MRT dump file manipulation toolkit (MDFMT) - version 0.2

Mattia Rossi
Centre for Advanced Internet Architectures, Technical Report 090730B
Swinburne University of Technology
Melbourne, Australia
mrossi@swin.edu.au

***Abstract*—The MRT routing information export format represents an effective way of storing BGP routing information in binary dump files. Although a few tools exist to extract data from MRT dump files, most of them do not allow repacking or creating such MRT files. The MRT dump file manipulation toolkit (MDFMT) allows to repack parts of large MRT dump files containing BGP update messages into smaller ones, and also allows to create them from Quagga bgpd text log files. The resulting MRT files contain a BGP update sequence of an initial routing table (RIB) propagation followed by the recorded BGP update messages, and may be used for recreating complete BGP sessions based on real data in test setups using the "Update Regenerator" (UR). It also includes a set of smaller scripts to inspect BGP messages within an MRT data set and to aid in operating the UR.**

I. Introduction

In order to allow analysis of BGP [1] traffic, routers and monitoring tools enable logging of BGP update messages and creation of routing table dumps. Although most of the routing tools use an own format for BGP update messages logs and RIB dumps, many of them also allow to use the binary MRT [2] format. The Quagga Routing Suite [3], the routing software in use for the BGP heuristics project [4] which generated the MRT dump file manipulation toolkit (MDFMT) [5], is able to produce human readable text log files and binary MRT dump files for further analysis of BGP update behaviour. While the text log files are usually extremely large as they are intended mainly for debugging purposes and may contain many unnecessary entries for BGP traffic analysis, MRT dump files contain just the BGP messages, preserving their full information in binary network byte order form and also consuming less disk space.

Quagga can produce two different types of MRT dump files for BGP: BGP4MP types which contain BGP UPDATE, NOTIFICATION, KEEPALIVE or OPEN messages, collected continuously in the order they arrive at the collecting router, or TableDumpV2 types which contain the full BGP routing table (RIB) of the router, dumped at certain time intervals to keep track of the current routing table status.

Not only do such MRT dumps allow to evaluate BGP behaviour in depth by analysing the collected dump file, but the preserved BGP packets may also be reused and injected into test routing setups in order to analyse the possible behaviour of adjacent routers.

As Quagga users sometimes have only enabled text logging but not MRT data collection, but would still like to have MRT files for BGP packet injection, MDFMT contains a tool for converting such Quagga text log files into MRT dumps.

Injection of BGP update sequences anyhow would require either that every adjacent router already contains the same starting RIB or that the injected update sequence also propagates an initial RIB.

Unfortunately such a RIB propagation is only logged by the routers when they initially connect to a peer, but oftentimes it is wanted to test BGP behaviour for an update sequence which took place at a certain time interval.

MDFMT contains also a tool to create slices of larger MRT update message dumps using MRT RIB dumps to create an initial routing table propagation followed by the BGP update sequence of the selected time interval.

The obtained MRT dumps can be used with the "Update Regenerator" (UR), which can inject the whole BGP session into a BGP speaker like Quagga.

The following sections will explain first how data collection can be set up in Quagga, then the libraries used for MRT file manipulation and following the tools included in MDFMT version 0.2 which can be obtained at [5].

As the MRT standard is not a standard yet, but only a draft, various versions exist. The draft Quagga and the described tools follow, has been determined as draft *draft-ietf-grow-mrt-08*. The minimum Quagga version which complies to that draft has been determined in 0.99.10.

II. Data Collection in Quagga

This section describes how data collection is enabled in Quagga through the bgpd configuration file or also through the command line interface (CLI). The commands are the same in both cases. It is necessary that the files to which data is going to be logged exist before logging is activated. It is also required, that the user that runs the Quagga process (commonly user Quagga) has write access to those files. Table I shows the commands to activate text logging in Quagga, Table II shows how to dump BGP packets in the MRT format (BGP4MP type). It also shows how to dump the RIB (TableDumpV2 type). As the routing table or RIB might be dumped only at certain time intervals, a simple *cron* [6] script as shown in Table III allows to rotate the dump files, and to rename them with a time stamp. It is important to precise at this point, that the data collected in MRT dumps is only the received data, not the sent data, while the text log files contain some information about which prefixes are advertised but without attribute information. The RIB dumps in text format also do not contain the whole attribute information but only the AS path, while the MRT RIB dumps preserve the whole attribute information.

