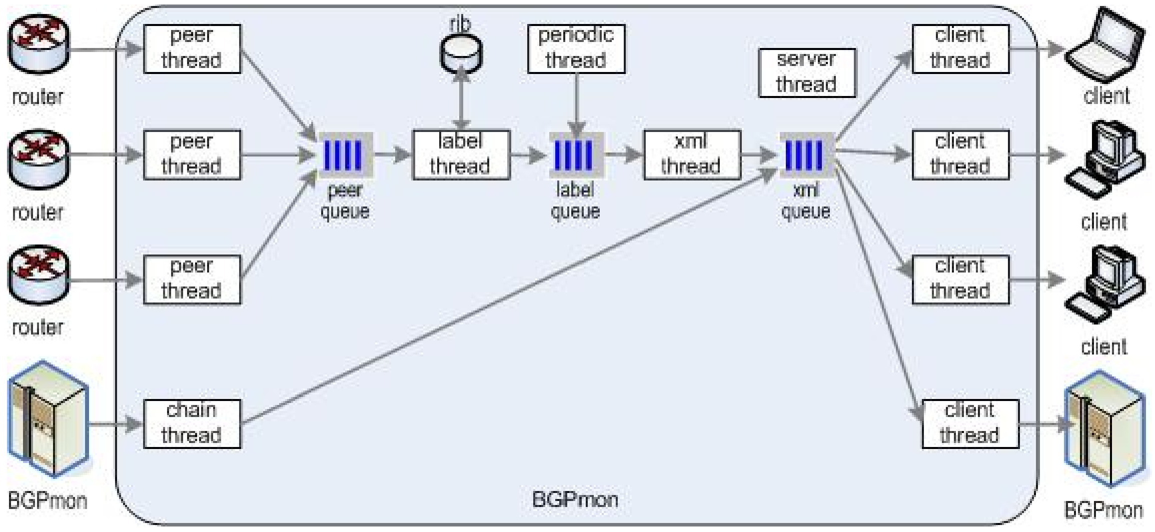


Figure 2. BGPmon architecture.

The XML thread processes events in the label queue, converts them to XML then places them into the XML queue. Events from another BGPmon instance can be aggregated into the XML queue to form BGPmon meshes, discussed later in the paper. Each client thread sends the entire stream of events from the XML queue to the client.

4.1. XML Event Stream

BGPmon can provide a real-time event stream to a large number of clients. XML was chosen as the message format for the even 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 events use XPath queries.

However, to scale to the demand for a large number of clients, there is no provision for a single client to request a table transfer from BGPmon. Instead, when a route-refresh is triggered by the router, BGPmon will incorporate the transfer directly into the event stream. Real-time applications must operate on partial information (without RIB information) until the route-refresh is initiated from a router in the event stream.

Several example clients that can make use of the XML event stream are different types of archivers. A simple log client can receive the event stream then write it directly to disk while a more complex log client can filter the XML messages then write them to disk. Another example of a client is one that can convert the XML messages back into MRT format and produce MRT formatted update files and rib tables.

One concern with XML is the space required to store the log files. Table 1 shows the space requirements of a

Table 1. Space requirements for text and compressed data. Ratios relative to MRT format.

Format	Data Size	Ratio	Compressed	Ratio
MRT	26711666	1.00	5614650	1.00
bgpdump	74551628	2.79	5645044	1.01
XML	264824363	9.91	13445451	2.39
XML-	218065044	8.16	6289003	1.12

sample log for a two hour period. MRT data is stored in binary format, adding only some additional header information to each update. The bgpdump [2] tool converts MRT into ASCII, increasing the size to 2.79 times the MRT sample. Our XML log includes the packet octets as well as the XML tags and is 9.91 times the MRT size. Eliminating the octets (line XML- in the Table), reduces the ratio to 8.16. However, most data is stored in a compressed format and after compressing the files using bzip2, the compressed log with octets is 2.39 times MRT and without octets is 1.12 times the MRT size.

4.2. Stream controller

BGPmon uses a design similar to the Staged Event Driven Architecture (SEDA)[17]. SEDA provides a reliable service that handles a large number of concurrent peers and clients. It divides the processing into stages connected via queues. And the resource controller is used to observe the incoming and/or outgoing rates of queues and adjust the queue length as needed.

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 sent. 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 queue and by pacing how fast the peers write.

Dropping slow readers: In the case of a very slow reader, the queue length may continue 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 and eliminates these slow readers. When the queue length reaches the maximum, the responsible reader is dropped and the queue is adjusted to the next slowest reader. This allows the remainder of the subscribers to continue processing the stream with no dropped events.

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 the 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 pacing is turned on and each writer is limit 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 disconnected.

This deletion 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 deleted, the oldest queue item also becomes read by all readers and is deleted, freeing at least one spot in the queue. If there are multiple equally slow readers, all of them are dropped to ensure some space is freed in the queue. Second, the remaining readers can process the data, possibly at a faster average rate.

Our experiment shows that pacing slows the rate at which BGP messages are read from the peer routers, but the connections are not dropped since the admission rate is still well above the keepalive and hold timers in the BGP sessions. Our system survives bursts in updates in has not dropped a connection in several months despite a wide range of reader speeds.

