BGP-Inspect - Extracting Information from Raw BGP Data

Dionysus Blazakis
Institute for Systems Research
University of Maryland
College Park, Maryland 20742
Email: dblaze@isr.umd.edu

Manish Karir
Networking Research and Development
Merit Network Inc.
Ann Arbor, Michigan 48104
Email: mkarir@merit.edu

John S. Baras
Institute for Systems Research
University of Maryland
College Park, Maryland 20742
Email: baras@isr.umd.edu

Abstract— **While BGP routing datasets, consisting of raw routing data, are freely available and easy to obtain, extracting any useful information is tedious. Currently, researcher and network operators implement their own custom data processing tools and scripts. A single tool that provides easy access to the information within large raw BGP data-sets could be used by both communities to avoid re-writing these tools each time. Moreover, providing not just raw BGP messages, but some commonly used summary statistics as well can help guide deeper custom analyses. Based on these observations this paper describes the first steps towards building a scalable tool. We describe the various techniques and algorithms we have used to build an efficient generic tool called BGP-Inspect. When dealing with large datasets, dataset size, lookup speed, and data processing time are the most challenging issues. We describe our implementations of chunked compressed files and B+ tree indices that attempt to address these issues. We then provide an evaluation of our implementations. Finally, we provide some example scenarios and case studies where BGP-Inspect can provide useful insight into the management and operation of complex BGP based networks. An efficient and flexible back-end custom BGP message database, coupled with an intuitive and easy to use web-based query front-end makes BGP-Inspect a unique and powerful tool.**

I. Introduction

Route Views [1] and RIPE [2] provide some of the most extensive routing datasets available. Each of those archives provide raw data collected from peering sites in Europe and North America. The Route Views archives date back to 2001, while the RIPE archives date back to 1999. Together, these archives comprise the most extensive collections of data related to the operation and performance of the BGP routing protocol on the Internet.

While these sources together provide roughly 66G of compressed raw data, there is no easy way for the network operations or research community to extract even basic information quickly from the raw data. Obtaining even basic information from the data requires custom scripts or tools that extract, collect, and analyze the data. Answering simple questions, such as how many unique prefixes were announced by an AS over the past month, requires the development of custom tools by each network operator. Similarly, network researchers attempting to conduct analysis on these datasets face the same problems. Moreover, most of the analyses performed on these datasets asks very similar kinds of question.

These basic observations regarding the utility of raw BGP routing data motivated us to implement a tool called BGP-Inspect. BGP-Inspect provides the network operations and research community preprocessed data in an easy to use and consistent manner. It also provides some basic statistics on queried data. While it does not answer all questions that might be asked, we attempt to build into BGP-Inspect the ability to generate statistics for some common queries which can in turn help to guide deeper analyses. Though BGP-Inspect is still under development, the preliminary release attracted a large amount of interest from the networking community. It also provided us with valuable insight into the needs of researchers and network operators.

While attempting to build this tool, we were faced with several challenges. The primary challenge is building a system that can scale to handle such large raw input datasets. Processing large amounts of information requires that we use carefully chosen algorithms and techniques to ensure a scalable solution. In this paper we describe some of the techniques that helped us to implement a scalable BGP dataset processing and query system. We hope that other researchers can make use of our experiences in their efforts to extract useful information from large BGP routing datasets.

1-4244-0143-7/06/$20.00 ©2006 IEEE.

The rest of this paper is organized as follows. Section II describes some related work in this field, both projects that provide BGP routing data as well as some projects that have run analyses on this data. Section III describes the overall architecture of the core message processing and query system, as well as specific techniques that we used to enhance the scalability of our tool. We also briefly describe the user interface that allows users to quickly query the processed data. Section IV provides results of our experiments used to validate some of optimization methods that we use to build our processed BGP messages database. Section V describes some simple case studies that illustrate how BGP-Inspect can be used to help in BGP routing analysis and network operational tasks. Finally, Section VI presents our conclusions and outlines some of our future work.

II. Related Work

There are various public archives where researchers can obtain BGP routing datasets. The most popular and extensive archives are maintained by the University of Oregon's Route Views Project [1]. The Route Views data archives contains data gathered from 40 BGP peers, at various prominent locations on the Internet. Together, these archives provide the most comprehensive collection of BGP routing datasets.

While the Route Views datasets are composed primarily of measuring points in North America, the RIPE archives [2] are composed of messages collected from peers in various parts of Europe. Both RIPE and Route Views provide access to raw datasets. Additionally, RIPE provides limited 3 month datasets via a query interface as well. Using the query interface, researchers can obtain raw update messages for their analysis. A similar service is provided by the BGP Monitor project at MIT [3]. However, both of these projects simply provide access to the raw messages. There is no attempt to provide prepackaged queries or summary statistics based on the raw data that can guide network operators and researcher in their analysis. While it is possible to use these services to obtain raw BGP data about a particular prefix or AS, we then need to write scripts and parsers to answer simple questions such as "how many unique prefixes were announced by a prefix?", or "how many times has the origin AS changed for a particular prefix?". BGP-Inspect is an attempt to provide answers to these questions efficiently. Our goal in building BGP-Inspect is to not only allow users to query BGP archived data, but to provide some simple analysis and statistics on that data as well. This can serve as a useful guide and a starting point for researchers and network operators in their own detailed analyses.

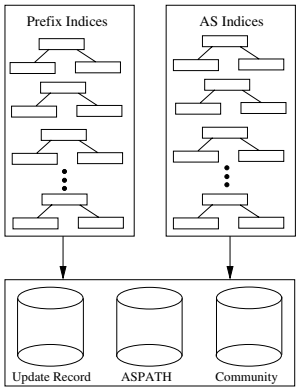

Fig. 1. System Architecture

In addition to the raw dataset archives, there are various projects that attempt to analyze BGP datasets in order to gain insight into its operation on a large scale, as well as attempts to detect anomalous events. BGPlay [4] displays animated graphs of the routing activity of a specific prefix. BGPlay animations aide in the understanding of how various updates affect a specific prefix. Similarly, the LinkRank Project [5] at UCLA attempts to analyze BGP datasets in order to visualize BGP routing changes as well to deduce the relative importance of various AS paths. Various other projects such as [6] [7] [8] [9] [10] [11] have also attempted to analyze BGP performance. However, these and other similar studies tend to focus directly on the specific analysis they are performing and do not put any emphasis on building generic tools. By building BGP-Inspect, we are attempting to provide a generic tool that can be easily used and extended by the network operator as well as the network research community. Some of the techniques and experiences that we have described can also provide valuable guidance towards building other tools in a scalable manner.

