Parallelization of BGP for route server functionality

A protocol and implementation study

UNIVERSITY OF AMSTERDAM

MSc Research Project
System and Network Engineering

4th August, 2017

Jenda Brands
Jenda.Brands@os3.nl

Patrick de Niet
Patrick.deNiet@os3.nl

Abstract

The Internet is growing and due to de-aggregation, many more smaller networks are being announced. This increase of routes, together with the sequential nature of the BGP best path calculation of routes, poses an issue for route servers operated by Internet Exchanges because of significantly increasing convergence times. If the calculation of routes could be done in parallel, route servers would benefit from the multi-core capabilities of processors. In this paper multiple solutions are proposed on the protocol design as well as a practical implementation to decrease convergence times and overcome the problem at hand.

Contents

1 Introduction

The Internet is growing in number of people connected to the Internet[1], as well as the number of prefixes that are being announced worldwide [2]. This increase is not only due to an increase of IP addresses in use, but also has to do with the de-aggregation of prefixes within the IPv4 address hierarchy. With de-aggregation of prefixes is meant that larger prefixes (for example /22 networks) are announced as multiple smaller networks (for example four /24 networks). This can have a great impact on the global routing table as this table will heavily expand since less summarizing can be done as a consequence.

The main routing protocol in the Internet is the *Border Gateway Protocol (BGP)*. BGP was introduced in 1989 and back then, its architects had not foreseen a number of shortcomings within the protocol. Many of these shortcomings were related to the increasing size of the Internet. Significant scaling additions have been done since, including the introduction of *Soft Reconfiguration (SR)* and *Route Refresh*. SR has become obsolete due to the fact that the underlying deficiencies have now been addressed within BGP itself, whereas Route Refresh is still actively used to request re-advertisements from a peer without resetting the entire connection.

Even though many shortcomings have been addressed, at this point in time there is still a mostly unanswered shortcoming with regards to the sequential processing of routes. When a peer announces new routes, BGP needs to calculate the best path to a certain prefix according to a *best path selection* algorithm. This calculation is nowadays still done in a sequential way, generally resulting in the inability to benefit from multi-core processors. This can cause additional instability if it renders the daemon unable to reply to *KEEPALIVE messages*.

BGP is used exhaustively in *Internet exchanges points (IXPs)*, where interconnections between peers are accommodated. These IXPs enable BGP speakers to setup BGP peering relations on a peering LAN. Peering at these IXPs is often done by means of a *route server*. Such a route server functions as a super-peer in an eBGP environment. This means the route server functions almost like a route reflector, with one of the core differences being that the route server works in eBGP and a route reflector in iBGP. Besides that, a route server commonly is not a router but an actual server. Customers of the IX can set up peering to the route server and thereby announce their routes to all other peers on the peering LAN. This eliminates the need to peer with all the other IX customers directly, as the routes are already being exchanged through the route server.

Route servers are a typical example where the shortcoming in sequential processing is observed. When a large amount of BGP speakers are sending UPDATE messages to a route server, it has to calculate the best paths from all these UPDATES, in a sequential

[1] http://www.internetworldstats.com/stats.htm
[2] http://www.cidr-report.org/as2.0/

way. This leads to one core of the server running at a 100% utilization, whereas other cores remain idle. This lack of multi-core support increases convergence times for the whole environment which is expected to worsen as the number of routes and peers grow larger.

1.1 Problem statement

This research focuses on improving this processing of routes and finding ways to improve (shorten) convergence times. As the bottleneck appears to be in the sequential processing of UPDATES, possibilities will be evaluated to multi-thread the BGP daemon. This research aims to improve both the protocol as described in RFC 4271 as well as find solutions on implementations themselves.

1.1.1 Research questions

Considering the problem described above, the main research question for this project is:

> *What improvements can be made to the Border Gateway Protocol (BGP) or its implementations to resolve current performance bottlenecks when processing updates?*

In order to formulate an answer to this question, the following sub-questions have been identified:

- What is the root cause of the perceived increasing convergence time?
- What past work has been done to solve this specific issue?
- What optimizations can be done to resolve this issue?

1.2 Outline

The rest of this document is set up as follows. First, in section 2, background information is given on BGP, as a protocol as well as the use of route servers and the decision process in BGP. Section 3 discusses the methodology used during the research. The testbed that was set up is also explained there. Then section 4 discusses the results of the tests that were conducted and several solutions are proposed there. In section 6 the research is discussed, after which section 7 concludes the paper including future work.

2 Background information

This chapter dives into BGP in more detail, aiming to provide the necessary background information needed to fully understand the rest of this paper. As BGP could cover a paper of its own, only the aspects relevant for this paper are explained in detail.

2.1 A Border Gateway Protocol

When it became apparent that the *Exterior Gateway Protocol (EGP)* first defined in RFC 904[1] was ill-equipped to scale with the expanding Internet, the *Border Gateway Protocol (BGP)* formalized in RFC 1105[2] which addresses some of these issues, has since become the de-facto standard for exchanging network reachability information over the Internet. Over the years, even this protocol has undergone further development, often as a result of practical shortcomings. This has resulted in the current standard *A Border Gateway Protocol 4* (RFC 4271[3]) more commonly known as BGPv4 or BGP-4.

The protocol is used by many organizations of all sizes. BGP allows one to control the flow of outgoing and possibly incoming traffic. In most cases, this is used to achieve multi-homing. This means traffic has two or more ways to enter or exit a given network. Although other use-cases exist, including ones outside the domain of Internet routing, the problem defined pertains to the original use case of BGP.

2.2 BGP protocol architecture

BGP is fundamentally built on the notion of *Autonomous Systems (ASes)* which are administrative areas where reachability information can traverse through. This is also where BGP inherits its loop-free properties from, by not importing routes that contain themselves in an *AS-path*. In order to exchange this kind of information, other mechanisms are in place which will be discussed in the sections below.

2.2.1 Peering

BGP runs on the *Transmission Control Protocol (TCP)* which handles complications such as retransmissions and sequencing. Using this technology, it allows BGP to set up peerings between instances such that data containing network information can be exchanged between them. To establish these peerings, BGP uses a *Finite State Model (FSM)*. Because this research does not revolve around the FSM theory, there is no need to delve deeper into this concept. It is only relevant to known that BGP has five potential states it can have during the formation of a peering, with the *BGP_Established* state being the desired state.

2.2.2 Messaging

When a peering is established, peers can exchange messages. A total of five message types exist that can be exchanged. Most relevant to this research are the *UPDATE* and *KEEPALIVE* messages.

2.2.2.1 UPDATE

The UPDATE message contains the actual network information that BGP wants to exchange. It contains prefix information formally named *Network Layer Reachability Information (NLRI)*, which is composed of *prefixes* (e.g. 128.66.0.0) and *lengths* (e.g. /16) in *Classless Inter-Domain Routing (CIDR)* format[4]. For each NLRI, attributes are added such as *AS_PATH* and *NEXT_HOP* for the peer to base decisions on. This type of message is used to both announce and withdraw information.

A notable composition of an UPDATE message, is one that does not contain any NLRI. This is specified as an *End-of-RIB* marker which indicates that a BGP speaker has communicated its entire routing table, known as a *Routing Information Base (RIB)*, through (multiple) UPDATE messages. This feature was introduced in the *Graceful Restart Mechanism for BGP* described in RFC 4724[5], and is not part of the initial BGPv4 standard.

2.2.2.2 KEEPALIVE

To validate the reachability of a peer, the KEEPALIVE messages within BGP are used rather than any TCP-based mechanism. KEEPALIVES are sent every one third of the *Hold Time* interval which is communicated in the OPEN message when peering is set up. If a peer does not receive a KEEPALIVE, UPDATE or NOTIFICATION message within the hold time, the peering will be closed.

Also, the KEEPALIVE message indicates an implicit End-of-RIB and can be used as such in some implementations[6].

2.2.3 Decision Process

Once peering is established and maintained, and UPDATES have been sent and received, BGP initiated the *Decision Process* to utilize the acquired information. This process consists of three phases. The RFC describes the structure of tables used to hold this information and also the rules as to the progression of phases. The structure itself is only conceptual and the actual implementation is left to the programmer. Figure 1 shows an abstract representation of the tables and processes defined in the RFC.

The *Adj-RIB-In* tables store the routing information sent by a remote peer where it awaits processing. For each peer, there is a separate table.

The *Loc-RIB* holds a list of the best path routes. This list is generally a subset of paths learned in all Adj-RIBs-In. From here the data is ordinarily imported into the *Forward Information Base (FIB)* where a router can use it to make routing decisions. This is not the case when it pertains to a *Route Server* implementation. The reason being that a route server does not have the function to route traffic itself. A more in-depth description regarding route servers will follow later in chapter 2.4.

The *Adj-RIB-Out* tables are similar to the Adj-RIBs-In but serve a reversed function of sorts. Each Adj-RIB-Out will contain information that is to be disseminated to its peer.

To emphasize, this is how the tables are described in the RFC. However, the RFC explicitly allows for tables to be combined within an implementation as to achieve better optimization of hardware.

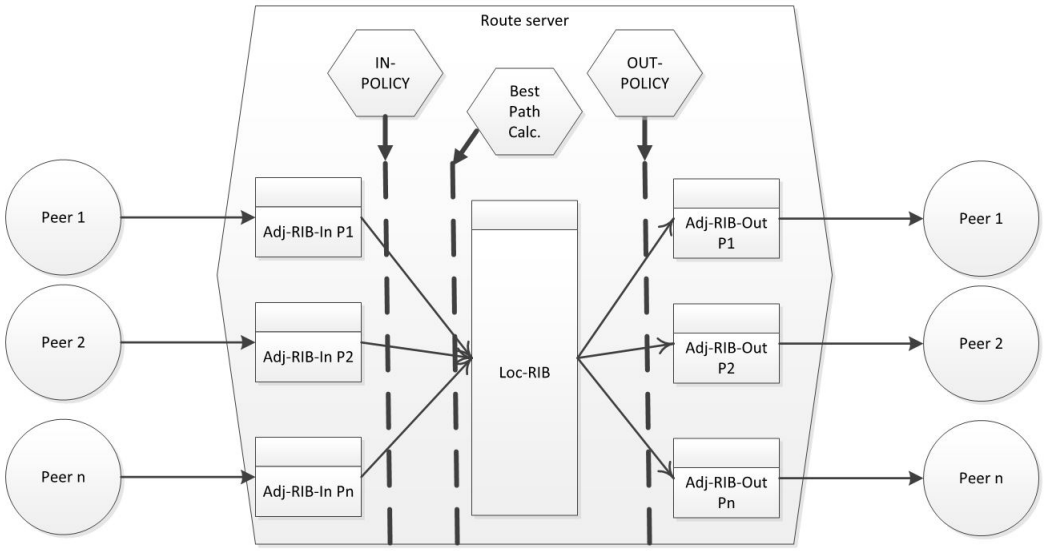

Figure 1: Mechanism described in RFC 4271 section 9.1

These tables are closely related to the three phases of BGP mentioned earlier, as the RFC specifies the rules for locking and distribution of data throughout the system as phase progression occurs.

2.2.3.1 Phase 1: Calculation of Degree of Preference

Once an UPDATE message has been received from a certain peer, phase 1 of the BGP protocol should commence. In practice, the phase is initiated when receiving the End-of-RIB mark as to ensure the remote peer had sent all its routing information[7][8]. First, any prefixes or attributes received that are marked as undesirable are dropped or altered by means of the *In-policy* [9].