5. Scaling Out: Chains and Brokers

While we have endeavored to design a BGPmon that scales to a large number of peers and clients, we allow BGPmon to scale out through the interconnection of multiple BGPmons. This allows the separation of the peer monitor from the client server with only a single connection maintained between them. A mesh of BGPmons may be used for redundancy as shown in Figure 3. Both BGPmon C instances monitor a unique set of peers and forward their events to both BGPmon S instances. Each BGPmon S will then log the event stream and forward their events to any clients attached.

The use of separate monitors helps insulate the monitor-

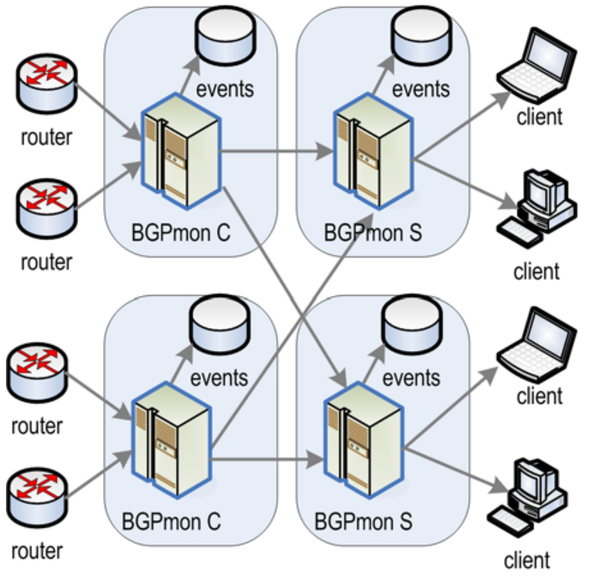


Figure 3. BGPmon mesh configuration to provide redundancy and scale.

ing, logging, and clients from individual failures. Failure of a BGPmon only affects the connected peers or clients. The other peers and clients in the mesh are not affected. Recovery is automatic since the failed BGPmon reconnects to the mesh when it recovers.

BGPbroker is the second stage of development to provide an overlay network that will support Internet scale and service extensions. Unlike BGPmons which simply collect and stream data, BGPbrokers provide a set of services for filtering and aggregation, allowing clients to subscribe to non-duplicate requests from a peer or to subscribe to an autonomous system that aggregates all of the appropriate peers. A BGPmon (or mesh of BGPmons) becomes a service to the broker in this model and provides the base data.

This powerful abstraction allows sites to design networks of brokers to provide a wide range of new services. For example, logging/playback services could allow applications to access archives through the subscription mechanism by specifying a time period in the subscription. A hybrid service could allow playback from a specific point that transitions into the current real-time feed. Other services may add additional information to suspect events in the stream, such as potential prefix hijacks or other security issues.

6. Deployment Results

This paper introduces a new BGP monitoring system that supports real-time monitoring, scalability, and extensibility. The system uses a publish/subscribe overlay network

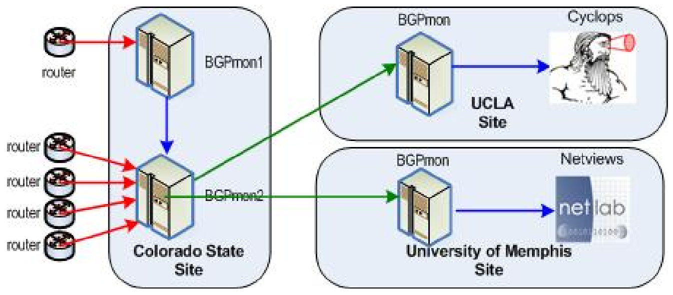


Figure 4. BGPmon testbed.

involving brokers, publishers, and subscribers to achieve these goals.

In order to deploy, debug, and evaluate BGPmon, a testbed has been in operation for roughly one year. This testbed involves 5 peers and 3 BGPmon sites is shown in Figure 4. In the Colorado State University (ColoState) site, BGPmon1 connects to 1 peer, BGPmon2 connects to 4 peers. These instances of BGPmon are then chained together. Recently, undergraduate course projects have experimented with clients connecting to BGPmon while other testing has shown that BGPmon can run smoothly with a varied client load from 1 to 200.

UCLA has its own local BGPmon which chains to BGPmon2 inside the ColoState site. Inside the UCLA site, the application Cyclops is integrating BGPmon data into its system. Cyclops will be discussed in detail in section 7.

Similarly, the University of Memphis site also runs its own BGPmon and chains their own BGPmon to BGPmon2. They also have a application Netviews[5] which is fed data by from their local BGPmon. Netviews provides a new visual interpretation of BGP data which network operators can quickly understand and analyze their own and others connectivity across the globe.

BGPmon provides raw data to clients such as Cyclops and NetViews. The only message processing that occurs in BGPmon is in the labeling system. By comparing a new update for a prefix to the last update for the same prefix received from the same peer, BGPmon labels updates as follows. If the peer is not currently announcing a route to a prefix and then a route to the prefix is reported, the update is labeled as a *new announcement*. Similarly if the peer is currently announcing a route to the prefix and then withdraws the route, the update is labeled as a *withdraw*. If a peer is currently announcing a route to a prefix and then changes the route to that prefix, BGPmon classifies the update as a *DPATH* update for different AS path or *SPATH* update if the path remains the same but some other attribute has changed.

