RIPE NCC

RIPE Network Coordination Centre
www.ripe.net

Research on RIS Route Collectors

W.A. Miltenburg
RIPE NCC
GII team
Supervisor: C. Petrie

Table of contents

1. Introduction

The current RIS implementation, that is used by the RIPE NCC, was released in 2001. Back then, use-cases and requirements were different from the ones people have now. The project “Replacement of RIS Route Collectors” has been initiated in the year 2013 and started in the year 2014. It is aimed at researching possible alternatives for the current implementation but also to develop and deploy a prototype.

This document is a research paper and will include the following items: a MoSCoW-scheme, researched alternatives, the outcome of the research and a proposal for a new implementation of the RIS Route Collectors. The MoSCoW scheme will be used for prioritising all the requirements. The proposal will result in a prototype that needs to be developed and deployed. It is used to prove if the proposed alternative can meet all the requirements and if it is possible to create the proposed alternative.

The next few chapters will give the reader a brief understanding why this project has been initiated and what the current implementation of RIS is.

2. General information

The Routing Information Service (RIS), provided by RIPE NCC, was established in 2001 and started with the idea to collect and store Internet routing data from several locations around the globe. This data, which is collected and saved by the RIS Route Collector, can be accessed via RIPEstat and can be used for further analyses. RIPE NCC also performs analyses on the data itself and publicise some of these researches through RIPElabs. The Remote Route Collector (RRC), as it name suggests, collects all the Internet routing data from several points around the world.

As more peers want to connect with the RRCs, it also takes more and more time to process all this data. This can cause some problems; people have to wait longer for the data, analyses have to wait because of the delay, and when IP hijacking occurs you want to know it immediately.

This project was initiated to research, develop and deploy a prototype that replaces the current RRCs and which invokes less delay. This document describes possible alternatives for the current implementation and a proposal to replace the current implementation of the RRCs.

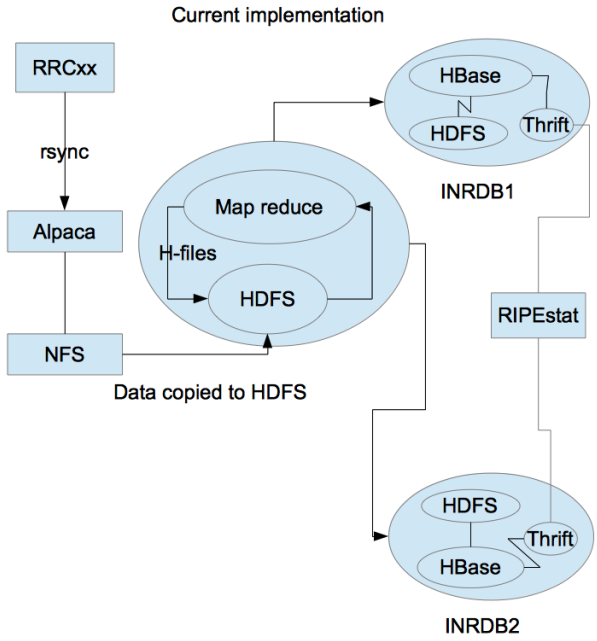
The document will contain the following information:
- Current implementation of the RRCs
- Requirements for the new implementation
- A MoSCoW scheme, which is comprised of the requirements
- Projects that have been researched and which are compared to the MoSCoW
- A proposal

3. RIS implementation and problems

This chapter will provide the reader with a general overview of the current implementation of RIS. It is a high-level overview of the current implementation and not all exact details will be discussed in this chapter.

3.1.Current implementation

Several RRCs at different worldwide locations, provide a periodic RIB dump or a dump of all the received updates during five minute periods. These dump files are saved in a MRT formatted file and will be periodically synced using rsync to a server called 'Alpaca'. A NFS share is mounted on "Alpaca", which contains all the MRT formatted files. The next step is to save this information in a database so that it can be retrieved at a later moment.



The current implementation uses Hadoop/ HBase as its database. The MRT formatted files that are stored on the NFS-share are copied to HDFS.

From HDFS, the data will be inserted into HBase which also uses HDFS as its file-system. Thrift is used as an API to access data that is stored in HBase and which needs to be accessible for public facing websites (i.e., RIPEstat).

3.2. Description of current limitations

This subchapter will provide the reader with a general overview of the problems that are involved with the current implementation.

Delay

One of the issues in the current implementation, is that there is a lot of delay involved in the processing part. The data must be copied several times and needs to flow to all different subsystems. This is in fact the main reason why this project has been initiated in order to develop a mechanism that will later allow to reduce the latency.

Quagga
Quagga is being used to receive all the BGP updates on the RRCs. It maintains BGP peering connections with multiple peers. Quagga also introduces some problems. There have been cases where Quagga missed some of the BGP updates and this caused a RIB that was not accurate. It is a single threaded application, which does not take advantage of a system with multiple cores. It also reaches the maximum utilisation of the single core that it uses and cannot handle any extra peers.

Some of the people at the RIPE NCC do not really trust the RIB dump implementation of Quagga. There were cases where the RIB table got corrupted or that Quagga missed updates during those dumps.

4. Research methodology

This chapter describes the methodology that is used in this research paper.

4.1. Main question

This project has been initiated to answer the following main question:
"How can the current implementation of the RIS Route Collector be replaced with a better alternative, aimed to process updates faster, make information easier to integrate in the RIPE NCC Hadoop storage backends (e.g. through XML, JSON, or YAML), and meet the gathered requirements?"

To answer this question the MoSCoW scheme is used to prioritise different aspects of this project. For example, is less delay more important than an information scheme that is easier to integrate? In subchapter 4.3 the MoSCoW scheme is introduced and it explains how it was used in the research stage.

4.2. Process

During the research stage different methodologies were used to gather all the required information, process the information and research it. It involved gathering information about the project, understanding RIS, gathering all the requirements by interviewing people and setting up meetings. All these requirements were prioritised using MoSCoW and was used to prioritise the different requirements of this project. Most of the research in the research phase was done be reading documentation and RFCs as part of desk research.

4.3. MoSCoW scheme

The MoSCoW scheme will give the requirements different priorities and is, in this case, a better methodology to use during this research stage. It is not the only methodology or toolkit that was used in this project. A few examples of other methodologies that were used are desk research and interviewing the stakeholders.

The MoSCoW scheme is also used to see if a possible alternative will meet all the requirements and how much work needs to be done when a requirements is not met. It is also used to see if all these requirements can be met with another alternative.

4.4. Disclaimer

The use of this methodology has been approved by the supervisors of this project[1].

[1] During the meeting on Tuesday February 11th

5. Proposed requirements

This chapter will show all requirements that were proposed during meetings and/or interviews. It does not mean that all the requirements will be considered in-scope of this project. All the requirements that were considered in-scope, and that have been used during the research stage, can be found in the next chapter. It is possible that some requirements listed here are out of scope or are post-processing requirements which is related to this project.

The terms “SHOULD, MUST, WILL” does not mean that this proposed requirement will make it to the definite MoSCoW scheme. This MoSCoW scheme will be discussed in the next chapter.

5.1. Same attributes should be provided in the output

The same BGP attributes, that are provided by the current implementation, should also be provided by the new implementation. The following attributes are outputted in the current implementation:

- Time
- MRT Routing information type
- BGP message type (Announce/Withdrawal)
- Peer IP
- Peer AS
- AS path
- Aggregation
- Local preference
- MED

These attributes are used internally in RIPE NCC but are also being used by the community for researching/analysing purposes. When attributes are missing, it could be possible that an analysing tool cannot use the data that is provided by another implementation.

5.2. Scaling

The new implementation should be able to handle more peers than the current implementation. It should be possible to expand the resources that are necessary to handle all of the data that is processed by this system. For example, a queueing system needs to be scalable in order to hold all the messages that need to be processed. This system should provide a mechanism where extra resources (i.e. servers) can be added to the cluster so that it can handle more messages.

5.3. MRT-formatted files

The Multi-Threaded Routing Toolkit (MRT) format is used as a file format for the current implementation's output. People at the RIPE NCC, and also the community, still rely on the MRT-formatted data that is provided by the current RIS implementation. Users can download the data, parse it and analyse the data for researching purposes. There are still tools used internally at RIPE NCC that rely on these MRT-formatted data, it is still a wish to maintain these files in order to fully support these tools. It is possible that the tools can be updated in order to support the new format, but this will take time and this needs to be accounted for.

5.4. Less delay

The current implementation involves a lot of delay. From the moment when the data is outputted and the data is processed, it takes a couple of hours before the data is available. Different processes, most of them are cronjobs, wait on each other which introduces delay. It would be useful for users, and RIPE NCC itself, that the flow of data is more a stream of data and not batch oriented. It also creates the ability to process the data as a stream instead of a batch oriented process.

Users could benefit from the fact that, if there is less delay involved with a new implementation, they can analyse data which is more current than the currently available data.

With less delay, some people mean that there should be an up-to-date RIB available when a request is made.

5.5. Live data stream

The most optimal and extreme case of 'less delay' is to have access to a live stream of data. Users could benefit from the fact if they can have access to a stream of live data. The users could then create tools that analyse the live stream and can send alerts if their filter matches the output.

5.6. Raw data

The output that is provided, by a certain application, should also contain the raw data. This is useful when a new attribute is introduced but cannot be parsed by the application itself. When another application can parse the attribute, it is still possible to do so if the raw data is provided in the output. Supporting this feature results that no attributes will be lost during processing and that users could always parse the raw data if they want to analyse the new attribute.

5.7. Aggregation of data

All of the data can be aggregated by an application. This aggregated data set could be used to get a total overview of all the data that has been received by all the RRCs. This can be useful when someone wants to see if a new announced prefix can be seen by all the RRCs and how many neighbours, from the RRC's point of view, have received the prefix in one of its updates.

5.8. Integrity of data
If an update is missed by one of the RRCs it will lead to an inconsistent or corrupt RIB table. Therefore it is important that all updates must be received and processed to maintain the integrity of the data.

5.9. Authorisation and encryption
Only authorised programs or users can access the systems that are used in the new implementation. Data that is transferred in the new implementation should also be encrypted so that the data cannot be read by unauthorised users. It is also a hot topic at the IETF that new implementations should provide security[2].

5.10. Announce ANCHOR/BEACON IPs
The current implementation announces anchor and beacon IP prefixes periodically. This is useful for studies concerning unwanted effects on convergence when applying Route Flap damping practices.

5.11. Extensible
There could be a scenario in the future were a new functionality is added to the new implementation. This happened in the past with the announcements of the anchor and beacon IPs. It was not part of the RRC till the decision was made to add this feature to the RRC. To support such use cases, the system has to provide a way for adding additional features.

5.12. Security
The system should somehow secure the messages that are sent between intermediate systems, so that messages cannot be malformed during transmission, and the data should remain confidential during this transmission.

5.13. eBGP multihop
Some peering relationships are set up using the eBGP multihop feature. This feature does not require a direct connection to be set up between the RRC and peer, which is already being used in the current implementation.

5.14. Correlation between atlas/anchor and RRC
Data received from the atlas and anchors should be correlated with the RRC's data. This is useful for research and analysis, and can be used to detect differences in routes from Atlas, Anchor or RRC its view.

---

[2] For more information about security and the IETF, please read the following article: http://www.circleid.com/posts/ietf_chairs_statement_on_security_privacy_and_widespread_internet_monitorin/ (09/09/2013, CircleID, Dan York)

5.15. Usability
The output from the RRCs should be usable, in the sense that it is easy to parse and the data can be easily processed. It must be clear for anyone what the output is and what the meaning of the output is.

5.16. Open Source
The implementation should be open source and may be published to be publicly available.

5.17. Spike detection
When a reset is issued by a peer, the RRC will receive all prefixes when the connection is re-established. As an example, this could create a spike in a graph that shows how much routes have been received. The implementation should detect this so that the graphs will not get malformed if a reset was issued by the peer.