Then, the corresponding Adj-RIB-In is locked as to avoid any inconsistencies during this phase. After this, the *degree of preference* is calculated for all entries within the Adj-RIB-In. When this is done, the lock is released and BGP progresses to phase 2.

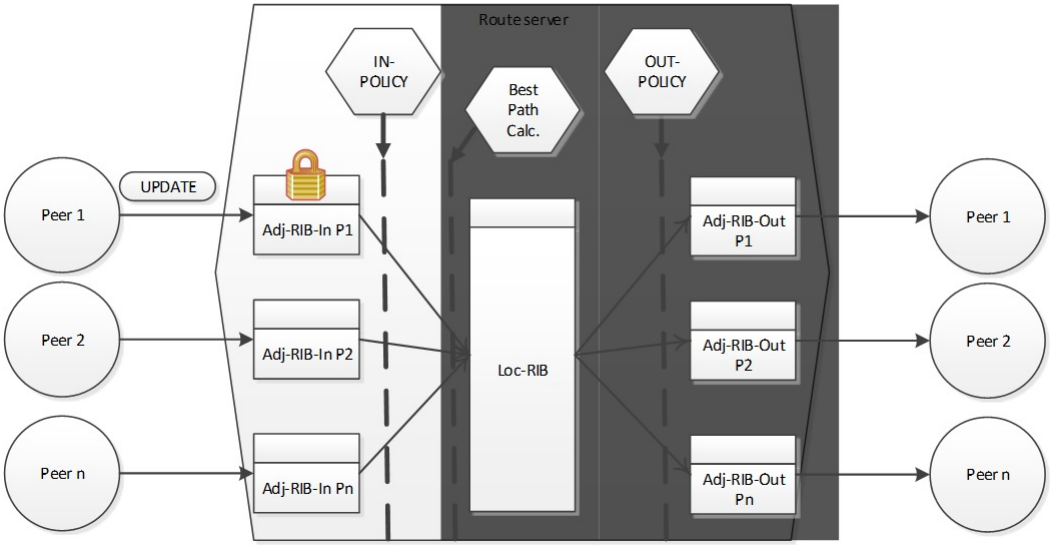

Figure 2: Graphical representation of phase 1

2.2.3.2 Phase 2: Route Selection

As phase 2 starts, all Adj-RIBs-In are locked. What follows is a *best-path calculation* that is run over the combined routing information of all Adj-RIBs-In. If only one peer announced a certain prefix, it is by definition the best (and only) path for that prefix. In many cases however, a path to a certain prefix may be announced by multiple peers. To determine the best path, the following criteria are described by RFC 4271:

- the highest degree of preference of any route to the same set of destinations, or
- is the only route to that destination, or
- is selected as a result of the phase two tie breaking rules specified in RFC 4271 Section 9.1.2.2.

By the end of this phase, each prefix will have a single route entry in the Loc-RIB. All processed routes will be cleared from the Adj-RIBs-In, excluding unresolvable routes and the locks are removed. Unresolvable routes refer to routes where the next-hop address can currently not be reached. It may be so that the *route refresh* feature is implemented, which also contains the UPDATE data received from peers. This data is not cleared.

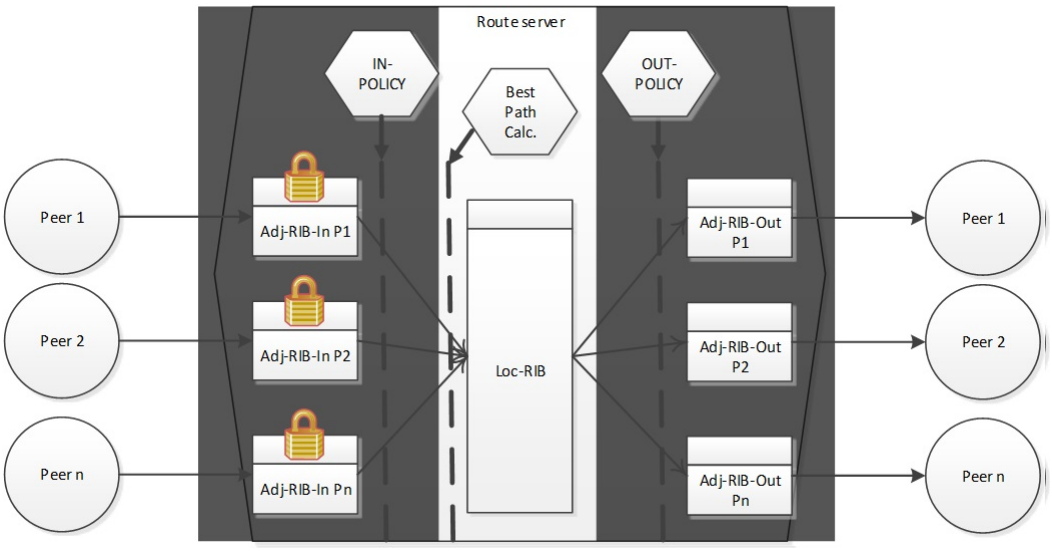

Figure 3: Graphical representation of phase two

2.2.3.3 Phase 3: Route Dissemination

When phase two has no more work to do, phase 3 kicks off. It takes the list of best paths from the Loc-RIB and may alter them in accordance with any *Out-Policy* applied. Afterwards, the routing information is disseminated to its peers.

2.2.3.4 Phase rules

The RFC specifies when a phase starts and ends. Additionally, there are rules as to what phases can run concurrently. These rules are summarized below:

- Phase 1 is invoked whenever receiving an UPDATE message.
- Phase 2 is invoked on completion of Phase 1.
- Phase 3 is invoked on completion of Phase 2 (more triggers exist).
- Phase 2 is blocked from running while Phase 3 is in process.
- Phase 3 is blocked from running while Phase 2 is in process.

2.3 BGP implementation architecture

While the RFCs describe the way BGP should behave, they do not result in a functioning application. The translation of the RFC into actual functioning code is referred to as the *implementation* of an RFC or standard. Although the RFCs strictly define some BGP mechanisms, others do not and leave it to the developer. One example of this

phenomenon is the way the conceptual RIBs are implemented, where it is left upon the developer to implement multiple or a single table in the application. This is why there is a large variety in BGP implementations which differ not only in programming language, but also in their internal workings. In practice, this results in many open- and closed source BGP software implementations available, each with different perks and quirks.

A commonality in these implementations is however, that they are all essentially single-threaded[10]. While applications such as OpenBGPd[11] and GoBGP accomplish some multi-threading to maintain peering and receive updates, work such as the best path calculation is still done within a single thread. For GoBGP this has been confirmed to us by the development team in response to an inquiry.

2.4 Route server

As mentioned, BGP speakers need to set up a peering in order to exchange routing information. If speaker A would want to converse with speaker B and C, it would need to construct a separate peering relationship with both these entities. Consider a network where seven BGP speakers exist and where each speaker wants to receive all available network information. In order to achieve this, a full-mesh topology has to be created as portrayed in figure 4.

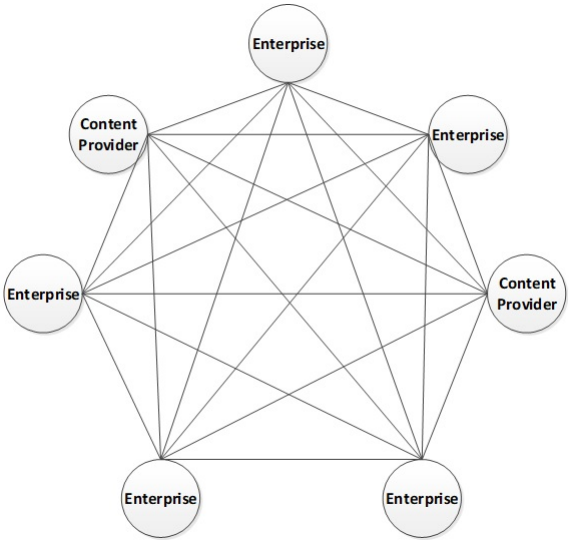

Figure 4: Full-mesh connection of seven BGP speakers

This results in 21 peering relations and each node maintaining 6 peering sessions. While these numbers are not necessarily a problem yet, it is evident that a higher number of participating nodes will stress the scalability of this architecture. Consider that in an environment such as the AMS-IX, which has about 1350 peers, it would require

every BGP speaker to maintain $1350 - 1$ peering sessions. The first challenge would be managing the list of peers from an administrative point of view. Secondly, it would take a considerable amount of resources in maintaining the peerings and processing all the network information.

In order to avoid this extensive architecture, one of the solutions that can be deployed is a *route server*. The route server functions as a super-peer, forming a peering relationship with every BGP speaker as depicted in figure 5. This is generally done in pairs as to improve availability. However, the use of a route server does not prohibit classical peerings from being formed.

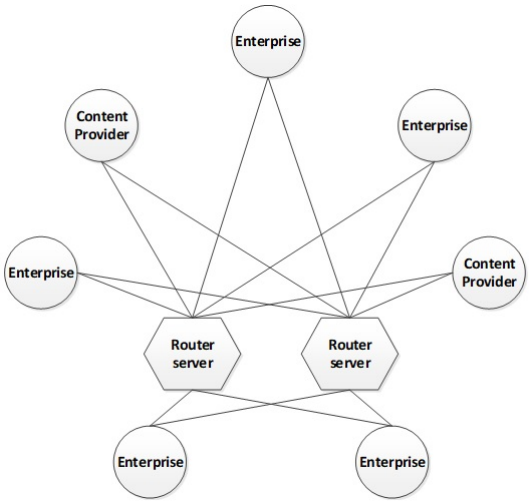

Figure 5: Route server connection of seven BGP speakers

This setup reduces the stress put on BGP speakers. Not only will the number of peering relations be equivalent to the amount of route servers, the amount of network information received is also reduced. This is due to a particularly pleasant side-effect that a peer will no longer receive multiple routes to the same prefix from other speakers. Instead, the route server had already accumulated all routes from all peers, selected the best path and informed its peers only of that particular route for a given prefix.

It is important to observe that while this greatly reduces the stress put on BGP speakers, the route servers themselves are not so fortunate. These servers still maintain a number of peering relationships equivalent to the number of BGP speakers. Additionally, they are confronted with potentially receiving multiple paths to a single prefix resulting in more calculation work when determining the best path. A trend that will only continue as the Internet grows in prefixes due to IPv4 de-aggregation and IPv6 traction increasing. Additionally, more organization join these Internet exchanges increasing the total amount of peers in such a network. As a result, the burdens placed on these single-

threaded route servers used today are already inducing increased convergence times.

2.5 Related work

Experiments have been conducted to modularize [12], distribute [13][14] or parallelize [15][16][17] BGP. Nevertheless, these innovations have not solved the problem at hand. The main reason being either test results showed unfavorable results or it would not address the focus of this project.

2.5.1 BGP protocol design

In the paper of Lina Ding and others[17] a method is proposed to split the BGP process in five self-specified phases. These phases are then running in a pipeline. This means that at any given time, all phases could be doing work simultaneously for independent tasks. The aim here was to have all five phases doing work continuously, rather than waiting for other phases to finish first. Some performance was gained, however, the core issue, namely that processing times of certain phases was long, has not been resolved. For example, while multiple phases could be active simultaneously, no two of the same phases would run at any given time.

Looking at the paper of Xuezhi Jiang and others[18], they propose a way to improve route processing by parallelization. However, in their paper they focus only on distributed control planes with multiple control elements. In this paper the aim is to have a broader focus, and actually focus on the protocol design itself rather than having dependencies on specific techniques.