In addition to the above, a peer may simply readvertise the exact same route to a prefix. This is labeled a *duplicate update*. In our system, duplicate updates are often the result

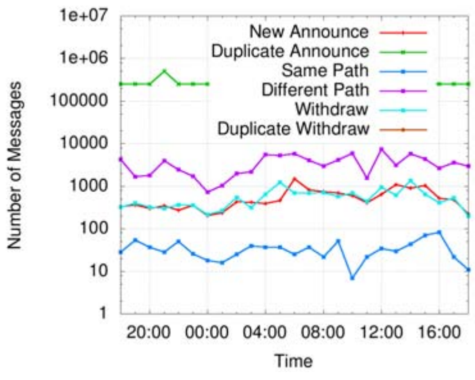


Figure 5. BGPmon peer statistics.

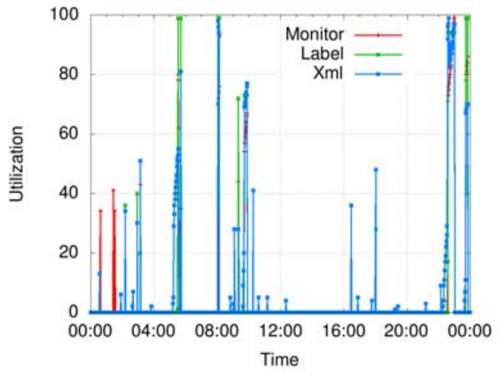


Figure 6. BGPmon queues.

of a request for routing table transfer. BGPmon periodically asks a peer to re-report all of its routes. This generates a large number of duplicate updates, but allows downstream BGPbrokers and clients to learn the full table of peer and/or synchronize any state associated with data from that peer. BGP routers may also withdraw the route to the prefix that has not been advertised in the first place. These *duplicate withdraws* are typically the result of bugs or sub-optimal implementation decisions at peer routers.

Overall, these labels can help clients or BGPbrokers quickly parse the vast number of updates. For example, tools such as Cyclops and prefix hijack detectors are primarily concerned with *new announcements* and *DPATH* updates. These tools can typically ignore the *duplicate announcements, SPATH updates, and duplicate withdraws*.

Figure 5 show the distribution of update messages received from a single peer. Each line represents one of the labels applied to the updates. As discussed above, the large number of duplicate announcements results from updates in the stream generated by the route refresh. Graphs for each peer are available at the BGPmon website[1] and are updated periodically.

Queue statistics in Figure 6 show the size of the peer, label, and XML queues.

Spikes typically occur during a route refresh. Pacing statistics in Figure 7 show the pacing limit, a weighted moving average, and the number of times pacing was enabled during the period. Queue and pacing parameters can be altered to change the size of the queue and the points at which pacing is turned on or off. This allows us to tune a monitor to the load generated by the peers and clients.

7. A BGPmon Client: Cyclops

Cyclops is a generic framework for routing monitoring that compares the intended behavior of the network with

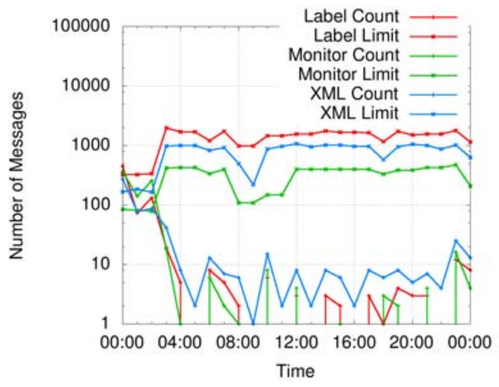


Figure 7. BGPmon queue pacing.

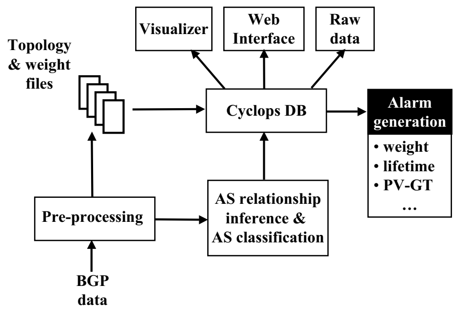


Figure 8. Cyclops implementation block diagram

the observed behavior. The intended behavior can either be explicitly entered by the user or statistically inferred. Network operators are automatically alerted (e.g. by email or sms) anytime there's a change in the network that deviates from the expected behavior.

Currently Cyclops data is updated once a day, which is not fast enough to react to network anomalies such as prefix hijacks. Cyclops is currently being integrated with BGP-mon so that it benefits from access to real-time BGP data. For example, observed data may trigger a routing security event that will be perceived within few seconds, rather than minutes or hours.

7.1. An overview of Cyclops

Figure 8 summarizes the Cyclops implementation. Daily collected BGP data from the *Public-View* first goes through a pre-processing stage where AS links and timestamps are extracted. More precisely, for each AS link in an AS path, we record the first and last times the link was seen, and whether the link was seen in the beginning, middle or end of the AS path. In addition, we save the last seen BGP update message that contained that link.

In the pre-processing stage, we also glean AS paths to infer business relationships between ASes, i.e. provider-customer, or peer-to-peer, and this relationship information is then used to do AS classification. The specific method we use to do AS relationship inference is deceptively simple. We extract the AS links from the BGP routes collected from the Tier-1 ISP monitors over a window of time which should span several months. In the AS path a_0–a_1– ... –a_n, the link a_0–a_1 can be either peer-peer or provider-customer (a_0 refers to the Tier-1 AS the monitor resides in), but the remaining links in the AS path should be of type customer-provider according to no-valley policy. Furthermore, if a_0–a_1 turns out to be a customer-provider link, it will be revealed in routes of another Tier-1 AS, therefore we will be able to accurately label it. The peer links are inferred by doing the diff between the entire set of links extracted from all the monitors and the set of customer-provider links, i.e. the peer links are all the links that are not propagated upstream to Tier-1s. In addition, we sort ASes into four classes based on the number of downstream customer ASes: *stubs* if they have 4 or less downstream ASes, *small ISPs* if they have between 5 and 50 downstreams, *large ISPs* if they have more than 50 downstreams, and finally *Tier-1 ASes*.