III. Data Mining Large BGP routing datasets

BGP-Inspect is composed of two parts. The first is the core BGP message processing system that is responsible for building a scalable database that stores BGP update messages - BGPdb. The second part of BGP-Inspect is the query API, the web interface, and the summary statistics, that allows users to extract information from the message database.

A. Scalable BGP Database - BGPdb

Though other projects such as [2] have been able to use a standard database such as MySQL as the back-end for their system, discussions with the operators of that system, our own initial experiments, as well as anecdotal evidence revealed that it was difficult to scale even a simple database containing just information from BGP messages. A generic database such as MySQL cannot be optimized to be able to effectively handle the extremly large amounts of raw BGP data. But we did'nt just want to be able to simply store raw BGP messages, we wanted to provide useful summary statistics on the data as well. This led us to consider a custom design which we would optimize to be specially suitable for the properties of BGP messages. Our goal was to optimize query time performance, and at the same time be able to scale to handle extremly large volumes of data.

The architecture of our design is illustrated in Figure 1. The BGPdb core consists of 3 databases: the update message database, the AS path database, and the community database. In addition, there are per prefix and per AS indices that reference these databases. The indices help speed up queries of these databases.

In order to build BGPdb we needed to process large amounts of BGP routing data. Doing this efficiently requires us to optimize several different components of the overall system. Some of the key design factors are disk usage, data loading efficiency, query processing time, and usability of the processed data.

To optimize disk usage, we eliminate redundancy in the BGP routing data. From experience, we know that large numbers of update messages share common fields, therefore in BGPdb we store these fields separately from our processed versions of the update messages. To further minimize the disk usage, we use a chunked compressed file to store these databases.

Data loading and processing is an important component of the overall system. In order to process the existing BGP dataset and convert it into our format, we need to process each update message, store the relevant information from it into our databases, and finally update our indices. To make this process more efficient we use simple caches that store recently used information. This speeds up the process of locating the relevant entry in the ASPATH and COMMUNITY databases.

Query processing time is a key factor of the system. The query interface is the public interface to BGPdb; it will be used directly only by those users performing custom analyses. Once again, we utilize the fact that most queries are either prefix or AS based to build B+ tree indices that speed up queries.

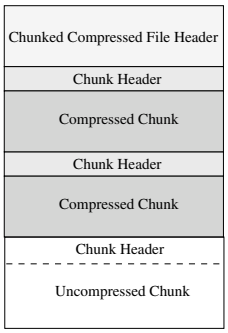

Fig. 2. Chunked Compressed File Structure

Last but not the least, we need to pay particular attention to the usability of the processed data. We need to ensure that our processing system does not abstract out or delete any information from the original data that might be important.

In the following subsections we provide further details regarding these key components of BGPdb.

1) Chunked Compressed Files: One of the first steps towards converting a large volume of data into a more manageable size is to eliminate redundancy. In the case of BGP routing update messages, there is an enormous amount of redundant data. Often, the only difference between update messages is in the timestamps at which the messages was transmitted. This redundancy implies that the data is extremely compressible.

The raw data archives generally store data in a compressed format, however in order to perform any analysis we need to both uncompress the data and run queries to extract the relevant portions. There is a trade off between making data easily available and disk usage.

In an attempt to minimize the amount of disk space and at the same time have the ability to extract relevant data quickly, we have implemented *chunked compressed files*. A chunked compressed file is a file that is divided into sections called chunks. Each chunk is individually compressed. The format of a chunked compressed file is shown in Figure 2. Each chunk header contains information that describes the following compressed chunk. This allows us to quickly identify which compressed chunk contains the relevant information. The last chunk is maintained uncompressed to speed up data insertion. The file header contains a reference to this last chunk. Though chunked compressed files give us slightly worse compression ratios than one would get by compressing the entire file, they provide us with much needed flex-

ibility by allowing us to navigate and access selected portions of the file without having to decompress parts that are not required.

Data insertion into a chunked compressed file is a simple operation. Data is only inserted into the last chunk of the file. As this chunk is uncompressed the insert operation is simple and does not require any other modifications to other parts of the file other than the file header. Once the last chunk reaches a preset limit of how many records can be inserted into a chunk, that chunk is then compressed and a new uncompressed chunk is created for new entries.

Querying for data in a chunked compressed file is also fairly simple. Knowing the record id which we are trying to access, we start at the top of the chunk file. We step through all the chunk headers and find the chunk that contains the record id. Once the relevant chunk is located, only this chunk is read into memory and uncompressed. Once uncompressed, we can easily find the record corresponding to the record id. It should be noted that though this is the current implementation, it is perhaps not the best way to search through this a large file; this linear search can be a performance and scalability bottleneck. We are looking into alternate ways of organizing this information. One method might be to create per-week chunked files, this will limit how large each file can get. This method is similar to the approach we used to limit the size of our B+ tree indices described in the next subsection.

The choice of the chunk-size is a key parameter in constructing an efficient chunked compressed file system. The larger the chunk-size the better the compression ratio would be. However, there would be a degradation in query performance as a larger chunk would need to be decompressed to locate a single record. In the next section we describe some experiments and discuss how sensitive the achieved effective compression ratios are to the choice of this parameter. We also discuss the impact this has on query processing.

2) B+ Tree Based Indices: Once we have reduced the redundancy in the dataset, the next step is implementing an indexing scheme to efficiently find and use meaningful subsets of this data. In the case of BGP update messages, finding messages originating from a given AS or advertising a given prefix are both examples of meaningful subsets. We utilize the fact that common queries are mostly prefix based or AS based to build our indices. In addition, we realize that most queries will have a time component. After some preliminary experiments with various data structures, we settled on B+ trees as the building blocks of our indices.