5.18. Separate full feed from partial feed
The BGP updates that the RRCs receive come from different peers. These peers can provide the RRC with a full or partial feed. It will be useful if there is a mechanism were the data of the full feed can be separated from the partial feed. This will enable the users to select if they want to receive information of a partial or full feed. Some people of the RIPE NCC say that the information is too cluttered at the moment because there is no distinction in the full or partial feed.

5.19. Metadata
The new implementation should output metadata about the RRC. Programs that use the RRC services, should be able to know the state of the peering connections. The new implementation must also send keepalive messages to the applications so that the applications know if the RRC is still alive.

5.20. Ordering
When output is generated by an application it should be possible to order this data. This can be done using a sequence number or it must be sure that everything, in the whole process chain, is processing and generating information in an ordered sequence. In some parts of the process chain, it is important that the information is ordered (e.g. to maintain a RIB state).

5.21. High resolution timestamp
The high resolution timestamp is a requirement that has been requested by the RIPEstat team. They want to have access to a higher resolution timestamp for their research and analyses.

6. MoSCoW

The MoSCoW scheme, which is presented in the next subchapter, is composed from the requirements that were received during the research stage. The following categories exist in the MoSCoW scheme of this project: “Must, Should, Could, Would and Out of scope.” The category ‘Out of Scope’ is added because some requirements are not in scope of the RIS project and are not considered to fit in the ‘would’ category. In a separate subchapter it will be explained why some requirements are considered to fall under the ‘out of scope’ category.

For the readers who are not familiar with MoSCoW, please read the first appendix. The first appendix will briefly explain MoSCoW.

6.1. Use cases

This chapter describes the use cases that currently exist. Some of the use cases are not supported by the current implementation. The use cases outlined below will give a better understanding why some of the requirements are listed in the MoSCoW scheme.

Analysing

People at the RIPE NCC and the community can download the MRT formatted files, which are provided by the current RIS implementation. These MRT formatted files hold the whole RIB state for each of the route collectors, or all the update messages that have been received by the RRC in a given time period. Users can download these files to do research on routing. For example, when someone wants to know if their prefixes are announced and are propagating through the network.

Note: This is possible with the current implementation

IP hijacking

When someone else is announcing a prefix that has been allocated to a certain entity, one wants to know this in a short amount of time. The person wants to know which AS is announcing the prefix and can take action to resolve this issue.

Note: This is not possible with the current implementation. Users have to wait for a long period before this data is available and it is only useful if this data is available in a short amount of time.

Live RIB state

Users can get a live RIB state at any given moment and can see what the state of the RIB is. This can be useful for users who want to get a live overview of a RIB and want to do research with it.

Note: This is not possible with the current implementation. Users have to wait for a long period of time before this data is available.

6.2. MoSCoW scheme

The following MoSCoW scheme will be used during research.

Must:	
	Announcements of anchor/beacon IP
	MRT-formatted files
	Raw data
	Metadata
	Ordering
	Less delay
	Same attributes
	eBGP multihop
	Scaling
Should:	
	Live data stream
	Integrity of data
	Extensible
Could:	
	High resolution timestamp
	Authorisation and encryption
Would:	
	None
Out of scope:	
	Spike detection
	Aggregation of data
	Correlation between atlas/anchor and RRC data
	Separate full feed from partial feed

The input for these requirements were provided during interviews and meetings.

This MoSCoW scheme was approved by the staff of the RIPE NCC during a meeting about the requirements and priorities[3].

6.3. Out of scope requirements

Out of scope means that the requirements is not considered in scope for this or future projects.

The following requirements are considered out of scope:
- Aggregation of data;
- Correlation between atlas/anchor and RRC data;
- Separate full feed from partial feed[4].

The reason that the 'aggregation of data' requirement is considered to be out of scope for this project. It is considered to be more a post-analysing item. The job of the RRCs is to receive the BGP updates and do further processing. A separate tool or project is needed to provide an aggregation of data between the RRCs.

The correlation between the Atlas probes, Anchor, and RRC data is also considered out of scope. This is also considered a post-analysing item which is not the RRC's job. It can cause more overhead and the RRC's job should be clear and should not take extra overhead to provide such features. There is a considerable chance that because of this overhead extra delay could be involved.

The 'separate full feed from partial feed' requirement is considered to be out of scope of this project. It can not really be determined on a RRC if a peer is sending a full feed or a partial feed and there is no standardised way of determining if a peer is sending its full feed to the RRC. Therefore, this needs to be determined after the routes have been collected, which is done in the post-processing state of the whole RIS system.

6.4. Requirements that are not in the MoSCoW scheme

Some requirements did not end up in the MoSCoW scheme. This can be caused when the requirement is too vague, is only considered a discussion item or is not part of the RIS project at all. If a requirement is not listed in the MoSCoW scheme it was the consensus outcome of the discussion during the requirements and prioritisation session.

---

[3] This meeting was on Thursday 27th of February

[4] When a peer sends its full feed the advertised routes are not filtered by the neighbour. With partial feeds the neighbour filters the routes that are send to the RRC.

7. Researched projects and programs

This chapter will describe the projects and/or programs that have been researched during the research phase. All projects and/or programs, that will be discussed in subsequent subchapters were considered to be in scope of this project[5].

Every possible alternative is compared with the MoSCoW scheme, which was presented in the previous chapter. A brief description is included in the diagram, which describes if the requirement is or can be met and how much work is involved to meet this requirement. A score is also included in the diagram that reveals, in a quick glimpse, if the requirement is met or how much work is involved to meet the requirement. The score is presented using the “++, +, -, --“ characters and mean the following:

++ Requirement is met and no further work needs to be done
+ Requirement is not met and little work is involved to meet the requirement
- Requirement is not met and some work needs to be done to meet the requirement
-- Requirement is not met and a lot of work is involved to meet this requirement

7.1. Custom solution with ExaBGP

This will be a custom solution which is a combination of multiple programs or projects. The BGP messages that are transmitted by the peers will be received by ExaBGP. ExaBGP can be compared with a Software Defined Networking program. It allows a user to transform BGP messages into plain text or JSON, which can then be received by another module or script for further processing. ExaBGP supports multiple drafts and RFCs[6] that are used by several other companies[7].

ExabGP is shipped with an API to extend its functionalities. The output that is generated by ExaBGP can be received by another application. A custom program can receive this information and process this data. It is possible to send data back to ExaBGP using the standard output stream, stdout) of the custom program. ExaBGP also provides the ability to specify which application must be invoked in certain circumstances. For example, a specific program is only invoked when an update has been received by ExaBGP or to invoke a different program when an IP-address needs to be announced. ExaBGP is a single-threaded program, but it is possible to create an ExaBGP instance per peer (e.g., process-per-peer-model).

---

[5] A project or program was considered in scope if it fulfilled one of the requirements that are listed in the requirements scheme

[6] To see which drafts and RFCs are supported, please visit: https://github.com/Exa-Networks/exabgp/wiki/RFC-Information (exaBGP, 17-02-2014)

[7] To see which companies use exaBGP, please visit: https://github.com/Exa-Networks/exabgp (exaBGP community, 14-02-2014)

This program does not maintain a RIB table, but an issue has been created on Github that this might be supported in future[8]. It is currently up to another program to maintain a RIB table[9].

ExaBGP only provides us, in this project, with functions to receive and send BGP packets. The received information must be further processed and this information must somehow be stored in HBase. It is not feasible to store the information directly into HBase as it can cause a lock on the program that was invoked by ExaBGP. The RRC's job is to receive and store the data, it is not up to the RRC to do further processing. A more logical choice would be to use a queueing mechanism between the RRCs and the datastore. The queueing mechanisms will be discussed in a separate subchapter.

After the data is stored in the queue, another system will consume the data from the queue and will do the post-processing stage. In this stage the output of ExaBGP will be stored in the datastore or the program will do some data manipulations.

There is also a need to create a custom application that receives the updates, stored in the queues, and create MRT formatted files for maintainability. It will periodically create a full table dump and will also periodically dump the updates in a MRT formatted file. These files will then be stored on the NFS share so that the community can still have access to these files.

This custom solution will also give us the ability to have access to a live data stream. The implementation depends on the queueing mechanism that is used in this custom solution. If the queue implementation can not provide multiple consumers with a live stream of data, the live stream must be stored in the datastore. This can cause a higher load on the datastore itself. An API needs to be created which relies on the queueing system or the datastore, for providing the data, and which can provide filters or authentication.

Pros

- Active project and regularly updated
- Exported data in JSON format
  - This format is already used internally for analysing data
- Raw data could be received from the receive-packets option
  - Only requires some work when both the raw and parsed update message must be in one JSON message
- It is open source and can be extended with features
  - It depends on programs/script to do something with the data (this is the mindset of ExaBGP)
- Open source BSD license

[8] For more information, please visit the following website: https://github.com/Exa-Networks/exabgp/issues/45 (ExaBGP, Github, 14-02-2014)

[9] For more information, please visit the following website: https://github.com/Exa-Networks/exabgp (ExaBGP, Github, 14-02-2014)

Cons

- High performance issue (when having more than 100 peers)[10]
- There is no RIB (only adj-RIB-out)
  - Must create own implementation to create a RIB
- Not multi-threaded or multi-process architecture
  - But it is possible to create a process per peer
- No feature to output MRT-formatted files for maintainability
- If JSON structure changes the post processing needs to be changed[11]
  - The version, since when the JSON output has been changed, is added in the JSON output itself

[10] For more information about this problem, please visit the following website: https://github.com/Exa-Networks/exabgp/issues/97 (ExaBGP, 17-02-2014)

[11] This is required since the consumers depend on the data that resides in the JSON messages. If the structure changes, the consumers need to be changed as well, otherwise it can not read the data.

MoSCoW comparison

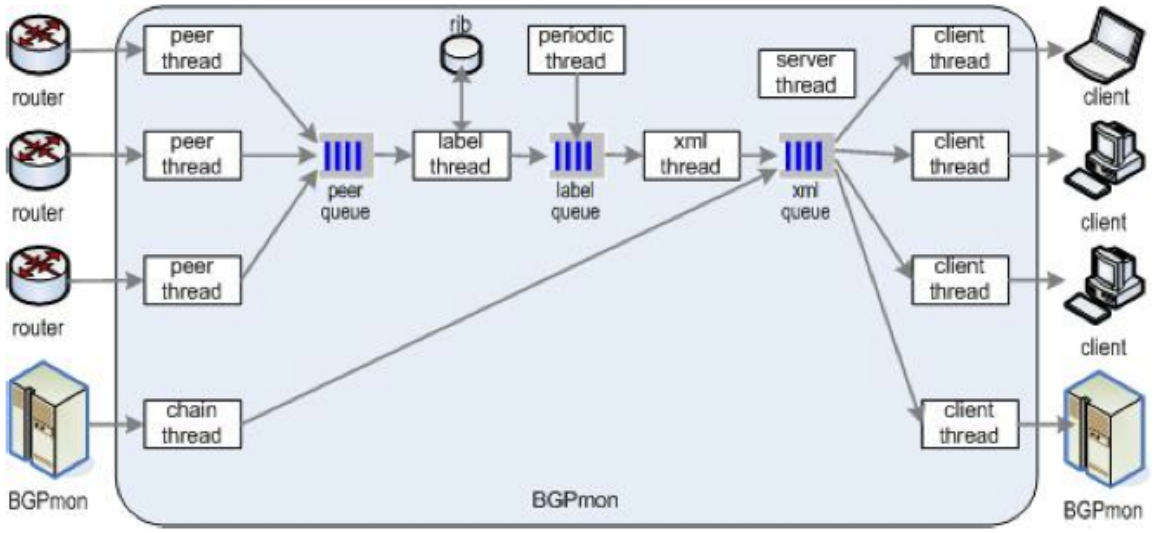
This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	Is possible, IP prefixes can be announced.	++
M2	MRT-formatted files	Need to create our own application which creates the MRT-formatted file	--
M3	Raw data	Provided by ExaBGP, need to modify code to have the parsed and raw data available in one JSON message.	+
M4	BGP metadata	Possible to output state messages, if more data is necessary the code needs to be modified.	+
M5	Ordering	Need to create a sequential ID in the output.	+
M6	Less delay	If data is inserted directly in the database it will involve less delay. It only requires that an application needs to be created which does the insertion into the database.	-
M7	Same attributes	Same attributes are provided in the output.	++
M8	eBGP multihop	Possible.	++
M9	Scaling	Can create thread-per-peer model and, depending on the queue mechanism, it is scalable.	++
Should:			
S1	Live data stream	Output is generated per received BGP update message. Message only needs to be stored somewhere and should be later retrieved. This needs to be created.	+
S2	Integrity of data	Depends on the queue mechanism that will be used.	+
S3	Extensible	Very extensible, shipped with an API.	++
Could:			
C1	High resolution timestamp	Is possible, will be implemented by the original developer of ExaBGP	++
C2	Authorisation & encr.	Not a feature, but could use SSH tunnel.	+

7.2. BGPmon

BGPmon, not to be confused with the protocol BMP or the service BGPmon[12], was created with the same philosophy as this project. The project uses a modular architecture to scale properly for all the monitored BGP peers and also supports a distributed deployment. All monitored updates, RIB dumps, and MRT files. This input can be outputted in a XML format. The output can be used as input for an analytic program and it is already used to analyse events[13]. The output can be retrieved from the default ports 50001 and 50002, the distinction between the two ports is that the first one only outputs the BGP update in XML-format and the second one only outputs the RIB tables in XML format.