To measure how much an AS link is used, we keep track of the number of BGP routes carried on each AS link. We call this number the *link weight*, a concept borrowed from our previous work [12]. To avoid measurement bias, the link weight measurement only uses data from the $N \simeq 120$ monitors in *Public-View* that provide full BGP tables and reside in different ASes. We denote $w_i^j(t)$ the number of routes of monitor j that use link i on day t, and $w_i(t) = \frac{1}{N}\sum_j w_i^j(t)$ the average weight of link i over all the N monitors. We further compute an expected weight of each link over time using a TCP RTT measurement-like smoothed average: $\hat{w}_i(t) = 0.8\hat{w}_i(t-1) + 0.2w_i(t)$, and keep track of the difference between the instantaneous link weight on day t and the expected weight of each link i: $\varDelta w_i(t) = w_i(t) - \hat{w}_i(t)$. A significant difference can be used to trigger alarms. Furthermore, we keep track of two different weights depending on the position of the eye of the Cyclops in the AS link seen in the routes. w_{to} represents the number of routes using the link x–y, where x is the eye of the cyclops and y one of its neighbors; w_{from} represents the weight of the link y–x, towards the eye of the Cyclops x. Usually for stub networks, Cyclops only displays the values w_{from}, since we would need a monitor at the stub to capture the other direction. Keeping both directions is important because it gives perspective on how the Cyclops eye slices routing through its neighbors, as well as how its neighbors point routes towards it.

All the above mentioned information, the AS links, the last BGP update reporting each link, the link weights, as well as the AS relationships and classification, is imported to the main Cyclops database on a daily basis to provide input into Cyclops.

7.2 Cyclops Web Interface

Cyclops web interface is designed to provide users a quick snapshot of AS connectivity surrounding the eye of the Cyclops, a given AS-x, within a given time window. It also complements Cyclops visualizer in scaling the topology display by allowing one to view a complete list of all the neighbors of large ISP ASes whose neighbor counts may go as high as thousands which makes visualization infeasible. The inputs to the web interface include the eye of the Cyclops, AS-x, the time period $[t_0, t_1]$ of interest, and a choice between showing all neighbor ASes and showing the connectivity changes only. In *connectivity* mode, a snapshot of the neighbors of x at time t_1 is displayed in the table. In *change-only* mode, all topology changes incident to x that occurred in the interval $[t_0, t_1]$ are displayed; the changes include both new AS links and disappeared AS links.

As an example, Figure 9 shows the Cyclops web user interface, together with the connectivity listing for AS174 (Cogent, an ISP). The table includes information for each neighbor of AS174, such as ASN, AS name, AS type (stub, small ISP, large ISP, Tier-1), number of downstream ASes within ()'s, relationship with x (customer, provider, peer), node degree, link appearance and disappearance date, link lifetime, link weight, and last BGP message observed which contained the link[1]. The "Weight (to)" column shows the

[1]The value shown in the "Last BGP Message" column is the prefix in

link weight in the direction of AS174 to neighbor AS; the "Weight (from)" column shows the link weight seen in the direction from neighbor AS to AS174. To fill in the "Weight (from)" column for a neighbor requires a monitor being hosted in that neighbor AS. The *Avg.* value is the expected weight as described in Section 7.1; the *Diff.* value is the percentage of difference between the link weight on the "End Date" and the expected weight.

The neighbor listing shown in Figure 9 is in the order of the node degree (the number of connections each neighbor AS has); one can click on any other parameter to order the neighbor list by the values of that specific parameter. For example, if one wants to know whether the AS is involved in any route hijack, one can sort the list by link lifetime and see the AS links with shortest lifetime on the top. The last BGP message informs the user who originated the BGP update message that caused the links to appear. Other columns in the table help users observe neighbors by degrees, relationship or AS types.

Figure 10 shows AS-174 connectivity in *change-only* mode, where all topology changes incident to AS-174 that occurred in the time interval $[t_0, t_1]$ are displayed. In this specific case, 18 links changed during the period of [5/23/08, 5/29/08] (the table is truncated due to space limit). The first row shows a link that disappeared since 5/25/08 with a lifetime of 947 days; the eighth row shows a new link to AS10279 that was added on 5/27/08. Note that the "Weight (from)" column shows no value, this is because there is no monitor inside any of those neighbor ASes.

7.3. Cyclops and BGPmon

The system above illustrates the potential for BGPmon data. BGPmon provides the raw data used by Cyclops. By delivering data from more peers, clients such as Cyclops can offer better analysis and improved inference. By delivering data in real-time, BGPmon allows Cyclops to report potential problems and attacks in real-time. Finally, by scaling to vast numbers of clients, BGPmon is not restricted to Cyclops alone. Any number of other services can obtain the same raw data.

As the BGPmon system matures, we anticipate a large mesh of BGPmon collectors and BGPmon chains. Large numbers of peers connect directly to one or more BGPmon collectors. The collectors in turn establish chains to other BGPmon instances and clients then connect to one of these instances. By providing a robust mesh, BGPmon instances can provide data from large numbers of peer routers and serve a large number of clients.