III. The "dpkt" packet manipulation library

As BGP packets and MRT dumps can become quite complex, it was necessary to use already existing libraries for packet manipulation. A set of libraries for various programming languages exists to extract information from MRT dump files. Some of them are rather complete, others provide just basic support. In this section the *dpkt* Python library will be explained along with the changes applied to make it more complete. *Dpkt* allows you to read and create MRT packets as well as BGP packets itself and can be easily integrated in any Python program. This library is the base for the tools explained in the following sections.

Dpkt is an open source project hosted on Google Code, and aims to be an easy packet construction library in Python for all sorts of network protocols. It has been decided to use this library over other ones because of its quite well developed BGP and MRT support and its

log file bgpd.log	Sets the text log file to the file bgpd.log in the current directory. It is necessary that the file exists and is writable by Quagga
debug bgp	Enables logging
debug bgp events	Enables logging of BGP events
debug bgp updates	Enables logging of BGP advertisements
debug bgp fsm	Enables logging of finite state machine events
debug bgp filters	Enables logging of filtering events
debug bgp keepalives	Enables logging of keepalive messages
debug bgp as4	Enables logging of information regarding the processing of advertisements containing 4 byte AS numbers
debug bgp as4 segment	Enables logging of information regarding AS path segments of advertisements containing 4 byte AS paths additionally to 2 byte AS paths
debug bgp zebra	Enables logging of events of the forward information base (FIB)

TABLE I
COMMANDS FOR TEXT LOGGING IN QUAGGA. THE 4 BYTE AS NUMBER DEBUGGING HAS BEEN INTRODUCED WITH QUAGGA VERSION 0.99.10

dump bgp all bgpd.dump	This command dumps all messages in MRT format into the file bgpd.dump. The file needs to be writable by Quagga
dump bgp routes-mrt \ rib/dump 24h	This command dumps the routing table into the file dump in the folder rib. The dump is repeated every 24 hours, and the first dump is created at the beginning of the next hour when the command is executed. The dump format is TableDumpV2

TABLE II
COMMANDS FOR PRODUCING MRT BINARY DUMPS IN QUAGGA

```
59 23 * * * cd bgproutedumps && \
        mv dump rib_‘date +\%d_\%m_\%Y‘ && \
        touch dump && chmod 0777 dump
```

This command is executed every day one minute before midnight. It renames the file to "rib" followed by the day, month, and year, then creates the original empty dump file and makes it writable for everybody.

TABLE III
AN EXAMPLE CRON COMMAND TO ROTATE THE RIB DUMPS

useful sister project *PyBGPDump* [7]. It has also been determined as the library which makes it easiest to create BGP and MRT packets from scratch, allowing not only to unpack MRT files, but also to repack them. Even though the latest release of the *dpkt* library contains quite complete BGP support, it lacks necessary features like 4 byte AS number support and has some bugs that need to be fixed in order to be able to use the library properly. It also lacks MRT TableDumpV2 support, which is required in order to handle the RIB dumps. Until the bug fixes as well as 4 byte AS number and TableDumpV2 support will be included in the next release of *dpkt*, it is necessary to use at least revision 52 of the library from the Subversion [8] repository at Google Code [9] and patch it with the dpkt patches from the MDFMT version 0.1 [5].

The *dpkt* library has no proper documentation. The scripts described in the following sections, will also give insight into the usage of the BGP and MRT part of *dpkt*. *PyBGPDump* is a Python script which uses *dpkt* and takes care of the extraction of MRT entries in a MRT dump file, giving easy access to the information of each entry. *PyBGPDump* has been used for the two tools contained in MDFMT version 0.1.

IV. Text to MRT Log File Converter

The first tool of MDFMT 0.2 is *log-to-mrt*. It is a tool created to allow a conversion from Quagga text log files into MRT dump files. It might be sometimes desirable to have the collected data in MRT format rather than text format for reasons already explained, but sometimes log files may be present only in human readable form.