BGPmon can handle multiple peers and uses a thread-per-peer model. Every thread is assigned to a specific peer and this causes a better throughput sine it takes advantage of a multicore system. The following image presents a high-level overview of the internals of BGPmon[14]:



12 The BGPmon service: http://www.bgpmon.net/

13 For more information, please watch the following video: http://www.youtube.com/watch?v=8Rqh4p4yzyQ&list=PLO8DR5ZGla8g24mzEdAKZ83byQuA6RIci&index=8 (BGPmon team, NANOG60, 2014)

14 Image acquired from BGPmon article. For more information, please visit the following website: http://bgpmon.netsec.colostate.edu/download/publications/catch09.pdf (BGPmon, 25-02-2014)

The system does not have the possibility to announce IP prefixes. This was a design decision to make sure that no bug or faulty configuration can announce an IP address. In this way, BGPmon is very neutral and cannot have a devastating impact on your network[15].

In BGPmon, writers add data to the queue and the data is only removed after all readers have accessed the data. Due to the fact that writers and readers can have different speeds, which could be caused by different bandwidth or processing speeds, BGPmon has two mechanisms that can handle these problems.

Pacing writers, a mechanism where the queue paces the writers according to the average reading rate across all readers. If a queue exceeds the length of a configurable threshold, it will enable pacing until the queue length drops below a second threshold .

Dropping a slow reader, a mechanism where a reader is dropped despite the attempts to pace writers to the average reader [16].

The community that supports BGPmon provides additional extensions, so called modules, to analyse or interpret the outputted information. There is a module that acts as an archiver[17], and which connects to the live feeds and save the information in a XML file. It is also possible that the output can be saved to a similar output[18] that bgpdump[19] provides.

---

[15] Explanation why BGPmon does not have the capability of announcing routes: http://bgpmon.netsec.colostate.edu/index.php/join-the-peering/peering-faq (BGPmon, Network Security Group, Colorado State University, 14-02-2014)

[16] For more information please read chapter 4.2 “Stream controller”: http://bgpmon.netsec.colostate.edu/download/publications/catch09.pdf (BGPmon: A real-time, scalable, extensible monitoring system)

[17] For more information, please read the following README file: http://cpansearch.perl.org/src/BGPMON/BGPmon-Archiver-2-10/README (BGPmon, Network Security Group, Colorado State University, 14-02-2014)

[18] To see this output, please visit: http://archive.netsec.colostate.edu/date/2014.02/14/UPDATES/updates.20140214.0000.2001:12f8::218:121.bgpdump-001 (BGPmon, Network Security Group, Colorado State University, 14-02-2014)

[19] To download the source of bgpdump, please visit the following site: http://www.ris.ripe.net/source/bgpdump/ (RIPE NCC, 14-02-2014)

Pros
- Multi threaded (per peer)
- Scalable
  - It is possible to create a cluster
- Queue mechanism
  - Maintains fast read and writes by
    - Pacing writers
    - Dropping slow readers
- Stores one RIB-IN for each peer
- Updates are accessible separately from the RIB table
  - Updates default port 50001
  - RIB table table transfers default port 50002
- Possibility to create ACL for retrieving information
- Can import MRT-formatted data
- Raw data available in XML output

Cons
- Mainly XML output
  - If XML structure changes, the post processing application changes
  - They just changed their format[20]
    - Next releases will still support the old format
    - This provides customers to change their converter or interpreter
- Cannot announce anchor or beacon IP addresses
  - Which is a must requirement
- It missed some updates in the past[21]
- Have had some problems with memory leaks in the past[22]
- Could be the case that readers are dropped when they are reading too slow and can cause corrupted data.

Most of the statements that are made above relies on the information that is available at the BGPmon website[23].

---

[20] Release notes of BGPmon v7.3.0: http://bgpmon.netsec.colostate.edu/index.php/live-data/69-bgpmon-730-released-alias (BGPmon, Network Security Group, Colorado State University, 14-02-2014)

[21] According to some contacts Emile spoke to and who can be trusted according to Emile.

[22] For more information, please read the bugfixes on the following page: http://bgpmon.netsec.colostate.edu/index.php/download (BGPmon, 25-02-2014)

[23] For more information about BGPmon, please visit the following website: http://bgpmon.netsec.colostate.edu/ (BGPmon)

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	This function is not provided by BGPmon on default.	--
M2	MRT-formatted files	Does not provide a feature to create a dump in a MRT-format. Need to create our own application that retrieves information from BGPmon and create a dump.	--
M3	Raw data	Raw data is available in the data stream.	++
M4	BGP metadata	Provides BGP states, but no keepalive messages.	-
M5	Ordering	Sequence number added in the output.	++
M6	Less delay	There is a live stream of data but need to create an application that inserts the data into the database.	+
M7	Same attributes	Same attributes are provided by BGPmon.	++
M8	eBGP multihop	Possible.	++
M9	Scaling	Is possible.	++
Should:			
S1	Live data stream	It only provides users/programs with a live data stream.	++
S2	Integrity of data	Had some memory issues and missed some updates in the past.	-
S3	Extensible	Can modify the source code, but there is not a real API and documentation. Besides the comments in the source code.	--
Could:			
C1	High resolution timestamp	Need to modify the code to add this attribute, requires some effort.	-
C2	Authorisation & encr.	Provides authentication but no encryption.	+

7.3. OSR Quagga

The Open Source Routing is supported by ISC and aims to support the community in releasing a mainstream, and a stable routing code to enable network innovation. They focus on Quagga and partners with the existing developer community, independent code committers, service providers and academic institutions to deliver a higher quality code base for Quagga.

They provide the following services to the Quagga community:
- Release Management
- Release Testing
- Bug Management
- Development
- Support

When OSR has developed new features or has changed the code, it will push the code upstream in the mainstream Quagga releases. A diff check on the source code of Quagga and the OSR's code reveals that there are no differences in the implementation.

The OSR team also advises to wait when the code is implemented in the Quagga mainstream releases and advises to use the OSR's code only for testing and development [24].

Pros
- Active community
- OSR Quagga will implement new features if they are requested[25]

Cons:
- There is no documentation available
  - Only comments in the code
- The OSR's code, that is accessible on Github, is nearly the same as Quagga[26]
- This will result in using the same application and structure as is being used now

[24] To see this statement, please visit the following website: https://github.com/opensourcerouting/quagga (17-02-2014)

[25] This only applies if it is strongly needed by the user base. For more information, please visit the following website: http://opensourcerouting.org/about-us (17-02-2014)

[26] If the differences of the compared code are related to this project

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	Is possible	++
M2	MRT-formatted files	Is possible, but exactly the same as Quagga	++
M3	Raw data	Is in the MRT-formatted files	++
M4	BGP metadata	Is in the MRT-formatted files	++
M5	Ordering	Is already done when it is dumped in the MRT-formatted files	++
M6	Less delay	Not really, same implementation as Quagga	--
M7	Same attributes	It is the same, since its core is Quagga	++
M8	eBGP multihop	Possible	++
M9	Scaling	Not really possible, as it basically is Quagga (which cannot handle any more peers)	--
Should:			
S1	Live data stream	Not provided or need to create locking and watch the dump files	--
S2	Integrity of data	As RIPE NCC currently suspects Quagga of missing updates it will also miss some updates here	-
S3	Extensible	If code needs to be changed it is better to modify the mainstream code of Quagga	--
Could:			
C1	High resolution timestamp	If code needs to be changed it is better to modify the mainstream code of Quagga, requires some effort	--
C2	Authorisation & encr.	Not provided by this implementation, but could use a SSH tunnel.	+

7.4. BIRD Internet Routing Daemon

The BIRD Internet Routing Daemon was developed as a school project at the Faculty of Math and Physics in Charles University Prague. It received contributions from Martin Mares, Pavel Machek and Ondrej Filip and is now sponsored and developed by CZ. NIC Labs. The project aims to develop a fully functional dynamic IP routing daemon primarily targeted on Linux, FreeBSD and other UNIX-like systems.

It supports multiple routing protocols such as BGP, RIP, OSPF, and maintains multiple routing tables. BIRD is single threaded and can save the states and/or messages in a MRT-formatted file when they are received by the application. It is not intended to store all BGP updates every five minutes. After a message or update message has been received it will directly append it to the file that was configured to store this information. It is more CPU and memory friendly than Quagga is. This is showed in the following image[27].

BIRD vs other daemons

- Test 1

Daemon	Memory (MB)	CPU (sec)
Quagga	90 + 77 = **167**	32 + 120 = **152**
BIRD	**30**	**14**

- Test 2

Daemon	Memory (MB)	CPU (sec)
Quagga	**87**	**30**
BIRD	**30**	**7**
OpenBGP	33 + 18 = **51**	10 + 7 = **17**

It is built with a modular design in mind and the BGP protocol is implemented in three different parts: bgp.c will handle the connections and most of the interfaces with BIRD its core, packets.c will handle the incoming and outgoing BGP packets and attr.c contains functions to manipulate the BGP attributes list[28]. In theory it is possible to add our own code to receive the packets and send it to a queueing system, which can be used to retrieve the information for post processing. It is likely that these code changes will not make it in the mainstream releases, since our use case is a special one. Newer version of BIRD could override the code changes or make it incompatible, which will cost time to fix such issues.

---

[27] Image acquired from the following website, it also provides more information about the comparison: https://www.nanog.org/meetings/nanog48/presentations/Monday/Filip_BIRD_final_N48.pdf (NANOG, NANOG meeting 48, 17-02-2014)

[28] Information gathered from comments in the code.

Pros
- Memory friendly
- CPU friendly
- Can save messages into MRT-formatted files[29]
  - Messages
  - States

Cons
- Single threaded, might face similar utilisation issues as experienced with Quagga
- Only able to write dumps to a file[30]
  - Not possible to invoke another program
- It will involve a similar implementation as the one that is used now
- Data can be corrupted while reading the output file
  - Quick glimpse at the output file reveals that it is not closed by BIRD when it is running
  - This can cause corrupted data or half received updates when the file is being read

[29] More information can be found in the configuration section of the website: http://bird.network.cz/?get_doc&f=bird-3.html#ss3.2 (BIRD)

[30] If you look at the configuration options, there is no configuration option available to invoke another program for processing the MRT data.