BGPmon intentionally does not process data. BGPmon instances simply collect and report data. But one can clearly envision scenarios where more robust data processing is required. BGPbrokers take this raw data as input and provide filtered output. For example, a BGPbroker might pass only BGP path changes to a client and filter out all other data such as keepalive messages and duplicate route announcements. Cyclops may one day receive BGP data from just such a BGPbroker. Cyclops primarily requires path changes and thus a broker that filters out all updates other than path changes could simplify the processing for Cyclops.

that update, embedding a hyperlink to the message.

The distinction between a BGPbroker and client is some arbitrary. To a BGPmon instance, the distinction is irrevelant. BGPmon simply provides unfiltered data via a TCP stream. This data may be fed directly to a tool such as Cyclops or may be fed a collection of BGPbrokers which provide filtered data to tools such as Cyclops. Finally, Cyclops itself may be viewed as a broker in the sense that it filters BGP and reports only events that deviate from the expected behavior.

8. Conclusions and Future Work

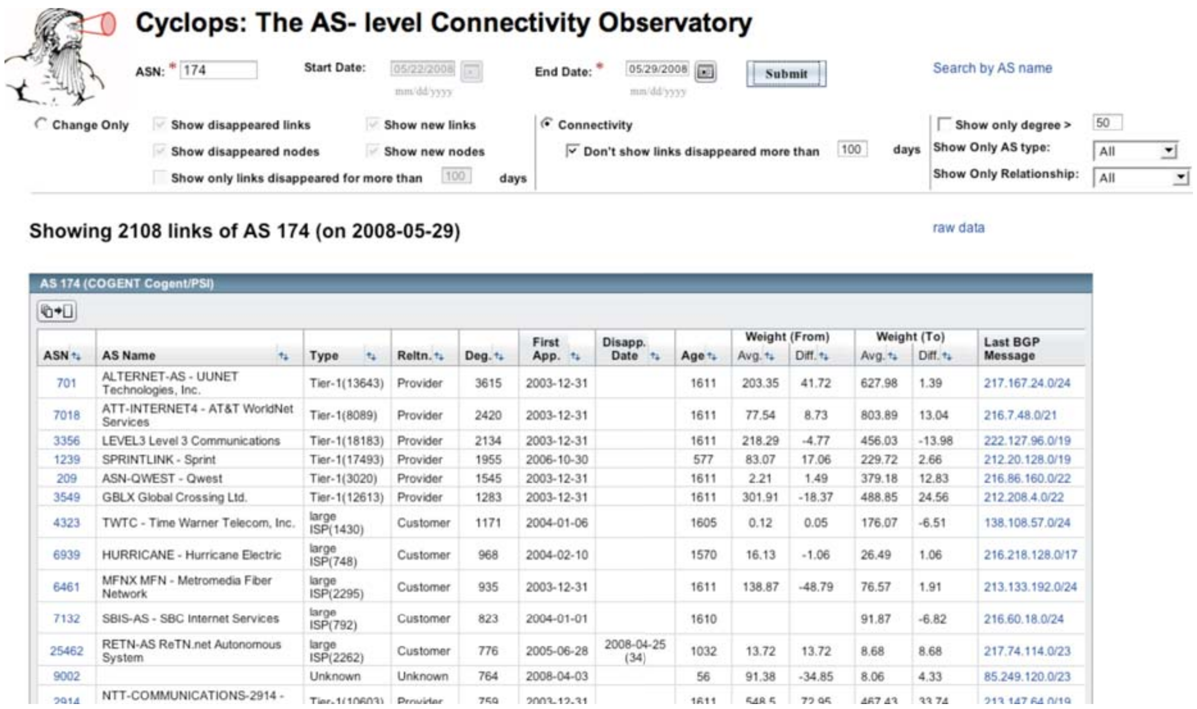
This paper presented the design of a new BGP monitoring system and showed how clients such as Cyclops can make use of the resulting data to better understand BGP behavior.

Based on discussions and user feedback, we are very encouraged by the deployment thus far. We have also learned a few critical lessons that have influenced the next release of BGPmon, BGPmon version 7. Chief among these lessons are a revised approach to handling slow clients, a revised approach to managing routing tables, and a new addition for integrating BGPmon into existing systems such as RouteViews. Each of these version 7 enhancements is discussed below.

8.1 BGPmon v7: Handling Slow Clients

Our current implementation of BGPmon (version 6) drops slow clients, as described in Section 4. It is essential that some action be taken to address the problem of slow clients. If no action is taken, a slow client can cause the BGPmon queues to overflow and eventually data would be dropped. This is particularly problematic if most clients could read at a high rate and receive all the data, but a few slow clients fill the queues and cause data loss. In BGPmon version 6, our solution is to identify and then terminate the slow clients. This has worked well in the deployment thus far.