A. Mode of Operation

The converter simply reads through the text log file line by line, and converts open, update and notification messages into binary BGP packets. It uses only received packets, as Quagga does not provide enough information about sent packets in its text log files. The text log files also do not contain certain information about the IPv4 address and AS number of the peers which is necessary for the construction of proper MRT entries, so it is important to know the configuration of the router that produced the log file, in order to hand this information over to the converter.

As input, the converter needs the text log file, the IPv4 address of the router that produced the log file, and a simple text file, which contains the IPv4 address to AS number mapping of all peering sessions recorded in the log file. This is necessary, as it is information which is needed for the MRT format, but not for BGP packets itself, and is therefore missing in the text log files, which contain just a textual representation of BGP packets. This mapping can be found in the Quagga bgpd configuration file. Automatic extraction from the configuration file is not yet supported. The mapping file consists of two columns separated by a white space. the first column contains the IPv4 addresses in dotted format, the second column contains the AS numbers, which may also be 4 byte numbers and may also be written in dotted AS notation [10].

The current version of the script needs a complete BGP session as input, including all the OPEN messages as it determines whether to encode AS numbers with 4 byte or 2 byte from the OPEN message of the session. If the OPEN message is missing or does not contain the 4 byte AS number capability code [11], [12], AS numbers are encoded with 2 bytes.

As the aim of this tool is to create MRT dumps which include an initial RIB propagation followed by the recorded update messages for BGP packet injection, it is necessary to have a text log file as source which also includes an initial RIB propagation. It is not possible to recreate an initial routing table propagation using text RIB dumps, as they lack of the necessary information about 4 byte AS numbers and certain attribute information.

B. Known Issues

Currently the converter ignores IPv6 sessions and IPv6 routing information. Eventual IPv6 entries in the text log file as well as in the mapping file will be ignored. As the converter is a Python script and parses the text log file line by line using regular expressions to filter information, it is a quite slow process. Using a log file with data of a months time, depending on the machine it can result in several hours of processing time, using quite a high amount of CPU power.

V. MRT Slicer

The second tool currently included in MDFMT 0.1 is the *mrt_slice* tool. *Mrt_slice* extracts slices of a certain time interval from a larger MRT BGP4MP type dump file using a MRT RIB dump (TableDumpV2) to determine the start of the time interval and to create the starting update sequence. It is not possible to create slices of MRT dumps starting at an arbitrary time, as the initial RIB and the following updates need to remain consistent. *Mrt_slice* creates a file that contains a whole update sequence in MRT BGP4MP type format for the time interval selected.

A. Mode of Operation

The script can create MRT slices in two different ways: either the slice starts from the beginning of the original data, which should include already an initial RIB propagation, and just extracts BGP messages up to the desired end time, or it uses the given MRT RIB dump file to determine the start time and proceeds from there with the creation of the slice.

If the slice starts from the beginning of the original file, the tool simply extracts the time stamp information of the MRT header of each entry, and compares it with the end date. As long as it is smaller, it repacks the MRT header adds it to the new dump file and appends the remaining payload of the entry, which contains the BGP update packet.

The reading of MRT entries is not done with *PyBGPDump*, as that would extract the whole BGP packet too, which is not necessary and would cause too much overhead. If the start time is defined by the MRT RIB dump, it is needed to convert the entries of the dump from TableDumpV2 into BGP4MP, the update message format. The subtype of BGP4MP used for storing update information, is BGP4MP_Message_AS4, which allows also to record 4 byte AS numbers. This format anyhow does not affect the way AS numbers are recorded in the BGP attributes for each prefix. If the BGP messages carries a 2 byte AS number, it is still recorded as 4 byte number in the MRT subtype After creating the MRT Message type with the appropriate subtype, the prefix and the attributes recorded in the RIB entry, are extracted and added to this newly created structure.

After adding all necessary information to the new BGP4MP structure, it is packed and written to the new dump file. The following part looks up the start time extracted from the RIB dump in the original update dump file, and proceeds with adding the entries to the new dump file as already described for the previous method. If for some reason the timestamps in the RIB dumps increase (which could happen if the dump process took several seconds at the time of creating a RIB dump), the script tries to mix the converted RIB dump entries with the update messages from the original update dump file, in order to keep the time line intact.