2.5.2 BGP Implementations

During this project, actual BGP implementations were also looked into. This will generally be open source daemons as then, there will be access to the code. There have been other initiatives before such as the 2009 Euro-IX project [19] to fork Quagga [20] and aim to improve, amongst other things, the BGP CPU limitations in the implementation. Likewise, some closed source software vendors have claimed to have worked on and implemented multi-threading [21][22]. This is almost certainly referring to only a part of the BGP process which is also seen in some open implementations.

2.6 Problem considerations

As mentioned, administrators of large scale BGP implementations are experiencing relatively long convergence times in their environments. The first step to proceed with this project is to confirm that the problem can indeed be reproduced by creating a testbed with a BGP route server and multiple BGP peers. There is a variety of reasons why this

may not be feasible. For instance, it could be that the hardware available is not capable of generating enough UPDATE messages to reach a point where this behavior can be observed. Alternatively, it may prove troublesome to identify what metrics (prefix size, communities or other attributes) are involved in this process.

Once the issue has been reproduced, an analysis can be done by repeating the tests and tweaking the metrics. In order to be able to conclude a specific section of the protocol is particularly at fault, it is necessary to make a distinction between functions within the protocol. It is practical to make this distinction based on the phases described in the background section. This does not only result in a comprehensible categorization, but is also based on well described BGP steps by means of the RFC. An additional factor to the three BGP phases is the time it takes for messages to travel over the network.

Consider the total time required to disseminate new routing information from another peer is 4.

$$T_{net}(1) + T_{phase1}(1) + T_{phase2}(1) + T_{phase3}(1) = T_{total}(4)$$

While in this example it is assumed that the cost of each step is equal, this is in fact very unlikely and is merely to illustrate the thought process. Ideally, the testbed will provide the ability to add, remove or adjust these metrics and to capture the effect on the total time. It is important to note that, while time is being measured, this in itself is not the cause of the problem. Instead, the time is a result of a fully utilized core unable to keep up with the work that has to be done. In essence, either the work has to be reduced or the capacity increased.

3 Methodology

During this research multiple tests have been conducted. This chapter explains in detail how the tests were conducted including definitions and assumptions made beforehand. Also the tools used to gather measurements are explained here.

3.1 Presuppositions

Before describing how the tests were conducted, it is important to state the definitions used during these tests. These definitions define the starting point and clarify what has effectively been measured. Besides these definitions, some assumptions had to be made in order to be able to conduct the experiments and get the needed results.

3.1.1 Definitions

One of the definitions that need clarification is the convergence time. This is the time it takes for the whole setup to converge. This means, the route server as well as all peers have the same information (shared amongst them) and the route server is done sending and receiving updates. To clarify, figure 6 shows the CPU usage on the route server during one of the conducted tests. The graph starts at the time the first UPDATE message is received, this is marked as the "START". Before that point the CPU was idle. In the graph one can see that the CPU usage spikes to 100% and stays around that number for a longer time. During the spike, the route server is busy processing UPDATE messages from its peers. When the whole environment is converged, the CPU usage drops to idle again, this is marked as the "END" in the graph.

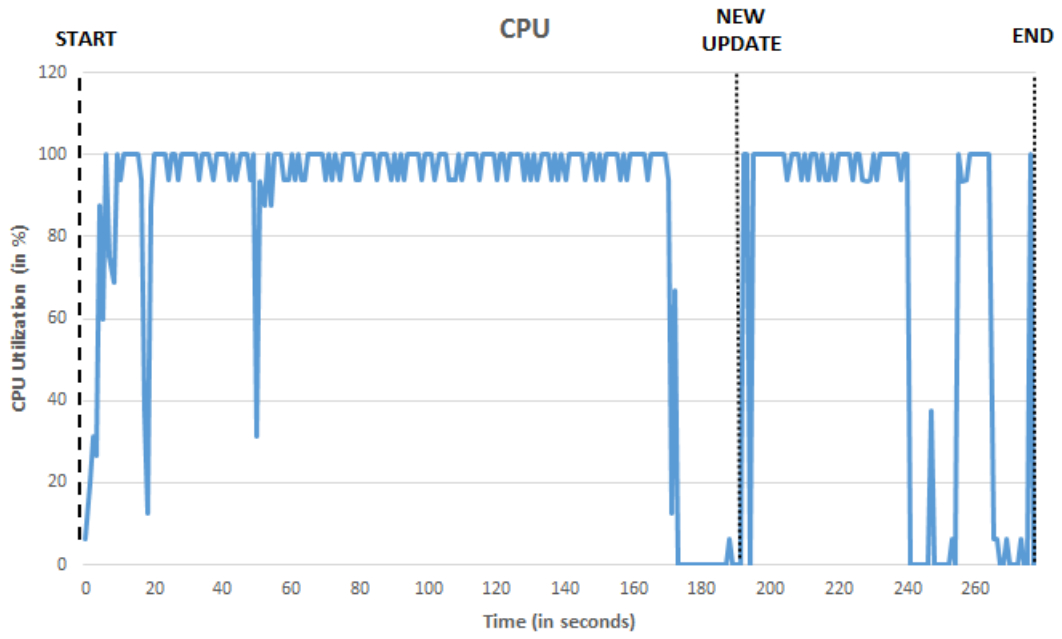

Figure 6: CPU utilization during testing

In some cases a dip occurs in CPU usage of the route server. This usually happens when a number of peers sent UPDATE messages, but some peers have not sent the UPDATE yet. As soon as the remaining peers start sending their UPDATE messages, the route server spikes to 100% CPU usage again. In the graph this happens at the "NEW UPDATE" marker.

3.1.2 Assumptions

BGP has quite some optional configuration choices, for example, the possibility to add policies. In the tests conducted, there were deliberately no policies configured, except for the forwarding of all routes. This was done because the exact problem should be pinpointed, which should occur even in the most basic BGP environments. Adding any extra complexity might have an impact on the route server performance so the decision has been made to keep the environment as clean as possible. This is also why no additional BGP attributes have been used in the experiments.

3.2 Testing framework

This section describes the testing framework used for describing and designing the tests that were conducted. First the different variables that were measured are explained, after which the testbed that has been set up is explained in detail.

3.2.1 Variables

As mentioned before, several variables were measured. Besides CPU utilization, also memory and bandwidth usage were measured. The reason being that bandwidth clearly shows a spike when (larger) UPDATE messages are being sent and received. When the environment is converging, bandwidth also shows a clear spike after which it drops down to a more idle state in which only KEEPALIVES are being sent. Lastly, also the logging of BIRD was looked at, which shows exactly what happens in the BGP daemon at a given time. From this logging it was possible to precisely pinpoint when the first until the last UPDATE was received.
During the tests the overall variable that is being measured, is the convergence time. In other relatively small tests, as compared to real world setups, convergence times of fifteen minutes have been demonstrated[23]. As convergence time also depends on how quick peers re-establish connections to the route server, the measurements are not supposed to be precise to the second. This means it is very likely that differences in convergence time, using the exact same configuration, can be off by multiple seconds.

3.2.2 Testbed

The testbed on which the experiments were conducted, consisted of one route server and eight servers where BGP peers were spawned in docker containers. Both are explained below in more detail.

3.2.2.1 Route server

On the route server BIRD was installed as primary BGP daemon, since this is the most commonly used route server implementation nowadays[24]. BIRD was configured as minimalistic as possible based on a template configuration file provided by BIRD [3]. Most of the configuration was kept intact, apart from the peer specific configuration and the route limit. Also, some additional logging parameters were added, which allowed for a more verbose debug logging. Several levels of debugging had been tested and from this it was concluded that the level used did not have any significant impact on the performance. More verbose levels were tested as well, which had tremendous impact on performance as every single route that was received was also being sent to syslog, leading to 50% CPU utilization being dedicated to syslog alone. The debug levels used in the tests were the following:

```
debug protocols { states, interfaces, events, packets }
```

Debugging on states was enabled in order to get a log entry for protocol state changes. Interfaces was enabled to observe any changes occurring on interfaces, that may affect BGP, like going up or down. Events was used to gather log entries from the internal

[3] https://gitlab.labs.nic.cz/labs/bird/wikis/Simple_route_server

protocol and packets was used to ensure a log line gets printed for each packet sent and received.

3.2.2.2 Peers

Eight servers were available to this project for use in the testbed. Docker was installed and configured on these machines, in order to run multiple BGP daemons efficiently. The BGP implementation used on the peer side is the open source ExaBGP implementation [4] which is primarily used for testing purposes. ExaBGP was used in the containers because it is flexible and allowed for a simple python script to run, in order to configure all routes that needed to be announced. As the goal is to measure the convergence time identifying a bottleneck on the route server, the BGP peer implementation is of little consequence as long as it does not pose a bottleneck itself. Preliminary testing indicated that increasing the amount of simultaneously active ExaBGP containers above one hundred, within a single server, would cause instability in creating and maintaining a peering relation with the route server.

3.3 Scenarios

To conduct the tests, a number of different scenarios were created. These scenarios differ in either the composition of peers or the composition of prefixes. For prefixes, only IPv4 addresses have been used since it is still the predominant network protocol. This is due to the work that would be required to set up the IPv6 environment, while it is not expected to react differently than IPv4.

All tests were conducted in the same manner, taking the same steps every time. These steps were the following:

1. Start BIRD daemon on route server
2. Start ExaBGP containers
3. Verify all peers are established
4. Simulate link flap (set link on route server down)
5. Re-enable link on route server
6. Start all measurement tools
7. Monitor first UPDATE message, this is the START marker
8. Verify all peers are established
9. Monitor last UPDATE message, this is the END marker

[4] https://github.com/Exa-Networks/exabgp

Each test was conducted three times to ensure the results are representative. After the three test runs, the BIRD BGP daemon was restarted to ensure previous tests could not influence other tests. This was done as a safeguard that BIRD was not left processing in the background, or using any additional memory from the previous test.

3.3.1 Peers

Three different scenarios were created with the differentiator being the number of peers, in order to measure when the problem would occur and become a bottleneck. The first tests were done with the least number of peers possible. Thereafter tests were done with as many peers as possible within the testbed. At last, a real world scenario was created to get a more realistic scenario instead of a 'clean' lab environment.

3.3.1.1 Three to one

The first scenario created included three peers and one route server. The decision was made to test with three peers instead of one. This was done because with only one peer, no best path calculation is done since there is only one path, which is then automatically the best path. If two peers had been used, the same issue would occur when bringing one BGP peer down, since if one peer would fail the other automatically becomes the best path. Ultimately with three peers, one peer could fail, leading to the route server still needing to calculate the best path between the two remaining peers, unless of course, the failing peer would not have been the best path for any prefixes.

3.3.1.2 Many to one

The second scenario involved as many peers as possible. In the testbed used for this research, it has been determined a number of 800 peers at maximum. This maximum number is a hardware limitation, so by adding more hardware it would have been possible to spawn even more peers.

3.3.1.3 Real world

The scenarios before are not typical examples one would find in a real world environment. All peers are initially announcing the same prefixes, which will not happen in real life. In order to find differences between the 'clean' lab environment and overlapping prefixes of different sizes, this real world example was created. While it is infeasible to approach the scale of the real world situation, it may be interesting to test the effects of overlapping prefixes with a realistic path to AS ratio. To emphasize, this setup is neither in scale nor attribute-use representative of existing BGP networks. We have merely used available data of real world implementations to help define the amount of prefixes for these tests. In order to differentiate different AS sizes in a coarse way, the categories defined in another BGP paper[25], have been adopted. In this paper, the following AS-size categories have been defined:

- Tier 1: AS customer cone size of 5000+
- Tier 2 Large: AS customer cone of 1000-4999
- Tier 2 Medium: AS customer cone of 100-999
- Tier 2 Small: AS customer cone of 10-99
- Tier 3: AS customer cone of 0-9

The customer cone seen in these categorizations refers to data that is publicly available from the *Center for Applied Internet Data Analysis (CAIDA)* website. Table 1 below shows an up-to-date count of all public ASes at time of writing, categorized by tier as mentioned above.