However, a potential problem is that the slow client may simply re-connect and thus drive the overall system into a state of persistent oscillation. The system runs well until the slow client joins. The slow client then causes queues to build up and the client is eventually killed. The queue



Cyclops: The AS- level Connectivity Observatory

ASN: * 174 | Start Date: 05/22/2008 (mm/dd/yyyy) | End Date: * 05/29/2008 (mm/dd/yyyy) | Submit | Search by AS name

( ) Change Only — [✓] Show disappeared links, [✓] Show new links, [✓] Show disappeared nodes, [✓] Show new nodes, [ ] Show only links disappeared for more than 100 days

(•) Connectivity — [✓] Don't show links disappeared more than 100 days

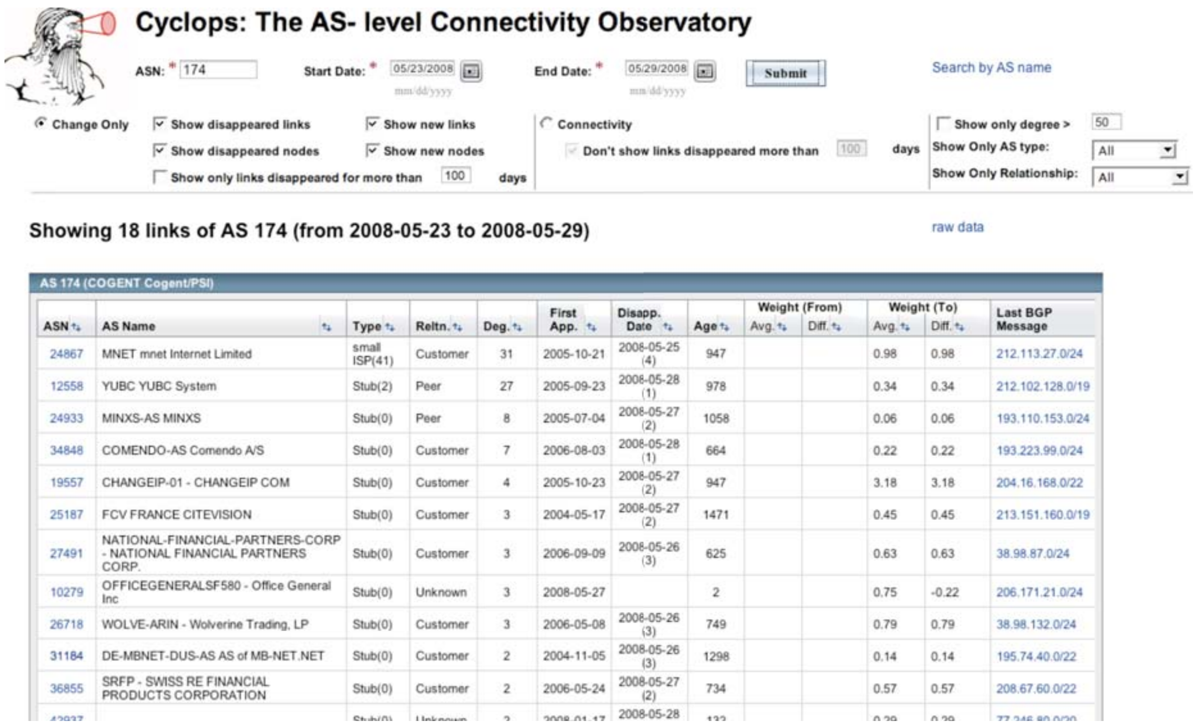
[ ] Show only degree > 50 | Show Only AS type: All | Show Only Relationship: All

Showing 2108 links of AS 174 (on 2008-05-29) raw data

AS 174 (COGENT Cogent/PSI)

ASN	AS Name	Type	Reltn.	Deg.	First App.	Disapp. Date	Age	Weight (From)		Weight (To)		Last BGP Message
								Avg.	Diff.	Avg.	Diff.	
701	ALTERNET-AS - UUNET Technologies, Inc.	Tier-1(13643)	Provider	3615	2003-12-31		1611	203.35	41.72	627.98	1.39	217.167.24.0/24
7018	ATT-INTERNET4 - AT&T WorldNet Services	Tier-1(8089)	Provider	2420	2003-12-31		1611	77.54	8.73	803.89	13.04	216.7.48.0/21
3356	LEVEL3 Level 3 Communications	Tier-1(18183)	Provider	2134	2003-12-31		1611	218.29	-4.77	456.03	-13.98	222.127.96.0/19
1239	SPRINTLINK - Sprint	Tier-1(17493)	Provider	1955	2006-10-30		577	83.07	17.06	229.72	2.66	212.20.128.0/19
209	ASN-QWEST - Qwest	Tier-1(3020)	Provider	1545	2003-12-31		1611	2.21	1.49	379.18	12.83	216.86.160.0/22
3549	GBLX Global Crossing Ltd.	Tier-1(12613)	Provider	1283	2003-12-31		1611	301.91	-18.37	488.85	24.56	212.208.4.0/22
4323	TWTC - Time Warner Telecom, Inc.	large ISP(1430)	Customer	1171	2004-01-06		1605	0.12	0.05	176.07	-6.51	138.108.57.0/24
6939	HURRICANE - Hurricane Electric	large ISP(748)	Customer	968	2004-02-10		1570	16.13	-1.06	26.49	1.06	216.218.128.0/17
6461	MFNX MFN - Metromedia Fiber Network	large ISP(2295)	Customer	935	2003-12-31		1611	138.87	-48.79	76.57	1.91	213.133.192.0/24
7132	SBIS-AS - SBC Internet Services	large ISP(792)	Customer	823	2004-01-01		1610			91.87	-6.82	216.60.18.0/24
25462	RETN-AS ReTN.net Autonomous System	large ISP(2262)	Customer	776	2005-06-28	2008-04-25 (34)	1032	13.72	13.72	8.68	8.68	217.74.114.0/23
9002		Unknown	Unknown	764	2008-04-03		56	91.38	-34.85	8.06	4.33	85.249.120.0/23
2914	NTT-COMMUNICATIONS-2914 -	Tier-1(10603)	Provider	759	2003-12-31		1611	548.5	72.95	467.43	33.74	213.147.64.0/19

Figure 9. A snapshot of Cyclops web interface: connectivity of AS174.