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	Possible.	++
M2	MRT-formatted files	Dumps updates straight into the dump file.	++
M3	Raw data	Is available in MRT-formatted file.	++
M4	BGP metadata	Stored in the MRT-formatted files.	++
M5	Ordering	Ordered in the MRT-formatted files	++
M6	Less delay	Not really, an application has to read the contents of the file. But the file is constantly opened by BIRD and BIRD can write into the file when our application wants to read from it. Need to add a locking mechanism if the application needs to read the dump file.	--
M7	Same attributes	Same attributes are stored in the MRT-formatted file.	++
M8	eBGP multihop	Possible.	++
M9	Scaling	Not really or an application must push the messages into a queue. Otherwise Alpaca has to retrieve all the dumps from the RRCs. But utilises less memory and CPU than Quagga, but still single threaded.	-
Should:			
S1	Live data stream	Need to modify the code, because every-thing is stored in a file and BIRD does not provide an API or locking mechanism able to read the contents of the file safely.	-
S2	Integrity of data	Updates are processed and can be dumped.	++
S3	Extensible	Source code is publicly available and there is some documentation in the source code.	-
Could:			
C1	High resolution timestamp	Need to modify the source code to add this functionality, requires some effort.	-
C2	Authorisation & encr.	Not provided by this implementation, but could use a SSH tunnel.	+

7.5. OpenBGPD

OpenBGPD is a free implementation of BGP version 4. The project started out of dissatisfaction of other implementations that existed at that time and is now a fairly complete BGP implementation. It is used by different users and advertises that users often praise its ease of use and high performance, as well as its reliability[31] on their website. The latest release went live on the first of November 2009, which carries the version 4.6.

OpenBGPD can create dumps at regular time intervals and will create a MRT-formatted file for it. It is possible to give all the files a different filename based on the current time. In this way it is possible to distinguish the different updates from each other[32].

The application uses multiple processes and is relatively as efficient as BIRD which is revealed by the following image[33].

BIRD vs other daemons

- Test 1

Daemon	Memory (MB)	CPU (sec)
Quagga	90 + 77 = **167**	32 + 120 = **152**
BIRD	**30**	**14**

- Test 2

Daemon	Memory (MB)	CPU (sec)
Quagga	**87**	**30**
BIRD	**30**	**7**
OpenBGP	33 + 18 = **51**	10 + 7 = **17**

OpenBGPD uses three processes which have the following tasks[34]:

- Session Engine (SE): Which manages the BGP sessions
- Route Decision Engine (RDE): Which holds all the BGP tables and takes the routing decisions
- Parent: Add routes to the kernel, starts SE and RDE

There is not much documentation available and more research on the source code needs to be done before the decision can be made to use OpenBGPD.

---

[31] For more information, please visit the following website: http://www.openbgpd.org/ (19-02-2014, OpenBGPD)

[32] For more information about the dump capabilities in OpenBGPD, please visit the following website: http://www.openbsd.org/cgi-bin/man.cgi?query=bgpd.conf (OpenBGPD)

[33] Image acquired from the following website, it also provides more information about the comparison: https://www.nanog.org/meetings/nanog48/presentations/Monday/Filip_BIRD_final_N48.pdf (NANOG, NANOG meeting 48, 17-02-2014)

[34] For more information, please visit the following website: http://www.openbsd.org/cgi-bin/man.cgi?query=bgpd (OpenBSD, Man page, 02-06-2014)

Pros
- Can create a dump file in a MRT-format
- Messages and full tables can be exported in separate MRT-formatted files
- A regular time interval can be configured for the dumps

Cons
- Does not provide users with a live stream of data
- There is no real difference in the implementation from the one that is now being used
  - It is only utilises less CPU and memory
- Takes more time to research the source code and create a module which can store the received packets in a queue
- Not really a modular design that allows to invoke other modules without changing the source code[35]

[35] This is concluded after researching the source code

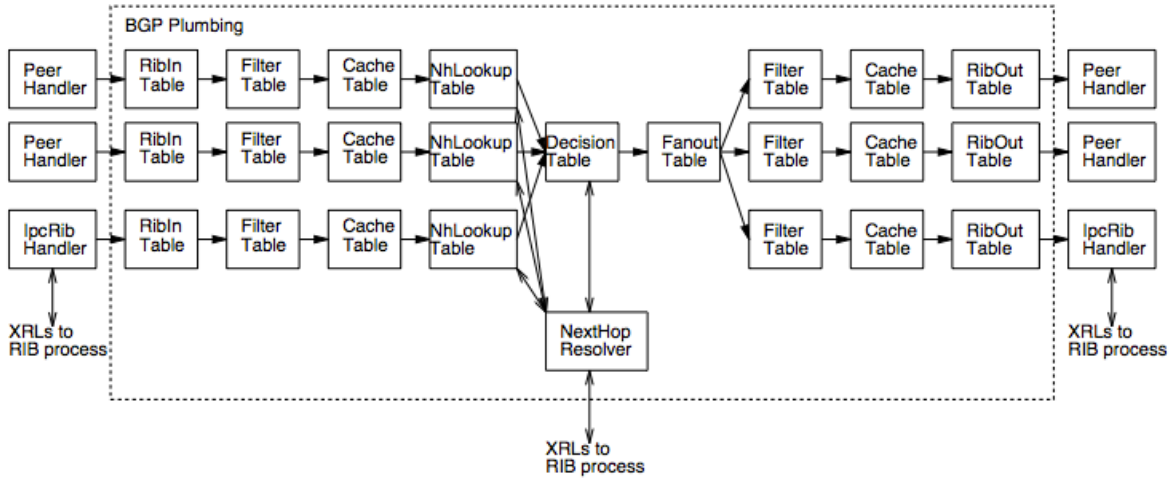
MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	Possible.	++
M2	MRT-formatted files	Periodically dumps updates in a MRT-formatted file.	++
M3	Raw data	Is available in the MRT formatted file.	++
M4	BGP metadata	Is available in the MRT formatted file.	++
M5	Ordering	Is available in the MRT formatted file.	++
M6	Less delay	Basically the same implementation as we have now, need to change the code to provide a live stream.	--
M7	Same attributes	Is in the MRT-formatted file.	++
M8	eBGP multihop	Possible.	++
M9	Scaling	Not really, if the default installation of OpenBGPD is used. Need to modify the code to get a live stream and push it somewhere. More CPU and memory friendly than Quagga and uses multiple processes.	-
Should:			
S1	Live data stream	Not possible with the default installation, need to modify the code.	--
S2	Integrity of data	Updates are saved in the MRT-formatted files.	++
S3	Extensible	Not much documentation online or in the code.	--
Could:			
C1	High resolution timestamp	Normal epoch timestamp, need to modify the code to add a high resolution timestamp, requires some effort.	-
C2	Authorisation & encr.	Not provided by this implementation, but could use a SSH tunnel.	+

7.6. XORP

XORP, which is an abbreviation for eXtensible Open Router Platform, supports OSPF, BGP, RIP, PIM, IGMP, OLSR. The project has been initiated to create an open source extendible routing platform where researches could perform tests on, but still make it a solid and stable platform. It is created with security in mind, and if one routing instance fails it will not drag the other ones down with it [36]. XORP creates a RIB per client, but it does not provide a built-in function to dump/save a RIB to a file. The following image is an overview of the BGP process as it is implemented in XORP[37].



The IPC system that XORP uses is named XORP Internet-Process Communication (XIPC) in combination with the XORP Resource Locators. The XRLs are responsible for describing the inter-process communication mechanisms and their associated arguments. It is represented in a human readable form which lends itself for easy manipulation with editing tools and can be invoked from the command line. XORP processes export XRL interfaces to a process that is known as the Finder and informs the system about which IPC schemes are available to invoke each XRL. The Finder is then able to provide a resolution service for other processes.

In order to get the update information out of XORP, a module needs to be developed which uses XRL to communicate with XORP. This only gets the information out of XORP and further processing needs to be done (i.e. putting it into a queue and inserting it into the database). Since XORP does not provide a feature or function to get the information out easily, it may take some time to develop such a module.

---

[36] For more information, please read the following paper: http://read.seas.harvard.edu/~kohler/pubs/xorp-hotnets02.pdf (19-02-2014, ICSI Center for Internet Research)

[37] For more information about the BGP implementation in XORP, please read the following document: https://github.com/wmiltenburg/xorp_documentation/blob/master/bgp/bgp.pdf (XORP, 19-02-2014)

Pros

- Multi process architecture (per protocol)
- Built from a research mindset
- Security has a high priority
- It is a modular design that allows to create a module that uses the API to gather information form XORP

Cons

- Does not have a built-in function to store or dump the RIB table or UPDATES
- Multiple RIBs exist and the to-be-developed module must retrieve the contents of multiple RIBs[38]
  - urib4
  - urib6
  - mrib4
  - mrib6
- It may take some time to develop a module which retrieves the information from XORP
- Latest official release was on January 11th 2012
  - This may cause a problem when new attributes are used in BGP updates and which need to be parsed (which is done by XORP)
- Invoking the API could cause a lock and could deteriorate the performance of XORP when there are a lot of prefixes in the RIBs
  - Need to perform measurements with the prototype if this is the chosen alternative

[38] According to the following article: http://mailman.icsi.berkeley.edu/pipermail/xorp-hackers/2009-November/002229.html (XORP-hackers mailing list, 19-02-2014)

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	Possible.	++
M2	MRT-formatted files	Need to build our own module to create MRT files.	--
M3	Raw data	Need to build our own module to retrieve the raw data from XORP.	--
M4	BGP metadata	Need to build our own module to retrieve the attributes from XORP.	--
M5	Ordering	Depends on the module that saves the attributes or prefixes.	--
M6	Less delay	It is a multi-process architecture and extensible. Need to create a module to store the RIB somewhere else.	+
M7	Same attributes	Need to build our own module.	-
M8	eBGP multihop	Possible.	++
M9	Scaling	It uses a process-per-protocol model and is extensible.	+
Should:			
S1	Live data stream	Not provided by the current default installation.	-
S2	Integrity of data	Need to build our own module, which receives the data and makes sure that the integrity of the data is not violated.	-
S3	Extensible	This is the main focus of XORP, to create an eXtensible Open Router Platform.	++
Could:			
C1	High resolution timestamp	Need to modify the source code or create our own module, requires some effort.	-
C2	Authorisation & encr.	Not a feature, but could use SSH tunnel.	+

7.7. Vyatta (Free Community Version)

Vyatta provides a software-based virtual router, virtual firewall and VPN solution for the Internet Protocol network. The network routing software engine was XORP and has been replaced by Quagga on April 2008[39]. In the year 2012, Brocade Communications Systems acquired Vyatta and renamed it: “Vyatta, a Brocade Company”[40]. In April 2013, Brocade renamed the Vyatta Subscription Edition (VSE) to Brocade Vyatta 5400 vRouter[41]. Their latest commercial release of the Brocade vRouter is no longer open source based.

Vyatta is not only a routing engine, it is much more than that. Since its routing engine core relies on Quagga, it does not provide any extra functions for logging or dumping the BGP updates, Vyatta itself has not implemented extra functionality to dump or export the BGP messages/updates.

Since their latest release is not open source anymore, a possible scenario is that the open source version will not be maintained by Brocade anymore. It must also be kept in mind that the open source project has been acquired by a commercial company which might no longer be involved in the open source project since they want to keep some of their code proprietary.

