DEGREE PROJECT IN ELECTRONICS AND COMPUTER ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2021

Performance analysis of the FRRouting Route Server

EMIL STÅHL

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

KTH Royal Institute of Technology
Stockholm

School of Electrical Engineering and Computer Science

DEGREE PROJECT IN ELECTRONICS AND COMPUTER ENGINEERING

Performance analysis of the FRRouting Route Server

Author
Emil Ståhl

Supervisor
Alexandros Milolidakis

Examiner
Marco Chiesa

June 2021

Abstract

The delivery of IP traffic on the Internet depends on the complex interactions between a set of autonomous systems that exchange routing information about IP prefix destinations utilizing the Border Gateway Protocol (BGP). Autonomous systems are often connected to a route server located at an Internet eXchange Point, which facilitates the administration of BGP peering arrangements for all parties connected to it. One of the most popular open-source implementations of BGP is the FRRouting software suite, making it an important part of the Internet infrastructure.

This thesis investigates the performance of FRRouting, configured as a route server, in terms of its capabilities of announcing routing information on the network to a set of peers emulating autonomous systems. The routing information consists of a set of distinct IP prefixes that FRRouting receives from its peers. With various benchmarks of different configurations, we relate the number of received prefixes to the number of prefixes that FRRouting has announced on the network to its peers in a given time span. The output of the thesis is a wide overview of how the performance of FRRouting is impacted by different configurations such as filtering of specific prefixes that are not announced to the peering networks.

The obtained results show that there exists a divergence between the number of prefixes that have been received and announced by FRRouting. Specifically, the discrepancy shows that FRRouting, in our benchmarks, is incapable of announcing prefixes at the same rate as it receives these prefixes from its peers. In general, the number of announced prefixes is dependent on how the prefix filter is configured. However, one can question what real-world limitations this may result in. Suggestions for future work include developing a more realistic benchmarking environment that does not rely on emulated peers as well as improving how the routing information is recorded. There also exists a wide variety of other metrics and configurations of FRRouting that may reveal further limitations.

Keywords: FRRouting, Border Gateway Protocol, Route Server, Internet eXchange Points

Sammanfattning

Leveransen av IP-trafik på Internet beror av komplexa interaktioner mellan en uppsättning autonoma system som utbyter dirigeringsinformation med hjälp av Border Gateway Protocol (BGP). Autonoma system är ofta anslutna till en dirigeringsserver belägen vid en Internetknutpunkt vilket underlättar administrationen av BGP-förbindelser mellan de parter som är anslutna till dirigeringsservern. En av de mest populära implementationerna av BGP med öppen källkod är FRRouting vilket gör denna mjukvara till en betydelsefull komponent för Internets infrastruktur.

Detta arbete undersöker prestandan av FRRouting konfigurerad som en dirigeringsserver vad gäller dess kapacitet att behandla och via nätverket vidarebefordra dirigeringsinformation till en uppsättning autonoma system. Dirigeringsinformationen består av en samling IP-prefix som FRRouting erhåller från de autonoma systemen. Genom att variera konfigurationen av FRRouting undersöker vi hur antalet mottagna IP-prefix relaterar till den mängd IP-prefix som FRRouting har vidarebefordrat till de autonoma systemen under en given tidsperiod. Arbetet resulterar i en bred genomgång av hur prestandan för FRRouting påverkas av olika konfigurationer såsom filtrering av specifika prefix.

De erhållna resultaten visar att antalet vidarebefordrade IP-prefix skiljer sig markant från antalet prefix som dirigeringsservern erhållit från de autonoma systemen. Denna avvikelse visar att FRRouting inte är kapabel att vidarebefordra IP-prefix i samma takt som dessa mottages från de autonoma systemen. I allmänhet beror antalet vidarebefordrade IP-prefix av hur prefixfiltreringen konfigurerats. Det kan dock ifrågasättas vilka verkliga begränsningar detta kan resultera i. Förslag på framtida arbeten inkluderar att utveckla en förbättrad testmiljö som inte förlitar sig på emulerade autonoma system samt att förbättra tekniken för insamling av vidarebefordrade IP-prefix. Det existerar även ett stort antal andra mätvärden och konfigurationer av FRRouting som möjligtvis kan resultera i att ytterligare begränsningar uppdagas.

Nyckelord: FRRouting, Border Gateway Protocol, Route Server, Internet eXchange Points

Acknowledgement

Throughout the writing of this thesis, I have received a great deal of support and assistance. I would first like to thank my supervisor, Alexandros Milolidakis, whose expertise was invaluable in formulating the research questions and methodology as well as improving the overall structure of the thesis. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. As well, I would like to acknowledge the constant engagement and guidance I have received from all of the ExaBGP and FRRouting community, your support was truly invaluable in configuring the software correctly. I would also like to thank Tâm Vũ for his comprehensive expertise in linguistics and LaTeX.