Category	**Amount of prefixes**	**AS customer cone**
T3	0-9	52945 (96.7%)
Small T2	10-99	1554 (2.8%)
Mid T2	100-999	233 (0.4%)
Large T2	1000-4999	30 (0.1%)
T1	5000+	10 (0.0%)
		54772 (100%)

Table 1: Basis for the real world scenario

Knowing that the testbed, consisting of eight servers, can run 601 BGP instances in a stable manner, the work has been divided as displayed in table 2. Note that the available resources have been divided roughly by the same ratios as seen in table 1.

Server name	**T3**	**T2s**	**T2m**	**T2l**	**T1**
Athens	62	3	1	1	0
Bern	82	4	1	0	0
Bordeaux	82	4	1	0	0
Brussels	82	4	0	0	0
Copenhagen	82	4	0	0	0
Dublin	87	4	0	0	0
Helsinki	92	4	0	0	0
Sheffield	0	0	0	0	1
8	**569 (94.7%)**	**27 (4.5%)**	**3 (0.5%)**	**1 (0.2%)**	**1 (0.2%)**

Table 2: Number (and types) of peers per server for the real world scenario

In selecting the exact numbers, some liberty has been taken to tip the scale by increasing the amount of nodes announcing larger spaces. For example, introducing a single tier 1 instance causes 0.2% of all instances to be tier 1, in reality there are only ten of these providers in the real world which is why there they account for less than

0.0%. Nevertheless a tier 1 node was included mainly due to the reason that it would otherwise be almost certain no significant convergence times would be seen, based on past measurements. In the end the main goal was to get a setup with somewhat realistic, but not necessarily exact, dimensions.

3.3.2 Prefixes

In order to get more insight in the impact of the number of prefixes, multiple numbers of prefixes have been tested, namely 100, 1.000 and 10.000. Between these numbers of prefixes a distinction was made between non-unique, unique and partially overlapping prefixes.

3.3.2.1 Non-unique

With non-unique prefixes is meant that all peers were announcing exactly the same prefixes. This was done because when all peers start announcing all the same prefixes, the route server would need to calculate the best path a lot of times, since all peers provide a path to the same prefix. The paths are thus equal to the number of peers.

3.3.2.2 Unique

To determine if any difference in convergence time exists when using all unique prefixes, the 'unique' scenario was composed. This consists of peers only announcing a set of unique prefixes. This means all announced routes will be best-path and the work put on the route server for best-path calculation is minimized. For this reason, it is expected that convergence time will be shorter than non-unique tests with an equivalent amount of peers and prefixes. By bypassing part of the phase 3 process, this scenario may help in determining where the CPU utilization is spent during convergence.

3.3.2.3 Partially overlapping

Also overlapping prefixes have been tested in the real world scenario. In that case a large tier 1 provider would announce a large network (for example /20), and a smaller tier 2 or tier 3 provider would announce multiple small networks (for example /24) of which a part were a subset of the large tier 1 /20 network. More specific routes are always preferred over less specific routes (set aside policies). When the tier 1 provider connects to the route server first and other providers connect later, the best path will be replaced by the most specific prefix announced. Only when multiple providers are announcing the same prefix and subnet mask (exact same NLRI) the best path calculation takes place.

3.4 Measurement tools

To measure CPU usage of a specific process, the `top` command was used with specific parameters to get the output for the BIRD process. The parameters used for `top` were

`-b`, to run in batch mode, `-n 1`, so only one iteration would be done. This output was piped through grep for the bird process (`grep bird$`). `ps` could also have been used with specific parameters, but the usage percentage that `ps` shows is the percentage of time spent running during the entire lifetime of a process[5]. Due to this difference `top` was the better choice. Memory usage was also measured with `top` so with one process execution both variables were measured at once. `top` ran every second during the tests.

In order to measure all traffic going to and from the route server, the bandwidth was measured with `ifstat`. By specifying the exact interface with `-i`, it was ensured only network traffic from that specific interface was measured. Since a dedicated network interface was used to connect to the peering LAN, no additional network traffic from the outside could influence the measurements.

[5] http://www.unix.com/man-page/all/1/ps/

4 Results

The results of the tests explained in chapter 3 are shown in this chapter. The results are ordered per scenario, after which observations are expressed.

4.1 Scenarios

The results of the three scenarios, three to one, many to one and the real world scenario, are shown in this section. Besides these scenarios some additional tests have been conducted that are explained in section 4.1.4 combined with its results.

4.1.1 Three to one

The first scenario involved the three to one tests. Table 3 shows the convergence time in seconds.

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	7	5	6
Run 2	6	8	6
Run 3	7	5	6
Average	6,67	6	6

Table 3: Convergence time in seconds with three peers

These results show that there is no significant difference in convergence time between the different number of prefixes. All nine tries took between five and eight seconds to converge. This shows that at least at this scale the problem does not occur. There is no notable difference between 100 prefixes and 10.000 prefixes.

4.1.2 Many to one

The many to one scenario was conducted with 10 peers and 100 until 800 peers, each time with a 100 peer interval. Tables of all these convergence times can be found in appendix A. Figure 7 shows the average convergence time plotted against the number of peers. Each peer was announcing 100 prefixes in this case. It clearly shows that the higher the number of peers, the longer it takes to fully converge with a maximum of 70 seconds.

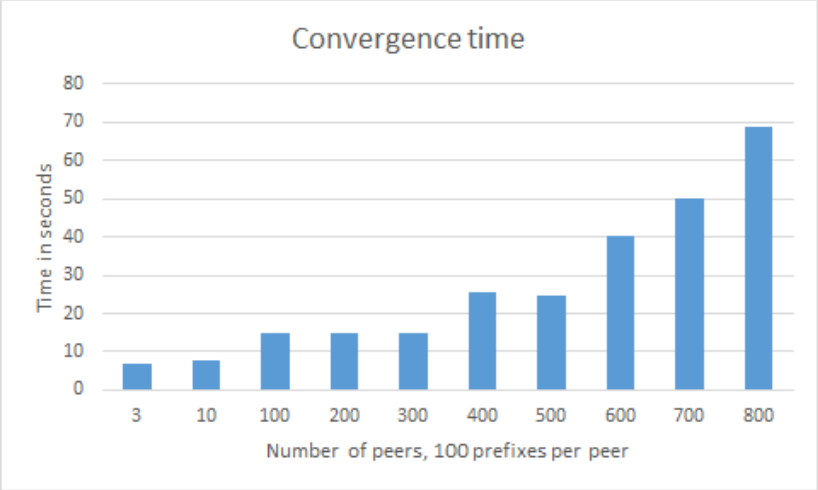

Figure 7: Convergence times with 100 prefixes per peer

Figure 8 is also showing the average convergence time plotted against the number of peers, now with each peer announcing 1.000 prefixes. The spike at 500 peers was caused by a hiccup in the system. Apart from the spike this behavior looks a lot like the previous, and maximum convergence time is around 70 seconds here as well. This shows there is no significant difference between 100 and 1.000 prefixes per peer.

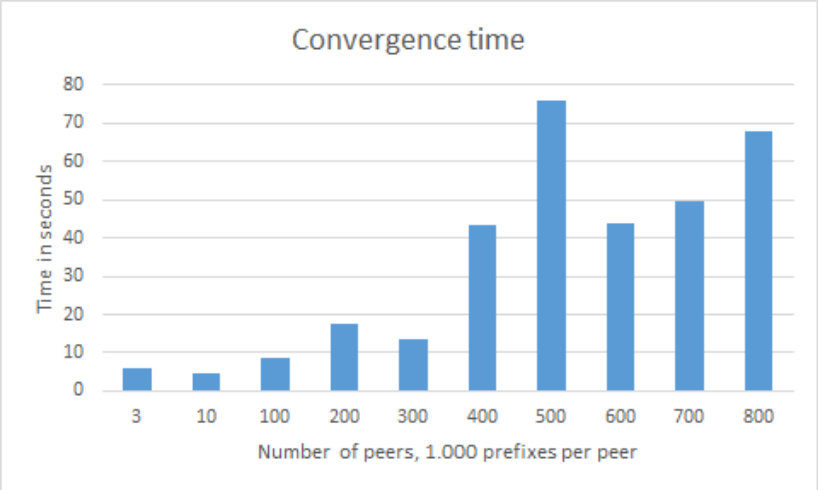

Figure 8: Convergence times with 1.000 prefixes per peer

Figure 9 shows the same graph as figures 7 and 8 but with each peer announcing 10.000 prefixes. The graph shows that from 500 to 600 peers there is a more steep increase in convergence time. The overall convergence time observed with this number

of prefixes resulting in 400 seconds is significantly higher than the 70 seconds observed before.

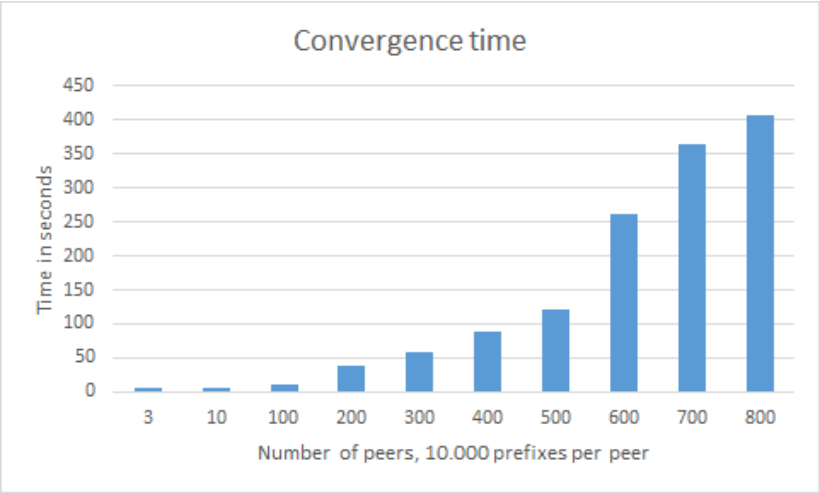

Figure 9: Convergence times with 10.000 prefixes per peer

When these graphs are plotted in the same figure, it shows in figure 10 that scaling up to 10.000 prefixes per peer has a major impact on convergence time compared to the lower amounts.