Pros

- Supports multiple routing protocols
- Has many extra/additional features

Cons

- Still uses the Quagga engine
  - The same implementation, as is in use now, must be used
- Shipped with many extra functions which could cause overhead
- Does not provide any functions besides the one provided by Quagga, that are considered in scope of this project

---

[39] For more information, please visit the following website: http://www.vyattawiki.net/wiki/Quagga (Vyattawiki, 02/06/2014)

[40] For more information, please visit the following website: http://newsroom.brocade.com/press-releases/brocade-acquires-vyatta-a-pioneer-and-leader-in-s-nasdaq-brcd-0949599#.U5WceJSSyf8 (Brocade, 2012)

[41] For more information, please visit the following website: http://www.brocade.com/products/all/network-functions-virtualization/product-details/5400-vrouter/index.page (Brocade)

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	The engine is Quagga, so it provides us with the functionality to announce anchor/ beacon IPs	++
M2	MRT-formatted files	The routing engine is Quagga, so it can make dumps in a MRT-format	++
M3	Raw data	Is in the MRT-formatted dumps	++
M4	BGP metadata	Is in the MRT-formatted dumps	++
M5	Ordering	Is in the MRT-formatted dumps	++
M6	Less delay	It uses the Quagga engine, and it can periodically dump its RIB and the updates it received. It does not differ from the current implementation and will cause the same delay	--
M7	Same attributes	Same attributes are provided as it is the same routing engine that is used now	++
M8	eBGP multihop	Is supported by the Quagga engine	++
M9	Scaling	The same scaling issues will be encountered using Vyatta	--
Should:			
S1	Live data stream	Does not provide a feature for a live data stream	--
S2	Integrity of data	Some updates are missed using Quagga; RIPE NCC experiences this problem with the current implementation, it is likely to encounter the same issue	-
S3	Extensible	It is open source with a community, but its routing engine still relies on Quagga which is not well documented	-
Could:			
C1	High resolution timestamp	Must change the source code for it, requires a lot of effort	--
C2	Authorisation & encr.	Not a feature, but could use SSH tunnel	+

7.8. PyRT

Python Routing toolkit is made for the purpose of collecting route information. The project was last updated on May 5th 2002. It only supports one peering session according to the documentation. This could be solved by running multiple instances of the program per session. It saves all information into a MRT formatted file, but this file will only contain a dump per BGP update that the program has received. In order to create a RIB table, the application needs to maintain the state of the RIB table and cannot miss any updates that are sent to the RRC. If this happens it will lead to an inconsistent or corrupted state of the RIB table.

The functions which the application provides are separated in different files. There is a file especially for handling the BGP connections, but also one for the MRT dump, and one for the table dump. PyRT does not specify its API anywhere, but there are some comments in the source code which clarifies what the use of some code is.

When the application was tested, in the week of 17th February, it crashed when it connected to a peer. Since there are not any updates available, and it seems that it is not that actively used, it is not a really feasible solution.

Pros

- Can save received information to a MRT-formatted file
- Can be a multi-process application when it is started multiple times with a different peer to connect to
  - A process-per-peer model

Cons

- Last update was on the May 5th 2002[42]
- Crashes when it connects with a peer
- Does not send keepalives according to the to do section[43] and does not send keepalive messages automatically[44]
- It is a listener, does not send prefixes
- If it receives invalid attributes it will close the connection or hang
- No eBGP multi-hop feature
- Not really scalable, connection gets closed since it is not sending keepalive messages. Only receiving initial RIB tables then eventually the connection will get closed.

Most of the statements that are made above relies on the information that is available in the comments of the source or the ‘README’ file[45].

---

[42] For more information, please download the package and read the ChangeLog: https://research.sprintlabs.com/pyrt/results/pyrt-2.5.tar.gz (PyRT)

[43] For more information, please visit the following website: https://github.com/mor1/pyrt/ (PyRT Github page)

[44] When you take a look at the while loop, it is not sending any keepalive messages: https://github.com/mor1/pyrt/blob/master/bgp.py (PyRT, Github, 02-12-2013)

[45] For more information, please read the following ‘README’: https://github.com/mor1/pyrt/blob/master/README (PyRT)

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	PyRT is only a BGP listener and does not announce prefixes.	--
M2	MRT-formatted files	It dumps the updates in an MRT-formatted file.	++
M3	Raw data	Is stored in the dump files.	++
M4	BGP metadata	Not all messages are adhered and parsed. Not all information is received and put into the MRT dumps.	--
M5	Ordering	It dumps the updates sequentially into the MRT-format.	+
M6	Less delay	It only writes to a file, same concept would be used.	-
M7	Same attributes	Same attributes are stored into the MRT-formatted files.	++
M8	eBGP multihop	It does not provide a feature for eBGP multi-hop.	--
M9	Scaling	It can only connect with one peer and does not send any information back (keepalives).	--
Should:			
S1	Live data stream	Puts all updates into a file. Need to create a module which retrieves this information and places it into a queue.	+
S2	Integrity of data	No keepalives are sent after the connection is established. Connection could get torn down which can result in an incomplete RIB.	--
S3	Extensible	It is open source and it is possible to create modules. A default installation does not work which needs to be fixed first, then needs to be checked if it supports AS 4 byte numbers.	--
Could:			
C1	High resolution timestamp	Not supported, need to modify the source code, but support high time stamps in MRT.	-
C2	Authorisation & encr.	Not a feature, but could use SSH tunnel.	+

7.9. BMP (BGP Monitoring Protocol)

Another solution could be the use of the BGP Monitoring Protocol (BMP) that is supported by Cisco and Juniper. It is specifically designed to monitor the BGP updates and states of peers. All messages that are stored in the Adj-RIB-In of the neighbour will be sent to the monitoring station, which then sends the message to the server for further processing. The BGP update messages received by the peer will be encapsulated in Route Monitoring messages.

The only drawback is that BMP is not widely in use today and that there are not many BMP clients or servers. This requires a lot of effort to receive and process the data as is required by the requirements.

Pros

- Using a protocol that is specifically designed to suit our desires
- Does not interfere with the routing engine, so you can use it on your production router[46]

Cons

- Need to buy hardware that supports BMP
- Not used in general
- Neighbouring peers must support this feature
- Not many BMP servers available
- It is still a draft[47]
- Need to create an application that generates the MRT formatted files

[46] It is stated in the draft that is is minimally service-affecting. For more information, please read the following website: http://tools.ietf.org/html/draft-ietf-grow-bmp-07 (BMP, IETF, draft)

[47] Current version is available on the following website: http://tools.ietf.org/html/draft-ietf-grow-bmp-07 (BMP, IETF, draft)

MoSCoW comparison

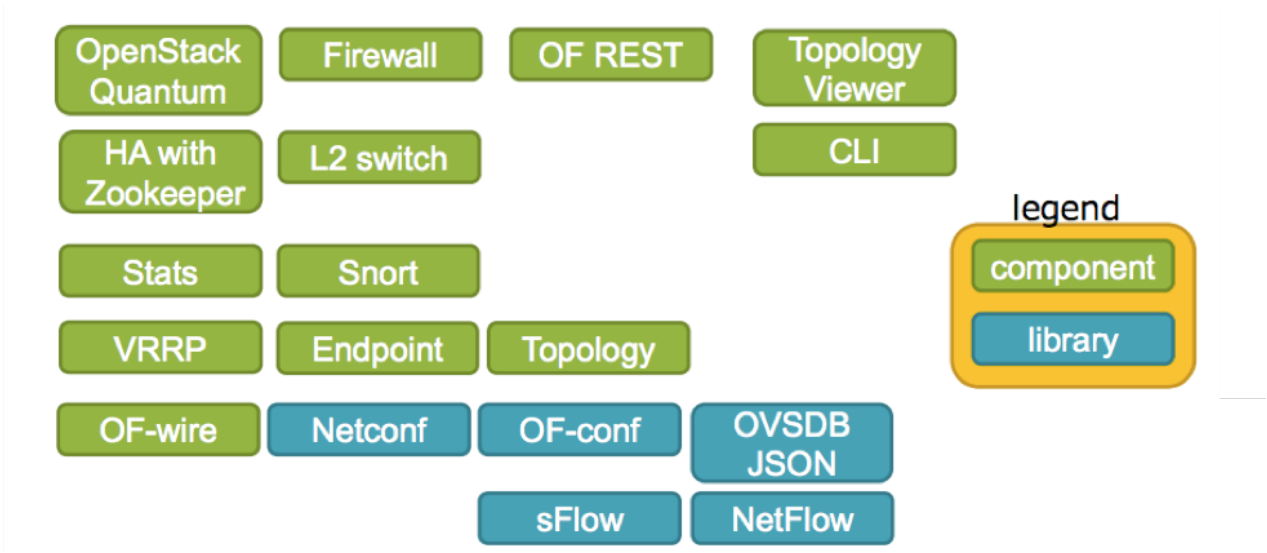
This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	Not supported by BMP itself, this needs to be done by another process on the router	-
M2	MRT-formatted files	Not supported by BMP on the routers	--
M3	Raw data	Is stored in the BGP Update messages	--
M4	BGP metadata	This protocol is especially developed to receive information about BGP.	++
M5	Ordering	There is no requirement on the ordering of messages in the peer dumps	--
M6	Less delay	All the information is saved on the router itself and not transmitted to or dumped into another process. Need to retransmit the traffic to another application	-
M7	Same attributes	Stored in the raw updates	++
M8	eBGP multihop	There is nothing noted in the draft that explicitly requires a peer to connect with a local neighbour	++
M9	Scaling	It is a really lightweight process on the routers itself as its utilisation is limited	+
Should:			
S1	Live data stream	BMP does not provide a client with a live data stream. The peer should attempt to generate updates as soon as they are received by the peer.	--
S2	Integrity of data	It also sends the raw BGP Update message, authentication to this data needs to be provided by a separate application	+
S3	Extensible	Need to develop an in house application that provides this extensibility	--
Could:			
C1	High resolution timestamp	Not supported by BMP on the routers, hard to change the OS of the routers	--
C2	Authorisation & encr.	Not a feature, but could use SSH tunnel	+

7.10. Ryu

Ryu is a component-based software defined networking framework that is actively maintained by a community. It is a framework that can be used to receive BGP updates and do further processing with this data. In order to support such a case, an application must be developed which uses the Ryu framework.

Ryu uses multiple processes and threads to provide all the functionalities. There is a lot of documentation describing their API and how it can be used in another program. It has a BGP parser and serialiser to perform the decoding and encoding of the BGP packets. The following image shows the components and libraries that are included in Ryu[48]:



To maintain the RIB state, information needs to be retrieved from Ryu by another application. This application should also maintain state and process the messages. Other 'possible alternatives' already provide similar functionalities and does not require that the application should maintain the BGP connection.

[48] Image acquired from the following website: http://osrg.github.io/ryu/slides/ONS2013-april-ryu-intro.pdf (26-02-2014)

Pros
- Active community
- Regularly updated
- Framework written in Python
- Ability to create our own BGP implementation

Cons
- Need to write our own module to save the received route information in an intermediate form
- Some other applications already provide us with more features and already handle the BGP connections
    - It requires more work to suit all our wishes, since the application that needs to be developed, also needs to maintain the BGP connection

MoSCoW comparison

This subchapter compares the possible alternative with the MoSCoW scheme.

Must:		Description	Score
M1	Announcements of anchor/beacon IP	As it is a library our own application must create this functionality relying on the library to create the packets	-
M2	MRT-formatted files	Need to create our own module to dump all updates into a MRT-formatted file	--
M3	Raw data	It is an API so it is very flexible. Our own application can store the raw update somewhere	-
M4	BGP metadata	As it is a library our own application must store the metadata somewhere	-
M5	Ordering	Needs to be done by our own application	-
M6	Less delay	Because it is an API, it depends on our own program if it involves less delay.	+
M7	Same attributes	Need to be facilitated by our application	-
M8	eBGP multihop	As it is a library, our own application must create this functionality using the library	--
M9	Scaling	Because it is an API, it depends on our own program if it could be scalable.	+
Should:			
S1	Live data stream	Every packet can be received by our own application. It is a design decision if a live data stream is provided by our in house developed application	+
S2	Integrity of data	Depends on our own application, if it processes every update and provides a feature for authorisation	+
S3	Extensible	It is an API and our own application could be extensible	++
Could:			
C1	High resolution timestamp	Need to be facilitated by our own application, requires a lot of effort	-
C2	Authorisation & encr.	Not a feature, but could use SSH tunnel	+

7.11. Summary

This subchapter will give the reader a quick overview of the total score of every possible alternative.

The total score is listed below in a diagram:

Possible alternative	Score ++ (+2)	Score + (+1)	Score - (-1)	Score -- (-2)	Total score
Custom solution with ExaBGP	6	6	1	1	15
BGPmon	6	2	3	3	5
OSR Quagga	7	1	1	5	4
BIRD Internet Routing Daemon	8	1	4	1	11
OpenBGPD	8	1	2	3	9
XORP	3	3	4	4	-3
Vyatta	7	1	2	4	5
PyRT	3	3	2	6	-5
BMP (BGP Monitoring Protocol)	3	3	2	6	-5
Ryu	1	5	6	2	-3

The following points are given to get to a total score:
- ++ is worth 2 points
- + is worth 1 point
- - is worth -1 point (subtraction)
- -- is worth -2 points (subtraction)

Be aware that the total score is not the only criteria that is used to draw a conclusion. It is really important that the ‘must have’ requirements can be all met, otherwise the application will miss some requirements that are the most important ones.