Contents

Chapter 1

Introduction

This chapter introduces the thesis. Section 1.1 presents the area to be researched. In Section 1.2, we define the problem that the work aims to address. The purpose and goal of the work are described in Section 1.3 and 1.4. Further on, Section 1.5 describes potential benefits with the work and how these relates to ethical and sustainability aspects. Next, Section 1.6 presents the methodologies used while Section 1.7 describes the delimitations for the work. Lastly, Section 1.8 presents the outline for the rest of the thesis.

1.1 Background

The internet consists of several inter-connected autonomous systems operated by Internet Service Providers, corporations, universities, and other organizations. These networks utilize the Border Gateway Protocol (BGP) in order to exchange routing information with each other. Routing information is the result of the higher-level decision making that directs network packets from their source toward their destination through intermediate network nodes and is exchanged each time there is an update on a certain route such as a new route, a change of some properties, or a withdraw of a route. BGP is therefore a critical component of the Internet infrastructure. One of today's most popular open-source BGP implementations is FRRouting (FRR). Given the large-scale of the Internet network, understanding the performance of BGP implementations has long been a goal of paramount importance in the networking community.

1.2 Problem

Analysing the performance of BGP implementations is key to tackle the scalability challenges associated with the large-scale in which BGP operates. Today, there exist over 800,000 destination prefixes that are announced through BGP [1]. Network devices processing BGP information must be highly optimized performance-wise in order to be able to process any incoming information in real-time. Processing hundreds of thousands announcements after restarting a process may take up to 10 minutes to complete [2]. In addition to this, when supporting security signatures in BGP the processing times would increase even more. Since FRR is one of the most popular open-source BGP implementations it is important to have an understanding of how well FRR is capable of processing incoming routing information in real-time.

1.3 Purpose

The purpose of this work is to analyse the performance of FRR by quantifying at what rate it is capable of processing BGP announcements. By analysing the performance of FRR in different network conditions, one can use this work for tailoring network configurations in order to optimize for performance. The results are potentially of interests for networking organizations, operators, and the open-source community.

1.4 Goal

The goal of this work is to run a set of experiments where the BGP implementation processes BGP messages under varying configurations. The results are then going to be described, analyzed, and discussed in order to determine if there are any bottlenecks regarding FRR's capability of processing BGP announcements under different configurations.

1.5 Benefits, Sustainability, and Ethics

There are many possible benefits that may result from understanding the performance of BGP implementations. Since BGP is part of the Internet's core infrastructure, which most of the modern society relies on, it is crucial to understand the underlying technology used in order to optimize for performance, up-time, and security [3]. An aspect of sustainability exists when considering the development and usage of the Internet. All forms of digitization inevitably lead to a further dependency on electricity. However, one could also argue that a well functioning Internet is a keystone for increasing social sustainability and achieving all of the sustainable development goals that have been established by the United Nations [4]. Regarding ethics, there exists several areas that are interesting from a ethics perspective. For example, showing limitations in how FRR performs and function might cause harm to the developers of the software, leading to a deteriorated reputation as well as deter individuals, companies, and organizations of using the software. Moreover, other ethical aspects may be related to that potential limitations of FRR can result in a new vector of potential attacks against organizations running the FRR software suite.

1.6 Methodology

The decision of methodology is important in a scientific study. In this work, the research method used is defined as *experimental* since the method focuses on finding aspects that affect how FRR handles BGP announcements [5]. Moreover, the research is also *applied* since it makes use of the FRR software suite as well as a wide variety of tools, which means that the methodology is based on work performed by other researchers and developers. With this in mind, the methodology used to perform this work consists of the choice and configuration of a benchmarking environment, a method for performing the experiments, and an evaluation method to analyze and evaluate the system itself. The chosen method ought to fulfill the purpose and goal of this thesis, which is to determine at what rate FRR is capable of processing BGP announcements under different configurations.

1.7 Delimitations

Due to the scope of this project as well as our knowledge and available resources, a set of delimitations are needed for this work. The delimitations can be constricted to those regarding our use of hardware and software configurations. The methodology and analysis used in this work does not, in any particular way, compensate for environmental differences, that could arise from our choice of hardware and software configurations of the servers used in the experiments. For example, the number of ways one could configure the FRR software suite is vast. Therefore, this work is limiting these configurations to a handful that only alter filtering of received routing information. In addition to this, due to limited resources such as servers and switches the methodology of this work make use of emulated BGP peers and virtualized network switches.