A B+ tree is a popularly used indexing scheme designed to allow quick insertion and retrieval of key/value pairs [12]. More importantly, B+ trees are specially useful for on-disk access and are used in many file systems. They have been extensively studied and are useful for quick lookups. A B+ tree is similar in structure to a B tree except that it contains a secondary index. This second index allows linear traversal of leaf nodes, which is suitable for range queries.

Our implementation of the B+ tree uses the date field of the BGP update message as the key. The value stored along with the key is a record id number. This record id references the actual update message which is stored in the chunked compressed file described in the previous section.

Based on our expected common case queries, we build two separate sets of B+ tree indices. One for prefix based queries and the other for AS based queries. A separate index is generated for each unique prefix, as well as for each AS. As new update files are loaded, each update message is processed in turn. The B+ tree corresponding to each prefix referenced in a particular update message is updated to reflect the new entry. Similarly, the origin AS is computed for announce messages, and the B+ tree index for that AS is also updated.

A simple prefix or AS based query would specify a time range for which update messages were required. Based on the query prefix or AS, the appropriate B+ tree index is accessed, and the relevant record ids are obtained by querying the index using time as the key. These records are then read from the corresponding chunked compressed files.

3) Input Data Processing and Caching: Due to the large volumes of data that we need to process, the data processing that we perform on the raw input data must be optimized.

Input processing consists of reading each update message from the raw BGP update message files. A lookup is performed to determine which peer originated that message. This lookup determines which database we will use to store the final processed message. We have 3 separate hash tables to cache frequently used values for each of the 3 databases: update record, ASPATH, and COMMUNITY. From the input message we extract the ASPATH and COMMUNITY attributes. A lookup is performed into the the 2 hash tables that store these values to determine whether this entry has recently been used. If it has, the hash tables provides the record ids of the matching entries in the databases. Once we have

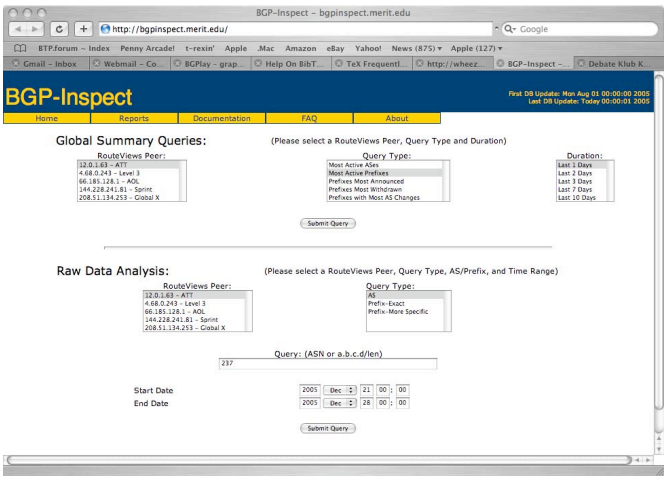

Fig. 3. BGP-Inspect Query Interface

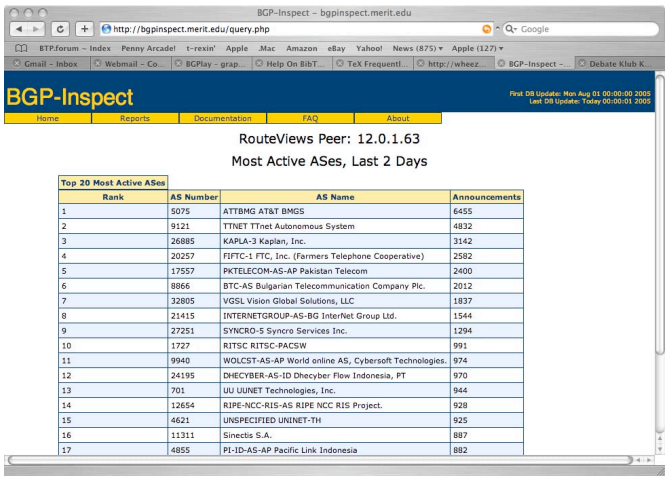

Fig. 4. Query Results for Most Active AS Numbers Query

these ids a third hash table lookup is performed on the entire update record, which includes these record ids. This lookup determines if the update record exists in the update record database or if it needs to be added. This lookup returns a single record id which identifies our re-constituted update record.

Once we have either inserted the update message into the database, or determined which record id indicates the entry in the database where it is stored, we then proceed to update our B+ tree indices. From each raw input update message we extract the affected prefix, the origin AS, and the time of the announcement. These values, together with the record id obtained from the database insert, are used to update the B+ tree. The prefix/origin AS indicates which B+ tree we should use, the time field is used as the key, and the record id is used as the value that is associated with that key. In section IV we discusses some experimental results that indicate the effectiveness of our caching scheme.

B. BGP Database Query Processing and Statistics

The overall design of BGP-Inspect has been motivated by the need to make queries efficient. The BGP-Inspect user interface presents users with the option of running 2 different types of queries. The first type of query is called a global query, the second type of query is called a raw data analysis query. A global query is a query that summaries data across a wide range of data, whereas a raw data query is a query that seeks detailed information about specific AS numbers of prefixes. The basic query web front-end is shown in Figure 3. The top portion of the page displays the interface for global queries, while the bottom half is for raw data queries.

1) Global Summary Queries: BGP-Inspect currently provides a basic set of five global queries which can be run over a variety of time intervals. These queries present a data summary distilled out of large amounts of data. The currently supported queries are:

- Most active AS numbers
- Most active prefixes
- Prefixes most announced
- Prefixes most withdrawn
- Prefixes with the most number of origin AS changes

A global query is composed by selecting a routeviews peer, a query type, and a duration. For example a valid query might be: ”as seen by Level 3, what were the most active AS numbers over the last 3 days”. This query will result in a reply containing a table that lists the top twenty most active AS numbers as observed on the routeviews peering session with Level 3. Figure 4 shows an example of the result of this query. The table in the figure shows the top twenty most active AS numbers terms of the number of update messages generated by them. The table lists the AS number, the AS name, as well as the total number of announce messages from that AS. A similar results page is generated for the other four types of queries as well. As the other queries are all essentially prefix based queries the resulting table lists, the top twenty prefixes in each category, the total number of announce messages for that prefix, the number of withdraw messages, as well as the total number of times the origin AS of that prefix changed.