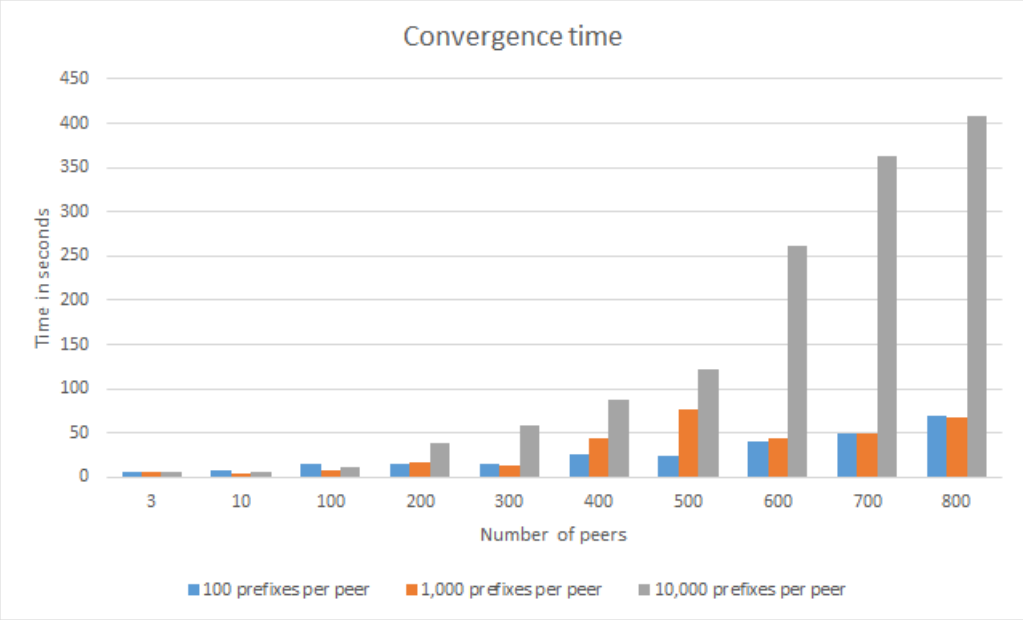

Figure 10: Convergence times combined

4.1.3 Real world

Testing the real world scenario led to behavior not seen before during previous tests. With all previous tests the End-of-RIB indicated the end of the convergence time. After receiving the last End-of-RIB the route server went idle and was just sending and receiving KEEPALIVE messages. However, when running the real world scenario, long after receiving the last End-of-RIB message the route server was sending UPDATE messages to its peers. Table 4 shows the times until the last End-of-RIB as well as the time until the last UPDATE message was sent. The latter indicates a converged state. As described in 3.3.1.3, the real world scenario involved 601 peers with a variety in the number of prefixes announced.

Real world	**Time to End-of-RIB**	**Time to last UPDATE**
Run 1	130	535
Run 2	139	512
Run 3	128	530
Average	132,33	525,67

Table 4: Convergence time in seconds during real world scenario

Comparing the real world scenario with the many-to-one scenario using 600 peers, it shows that the real world scenario overall took more time to converge.

Number of prefixes	**100**	**1000**	**10000**
Run 1	43	39	241
Run 2	38	38	265
Run 3	40	54	281
Average	40,33	43,67	262,33

Real world End-of-RIB	**Real world time to last UPDATE**
130	535
139	512
128	530
132,33	525,67

Table 5: Convergence times of 600 peers vs real world

4.1.4 Additional tests

As described in 3.3.2, the aim was to test multiple types of prefixes. Besides the real world tests, which were a mixture of unique and non-unique prefixes, all tests above were conducted using non-unique prefixes. When the tests with only unique prefixes were attempted, previously unseen problems would occur. The route server was running out of memory very quickly, starting to swap and becoming very slow and unresponsive. After some time either the BIRD daemon crashed, or KEEPALIVE messages were not replied anymore, leading to sessions timing out.
The additional use of memory is not a surprise as the Loc-RIB will grow way larger than when all prefixes are the exact same. With 800 peers announcing 10.000 of the same prefixes, the Loc-RIB on the route server only contains 10.000 routes, as it decided on

the best path for these prefixes. However, when 800 peers start sending 10.000 unique prefixes the Loc-RIB would need to store 8.000.000 routes, obviously using more memory than with 10.000 routes. Depending on the implementation, it could be that when routes are received in the Adj-RIB-In and they are being copied to the Loc-RIB, the entries are also copied in memory, thereby occupying double the memory for a short time. When a machine is low on memory (like the route server used here) it starts swapping and the process can get slow because it needs to read/write to/from the slower disk instead of the faster memory. This could explain why BIRD was not answering KEEPALIVE messages anymore, if the process would be waiting for blocking I/O.

4.1.4.1 Export

Another test that was conducted, was based on the hypothesis that the bottleneck could be in the transmission of UPDATE messages from the route server to peers. The main argument for this was that a route server with 700 peers, that receives one route, needs to send UPDATE messages to 699 peers with that new route.
A good way to test if this indeed poses a problem, is to configure BIRD not to export any routes. This was done by adding `export none` in the BIRD configuration. This tells BIRD to not export any routes to its peers. The tests done involved 800 peers and 10.000 prefixes per peer.
Figure 11 shows the difference in average convergence time. This figure shows the convergence time with export disabled being even higher than with export enabled.
Table 6 shows why that is the case. Out of three tests with export enabled, the third test had an exceptional convergence time of 322 seconds, which compared to the other two tests differs a lot. When discarding this third test, convergence time comes at an average of 451,5 seconds. This compared to the convergence time with export disabled (438,3 seconds) shows a slight increase in convergence time, however this difference in convergence time is too small to indicate the bottleneck would be in the exporting of routes.
Also during the real world example, where the route server was for a longer time sending UPDATE messages to its peers, CPU utilization dropped way before the route server stopped sending UPDATE messages, indicating the sending of messages is not a CPU intensive action.

Export	**Enabled**	**Disabled**
Run 1	446	425
Run 2	457	436
Run 3	322	454
Average	408,33	438,33

Table 6: Convergence times with and without export enabled

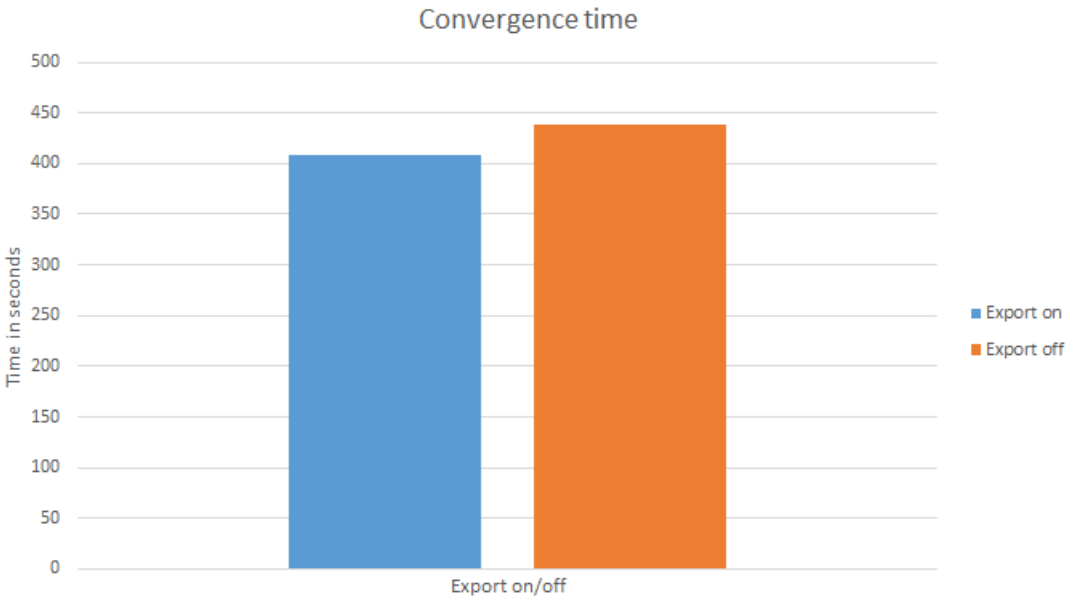

Figure 11: Convergence times with and without export

4.1.4.2 GoBGP

In order to verify that the increasing convergence times are also observed in other multi-threaded implementations, a last additional test was conducted. This last test covers a small part of the extensive BIRD tests with using the same scenarios. This test involved the GoBGP implementation. Due to time constraints it was not possible to conduct all tests done with BIRD however, the results do show an increase in convergence time just as the BIRD test results show.
During the GoBGP tests, only one set of prefixes was used, all peers were announcing the same 100 prefixes. The number of peers was increased from 50 to 100 and 150. To configure GoBGP as route server, a configuration sample from the official documentation [6] was modified to suit these tests. The GoBGP configuration can be found in appendix C.
Table 7 shows the results for the tests conducted. The convergence time was measured with 50, 100 and 150 peers all announcing 100 prefixes. Figure 12 shows that, just as during the other tests, the convergence times do increase significantly when adding more peers.

[6] https://github.com/osrg/gobgp/blob/master/docs/sources/route-server.md

Number of prefixes	50	100	150
Run 1	33	57	116
Run 2	69	59	132
Run 3	57	61	120
Run 4	64	103	122
Average	55,75	70	122,5

Table 7: GoBGP convergence time in seconds

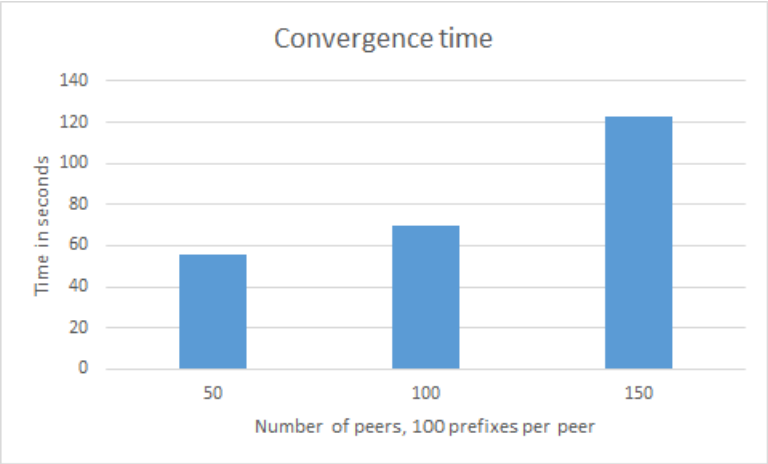

Figure 12: GoBGP Convergence times

It should be noted that because of the low number of test runs for the GoBGP tests, no hard conclusions could be drawn. However, the results do show the same trend that was observed in the results of the tests with BIRD. Furthermore, the number of peers and prefixes were decreased in these tests compared to the BIRD tests. Still the convergence times turned out relatively high compared to BIRD. There are two reasons why this may be the case. BIRD uses a single-table which has shown to drastically improve performance [23]. While this is not yet the case for GoBGP, this is currently the main focus of the development team and these numbers may be a result of the single-table development implementation of GoBGP. Additionally, the GoBGP route server ran on a different server due to hardware failure which may also have an effect on the results.

4.2 Observation summary

In the *three to one* setup, no significant convergence times have been observed. This is not at all surprising considering this does not in any way begin to approach the scale of IXPs which are afflicted by this issue. Nevertheless it is valuable to observe that a setup with 800 peers itself does not cause any issues, which could have been possible

considering the amount of UPDATE messages that need to be crafted to inform all peers.

After scaling up the amount of prefixes announced by each peer, the convergence problem does appear in the *many to one* tests. The convergence times of 100 and 1.000 peers show an increasing trend which is to be expected when increasing the amount of NLRIs that need to be processed. It is in the 10.000 prefixes test, which has a more realistic scale compared to IXPs, that increasingly problematic convergence times are observed. Within these tests, we can also see that the increase is not linear. That is to say, the convergence time of 10.000 prefixes with 400 peers is not half of 800 peers. Unfortunately, we do not have a solid theory as to why this is the case.

Additionally, it is not possible to deduce to what degree either the prefixes or peers are causing the issue. This is relevant because if route dissemination (phase 3) is causing most of the delay, adding another peer (containing 10.000 prefixes) should cause a greater increase in convergence time than when adding an additional 10.000 prefixes over existing peers. Unfortunately no results for this hypothesis can be generated in this testbed due to constraints in the available hardware.