B. Known Issues

The *mrt_slice* script does not recreate BGP OPEN messages [1]. The slice anyways might contain such messages if they were recorded in the original dump file within the period of time which the slice represents. OPEN messages may be easily recreated using *dpkt*. With the conversion from TableDumpV2 to BGP4MP every prefix is put in a single BGP message, which impacts negatively on the size of the MRT slice and creates additional network overhead when injecting the packets.

VI. Update Regenerator (UR)

The UR is the most interesting tool of the MDFMT, and needs almost all other MDFMT tools and scripts to be able to operate properly. The UR is a pseudo BGP speaker, it connects to a live instance of Quagga (or possibly any other BGP speaker, although not tested), and injects a full BGP session, as included in the MRT dump files generated by the MRT slicer and/or the log-to-mrt converter. It is based on *dpkt* and *PyBGPDump*, and uses a real network interface to communicate to a BGP speaker.

A. Mode of Operation

The UR can only work with a full BGP session, which includes an initial RIB announcement. It opens a real TCP connection to a BGP speaker, eventually also via a loopback interface, with the peering BGP running on the same machine as the UR, and sends the BGP updates extracted from the MRT dump file. As the MRT file contains the updates received by the collector, the UR takes the place of the peers connected to the collector, while the collector is replaced by the BGP speaker that peers with the UR as shown in Figure 1. The updates anyhow do not contain information about the destination, so the updates can be sent to any destination IP address or AS, making the choice for the destination BGP speaker flexible.

The UR takes 3 arguments as input:

- The MRT dump file
- The destination IP address
- A text file containing an AS number - IP address mapping, including information about the use of 4 byte AS numbers or the common 2 byte AS numbers, of the peering sessions contained in the MRT file.

The AS to IP mapping file is needed in order to create the necessary OPEN message(s), and also to filter only wanted peering sessions from the input MRT file. The information can be extracted by the additional *asip.py* script described in section VII. Each line in the AS-IP mapping file creates a regenerated peering session. If a

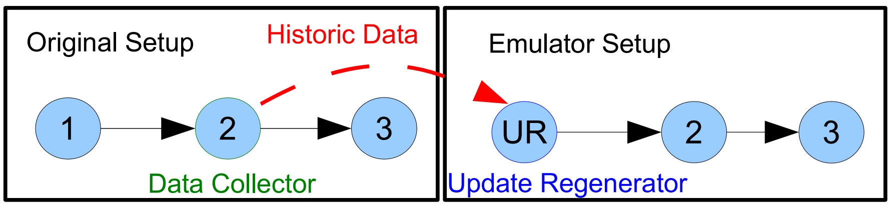

Fig. 1. The UR takes the place of the BGP speaker peering with the collector

recorded session doesn't need to be regenerated, the line describing that session can simply be omitted.

An entry in the AS-IP mapping file is an AS number followed by a whitespace, followed by an IP address and another whitespace, follwed by either 0 or 1. A 0 means that the peering session was recorded using 2 byte AS numbers, while a 1 means the session used 4 byte AS numbers. This information is necessary in order to create an OPEN message which is consistent with the updates regenerated by the UR simulating the affected peer.

The UR parses the AS-IP mapping file and uses the IP addresses in the file to establish a TCP connection to the peer. In order to establish a BGP connection, an OPEN message is constructed and sent, by using the information about AS number, AS number type and IP address. Most of the capabilities [11] sent in the OPEN message are currently hard coded in the UR: MULTIPROTO (IPv4/Unicast — *PyBGPDump* uses only IPv4/Unicast by default), Route Refresh and Route Refresh OLD [12]. The AS4 capability is set dynamically, depending on the contents of the AS-IP file.

The UR doesn't implement a "server" part of the BGP speaker, so it ignores all messages sent by the peering BGP speaker. Instead it relies on the intelligence of the peer to accept the OPEN message and sends a KEEPALIVE message one second after the OPEN message in order to signal it's readiness. It assumes that the peer is ready to receive data within this time span (an assumption that proved to work with a Quagga instance as peer). Once the KEEPALIVE is sent, a timer is started to send KEEPALIVE messages every 30 seconds, also hard coded in the UR.