Global summary queries require us to collect statistics from a very large number of entries in the database. For example, a simple query of the most active AS over the last seven days would require us to access

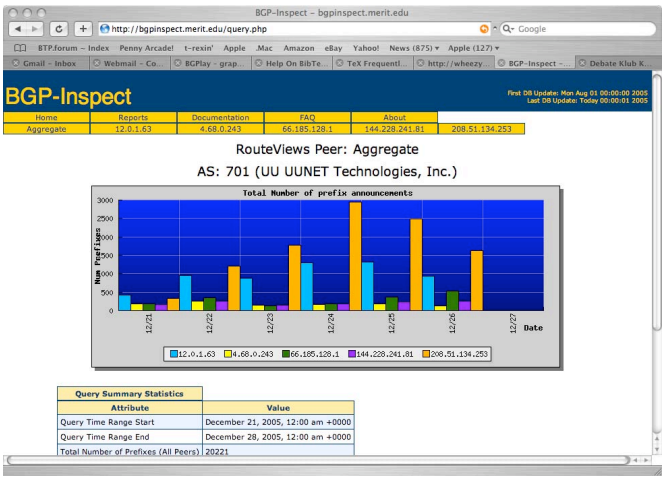

Fig. 5. Query Results Page for Specific AS Number Query

database files for each AS number over all seven days for each routeviews peer. However, the static nature of these queries helps us address this difficult problem. These query results remain the same across all users. They are presenting a global overview of BGP activity. Based on this observation we can pre-compute the results for these queries each time the database is updated with new data and save the results in a separate file for quick lookups at query time. The current version of BGP-Inspect available at http://bgpinspect.merit.edu updates the BGP message database once per day. These statistics are updated automatically right after the database update completes.

2) Raw Data Analysis Queries: Raw data analysis queries can be of three different types:

- AS number
- Prefix
- Prefix more specific

These queries are composed in the bottom half of the web based query interface. Users select a routeviews peer, the query type, enter the query value in the text field, and finally select the start and end times for which this query should be run. For example a valid query might be: ”as seen by Level 3, what were the update message announcements originated by AS 237 between July 25th and August 1st 2005”. This query will result in a reply that lists all of the update messages stored in the message database. In addition a summary statistic table is shown at the top that summarizes some basic properties of the reply dataset. A simple graph at the top of the page presents some basic per day activity information about AS 237. An example of the resulting page from this query is shown in Figure 5. The figure shows that AS 237 announced a total of 45 unique prefixes in a total of 137 update messages during the query duration. Some of the other dynamically computed summary statistics include, total number of update messages in query reply, number of unique prefixes and AS numbers, complete list of unique prefixes and AS numbers, minimum/maximum ASPATH length, and the query run time.

Raw data analysis queries are fundamentally different from global queries. Raw data queries dynamically access the BGP messages database in order to extract the relevant update messages. This is where our use of B+ trees enables us to rapidly run queries that would result in large response set. AS queries are generally simpler, we simply use the AS number provided by the user to determine directly which AS B+ tree we should query. Prefix queries can be more complex, for example users can ask for not only a specific prefix, but also any prefixes that are more specific than the one they query for.

In order to quickly locate both the B+ tree associated with a particular prefix along with those more specific prefixes, we use a binary index file naming scheme. Using our scheme, each per-prefix B+ tree file is named with the binary equivalent of that prefix. With this convention in place locating a particular prefix is fairly straight forward; we convert the query prefix into binary and use that as the B+ tree filename. For a query that was attempting to look for more specific prefixes, we can use the binary filenames to determine which prefixes are more specific than the query prefix.

Most queries have an associated time range component. We use B+ tree lookups to quickly determine the start and end record matches. In order to extract the complete set of all the records in this range, we simply follow the links between the records from the start time until we reach the end time. The following section presents experimental results that illustrate the performance of our B+ tree query processing implementation.

IV. Experiments and Analysis

In this section, we describe some experiments and measurements that we have conducted in order to evaluate the performance of our implementation of the BGP update message database. We used Route Views BGP data sets [1] for the month of January, 2005. Different portions of this month long dataset were used as necessary to evaluate different parts of BGPdb.

The Route Views datasets contain BGP information collected from 40 different peers. For our evaluation we focused on information from the following 5 major peers.

- 12.0.1.63 - ATT
- 4.68.0.243 - Level3
- 66.185.128.1 - AOL
- 144.228.241.81 - Sprint (Stockton)
- 208.51.134.253 - Global Crossing

Table I shows the number of messages processed for all 5 peers, for each week in our dataset. In all, the 4 week dataset contains 10.01 Million update messages from the 5 peers listed above. It is important to keep these per week message counts in mind as we examine the performance of BGPdb. For example, week 4 has more than double the number of messages compared to week 1 which needs to be factored into our analysis of the results.

Week	Messages	Cumulative Message Count
1	1.74M	1.74M
2	2.44M	4.18M
3	2.15M	6.33M
4	3.68M	10.01M

TABLE I
THE INPUT DATA SET

A. Evaluation of Chunked Compressed File Mechanism

Our first set of experiments attempted to examine the *chunk-size* parameter associated with our chunked compressed file implementation. The chunk-size parameter controls the number of records contained in each compressed chunk. The larger the chunk-size, the better the compression ratio will be, however, the larger chunk size will also result in a much larger uncompressed last-chunk. We ran experiments using the first 2 weeks of January 2005 as the input dataset. For each run we use a different chunk-size. For each experiment the compression ratio achieved is recorded. Figure 6 shows a graph of these recorded compression ratios. The graph shows how the filesize grows as a function of the number of messages processed. As expected, the two largest chunk sizes show the greatest compression ratios, and the results for a chunk-size of 2048 and 4096 records per chunk are very similar. Therefore, it seems a chunk-size of 2048 is adequate for our input data.