However, the effects of phase 3 have been tested in an alternative way. This is done in the tests where the dissemination of routes was disabled by means of the export feature. The results show that no serious difference in convergence times occur.

In the *real world* tests, the only notable observation was that after receiving the last End-of-RIB, the route server was still sending out UPDATE messages. This can be explained due to the existence of more unique prefixes in this setup than compared to the 10.000 unique prefixes found in the *many to one* tests.

Lastly, when running similar tests in GoBGP, comparable increases are found. A striking difference though is that, in these tests, GoBGP was significantly slower than BIRD. This does not reflect the findings presented at the Euro-IX forum[26]. Two possible reasons may be related to a change in hardware and the use of a different GoBGP (developer) build. It is evident that no comparisons are to be made between the GoBGP and BIRD convergence times based on these measurements. We can only state that within GoBGP, a similar trend can be observed with regards to the increase in convergence time.

5 Proposed Solutions

In this section multiple solutions will be proposed to improve the convergence time observed before. On the protocol side, two solutions are proposed and also a solution on implementation side is proposed.

5.1 Considerations

The results show similar behavior to what has been described by administrators of large BGP implementations. It is fair to say that, when reaching the 600 peers in any of the BIRD tests, the route server clearly shows a delayed converged state. Servers with more powerful hardware may be able to prolong this from occurring, therefore the numbers at which a delayed converged state occurs is only significant within this testbed. More importantly, the tests have been able to induce this effect with tweaking solely the amount of peers connected and the amount of prefixes they announce. This result somewhat simplifies the problem as more complex setups (such as using large communities) do not appear to be necessary and can be avoided.

It may seem plausible to suggest that phase 1 of BGP is causing these growing convergence times based on the knowledge that the amount of prefixes being received is constantly increasing due to IPv4 de-aggregation and IPv6 gaining traction. However, the background research has uncovered there are implementations, such as GoBGP, which have multi-threaded this phase. However, the results of GoBGP tests show that even then the same convergence problems occur. Because of this, it is unlikely much can be gained here to reduce the issue.

As mentioned before, there is a theory that route dissemination (phase 3) is the main culprit in increasing the convergence time. The main hypothesis for this, is that the crafting of update packets is the most resource demanding step in the BGP process. If this were to be true, then increasing the amount of peers would contribute more to the problem than increasing the prefixes. Unfortunately, the available testbed was unable to scale to a point to do valid comparisons for this, such as $600 peers \times 1.000 pfx = 600.000 totalpfx$ versus $1.000 peers \times 600 pfx = 600.000 totalpfx$. While comparing this was not viable, it is possible to avoid the dissemination of routes altogether. In BIRD this was controlled by toggling the "export" feature. The results then show that even with route dissemination disabled, no significant changes in convergence times could be observed. For this reason, it is also less likely for phase 3 to be the main contributor to the problem.

What remains is phase 2 where the best path selection occurs. During the background research, the documentation of multiple implementations have been examined and indicate that no multi-threading for phase 2 is implemented in any of these. Because of this, the increase in prefixes that have to be processed cannot be shared across multiple cores as is the case for phase 1.

Based on this, it seems plausible that in current implementations, the sheer amount of prefixes to be processed in a serial manner in phase 2, is likely to be a major contributor to the problem. Because of this, the most effective solution would be to devise a means to be able to share the work, as is the case in phase 1. Therefore, in section 5.2.1 an addition to the BGP defined structures is proposed to allow multiple threads to simultaneously operate without violating the existing BGP rules. The decision not to meddle with the core mechanics, such as locking and phase progression, has been made because insufficient time is available in this project to fully grasp the consideration and consequences that fundamentally changing them may have on the operation of BGP.

This project recognizes, however, that even a mere addition to the BGP structure is not done overnight. If the solution proves to be effective in a proof of concept, it would still take more time before this would be implemented in a production viable BGP implementation. For this reason, another solution has been proposed in section 5.3 that can be used to share the work of phase 2 without requiring an alteration in BGP or its implementations. Because these solutions propose the use of existing technologies, such as load balancing and iBGP, this can be tested and implemented relatively quickly. Although this solution does not allow for multi-threading in a classical sense, it does allow for the work to be distributed over multiple BGP instances.

A more general observation done during this research, is that reducing the amount of UPDATE messages sent in the first place, may somewhat relieve the entirety of BGP from some load. A suggestion as to how this can be achieved is proposed in section 5.2.2, where we suggest working with hashes to reduce the amount of data that has to be exchanged.

5.2 Protocol

On the protocol, two changes are proposed which should improve the convergence time of the BGP system. Below the two solutions are explained in detail.

5.2.1 Phase 2 solution

When we presume the best-path selection (phase 2) is indeed the main issue with regards to the increased convergence time, it is only logical that the problem should be addressed here. Because the observation in the testbed has been that indeed an extended period of time exists where a single core is fully utilized, it makes sense to attempt to multi-thread this, as to increase the amount of processing that can take place. When following the RFC rules, it is mandatory to lock all Adj-RIBs-In before proceeding with the best-path calculation. In doing so, any attempt to run these calculations in parallel would not be possible as the data required to do so would be inaccessible. While the reason for this is not mentioned in the RFC, one can see it is an effective measure to maintain consistency throughout the phases and their progression. In other words, while phase 2 is running, no other processes can access the data mid-calculation. In order to allow for

this, one solution would be to alter the rules of locking and access. However, because such a change can fundamentally impact the operation of BGP and we do not possess enough insight in the software engineering field to assure no other additional problems would be created, we have opted for a solution that abides by the existing RFC rules.

Therefore, the first change proposed here is the addition of another Adj-RIB-In between the Adj-RIB-In and the Loc-RIB. This extra Adj-RIB-In is sorted on prefix rather than sorted on peer, as is the case with the original Adj-RIB-In. The main advantage this offers, is that the locking issue in phase 2 as explained in section 2.2.3 does not pose an issue anymore.

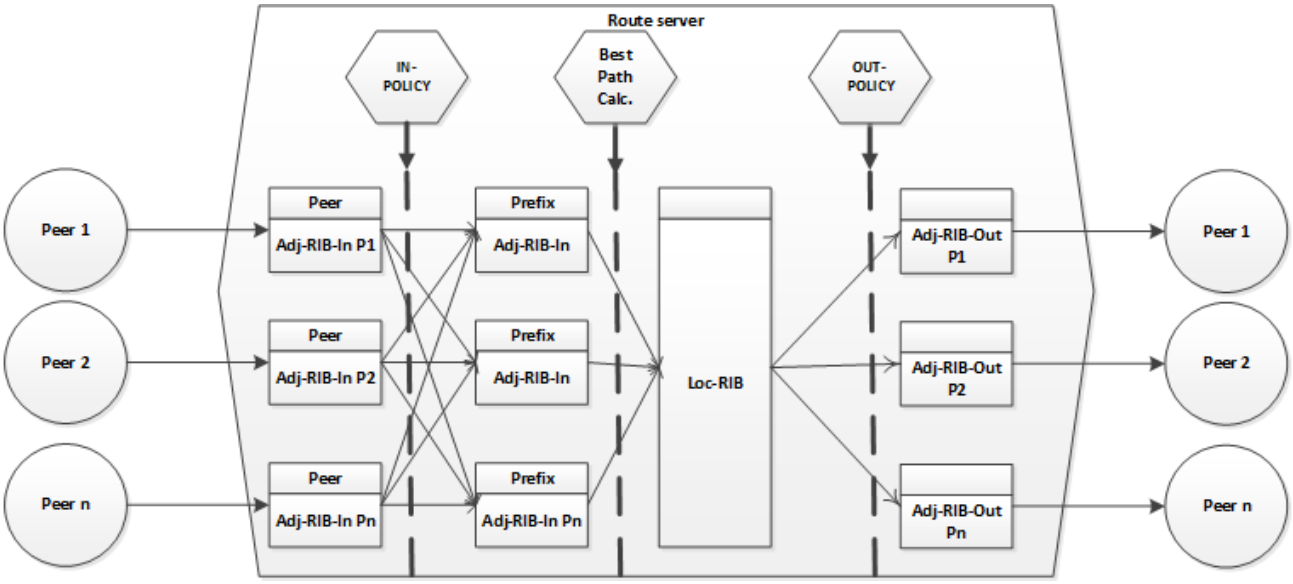

Figure 13: Graphical representation of the proposed solution in phase two

In this new proposed flow, shown in figure 13, whenever a peer sends an UPDATE message, first all prefixes are put into the Adj-RIB-In for that specific peer, just as in the current situation. Phase one, as described before in section 2.2.3, is executed as normal. The differences start from this point on, as between phase one and phase two, all prefixes in the Adj-RIB-In are put in another Adj-RIB-In. However, this Adj-RIB-In is sorted per prefix. This means each prefix gets its own Adj-RIB-In. All peers can write new paths for that prefix in the prefix's Adj-RIB-In.

When phase two wants to start, it only requires to lock the Adj-RIB-In for the prefix it wants to calculate the best path for. This means no locking issues occur with this process. When a peer sends an UPDATE message containing a prefix for which the BGP process is calculating the best path for, thus the Adj-RIB-In for that prefix is locked, this path will be held in the peers Adj-RIB-In until the BGP daemon is done

with calculations for that prefix and unlocks the prefix' Adj-RIB-In.

The main benefit of this solution is that now phase two could be multi-threaded, one thread per prefix. This could potentially drastically improve convergence time, as the BGP daemon can finally utilize all available processing power during phase two.

It is good to note that, just as in the original RFC 4271, the solution provided above is a conceptual solution, meaning it does not have to be implemented precisely as described as long as the implementations support the described functionality and they exhibit the same externally visible behavior. This is specifically mentioned here because in some implementations the second Adj-RIB-In, based on prefix, could be optimized in terms of memory usage, as a table per prefix could require lots of memory.

5.2.2 Hashing mechanism

While performing the link-flaps in the testbed, we considered that it might not even be necessary for all peers to disseminate their entire RIBs again. If the amount of UPDATE messages can be reduced, this may contribute to reducing the load on a BGP implementation for all phases. Admittedly, this suggestion would not help as much in reducing the work that has to be done particularly in phase 2 as compared to the prefix-based Adj-RIBs-In. The reason being that if the peer providing the current best route to a prefix disconnects, its entry in the Loc-RIB should be invalidated, recalculate and reconsider all remaining available routes. However, avoiding any unnecessary workload on the route server may help to spend these resources elsewhere, where they are needed to process UPDATE messages.

The second solution on protocol level is therefore based on a hashing mechanism that should take place on both the peer (that sends an UPDATE) and the route server (that receives an UPDATE). These large UPDATE messages full of NLRIs can be avoided by means of exchanging just hashes to validate the data in the route server still matches the data in the peers Adj-RIB-Out.

The peer would send an OPEN message to the route server. This OPEN message should include a hash of the routes it will be sending to the route server. The route server should then keep track of all routes and hashes it got from its peers in memory. As soon as the route server receives an OPEN message from a peer, after a link flap, the route server compares this hash with the previous hash it had calculated for that peer. When the hashes match, the route server still has the latest information from that peer thus that peer does not have to send its full table to the route server.

An important caveat here is that while the information sent by the peer may not have changed, the policy on the route server potentially could have changed, essentially necessitating the reprocessing of routing information. If this situation occurs, the peer could be asked to send the RIB nonetheless or *soft-reconfiguration* can be used when

enabled. In that case, the soft-reconfiguration table for that peer will already have all data available and validates this is still up-to-date by means of the hash.