After the BGP connection is up, the UPDATE messages are sent each second. The timestamp is read from the MRT header, and all updates within a second of recorded time are sent as quick as possible. As soon as the recorded time increases by a second, the real time is controlled, and the UR sleeps for the eventual remainder to the end of the real time second. This ensures a precision on a per second basis. The UR sends only updates that belong to an IP address in the AS-IP mapping list. If an update belongs to a different IP address, it is discarded. The UR quits as soon as it finishes sending, closing the TCP connections beforehand.

As the MRT dump file contains data collected at a single router, the UR can only peer to a single BGP speaker, even though it regenerates sessions from multiple peers. The best practise so far has been to setup the UR together with a Quagga instance on a single machine, and let them peer via a loopback interface, and let the Quagga instance peer with any other BGP speaker necessary.

B. BGP peer configuration

The UR sets up a real BGP session to a BGP speaker, thus it is necessary to configure such a router properly. The suggested, yet not so resource friendly way of operation is to use the UR in conjunction with a Quagga instance on the same machine, using a common interface for communication. The interface needs to be configured with the destination address, as well as the IP addresses recorded in the AS-IP mapping file, so that the UR and the Quagga instance can bind these addresses.

The Quagga instance then needs to be configured for peering to the BGP speakers which the UR claims to be - also as recorded in the AS-IP mapping file. If the Quagga instance wants to peer to a remote BGP speaker, all the addresses need to be set up at the interface that is able to reach the remote BGP speaker. Table IV shows an example Quagga BGP configuration file. Such a setup might create some confusion in some networks, therefore other solutions like multiple instances [13] might be approached.

C. Known Issues

The Update Regenerator is highly experimental, and could be improved and extended in many ways. It has been proved to work quite well in specific context, but it might need some changes in the source code to make it work under different circumstances. The UR has only been used in conjunction with Quagga version 0.99.10 and higher, but it might also work with other BGP

router bgp <AS number>	The AS number of the Quagga BGP speaker - not important for the UR
bgp router-id <router IP>	The IP address of the Quagga BGP, to be passed to the UR as destination IP address
neighbor <remote-ip> remote-as <AS number>	The AS number of the remote peer
neighbor <remote-ip> update-source <router IP>	The IP address of the Quagga BGP. This is necessary if multiple IP addresses are configured on a single interface
neighbor <remote-ip> filter-list 20 in	Does not accept updates from the remote peer. This makes sense, as is acts only as intermediate for the UR
neighbor <UR IP address 1> remote-as <UR AS number 1>	The AS number of the first UR peer
neighbor <UR IP address 1> update-source <router IP>	The IP address of the Quagga BGP. Same as above
neighbor <UR IP address 1> filter-list 20 out	Does not send updates to the UR peer. The UR doesn't accept them. This is important in order not to disrupt functionality
neighbor <UR IP address 2> remote-as <UR AS number 2>	The AS number of the second UR peer
neighbor <UR IP address 2> update-source <router IP>	The IP address of the Quagga BGP. Same as above
neighbor <UR IP address 2> filter-list 20 out	Does not send updates to the UR peer. Same as above
ip as-path access-list 20 deny .*	Applies the filter-list 20 to all addresses

TABLE IV
AN EXAMPLE CONFIGURATION FOR A QUAGGA BGP RUNNING ON THE SAME MACHINE AS THE UR AND USING THE LOOPBACK INTERFACE FOR COMMUNICATION

speakers. It is completely missing a "server" part, which might confuse certain BGP speakers, and make them refuse to connect. The UR also does not recover from connection loss, as it does not contain the necessary FSM. The UR also needs the IP addresses contained in the MRT dump file to be available on the interface used for BGP peering, which might reduce it's range of employment. The UR also uses a quite high amount of resources, specially if there are many updates to be sent (like the initial RIB propagation) as it reads the BGP message from a file, performing a high amount of I/O operations.