B. Evaluating B+ Tree Indices

The goal of the B+ tree indices is to quickly return a set of records queried by prefix or AS while minimizing the amount of extra storage space used. Once again, the conflicting goals of small storage space and fast lookups drive the parameter selection. Our second set

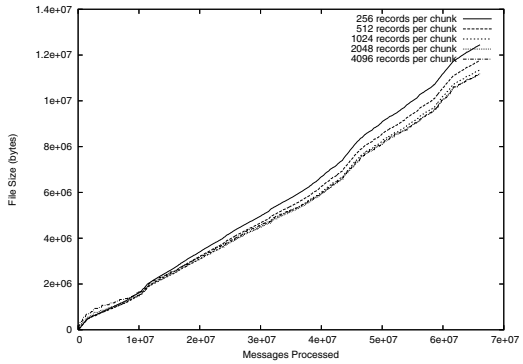

Fig. 6. Chunk Sizes vs. File Size

of experiments attempted to evaluate the sensitivity of index sizes and query speeds for 5 different values of block sizes. For this set of experiments we used the first 2 weeks of January 2005 as the input dataset.

For each block size, we load the 2 week input dataset and generate the B+ tree indices. After the load has been completed, the overall size of the database and the time to query the entire range of the database is recorded. For our experiments, we ran queries for the 5 largest AS indices over the entire 2 week dataset resulting in a total of 130K returned entries. The total time to return these entries was also recorded. Since the time to query the entire database is the worst case query, it represents a good benchmark to determine the speed of the B+ tree. The size of the database is measured to determine which block size minimizes disk use.

Block Size	Total Query Time	Database Size
512	0.241593 s	6,114M
1024	0.174691 s	6,092M
2048	0.147330 s	6,070M
3072	0.142825 s	11,802M
4096	0.133109 s	11,800M

TABLE II
B+ TREE QUERY PERFORMANCE

Table II summarizes the results of our experiments. As the block size increases the time to query the resulting indices also decreases. Using a larger block size results in more shallow B+ trees resulting in faster query times. However the size of the resulting dataset first decreases then increases as the block size increases. For smaller values of block size increasing the block size results in fewer nodes being created, resulting in savings in terms of size. For larger values of block size, increasing the block size only results in more empty slots in the tree. Based on the above data, the choice of 2048 as the block

size seems to be appropriate for our input dataset as it provides the best compromise between database size and query time.

C. Impact of Caching

Caching plays an extremely important role in enhancing the performance and scalability of BGPdb. This is because of the large similarities that exist between raw update messages inside the BGP datasets. BGPdb uses caching to speedup both record insertion as well as queries

Our initial caching scheme was a simple one level scheme where we would cache the entire BGP udpate message. This method required very large caches to achieve even small hit rates. To fix this shortcoming, the messages were broken into 3 parts and each part was cached on its own. We then ran experiments to evaluate and measure the effectiveness of our caching scheme using a dataset composed of the first two weeks of January 2005. The results of are shown in Table III.

In addition to the speed up in data insertion and queries, a cache also helps us to decrease disk usage. As we cache entries that we have inserted, future entries simply refer to the earlier entry. The impact shown in Table III upon storage costs is significant. In addition, Table III describes our cache hit ratios. The cache that stores COMMUNITY information from the raw BGP update messages has a 99% hit rate. The resulting file size was only 100K instead of the expected 126M. This clearly indicates the importance of using a cache.

Type	Hit Ratio	Bytes Saved	Database Size	Compression Rate
Update	%42.14	210M	93M	%75.82
AS Path	%69.24	139M	15.2M	%75.85
Community	%99.23	126M	100K	%93.37

TABLE III
SUMMARY OF CACHING IMPACT

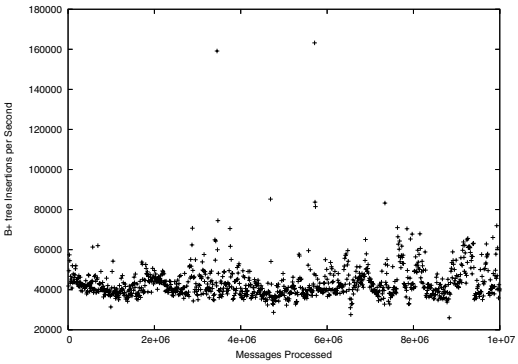

Fig. 7. B+ tree Insertions over time

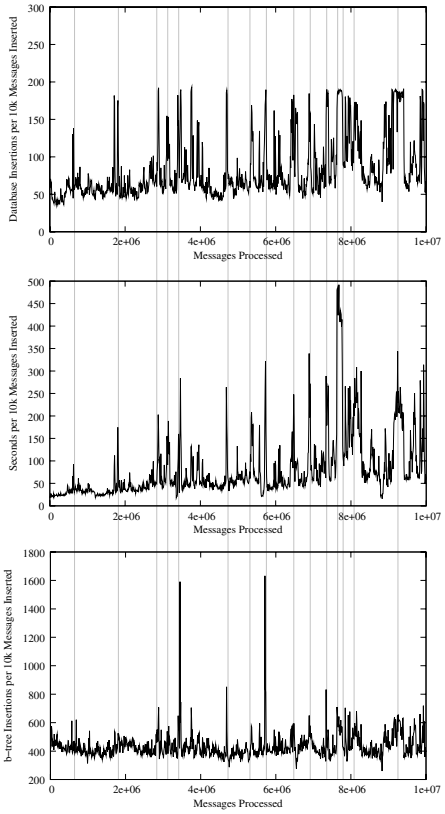

Fig. 8. Correlation between Insertion count and Processing Time

D. Scalability

While attempting to characterize the scalability of BGPdb, we attempted to determine if there was a slowdown in our message processing rate over time. Figure 8 shows the performance of BGPdb over a 1 month long input dataset. This dataset required roughly 24 hours to process and convert into our modified format. The graphs in the middle of Figure 8 shows the time taken to process blocks of 10,000 BGP udpate messages from the raw BGP dataset over time. Initially we were concerned by the large spikes in the graph, however detailed examination revealed that these spike were perfectly correlated with spikes in the number of database and index insertions. This implies that these spikes are actually a result of increased processing load on BGPdb by those particular update messages. This probably indicates BGP messages where a single message announced or withdrew a large number of prefixes. Based on Figure 8 BGPdb seems to be performing fairly well though it does not appear to be completely scalable yet. We are still investigating particular enhancements