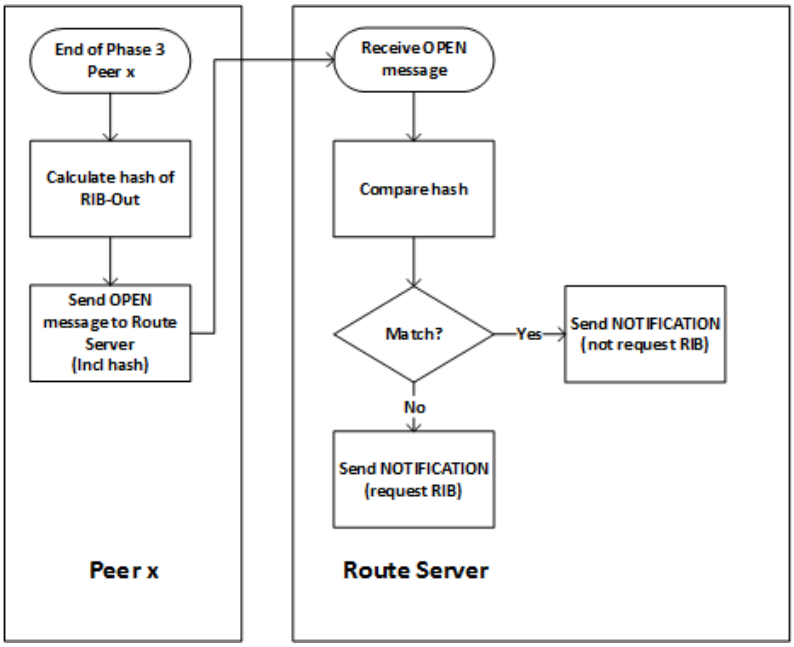

Figure 14: Graphical representation of the hashing solution

Figure 14 shows the flow diagram of how the hashing should work in practice. When peer X ends phase three, the peer should calculate a hash from the contents of its RIB-Out. The contents of its RIB-Out should be similar to the contents of the Adj-RIB-In on the route server before processing. The peer sends UPDATE messages with its routing information to the route server. As soon as the route server gets an End-of-RIB, the Adj-RIB-In should be filled with all information peer X wanted to send to the route server. At this point the route server should calculate a hash of the contents of the Adj-RIB-In for that peer. Important to notice here is that the Adj-RIBs-In need to be persistent, as opposed to normal operation. The route server now has a persistent Adj-RIB-In for that peer, including a hash calculated from the contents.

In case of a link flap, the peer notices the route server was unavailable for some time due to the KEEPALIVE and Hold-Down timers timing out. The peer tries to send OPEN messages to the route server to establish the BGP session again. This OPEN message should now also include the hash it had calculated before, on the RIB-Out. The route server kept all Adj-RIB-In information, including the hashes. As soon as the route server is reachable again, it receives the OPEN message from the peer. The route server will extract the hash from the OPEN message and compare that hash with the hash it had stored for that peer. When the hashes match, thus the routing information from the

peer did not change, the route server sends a NOTIFICATION message to the peer, to signal it should not send its full table. When the hashes do not match, the route server sends a NOTIFICATION message to the peer to signal it should send its routing information.

A benefit of this hashing mechanism is that in case of a link flap on the route server, or on multiple peers, not every peer will need to send its full routing information again if nothing changed. It is a quick and easy way to verify if any changes occurred and if not, it lowers the load on the route server to process all routes for all peers.

5.3 Implementation

On the implementation side a load balancing solution is proposed. While this does not directly multi-thread the BGP daemon itself, it does allow the work, normally done by a single BGP phase 2 process, to be performed in parallel. This solution may be particularly interesting as a short-term solution because it can be set up without altering any existing RFC standards or code. In essence, best-path selection can then be calculated in parallel as would be the case in a multi-threaded daemon. On one side clients can set up peering to the load balancer. On the other side multiple route servers are connected in a full iBGP mesh with each other, besides having an eBGP connection to the load balancer.

Figure 15 shows the external side in front of the load balancer. On this side the load balancer exposes a single IP address where all customers can connect to by means of eBGP. The load balancer will then forward the BGP packets to a route server, while balancing the number of peers per route server. Effectively the BGP sessions are set up to one of the route servers through the load balancer.

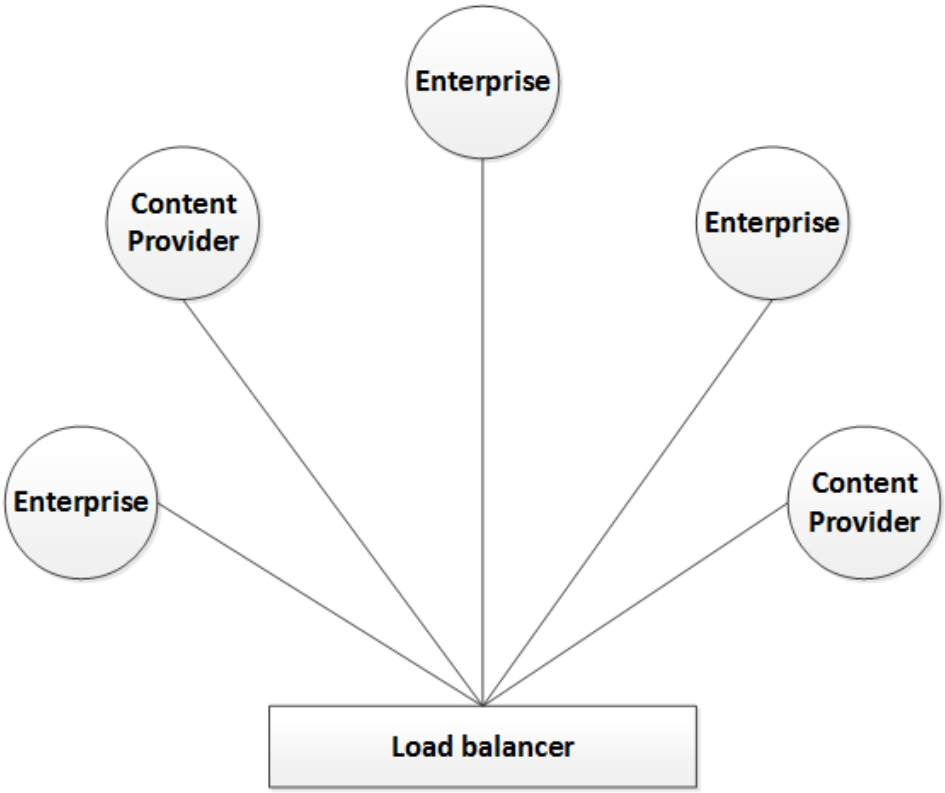

Figure 15: Graphical representation of the load balancing solution

Figure 16 shows the internal side behind the load balancer. On this side a cluster of route servers is connected in a full iBGP mesh, including an eBGP connection towards the load balancer. The example in figure 15 shows a total of five route servers. Say that 500 peers set up peering to the load balancer. The load balancer does its job and balances the load so that each route server has 100 established external peers. Besides these peers, the route servers also peer with each other resulting in a total of 104 peers per route server. In general one can say that the number of peerings per route server is equal to $\frac{i}{j} + j - 1$ where i = number of customers, j = number of route servers and z = number of peerings per route server. The number of peerings per route server is equal to the number of customers divided by the number of route servers plus the number of route servers minus 1 (itself).

In this example it comes down to $\frac{500}{5} + 5 - 1 = 100 + 4 = 104$.

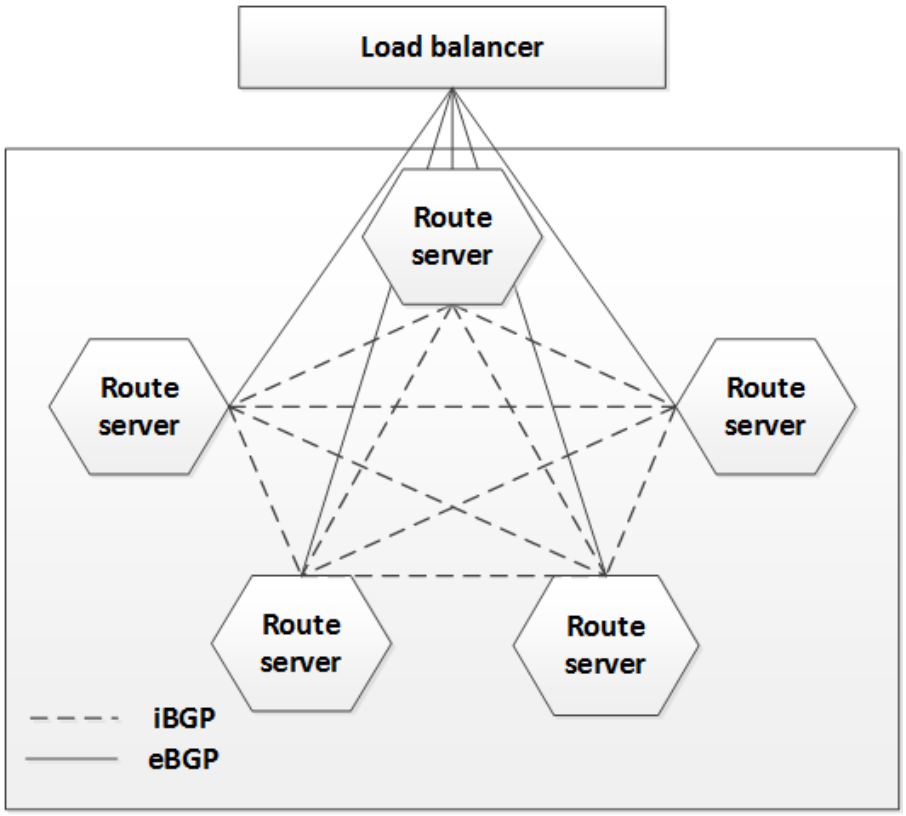

Figure 16: Graphical representation of the load balancing solution

The load is now spread over multiple servers, so instead of one route server having to calculate paths for 500 peers, it now has to calculate paths for 104 peers. The best path calculation is now divided over the other route servers which will only announce a best path. Although the graphical representation shows multiple route servers, this could also be implemented on one physical server with multiple BGP daemons running. Another benefit is that all customers can configure their routers to connect to a single IP address, namely the address of the load balancer. The load balancer will then forward connections to the appropriate route server. Implementing this without a load balancer, but with multiple route servers, would require additional administration of which customer is configured on which route server. As [23] shows, it could be beneficial to, when load balancing is implemented, also tweak the maximum MTU size. In the presentation it shows that changing the MTU from 1500 to 9000 bytes, improves convergence time by 33%.

Ideally, the load balancer and multiple iBGP processes would run on the same server. By doing this, the resources of commodity hardware are utilized more effectively. This also reduces some reliability on the network infrastructure for iBGP communication.

While configuration management tools may allow for effective configuration of such an environment with multiple daemons, it may be more beneficial to let a single implementation set up multiple iBGP processes in a smarter way to attain improved (dynamic) distribution of work and manageability.

6 Discussion

Due to BGP currently being the only EGP in use, any shortcomings that may arise will have to be addressed in order to continue the exchange of route information over the Internet in a reliable manner as the Internet continues to grow.

Currently, only relatively large BGP implementations, such as IXPs are experiencing scaling problems. The testbed used in this project aimed to roughly mimic these environments. Notably, BGP attributes have been intentionally unmodified as there was no indication this would affect the convergence times while only increasing complexity. The tests conducted showed a significant increase in convergence time while solely tweaking with the amount of peers and prefixes. It is worth noting that the measurements produced by conducting the tests are not significant in an absolute sense. If one was to use a testbed with a high clock rate, the convergence times would probably be lower. What the research does show is that a significant increase can be observed relative to other measurements done within the same testbed.