VII. Additional Scripts

The MDFMT contains additional small scripts based on *dpkt* for extracting information from MRT dump files. The *asip.py* script is needed in combination with the UR, as it extracts the AS number and IP addresses of all peering sessions from a dump file, and places them into a text file. It also detects whether a peering session uses 4 byte AS numbers or 2 byte AS numbers.

The *follow_prefix.py* script is a simple script to extract updates regarding a certain prefix from a set of MRT dump files and to list the updates in chronological order. This allows to examine the announcements or withdrawals of the prefix in a whole BGP system, and helps with the discovery of eventual misconfiguration.

The *find_withdrawals.py* script discovers withdrawals within a MRT dump, making sure the prefix has been previously announced, while the *check_path_hunting_intervals.py* script searches a dump file for path hunting events [14], and calculates how long such events last.

While this scripts have been written for specific contexts which may not seem useful to anyone else, it is rather easy to create a your own scripts using dpkt using the presented scripts as aid.

VIII. Conclusions

The MRT dump file manipulation toolkit version 0.2 provides tools which can be used for manipulating and creating MRT files, which again can be used for BGP packet injection. The MRT files created by the tools always contain a consistent sequence of BGP packets representing an initial RIB propagation followed by BGP updates. The UR can be used to repeatedly inject the same data into a BGP setup, providing a useful tool for data and BGP configuration analysis. The tools of MDFMT 0.2 have been used extensively in specific contexts [4], although they have been created with a certain level of generality, they may not yield the wanted results under specific circumstances.

IX. Acknowledgements

The development of MDFMT has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

References

[1] Y. Rekhter, T. Li, and S. H. (Editors), "RFC 4271: A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), january 2006, obsoletes RFC 1771. [Online]. Available: http://tools.ietf.org/html/rfc4271

[2] L. Blunk, M. Karir, and C. Labovitz, "MRT Routing Information Export Format," February 2009. [Online]. Available: http://tools.ietf.org/html/draft-ietf-grow-mrt-09

[3] K. Ishiguro, "Quagga Software Routing Suite." [Online]. Available: http://www.quagga.net

[4] G. Armitage, G. Huston, and M. Rossi, "Reducing BGP Update Noise." [Online]. Available: http://caia.swin.edu.au/urp/bgp/

[5] M. Rossi, "MDFMT - MRT dump file manipulation toolkit - version 0.2." [Online]. Available: http://caia.swin.edu.au/urp/bgp/tools.html

[6] "Cron, a time-based Scheduling Service." [Online]. Available: http://en.wikipedia.org/wiki/Cron

[7] J. Oberheide, "PyBGPDump, BGP Messages Parser of MRT Dumps." [Online]. Available: http://code.google.com/p/pybgpdump/

[8] "Subversion, an Open Source Version Control System." [Online]. Available: http://subversion.tigris.org

[9] "Dpkt Subversion Repository." [Online]. Available: http://code.google.com/p/dpkt/source/checkout

[10] G. Michaelson and G. Huston, "Canonical Textual Representation of Four-octet AS Numbers," December 2007. [Online]. Available: http://tools.ietf.org/html/draft-michaelson-4byte-as-representation-05

[11] R. Chandra and J. Scudder, "RFC 3392: Capabilities Advertisement with BGP-4," RFC 3392 (Draft Standard), november 2002, obsoletes RFC 2842. [Online]. Available: http://tools.ietf.org/html/rfc3392

[12] C. Appanna and E. Chen, "IANA Capability Codes," last Updated on 06/08/2008. [Online]. Available: http://www.iana.org/assignments/capability-codes/capability-codes.xhtml

[13] K. Ishiguro, "Quagga Software Routing Suite - BGP configuration documentation." [Online]. Available: http://quagga.net/docs/docs-multi/BGP.html#BGP

[14] M. Rossi, "Implementing path-exploration damping in the Quagga Software Routing Suite Version 0.99.13 - patch set version 0.3," Centre for Advanced Internet Architectures (CAIA) - Swinburne University of Technology, Tech. Rep., July 2009. [Online]. Available: http://caia.swin.edu.au/reports/090730A/CAIA-TR-090730A.pdf