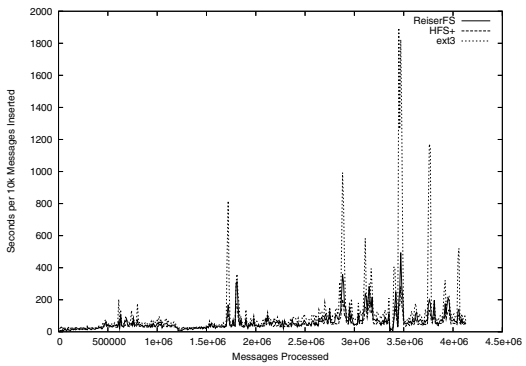

Fig. 9. Database Speed vs. File System

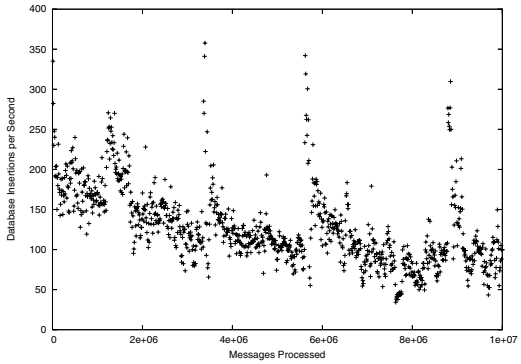

Fig. 10. Database Insertions over time

to the current system, as described in Section III.A.1 on chunked compressed files, which we believe will further improve performance.

The B+ tree based indices in BGPdb appear to scale extremely well. Figure 7 shows the rate at which messages were inserted into the B+ tree indices as a function of the total number of messages processed. The performance seems to be fairly steady implying stability in the performance over time.

E. External Factors

Building a scalable BGP dataset processing system is a difficult task. Aside from the need to use efficient techniques and algorithms we also need to be aware of and work around several external factors that can have an impact on the performance of our system. Since the processes are likely to be I/O bound, some of the key factors beyond the design that impact scalability are filesystem choice (ReiserFS, ext3, or HFS+), and disk speed. We ran several experiments to shed some light on these factors.

Figure 9 shows a comparison of the performance of BGPdb when different filesystems are used. We experimented with ReiserFS, HFS+, and ext3. It is interesting

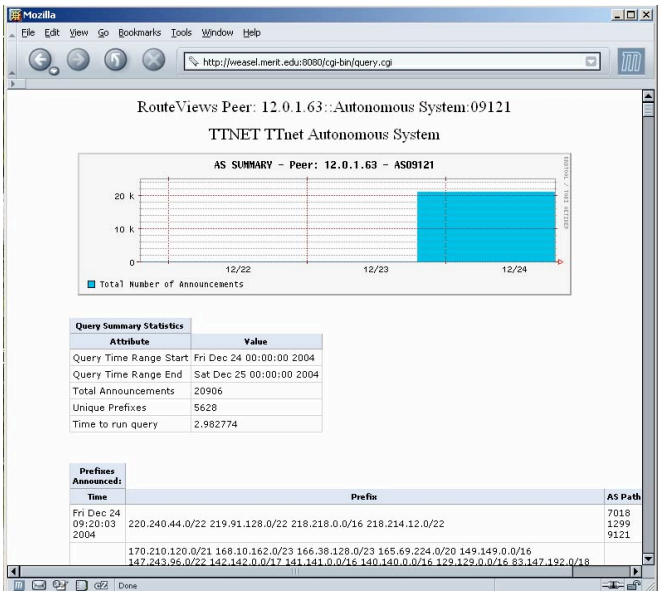

Fig. 11. Raw Data Query for AS9121

to note that the performance of ReiserFS and HFS+ is similar, which ext3 performs considerably worse.

The BGPdb message processing system uses a large number of index files. This brings to light some interesting tradeoffs which influence our system design. Initially, we based our design on maintaining single per-prefix and per-AS indices, however, as these files get large performance degrades. Next, we moved to using per-week index files for these indices. However, this now results in the creation of a large number of inodes. Interestingly, ReiserFS filesystem performance seems to deteriorate as a large number of files are created within a single directory. The next step was to create these per-week indices in separate directories, which resulted in a much more scalable system. Figure 10 shows the performance of BGPdb in terms of insertions/sec as a function of the number of messages. Weekly, boundaries can be clearly seen in the graph. Each time we start using a new directory performance improves and drops as the number of files in that directory increase.

V. Case Studies

In this section we illustrate with the help of specific examples how BGP-Inspect can be used to help identify anomalous routing events, as well as for forensic analysis. We use two examples, the AS9121 route leakage incident from December 24th 2004, and a prefix hijacking event on February 10th 2005.