Cyclops: The AS- level Connectivity Observatory

ASN: * 174 | Start Date: * 05/23/2008 (mm/dd/yyyy) | End Date: * 05/29/2008 (mm/dd/yyyy) | Submit | Search by AS name

(•) Change Only — [✓] Show disappeared links, [✓] Show new links, [✓] Show disappeared nodes, [✓] Show new nodes, [ ] Show only links disappeared for more than 100 days

( ) Connectivity — [✓] Don't show links disappeared more than 100 days

[ ] Show only degree > 50 | Show Only AS type: All | Show Only Relationship: All

Showing 18 links of AS 174 (from 2008-05-23 to 2008-05-29) raw data

AS 174 (COGENT Cogent/PSI)

ASN	AS Name	Type	Reltn.	Deg.	First App.	Disapp. Date	Age	Weight (From)		Weight (To)		Last BGP Message
								Avg.	Diff.	Avg.	Diff.	
24867	MNET mnet Internet Limited	small ISP(41)	Customer	31	2005-10-21	2008-05-25 (4)	947			0.98	0.98	212.113.27.0/24
12558	YUBC YUBC System	Stub(2)	Peer	27	2005-09-23	2008-05-28 (1)	978			0.34	0.34	212.102.128.0/19
24933	MINXS-AS MINXS	Stub(0)	Peer	8	2005-07-04	2008-05-27 (2)	1058			0.06	0.06	193.110.153.0/24
34848	COMENDO-AS Comendo A/S	Stub(0)	Customer	7	2006-08-03	2008-05-28 (1)	664			0.22	0.22	193.223.99.0/24
19557	CHANGEIP-01 - CHANGEIP COM	Stub(0)	Customer	4	2005-10-23	2008-05-27 (2)	947			3.18	3.18	204.16.168.0/22
25187	FCV FRANCE CITEVISION	Stub(0)	Customer	3	2004-05-17	2008-05-27 (2)	1471			0.45	0.45	213.151.160.0/19
27491	NATIONAL-FINANCIAL-PARTNERS-CORP - NATIONAL FINANCIAL PARTNERS CORP.	Stub(0)	Customer	3	2006-09-09	2008-05-26 (3)	625			0.63	0.63	38.98.87.0/24
10279	OFFICEGENERALSF580 - Office General Inc	Stub(0)	Unknown	3	2008-05-27		2			0.75	-0.22	206.171.21.0/24
26718	WOLVE-ARIN - Wolverine Trading, LP	Stub(0)	Customer	3	2006-05-08	2008-05-26 (3)	749			0.79	0.79	38.98.132.0/24
31184	DE-MBNET-DUS-AS AS of MB-NET.NET	Stub(0)	Customer	2	2004-11-05	2008-05-26 (3)	1298			0.14	0.14	195.74.40.0/22
36855	SRFP - SWISS RE FINANCIAL PRODUCTS CORPORATION	Stub(0)	Customer	2	2006-05-24	2008-05-27 (2)	734			0.57	0.57	208.67.60.0/22
42937		Stub(0)	Unknown	3	2008-01-17	2008-05-28	133			0.29	0.29	77.246.80.0/20

Figure 10. AS174 connectivity changes during the period of [5/23/08, 5/29/08].

then quickly drains when the slow client is killed. Note that the queue contains at least one update that has been read by everyone except the slow client. When the slow client is killed, that update can be discarded. In our experiments thus far, a typical slow client has hundreds of updates that are waiting only for the slow client; killing the slow client immediately removes these updates and frees hundreds of slots in the queue. But oscillation occurs if the slow client immediately connects. The queue of unread updates begins to build again as soon as the slow client joins and the cycle repeats. One can easily imagine a poorly written slow client that automatically reconnects anytime it is disconnected.

An alternate approach is to better manage, but not kill the slow clients. In BGPmon version 7, the slow client is not deleted from the system. Instead, slow clients are forced to skip messages. From a queuing standpoint, the effect is similar to killing the slow client and works as follows. When BGPmon determines a client is reading updates too slowly, all messages that have yet to be read by that slow client are immediately marked as read. The client is informed it has missed several messages, but it is allowed to continue. If the message loss is unacceptable to the client, we leave it to the client to terminate the connection.

For example, suppose 100 messages have been read by everyone except a slow client. Rather than waiting for the slow client to read the 100 messages, the client is sent an XML message indicating that 100 messages will be skipped. All 100 unread messages that are waiting only on the slow client are immediately marked as read and removed from the queue.

The same pacing rules discussed for killing clients apply to skipping clients forward. The algorithm works exactly as described in Section 4, but rather than terminating the TCP connection BGPmon instead skips the client forward. A command line interface allows the BGPmon administrator to set minimum client rates. The BGPmon administrator can also terminate clients that fall behind too often, but this now becomes a decision by an administrator rather than an automated behavior.

8.2 BGPmon v7: Sending Routing Tables

BGPmon version 6 sends both incremental updates and periodic RIB table transfers in the same XML stream. The periodic table transfers are added to the stream using the BGP Route Refresh capability. This works as follows. At periodic intervals, BGPmon sends a route refresh request the ISP router. In response to the request, the ISP router re-announces all routes in its table. This allows clients who have recently joined the XML stream to learn the full table. It also minimizes the possibility of monitoring errors since the re-announced table comes directly from the ISP router.