The way forward to go about resolving it was to identify the main issue. While it was clear the single-threaded nature of BIRD was causing the increased time before work was completed, other implementations seemed to suffer the same problem. Because of this, the search for the root cause was likely in the BGP standard itself. Some suspicion existed that particularly the route dissemination phase was burdening the BGP process. Although we cannot conclude it with unequivocal certainty, experiments conducted show this is likely not the case. Also, the receiving and parsing of packets is also unlikely to be the culprit due to this part being multi-threaded in some implementations.

Because of this, the best-path calculation is suspected being the lead cause. This idea is supported by the fact that no implementation in production has multi-threaded this so far, explaining the observation of a single core being pushed to its limits.

One of the proposed solutions called for an addition of tables to the ones currently described in the BGP standard. Because the change does not really touch existing BGP mechanisms, implementing it into existing BGP applications is realistic. Additionally, the processing of updates is a local matter and it is not required for both BGP speakers to have a prefix-based Adj-RIB-in capability. Although this may require some coordination between the parallelized phases with regards to phase progression, there is no indication that limiting the scope of a process to a single prefix changes the end result of the best-path calculation in any way. However, standardization would be required before the general public would accept this addition, which would be a lengthy process.

The load-balancing solution, on the other hand, would not need to be standardized in this way. The main aim here is to propose an intuitive solution that is implementable today. While this can be distributed over multiple machines in the network, doing this

on a single machine would reduce complexity and be more efficient. Additionally, most commodity hardware would be sufficient as multi-core CPUs are widespread. Although a BGP load-balancing system may be created by installing and configuring the software packages manually or through configuration management tools, it may be beneficial to put all features (load-balancing, multiple BGP instances) under a multi-threaded process for (dynamic) distribution of work and manageability.

7 Conclusion

This project has contributed to the identification of processes which are causing excessive convergence times in large scale deployments of BGP. It has shown the reported behavior is reproducible and determines the scale and properties in which the phenomenon can be expected to occur. Furthermore, this project has shown that it is not bound to BIRD but that it is likely a broader issue.

While working with the testbed developed for this research, measurements have shown no significant difference in completion time with and without route dissemination enabled. This is contradictory with the supposition, found within the networking community, that the transmitting of UPDATE messages (phase 3) is the main culprit of CPU resource consumption[27].

Based on the background research and experiments performed within this project, the hypothesis that the best-path selection (phase 2) within BGP is hogging the resources in lieu of route dissemination (phase 3) is more probable. In order to counter increasing convergence times affecting the stability of some BGP environments in the future, this project has proposed multiple solutions to do so.

The first recommendation is to append prefix-based Adj-RIBs-In to the BGP standard. This allows for multi-threaded best-path selection while leaving the existing BGP locking and phase-progression mechanics untouched. It is also recognized that avoiding large updates by means of hashing can provide additional benefits by reducing the overall workload.

Another solution proposes the division of work over multiple threads without changing existing BGP mechanics. Central to this, is the use of load-balancing and iBGP. While overhead can be expected even if the optimizations suggested in this research are implemented, there will be a tipping point at which it will outperform current single-process implementations.

8 Future work

While this research has shown it is reasonable to assume that the problem described is due to BGP phase three serialization, it has not been definitively proven. In order to definitely confirm this, the code of a BGP implementation should be thoroughly debugged while running test scenarios. Although this project has partially done this by using the debug feature, it is unable to clearly distinguish the time spent in each phase as to identify the culprit. Even then it may be implementation dependent. However, the fact that more, if not all, current implementations suffer the same issue suggests a common bottleneck.

Additionally, during this project the testbed reached a scale in which the increased convergence time was observed. Whether the increase in peers or the increase in prefixes played a primary role in this, remains uncertain. The testbed had reached its limit due to the available hardware for this project. This limited the tests to 800 peers with 10.000 prefixes for the non-unique scenario. For unique prefixes, the route server itself ran out of memory. It would be interesting to scale up the testbed further to identify:

- The progression of convergence times as the number of peers and prefixes increase.
- Comparing the convergence time of unique to non-unique as to rule out the tie-breaking mechanism contributing to the high utilization.

Lastly, both proposed implementations will need to be tested in a proof of concept. With the load balancing solution, the main concern will be reducing the overhead cost and finding a point where it is more feasible to implement such an architecture in favor of a single BGP instance. For the addition of prefix based Adj-RIBs-In, the challenge lies in efficient use of memory and complying with the RFC.

Acknowledgements

We hereby would like to thank Stavros Konstantaras and Aris Lambrianidis of AMS-IX for supporting and guiding us thoroughly during our research. We often have had good sessions to discuss the progress of the research. We also would like to thank Job Snijders (NTT Communications) for taking the time at NLNOG (June 16th, 2017) to discuss our research and proposing an initial idea about load balancing BGP.

References

[1] D.L. Mills. Exterior gateway protocol formal specification, 1984.

[2] K. Lougheed. A border gateway protocol (bgp), 1989.

[3] Y. Rekhter, T. Li, S. Hares. A border gateway protocol 4 (bgp-4), 2006.

[4] InetDaemon. Bgp network layer reachability information (nlri), 2017.

[5] S. Sangli, et al. Graceful restart mechanism for bgp, 2007.

[6] Cumulus Networks. Configuring border gateway protocol - bgp; enabling read-only mode, 2017.

[7] Cumulus Networks. Configuring border gateway protocol - bgp, 2017.

[8] Juniper Systems. hold-time (protocols bgp), 2017.

[9] K. Ramasamy D Medhi. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann, 2010.

[10] Elisa Jasinska, Chris Malayter. (ab)using route servers, 2010.

[11] Claudio Jeker. Routing with openbsd using openospfd and openbgpd, 2010.

[12] T.Klockar, M. Hidell, et al. Modularized bgp for decentralization in a distributed router. 2005.

[13] X. Zhang, P. Zhu, X. Lu. Fully-distributed and highly-parallelized implementation model of bgp4 based on clustered routers. 2005.

[14] K. Wu, J. Wu, K. Xu. A tree-based distributed model for bgp route processing. 2006.

[15] Kefei Wang, Lei Gao, Mingche Lai. A scalable multithreaded bgp architecture for next generation router. 2011.

[16] Gao Lei, Lai Mingche, Gong Zhenghu. Practical non-blocking route propagation-technology for threaded bgp. 2009.

[17] Lina Ding, Xingwei Wang, Fuliang Li, Min Huang. A parallel processing method for border gateway protocol update messages. 2015.

[18] Mingwei Xu Xueshi Jiang and Qi Li. Compact route computation: Improving parallel bgp route processing for scalable routers. 2011.

[19] Arnold Nipper (DE-CIX). Nick Hilliard (INEX), John Souter (LINX). Euro-ix quagga development. page 6, 2009.

[20] Boian Bonev. Euro-ix branch of quagga, 2017.

[21] Mikrotik. Cloudcorerouter and routeros v6.x, 2014.

[22] Cisco Systems. Virtual route reflector using cisco ios xrv, 2014.

[23] Eric Nghia Nguyen-Duy, Aris Lambrianidis. Ams-ix route-server performance test euro-ix 20th, 2012.

[24] Euro-IX. Internet exchange points. Technical report, Euro-IX, 2015.

[25] Bryan Eikema. Bgp routing security and deployment strategies, 2015.

[26] Shu Sugimoto, Wataru Ishida. Performance evaluation of bird and gobgp, 2016.

[27] Ondrej Zajicek. Identifying bgp convergence bottleneck, 2017.

Appendix A Many-to-one scenario results

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	12	4	4
Run 2	4	4	3
Run 3	7	6	10
Average	7,67	4,67	5,67

Table A8: Convergence time in seconds with 10 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	31	5	11
Run 2	7	5	12
Run 3	7	16	12
Average	15	8,67	11,67

Table A9: Convergence time in seconds with 100 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	8	21	27
Run 2	29	12	49
Run 3	7	19	40
Average	14,67	17,33	38,67

Table A10: Convergence time in seconds with 200 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	20	12	49
Run 2	13	16	78
Run 3	12	13	48
Average	15	13,67	58,33

Table A11: Convergence time in seconds with 300 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	20	42	75
Run 2	32	35	83
Run 3	25	53	105
Average	25,67	43,33	87,67

Table A12: Convergence time in seconds with 400 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	14	94	79
Run 2	16	71	115
Run 3	44	63	170
Average	24,67	76	121,33

Table A13: Convergence time in seconds with 500 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	43	39	241
Run 2	38	38	265
Run 3	40	54	281
Average	40,33	43,67	262,33

Table A14: Convergence time in seconds with 600 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	29	46	380
Run 2	90	52	431
Run 3	31	51	280
Average	50	49,67	363,67

Table A15: Convergence time in seconds with 700 peers

Number of prefixes	**100**	**1.000**	**10.000**
Run 1	75	49	446
Run 2	65	82	457
Run 3	66	73	322
Average	68,67	68	408,33

Table A16: Convergence time in seconds with 800 peers

Appendix B BIRD Configuration

```
 ### Begin Example Configuration

log "/var/log/bird.log" all;
log syslog all;
watchdog warning 1s
debug protocols { states, interfaces, events, packets }

debug latency on
debug latency limit 1s
router id 10.0.8.2;
define myas = 65000;

protocol device { }

### The following function excludes weird networks such as
### rfc1918, class D, class E, and too long or too short prefixes

function avoid_martians()
prefix set martians;
{
martians = [ 169.254.0.0/16+, 172.16.0.0/12+, 192.168.0.0/16+, 10.0.0.0/8
224.0.0.0/4+, 240.0.0.0/4+, 0.0.0.0/32-, 0.0.0.0/0{25,32}, 0.0.0.0/0{0,7}

### Avoid RFC1918 and similar networks
if net ~ martians then return false;
return true;
}

function avoid_crappy_prefixes()
{
if net.len < 8 then return false;
if net.len > 29 then return false;
return true;
}

### Protocol Template

template bgp PEERS {
local as myas;
import all;
export all;
```

```
import limit 1000000000;
rs client;
passive on;
}

### Generic Input Filter

filter bgp_in {
if avoid_martians() && avoid_crappy_prefixes() then accept;
else reject;
}

### BGP output filter (based on communities)

function bgp_out(int peeras)
{
if ! (source = RTS_BGP ) then return false;
if peeras > 65535 then return true; ### communities do not support AS32
if (0,peeras) ~ bgp_community then return false;
if (myas,peeras) ~ bgp_community then return true;
if (0, myas) ~ bgp_community then return false;
return true;
}
protocol bgp R_654 from PEERS {
description "R_654 - peer 4";
neighbor 10.0.8.4 as 654;
import filter bgp_in;
export where bgp_out(654);
}
protocol bgp R_655 from PEERS {
description "R_655 - peer 5";
neighbor 10.0.8.5 as 655;
import filter bgp_in;
export where bgp_out(655);
}

. . .
```

Appendix C GoBGP Configuration

```
[global.config]
  as = 65000
  router-id = "10.0.8.2"

[[neighbors]]
  [neighbors.config]
    neighbor-address = "10.0.8.4"
    peer-as = 654
  [neighbors.transport.config]
    passive-mode = true
  [neighbors.route-server.config]
    route-server-client = true
[[neighbors]]
  [neighbors.config]
    neighbor-address = "10.0.8.5"
    peer-as = 655
  [neighbors.transport.config]
    passive-mode = true
  [neighbors.route-server.config]
    route-server-client = true

. . .
```