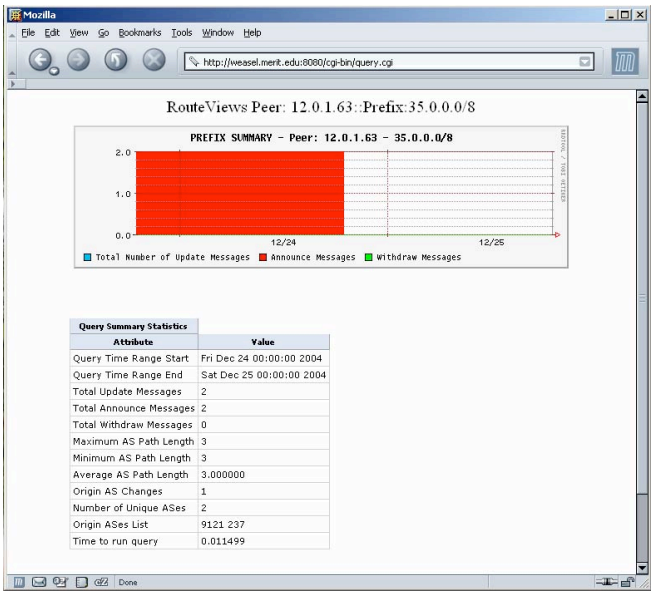

Fig. 12. Raw Data Query Prefix 35.0.0.0/8

A. *Route Leakage – AS9121 Incident*

The AS9121 route leakage incident is one of the most recent examples of widespread BGP instability caused by a single mis-configured BGP session. At roughly 9:20 UTC on December 24th, 2004, AS9121 began re-originating a large number of globally routed prefixes. We loaded data obtained from routeviews for this time period (Dec 22th and December 25th) into BGP-Inspect and attempted to determine how BGP-Inspect might have provided valuable information about this event.

One of the earliest signs of trouble emerges when we examine the global summary query results for ”Most Active AS Numbers” for the last two days. AS9121 shows up on this list at number 11. This is the first indication of potential problem. Next running a raw data query for routeviews peer, AT&T, for AS number 9121, over the time period December 23rd 2004 to December 25th 2004, we see that AS9121 is originating 5628 unique prefixes in a total of over 20K update messages. Figure 11 shows the query result page. This by itself does not indicate anomalous behavior, we need to know what the behavior of this AS is like on other days. Therefore, next we repeat the query for the time range December 22nd 2004 to December 24th 2004. This shows, AS9121 only originating 17 unique prefixes with only about 50-150 update messages per day. This clearly establishes that AS9121 is exhibiting anomalous behavior on December 24th.

The next step for a network operator might be to determine impact on their own networks. Returning to the main query page and running a raw data query on their specific prefix would return this information. Figure 12 shows an example where we ran a query for the prefix 35.0.0.0/8 for the time period December 24th 2004 to December 25th 2004. The top portion of the figure displays a summary graph that shows that there were two BGP announce messages for this prefix on December 24th. The bottom portion of the figure shows a table containing summary statistics for our query which clearly lists that this prefix was announced by 2 unique origin AS numbers. It also lists this prefix as being originated by AS 9121 and AS 237 over the query period. This shows us that this prefix was indeed affected by this incident, atleast as seen by AT&T. Repeating this query for other routeviews peers shows that only AOL and Sprint report this prefix as having been originated by 2 origin AS numbers. Level 3 and Global Crossing only see this prefix originated by AS 237 which is correct origination for this prefix. This example demonstrates that by running a sequence of queries using BGP-Inspect a network operator is easily able to obtain information about routing anomalies, whether they are impacted by it and some estimates about how widespread the anomaly is. This combination of functionality and ease of use is not available via any of the existing BGP tools.

Using BGP-Inspect it is possible to easily perform even more detailed analysis of this event. For example, by repeating our raw data query for AS 9121 for the time range December 24th to December 25th for various routeviews peers, and checking to see the number of unique prefixes originated by this AS number, we can see that some networks, were affected to a much smaller degree than others. The Sprint routeviews peer information for example shows AS 9121 as having originated 8K unique prefixes, Global Crossing shows 7.5K, AOL 12K, and Level 3 only 3.7K unique prefixes. We can perform even more detailed analysis of the event by modifying the query time interval. By changing our query time interval to query data for 1 hour intervals, we see that there appear to have been two separate incidents not just one. The first incident occurred between the hours of 9-10 UTC and the second incident between the hours of 19-20. Table IV lists the number of unique prefixes announced by AS9121 as seen by the SPRINT routeviews peer.

It should be noted that not only does BGP-Inspect list the summary statistics we have cited in the previous analysis, but it lists the complete update messages as well including time of announcement, the set of prefixes in the update message, the ASPATH, and the COMMUNITY.

This provides valuable information that can help identify potential sources of routing anomalies. Providing network operators with this powerful capability in an easy to use form can help enhance the security and robustness of the Internet.

B. Prefix Hijacking

Our second example is based on an accidental route hijacking incident that occurred on February 10th, 2005. A class C prefix 207.75.135.0/24 was announced into BGP by AS 2586. This prefix is actually part of the larger 207.72.0.0/14 under the control of AS 237. Once again, we try to show how BGP-Inspect can help in the analysis of such events. The query type most useful for this type of an event is a raw data analysis query of type "prefix more specific". This queries the BGP message database for all prefixes more specific than the one specified. In this case the query is for the prefix 207.72.0.0/14. The resulting response page is shown in Figure 13. The summary statistic table at the top of the page clearly lists, that there were 2 prefixes that matched this query. It lists both the /14 as well as the more specific /24 prefix. It also shows that there were two unique AS numbers 237 and 2586 originating these prefixes which is an indication of a potential prefix hijack event. Next in order to determine the scope of this event, we repeat our query on the different routeviews peers. We see that this prefix hijack was indeed visible in all those networks.

Closer examination of the update messages listed in the BGP message table, we are easily able to identify the specific update message that caused the prefix hijack.

Time (UTC)	Number of Unique Prefixes
07-08	0
08-09	0
09-10	4604
10-11	56
11-12	804
12-13	56
13-14	196
14-15	159
15-16	34
16-17	92
17-18	54
18-19	172
19-20	4496
20-21	229
21-22	15
22-23	0