8. Other modules/implementations

This chapter will describe other modules or implementations that are needed to deliver a complete product[49]. Some applications require other software or products to be able, for example, to do further processing.

8.1. Queueing mechanisms

This chapter will describe the queueing mechanisms that could be used when another program/module wants to put all the updates into a queue. The queueing mechanisms that are described in subsequent chapters are the most known well-used queueing mechanisms that are available.

8.1.1. Apache Kafka

Apache Kafka is a queueing system that originates from a LinkedIn project to build a more scalable queueing system which can handle more requests than other traditional systems back then. They have used the system in production and open sourced the code, Apache Kafka took the project and it is now an 'incubated' project[50].

As Apache Kafka advertises itself, it is a high-throughput distributed messaging system that can handle more requests than their competitors[51]. Kafka works with Zookeeper to be able to create a cluster of Kafka servers and provides a user with different topologies. A user can create different topics, let's say per RRC, and can create multiple partitions, let's say a peer neighbour. A single partition must fit on the server that hosts it, but an arbitrary amount of partitions can be created and can be replicated to other servers. One server always acts as the leader, and several other servers can act as followers. When the leader crashes, one of the other followers will take over his role. No data is lost in this case and the consumer can still retrieve the information that exist in these queues. Contents of the messages are saved on disk, which can be configured, so if one machine crashes it can create the queue as it was based on the contents of the backup file. Data that is stored in a partition is always in an ordered sequence.

When a consumer retrieves information from a queue, the message will still exist after the consumer has completed the task. This makes it possible to retrieve the information using another consumer, or to re-read messages when a consumer wants to do so.

When a producer wants to add messages to the queue, it can wait for an acknowledgement from Kafka. In this way it will know if the message has been added to the queue. From a consumer's point of view, it can periodically commit its offset to Zookeeper.

---

[49] Only the implementations that made a reference to this chapter

[50] For more information about the incubation process of Apache, please visit the following website: https://incubator.apache.org/incubation/Process_Description.html

[51] For more information, please visit the following websites: http://www.quora.com/RabbitMQ/RabbitMQ-vs-Kafka-which-one-for-durable-messaging-with-good-query-features, http://www.slideshare.net/charmalloc/apache-kafka

8.1.2. RabbitMQ

RabbitMQ is an open source message broker software that uses the Advanced Message Queueing Protocol (AMQP). It is written in Erlang and is built on the Open Telecom Platform framework and is released under the Mozilla Public License. Rabbit Technologies Ltd. develops and provides support for RabbitMQ. Rabbit Technologies was founded in the year 2007 and was back then a joint venture between LShift and CohesiveFT. It was acquired in April 2010 by SpringSource, which is a division of VMware and it became part of GoPivotal in May 2013.

RabbitMQ can process less messages per second than Kafka can, but it is a very solid implementation of a message queue. It provides both at the producer's and consumer's side features to acknowledge the push-and-retrieve operations. Using these acknowledgements the producer is assured that RabbitMQ has received and processed the message. The consumer acknowledges it when it has received the message and after this acknowledgement the message will be deleted from the queue.

It is possible to create a cluster of RabbitMQ servers, but messages are not replicated among the other server on default. RabbitMQ allows to mirror the queues among the other servers and create a master/slave topology. If the master crashes, one of the eldest servers will become the master, but it is possible that some messages are re-delivered to the queue, due to the fact that their acknowledgements have not been received by the master and the messages were simply not removed from the queue.

Messages are written to disk, so when there is a power failure, RabbitMQ will read the content of the queues from disk. The producer has only to implement the publisher's confirmations to be assured that the messages have been written to disk. It also supports SSL for securely transmitting the messages and authentication to give access only to the machines which may access the queues.

8.1.3. HornetQ

HornetQ is an open source project for building a multi-protocol, embeddable, very high performance, cluster, asynchronous messaging system from JBoss. The code base was originally developed under the name JBoss Messaging 2.0. It started with Tim Fox as the project leader[52]. The current project leader is Clebert Susconic with his core engineers Andy Taylor, Francisco Borges, Howard Gao, and Jeef Mesnil. The project itself was released on 24 August 2009.

HornetQ supports the Stream Text Oriented Messaging Protocol (STOMP) and is JMS compliant. It has the capability to create clusters and messages which can be replicated among other servers. To create a cluster a user can either configure the servers to form a cluster, or a user can use the server discovery feature. If server discovery is enabled, the servers will propagate its connection details to other servers. This requires less configuration than the manual configuration.

The project praises itself for their fast journaling system. They say that the journaling system is faster than their competitors [53]. In case the server crashes, it will be faster up and running than other queueing/message brokers implementations.

When a consumer retrieves information from a queue, HornetQ will automatically delete the message. The consumer has to send an acknowledgement before the message gets deleted from the queue.

8.1.4. ØMQ/ZeroMQ

ØMQ, also known as ZeroMQ, is a high-performance asynchronous messaging library which can be used in scalable or concurrent applications. The library provides a message queue, but unlike message-oriented middleware, a ØMQ system does not need a dedicated message broker. The library is designed to have a familiar socket-style API.

It is developed by a large community of contributors which is originally founded by iMatix Corporation. iMatix holds the domain name and the trademarks and have third-party bindings for many popular programming languages.

As it is a library, it depends on our own implementation if it will support clustering and replication. It will involve more work, since the whole queue facility has to be developed. On the other hand, it allows us to create a queueing solution that is developed to meet our requirements.

---

[52] Until October 2010

[53] For more information about the journaling system, please visit the following websites: https://community.jboss.org/wiki/HornetQFeatures, http://hornetq.blogspot.nl/2009/08/persistence-on-hornetq.html (21-02-2014)

9. Conclusion

This chapter draws a conclusion from the information that has been outlined in earlier chapters.

9.1. Total score overview

The following diagram will show a total overview of the scores that are listed in the MoSCoW scheme per 'possible alternative':

Possible alternative	Score ++ (+2)	Score + (+1)	Score - (-1)	Score -- (-2)	Total score
Custom solution with ExaBGP	6	6	1	1	15
BGPmon	6	2	3	3	5
OSR Quagga	7	1	1	5	4
BIRD Internet Routing Daemon	8	1	4	1	11
OpenBGPD	8	1	2	3	9
XORP	3	3	4	4	-3
Vyatta	7	1	2	4	5
PyRT	3	3	2	6	-5
BMP (BGP Monitoring Protocol)	3	3	2	6	-5
Ryu	1	5	6	2	-3

The following points are given to get to a total score:

- ++ is worth 2 points
- + is worth 1 point
- - is worth -1 point (subtraction)
- -- is worth -2 points (subtraction)

The 'possible alternative' with the highest score is the custom solution that uses ExaBGP as its core. This is a criterion which can be used to evaluate what is the best solution to use as an alternative. Another criterion is, if it is possible at all, to develop and deploy a prototype that is based on this 'possible alternative'. To measure, if this is possible indeed, the scores in the diagram above are used as the score also reflects how much work need to be done in order to meet all the requirements.

Be aware that the total score is not the only criteria that is used to draw a conclusion. It is really important that the ‘must have’ requirements can be all met, otherwise the application will miss some requirements that are the most important ones.

9.2. Conclusion

The conclusion of the above is to use the custom solution based on ExaBGP. It is a very extensible, open source, and lightweight application. It does its job, which is to give another program its output and that program should do the further processing.

Every possible alternative is compared with the requirements that are listed in the MoSCoW scheme. This is to get a good overview if all the requirements can be met and if much effort must be done to meet all the requirements. What also needs to be considered as an important criterion is, if all the ‘MUST’ requirements can be met.

Not all the requirements are met when ExaBGP is used as a standalone application. Another application needs to be developed to do the further processing and that meets some of the “must” requirements. To meet all the “must” requirements it also involves modifying the source code. But when you look at the overall picture, ExaBGP is more flexible and extensible than the other applications and it is possible to meet all the ‘must’ requirements. It allows RIPE NCC to use this implementation even if the use cases will differ in the next years to come. As a side note, the original developer of ExaBGP has shown interest to add the requested features in ExaBGP.

The conclusion of this report is:
“The best alternative is ExaBGP, but some modification is required, as well as some other applications that need to be developed. It is advised to develop a prototype to see if all requirements can be met and if it is the alternative as we have outlined in this research report.”

9.3. Evaluation

The reason for not choosing one of the other possible alternatives is related to the requirements list. For all the possible alternatives, more work is required to meet the requirements. Some of the possible alternatives do not provide us with the basics, this is the case with Ryu where an application needs to be developed that maintains state. The other possible alternatives will require a same implementation that is used now with Quagga. It will lead to a batch oriented process while it is preferred to go to a stream oriented process. Some other alternatives are still drafts or not mature enough to use right now, but it could be promising for in the future (i.e., the BMP protocol). Therefore, the conclusion, which is described above, is that ExaBGP is the ‘possible alternative’ that should be used.

10. Proposal

This chapter describes the proposal that follows the conclusion which has been outlined in the previous chapter.

10.1. Custom solution using ExaBGP

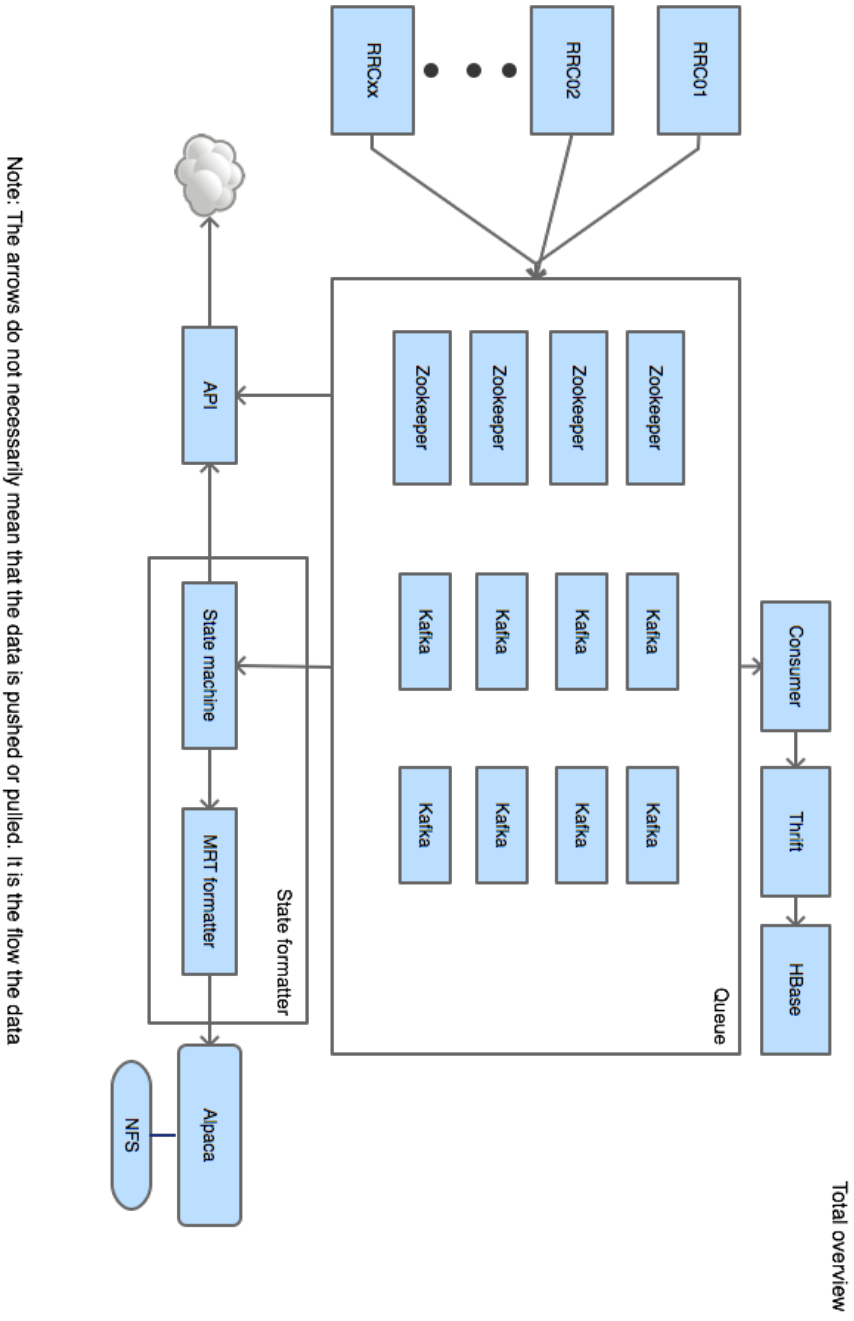
As explained in the previous chapter, the best alternative is ExaBGP with a queueing mechanism. Some additional components need to be developed that rely on a queueing mechanism.