However, periodic route refreshes have two major negative consequences. First, ISP routers may not be willing to periodically resend the entire table. Sending a route refresh requires processing and bandwidth from the ISP router. In some cases, this added load may be considered too costly for the ISP router. This problem is easily solved by storing the routing table at BGPmon. Rather than requesting a table from the peer, BGPmon simply re-announces its copy of the routing table. No action by the peer router is required. In fact, the peer router is not aware that BGPmon is re-announcing the table. All updates related to this BGPmon generated table transfer are clearly labeled so a client can easily distinguish an actual update by the peer router from a simulated table transfer generated by BGPmon. This functionality already exists in BGPmon version 6 and we believe this will become the standard way to periodically re-announce the routing tables.

Second and perhaps more problematic, the re-announced table is added to the XML stream and received by all clients. This is desirable for new clients who would like to learn the full table and may also be useful for existing clients who want to refresh their state, but the re-announcement adds a vast number of messages. Clients who do not want (or need) a routing table are forced to receive and ignore periodic bursts of table transfer messages. Worse still, these periodic bursts increase the number of updates by an order of magnitude; a high price to pay if these updates are to be simply ignored by many (if not most) clients.

Our solution in BGPmon version 7 is to introduce a second XML stream. The first XML stream contains only BGPmon updates. As updates are received from peer routers, they are immediately added to this stream as discussed above. A new second XML stream will contain only periodic routing table snapshots. The BGPmon administrator (not the clients) chooses how often routing table snapshots are announced via this stream. In this model, a new client who wants both an initial table and incremental updates subscribes to both the standard update stream and the periodic table stream. Once an table transfer is received over the periodic table stream, the client can disconnect from this stream and receive only the BGP updates sent in the first stream. Clients who do not want periodic table transfers will not be subscribed to the periodic table transfer stream and thus will not receive them. Clients who do seek periodic table transfers can subscribe to this stream whenever a table transfer is needed. As with BGPmon version 6, the clients who have subscribed to the stream must wait for the next periodic transfer interval before the table transfer is sent by BGPmon.

8.3 Integration With Existing Monitors

Along with the RouteViews team, our objective is to replace this routing software with BGPmon. However, the transition can be challenging. Ideally, one would like to gradually phase in the new BGPmon collectors before disconnecting the existing collectors. Existing collectors use open source routing software and log updates to files. In a transition to BGPmon, the monitored ISPs routers could peer with both the existing collector and a new BGPmon collector. But this places a burden on the monitored ISPs to modify configuration files and adds additional load to the production BGP routers being monitored. We instead have developed an alternate incremental deployment plan.

In order to phase BGPmon in, we are working to modify the existing collectors slightly so they feed updates to BGPmon rather than writing updates to files. An existing collector already writes BGP updates to a file. If one thinks of a file as simply another stream, there is no conceptual reason why these updates can't be written to a stream and BGPmon could read from this stream. In other words, the existing collector opens the BGP connection to the ISP router and begins receiving updates. Each update is written to a stream instead of written to a file. BGPmon then processes the update stream. Rather than directly receiving an update from a peer, BGPmon indirectly receives the update via this stream. BGPmon labels the updates, converts it to XML, publishes to the event stream, and so forth. This allows us to test BGPmon on the full RouteViews update load. The only part of the BGPmon code not tested is the actually opening on the peering session itself, which can be thoroughly evaluated using other means.

Once BGPmon has proven itself handling the full RouteViews update load, BGPmon can replace the existing open source collection software. Note the ISP routers do not know or care what underlying software is used to establish and maintain a peering session. We believe this approach will allow a rapid rollout of BGPmon into RouteViews and provide a potential model for other monitoring sites to deploy BGPmon.

Overall, we believe BGPmon represents an important change in how BGP route monitoring is accomplished in the Internet. We hope that the addition of BGPmon will make it much simpler for researchers and operators to obtain BGP data and the addition of widely available real-time BGP data will lead to the development of new tools for better understanding Internet routing.

9 Acknowledgments

Many of the design insights and current BGP peering sessions would not have been possible without the help of the Oregon RouteViews team, the UCLA Internet Research Lab, and the Networking Research Lab at University of Memphis, and the many contributors from the Colorado State Network Security Group.

References

[1] Bgp monitoring system. `http://bgpmon.netsec.colostate.edu/index.html`.

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

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

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

[5] Netviews. `http://netlab.cs.memphis.edu/projects_netviews.html`.

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

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

[8] 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.

[9] Y.-J. Chi, R. Oliveira, and L. Zhang. Cyclops: The Internet AS-level Observatory. In *ACM SIGCOMM Computer Communication Review*, 2008.

[10] N. Feamster, R. Johari, and H. 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.

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

[12] M. Lad, R. Oliveira, D. Massey, and L. Zhang. Inferring the Origin of Routing Changes using Link Weights. In *Proc. IEEE ICNP*, 2007.

[13] D. Pei, M. Azuma, D. Massey, and L. Zhang. Bgp-rcn: improving bgp convergence through root cause notification. *Comput. Netw. ISDN Syst.*, 48(2):175–194, 205.

[14] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. Hlp: a next generation interdomain 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.

[15] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. 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.

[16] 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. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 183–195, 2002.

[17] M. Welsh, D. Culler, and E. 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.