TABLE IV
NUMBER OF UNIQUE PREFIXES ORIGINATED PER HOUR BY AS9121 ON DECEMBER 24TH 2004

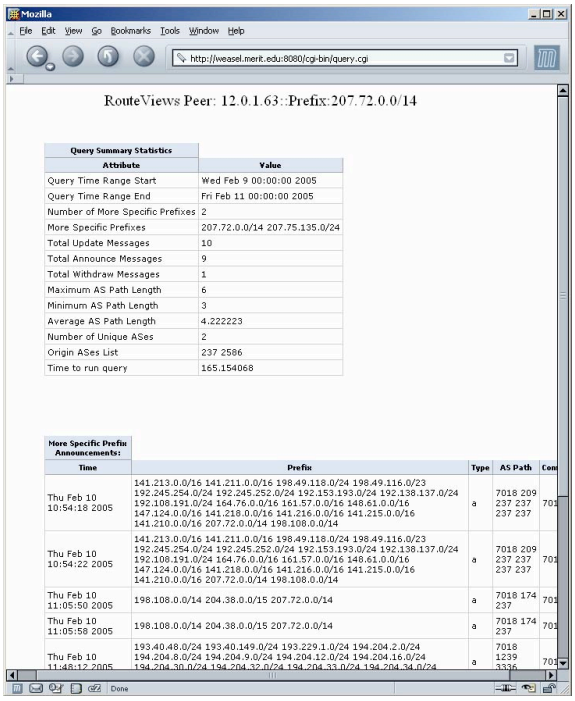

Fig. 13. Raw Data Query for Prefix 207.72.0.0/14 and More Specific Prefixes

In this case there was an update message at 11:48:28 which mistakenly announced the /24 prefix. Querying BGP-Inspect for AS 2586 for Feb 7th 2005 to Feb 13th 2005, shows who the originator is. It also reveals that on Feb 20th 2005, they originated 80 BGP update messages, whereas they usually only originate less than 10 messages per day. This seems to indicate a possible router reconfiguration. Also listed is the withdraw message at 19:22:14 which reverted the mis-configuration. This example shows how network operators can benefit by having a globally accessible repository of easy to query BGP update messages.

C. Common Network Management Tasks

Common network management tasks are easily accomplished with BGP-Inspect. Properties such as connectivity, historical origin AS, and average BGP UPDATE frequency are quickly determined. Each of these properties are determined by setting 2 dialog boxes, entering a single value, and clicking a button. For example, how each of the loaded peers get to a specific prefix involves selecting all interested peers, selecting the query type (Prefix-Exact), entering the prefix, selecting the date range, and, finally, submitting the query. The response will show an aggregate view, showing the frequency of update messages recieved at each peer (for the given

prefix), and tabs for each peer, giving a full account of update messages recieved. The other properties are either infered from this response or completed in a similar way. While the type of information presented for day to day network management is not unique, the speed and ease at which BGP-Inspect delivers the information is optimized.

VI. Conclusions and Future Work

There has been an increasing awareness of the need to use data collection and analysis to study the performance of various Internet protocols. This has resulted in the deployment of large scale data gathering infrastructures. However, it is difficult to extract useful information from large raw datasets. Analyzing large datasets requires one to use tools, techniques and algorithms that are able to scale to handle such large input datasets.

In this paper we have described our experience in attempting to build a scalable tool for mining large BGP routing datasets as well as in building an effective query mechanism for this information. In particular, the use of chunked compressed files and B+ tree indices enables us to balance the need for compressing the data while allowing us to extract information quickly via scalable queries. The use of an extremely simple and intuitive interface to compose powerful queries is also important. BGP-Inspect provides not only raw update messages in response to queries, but also computes statistics which summarize the information. We hope our experiences will provide a valuable guide to others attempting to build similar systems. We have described some example scenarios where BGP-Inspect provides valuable information about anomalous events easily. Current BGP tools lack the ability to allow such powerful analysis of anomalous events in such an easy to use manner.

We have released an initial version of our tool, BGP-Inspect [13], [14] and have received valuable feedback from the network research and operator communities regarding its features and usability. We are incorporating this information into the next release. BGP-Inspect is currently available via our website http://bgpinspect.merit.edu. It is updated on a daily basis. We are continuing to investigate the scalability of both the BGP-Inspect back-end update message database, as well as the query front-end. We are also working on adding addition query capabilities and statistics which will serve to further enhance its utility.

References

[1] University of Oregon Route Views Archive Project. http://www.routeviews.org.

[2] RIPE-NCC. Routing Information Service Project. http://www.ripencc.org/projects/ris/tools/index.html.

[3] N. Feamster, D. Andersen, H. Balakrishnan, and F. Kaashoek. BGP Monitor. http://bgp.lcs.mit.edu/.

[4] BGPlay. http://bgplay.routeviews.org/bgplay/.

[5] M. Lad, D. Massey, and L. Zhang. LinkRank: A Graphical Tool for Capturing BGP Routing Dynamics. *IEEE/IPIF Network Operations and Management Symposium (NOMS)*, April 2004.

[6] Z. Mao, R. Govindan, G. Varghese, and R. Katz. Route Flap Damping Exacerbates Internet Routing Convergence. *Proceedings of ACM SIGCOMM*, November 2002.

[7] A. Feldman, O. Maennel, Z. Mao, A. Berger, and Maggs B. Locating Internet Routing Instabilities. *Proceedings of ACM SIGCOMM*, August 2004.

[8] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. *2nd Symposium on Networked Systems Design and Implementation(NSDI)*, May 2005.

[9] D. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan. Topology Inference from BGP Routing Dynamics. *Proceedings of SIGCOMM Internet Measurement Workshop*, November 2002.

[10] Xenofontas Dimitropoulos, Dmitri Krioukov, Bradley Huffaker, kc claffy, and George Riley. Inferring AS Relationships: Dead End or Lively Beginning? *Fourth Workshop on Efficient and Experimental Algorithms (WEA)*, 2005.

[11] Andrea Carmignani, Giuseppe Di Battista, Walter Didimo, Francesco Matera, and Maurizio Pizzonia. Visualization of the Autonomous Systems Interconnections with HERMES. *Proceeding of Graph Drawing*, 1984:150–163, 2001.

[12] R. Bayer and E.M. McCreight. Organization and Maintenance of Large Ordered Indexes. *Acta Informatica 1*, pages 173–189, 1972.

[13] BGP::Inspect. http://bgpinspect.merit.edu.

[14] M. Karir, D. Blazakis, L. Blunk, and J. Baras. Tools and Techniques for the Analysis of Large Scale BGP Datasets. *Presentation at North American Network Operators Group(NANOG-34) BoF*, May 2005.