Multiple queueing solutions were evaluated during the research stage. The best queueing mechanism that could be used in the new implementation is Apache Kafka. The most important argument is that it provides the ability for a consumer to re-read packets from the queue. This can be useful in cases were multiple consumers want to read packets from a queue without destroying the data that is stored in this queueing mechanism. In our case, this would be useful if an API is provided to the community. The community can then create a tool that uses this API.

The following needs to be done to meet all the must requirements:

- State machine; a RIB state needs to be created, which is not done by ExaBGP, this information is used to create a RIB dump;
- MRT formatted files; application that retrieves information from the queue and stores it in a MRT formatted dump file;
- BGP metadata; provided by ExaBGP but must be stored somewhere, is also vital to create a RIB-state (if a connection to a peer is lost, the RIB entries from that peer must be flushed);
- Ordering; ExaBGP needs to add a sequential serial number in its output;
- Less delay; mechanism that inserts the data directly into the database;
- RAW data; must be added to the output or be able to receive this.

If all the “must” requirements must be met, it requires developing an application for putting all the ExaBGP messages in a queue system, which can be later on retrieved for further processing. The following image will give the reader a total overview of the system that needs to be developed.



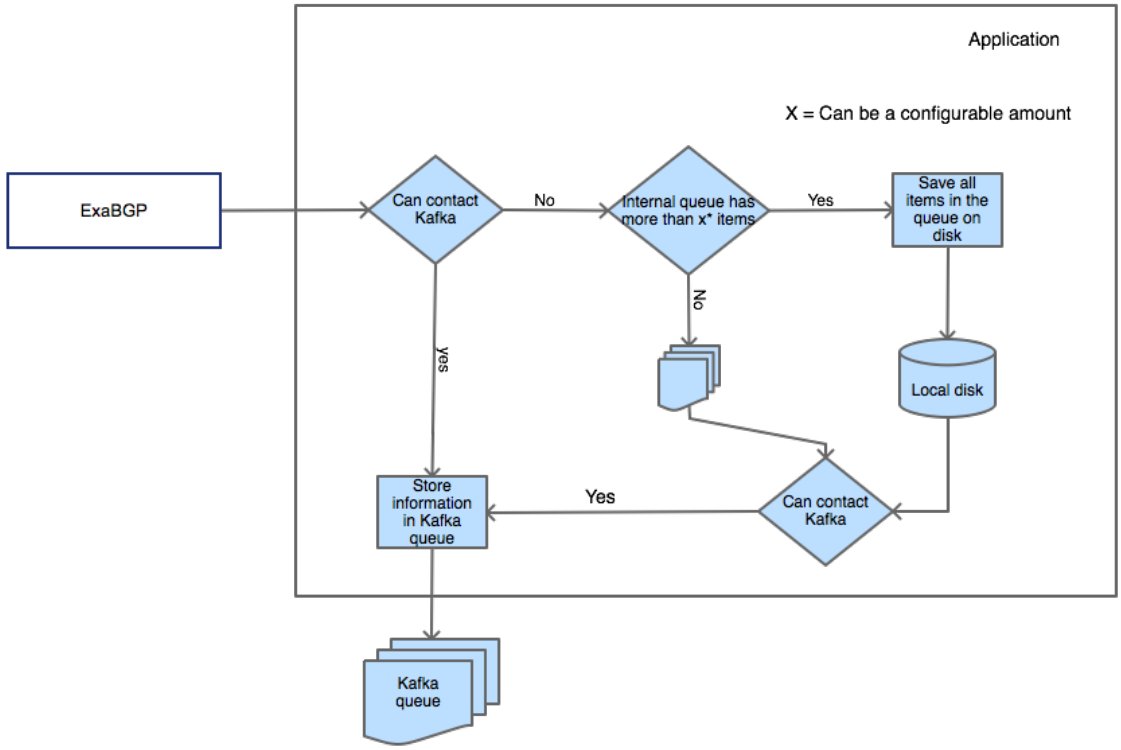
Another separate application retrieves information from the queue, creates a RIB state and stores this in a MRT formatted file. The MRT formatted file contains all updates that were received during the interval. As explained earlier, there is also a separate file, which contains the RIB state which is formatted in a MRT format. This application could be configured to do this periodically, and efficiency could be gained in the way the MRT formatted file is created. This could reduce the latency, which is involved in the current implementation.

The information that is stored in the queue must also contain the BGP state messages and could be stored in a separate topic. Delay is involved when the MRT formatted files are copied from the “state formatter” to Alpaca, and later on, copied to HDFS. To overcome this delay, an application needs to be developed, which directly inserts the information into the database.

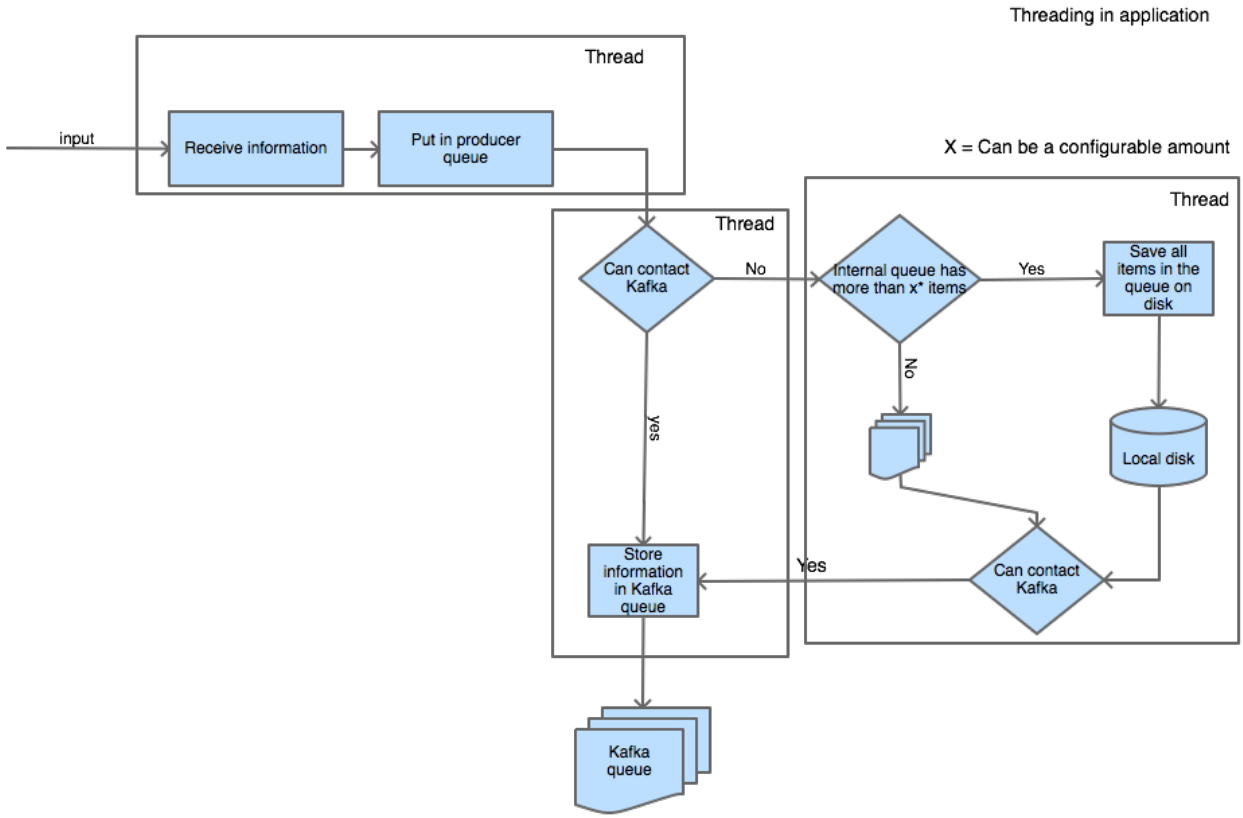
10.2. Producer

The application, which receives the information from ExaBGP on the RRC itself, must provide a mechanism to store all the information that it received from ExaBGP whenever it cannot connect to Zookeeper/Kafa. This is illustrated in the following image.

Application overview, communication between ExaBGP and our own application



The application will use multiple threads to process all information that it receives from ExaBGP. This is illustrated in the following image.

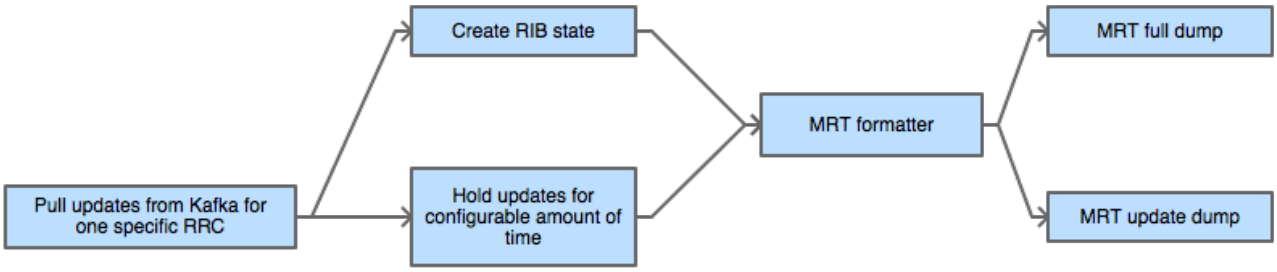


This keeps all the functions separated from each other. There are three threads that provide the following functions:

- **Receiver**: Sending the received information, from ExaBGP, to a producer queue
- **Producer**: Put the information into the queue system or pass the information to another thread if the queue cannot be reached. If the queue is reachable again it will put all the locally stored information in the queue first before it puts the new information into the queue
- **Store**: If the queue cannot be reached this thread will save all the information in another queue or store the information locally on disk.

10.3. State formatter

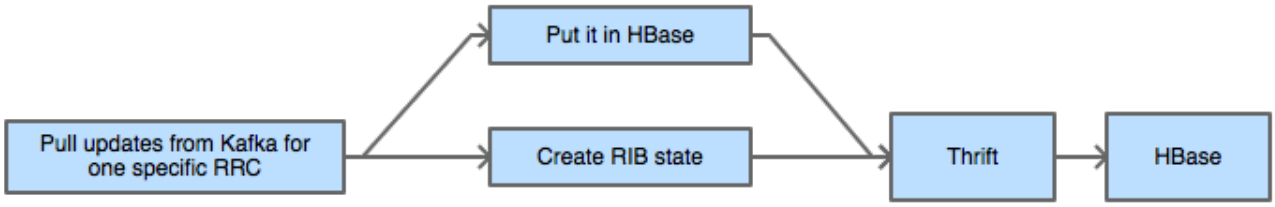
Another application needs to be developed that retrieves the information, from the same queue, and creates a RIB state. This is illustrated in the following image:



The application will receive the information from the queue and stores it in a local queue. After a configurable amount of time it will create two MRT formatted files. One which contains all the updates that ExaBGP received and one that holds the complete RIB state. These MRT formatted files are then copied to Alpaca that stores these MRT formatted files on its NFS-share. This is done to maintain the MRT formatted files that are used by the community, but also by other programs that do further processing on them and insert it into the HBase database. It allows the currently existing mechanisms to continue working and maintains backwards compatibility.

10.4. HBase consumer

To reduce the latency even more, it would be useful to insert the updates straight into HBase, which is illustrated by the following image:



The information that is retrieved from the queue is directly inserted in HBase using Thrift as its 'gateway'. It will also create a RIB state so that it can periodically save a complete RIB state in HBase, this is also done in the current implementation.

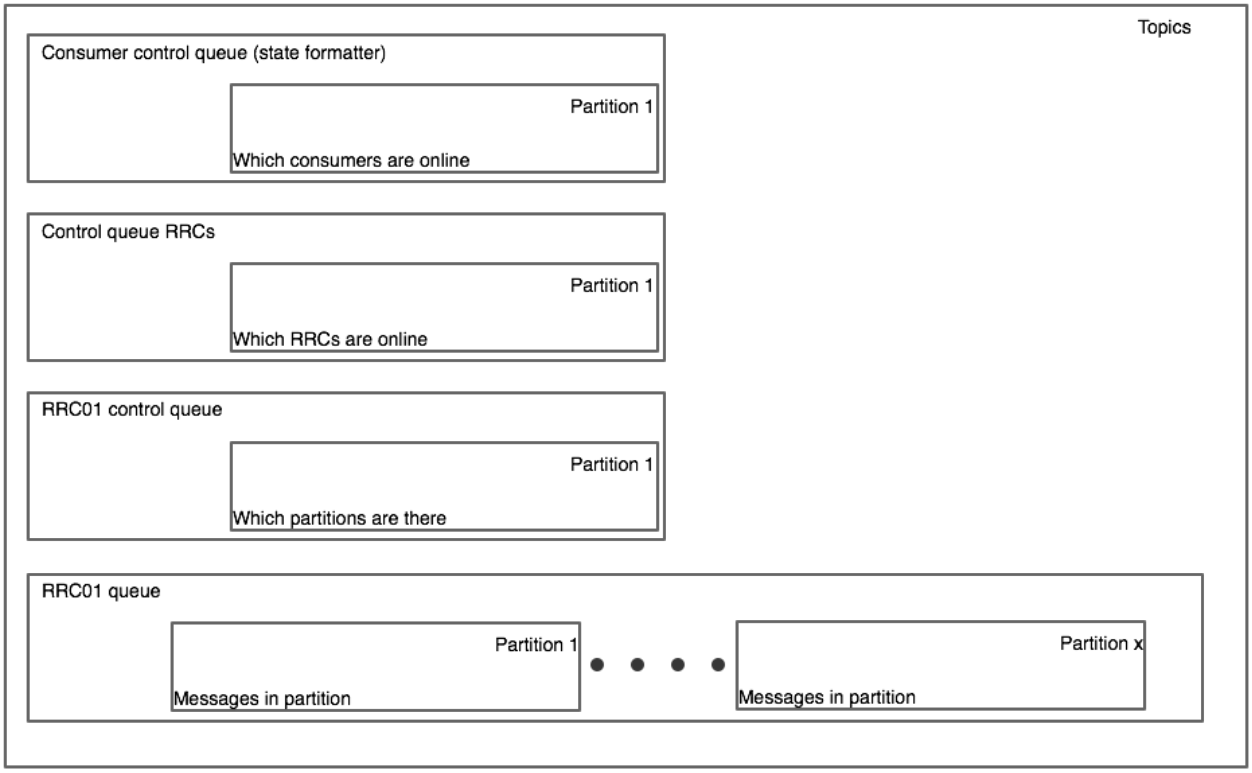
It is basically the same functionality, besides from inserting it into HBase, as the "state machine". The reason that this application does not rely on the data from the "state formatter" is that it could be the case that the MRT formatted files might be deprecated in future. If this is the case, the whole "state formatter" does not need to be any longer used as it does not have any dependencies. There is also the case of locking that might be involved if the HBase consumer relies on the data in the state machine. This involves complexity and could cause race conditions in the HBase consumer if it is not implemented correctly.

If there is an API that relies on the data from the state machine, it could rely on the data of this application or use the state machine in another program.

10.5. Queue

The queueing system is a very important part of the overall working of the prototype too, it is the central system where all information is logged and that lends itself to be used as an inter-process communication (IPC) mechanism. For example, producers write their status information to a specific topic. This information is used by the consumer to detect the RRCs that are alive and the ones that publish the information. Then the consumer can look at the 'Consumer control queue' to see if all the information that is published by a specific RRC is also processed by another queue. When a new RRC comes online it can register itself to the control queue of all RRCs, the consumers see this and a consumer can subscribe to the topics, that the specific RRC has created, and process all the information. When a new RRC is deployed, a consumer does not need to be configured to subscribe to these topics, but automatically detects it and retrieves the information from the topics.

The following image illustrates the queue mechanism:



The auto-detection and subscription mechanism is not implemented in the prototype, but the queue system and prototype will be constructed to facilitate such a need for the future.

The queues of the RRC are divided in multiple partitions, this is for the sake of scalability. If there is only one partition per RRC the system does not provide us with the scalability that is needed in the RIS project. All these different partitions are stored in, for example the RRC01 control queue, so that the consumers know how many partitions there are and the sequence ID that is stored in a specific partition.

10.6. Developer planning

This chapter will describe the planning of the development stage.

The planning will be divided in several stages as outlined in the following table:

Stages	Description	Estimated time
Stage 1 **Preparation**	Virtual Machines: Virtual machines need to be created to use it as a development environment. The applications will be developed on these virtual machines but are also tested. Involves: Installing Virtual Machines, Zookeeper/Kafka, HBase, ExaBGP, libbgpdump.	1 week
Stage 2 **ExaBGP and Kafka**	Change ExaBGP code to meet all the “must” requirements. Put the messages in the queue system. Messages must be saved locally if the connection is lost with Zookeeper/Kafka. Add keepalive messages to the output/BGP metadata.	2 weeks
Stage 3 **State formatter**	Create state machine. Create code to create MRT-formatted files.	3 weeks
Stage 4 **Insert data into HBase**	Insert data straight into HBase.	2 weeks

10.7. What is necessary

Multiple virtual machines are necessary to develop and test the code. In the beginning of the development stage the virtual machines will run on a laptop, but it may be useful if some parts of the project are installed on a real server. When a real server is used and the application is tested, some requirements could be set on the required specifications of a server that must run a RRC.

It must also be possible to connect a test RRC with a test peer. This could be done to determine the scalability of the application. If the HBase consumer is ready for use, it will be tested to see if the HBase consumer can insert data in a test setup.

10.8. Note on prototype

The prototype is built to see if it is possible at all to meet all the requirements. It is possible that the outcome of the prototype is that this is not the perfect alternative as we may have thought. If this is the case, it should be documented why it is not possible to create an RRC using ExaBGP and what the recommendations are for the next project.

11.Resources

This chapter lists all the resources that have been used during this project.

Publication of an institute:

- *A Guide to the Business Analysis Body of Knowledge,* International Institute of Business Analysis, Whitby, ON (Canada), 2009

Websites:

- http://bird.network.cz/
- http://www.rabbitmq.com/
- http://hornetq.jboss.org/
- http://zeromq.org/
- http://kafka.apache.org/
- https://github.com/Exa-Networks/exabgp
- http://www.openbgpd.org/
- https://github.com/mor1/pyrt/
- http://wwww.opensourcerouting.org/
- http://www.xorp.org/
- http://www.vyatta.org/
- http://osrg.github.io/ryu/
- http://bgpmon.netsec.colostate.edu/

Specific page of websites:
- http://www.circleid.com/posts/ietf_chairs_statement_on_security_privacy_and_widespread_internet_monitorin/ (IETF chairs statement on security)
- https://github.com/Exa-Networks/exabgp/wiki/RFC-Information (RFCs supported by ExaBGP)
- http://www.youtube.com/watch?v=8Rqh4p4yzyQ&list=PLO8DR5ZGla8g24mzEdAKZ83byQuA6RIci&index=8 (BGPmon presentation)
- http://bgpmon.netsec.colostate.edu/download/publications/catch09.pdf (BGPmon article)
- http://bgpmon.netsec.colostate.edu/index.php/join-the-peering/peering-faq (BGPmon peering FAQ)
- https://github.com/opensourcerouting/quagga (Github page of OSR Quagga)
- https://www.nanog.org/meetings/nanog48/presentations/Monday/Filip_BIRD_final_N48.pdf (Presentation about BIRD and speed comparison)
- http://bird.network.cz/?get_doc&f=bird-3.html#ss3.2 (BIRD configuration options)
- http://opensourcerouting.org/about-us/ (OSR Quagga about-us page)
- http://www.openbsd.org/cgi-bin/man.cgi?query=bgpd.conf (OpenBSD configuration options for bgp.conf)
- http://www.openbsd.org/cgi-bin/man.cgi?query=bgpd (OpenBSD bgpd options)
- http://read.seas.harvard.edu/~kohler/pubs/xorp-hotnets02.pdf (XORP paper)
- https://github.com/wmiltenburg/xorp_documentation/blob/master/bgp/bgp.pdf (XORP BGP documentation)
- http://mailman.icsi.berkeley.edu/pipermail/xorp-hackers/2009-November/002229.html (XORP message about RIB)
- http://www.vyattawiki.net/wiki/Quagga (Page describing the change from XORP to Quagga)
- http://newsroom.brocade.com/press-releases/brocade-acquires-vyatta-a-pioneer-and-leader-in-s-nasdaq-brcd-0949599#.U5Wj5pSSyf9 (Information about Brocade acquiring Vyattta)
- http://www.brocade.com/products/all/network-functions-virtualization/product-details/5400-vrouter/index.page (Vyatta page)
- https://github.com/mor1/pyrt/blob/master/bgp.py (The bgp.py file that shows how it creates the MRT objects)
- https://github.com/mor1/pyrt/blob/master/README (README file of PyRT)
- http://osrg.github.io/ryu/slides/ONS2013-april-ryu-intro.pdf (Ryu introduction presentation)
- https://incubator.apache.org/incubation/Process_Description.html (Incubation process description of Apache)
- http://www.quora.com/RabbitMQ/RabbitMQ-vs-Kafka-which-one-for-durable-messaging-with-good-query-features (Information about RabbitMQ and Kafka)
- http://www.slideshare.net/charmalloc/apache-kafka (Information about Kafka performance)
- https://community.jboss.org/wiki/HornetQFeatures (Feature list of HornetQ)
- http://hornetq.blogspot.nl/2009/08/persistence-on-hornetq.html (Presistence in HortnetQ)

RFCs/drafts:
- https://tools.ietf.org/html/rfc6396 (MRT RFC)
- http://tools.ietf.org/html/draft-ietf-grow-bmp-07 (BMP, draft)
- http://www.ietf.org/rfc/rfc4271.txt (BGP RFC)

12. Appendix

This chapter is used to store all appendixes.

12.1. MoSCoW

MoSCoW, also known as MoSCoW prioritisation or MoSCoW analysis, is used to reach an understanding about the importance that all stakeholders have on a requirement.

The MoSCoW contains the following categories according to *A Guide to the Business Analysis Body of Knowledge*[54]:

Must: Describes a requirement that must be satisfied in the final solution for the solution to be considered a success.

Should: Represents a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary.

Could: Describes a requirement which is considered desirable but not necessary. This will be included if time and resources permit.

Won't: Represents a requirement that stakeholders have agreed will not be implemented in a given release, but may be considered for the future.

Note: Sometimes the word "Would" is substituted for "Won't."

54 A Guide to the Business Analysis Body of Knowledge, *International Institute of Business Analysis*, Whitby, ON (Canada), (2009, page 102)