1.8 Outline

This thesis is organized as follows. Chapter 1 gives an introduction to the thesis where the background is briefly explained. In addition to this, the purpose, the goal, and the methodology used are shortly explained along with its connection to ethics and sustainability. Chapter 2 is a recess of the theoretical background of the area to be researched. The field in computer science that is networking and its sub-field inter-domain routing are presented. Further, the underlying functionality of IXP's and route servers are presented as well as the Border Gateway Protocol and its implementation in the FRR software suite are elucidated. Lastly, the most essential previous work is briefly explained. Further on, Chapter 3 describes the methodologies and methods used to perform the work, which include the benchmarking environment and metrics. Next, Chapter 4 presents the results of the performed benchmarks, mainly focusing on the quantity of processed BGP announcements and how these differ in-between configurations of FRR. Chapter 5 analyses and discusses the results. In other words, what the results mean and why they look the way that they do. We also discuss how the experiments could be changed to improve the accuracy and reliability of the benchmarks. Finally, Chapter 6 presents a conclusion of the thesis in relation to its purpose and goal, as well as potential future work that can be performed in order to improve the conclusions of this work.

Chapter 2

Background

This chapter introduces briefly the area of networking and its sub-field inter-domain routing. Firstly, we describe autonomous systems and the standard inter-domain routing protocol that is the Border Gateway Protocol and its open-source implementation FRRouting. It continues by describing more specific topics of the area such as the critical crossroads of Internet traffic, *i.e.*, Internet eXchange Points, and their route exchange services, *i.e.*, route servers. Finally, we discuss the related work on the topic.

2.1 Autonomous Systems

The Internet consists of a large number of distinct independent entities, called *Autonomous Systems* (ASes). The administrators of an AS have full control over the set of Internet Protocol (IP) prefixes owned by the AS as well as its routers and links. Examples of ASes are Internet Service Providers (ISPs), Content Delivery Networks (CDNs), companies, and universities. An AS can be defined as a domain owning a set of IP prefixes. Multiple ASes are interconnected and exchange routing information about IP prefix destinations that reside in each AS. A unique identifier called *Autonomous System Number* (ASN) is assigned to each AS by a Regional Internet Registry (RIR). Currently, there are five RIRs that manage ASNs as well as the regional unallocated IP address space. The Internet Assigned Numbers Authority (IANA) delegates the assigning of ASNs to the five RIRs, each responsible for a region of the world. The five RIRs consists of AFRINIC serving Africa, ARIN is responsible for North America and Antarctica, APNIC serves most of Asia and Oceania, LACNIC controls most of Caribbean and all of Latin America while RIPE serves Europe, Central Asia, Russia, and West Asia. The topology within an AS, in which multiple internal routing protocols may be used such as RIP, OSPF, IS-IS, is commonly not visible to outside the specific domain by design. However, the communication to neighboring domains is done through border routers that run an Exterior Gateway Protocol [6].

2.2 Intra-domain and Inter-domain Routing

In order to connect several ASes, the internal routing inside a domain, called *intra-domai*n routing, is separated from the routing between domains defined as *inter-domain* routing. Intra-domain routing is the exchange of routing information between routers that exist within the same AS. As commonly a single entity controls the domain, the protocols used are optimized for low convergence time. Inter-domain routing, on the other hand, manages reachability information shared between routers of different domains which consequently results in a higher convergence time. This is defined as the time it takes for a set of routers to obtain the same routing information about their inter-connected peers. Furthermore, the transition between inter-domain and intra-domain routing is established at special border routers that sit at the edge of ASes and route traffic between the internal network and neighboring domains. Just as internal routers use multiple types of the Internal Gateway Protocol (IGP) to exchange routing information and to build routing tables, border routers use External Gateway Protocols (EGPs). The EGPs are optimized to be able to handle external routing since it differs from how internal routing is applied within an AS. One of the first protocol implementations for routing between domains was EGP originally defined in 1982 [7]. The modern protocol for inter-domain routing is BGP4 and is further presented in section 2.3. By utilizing BGP, border routers summarizes, receives, and advertises BGP messages which includes information regarding the internal routes of the AS. Apart from advertising the internal routes of the border router's own AS, it also advertises any external connections it might have to neighboring ASes. Moreover, only reachability information is shared with external routing protocols which gives the advantages that the internal topology of the neighboring networks is hidden simplifying configuration. As a result, the nodes only need to determine the next hop toward a destination [8].

2.3 Border Gateway Protocol

In 1995, RFC 1772 proposed an outline for the migration from EGP to the Border Gateway Protocol (BGP) to solve serious scaling problems such as routing tables becoming too large for EGP to handle [9]. Today, BGP version 4 (BGP4) has replaced EGP as the standard for inter-domain routing on the Internet. BGP makes use of the Transport Communication Protocol (TCP) to ensure reliable communication between peers. It uses a path-vector protocol for calculating loop-free paths between other BGP peers. There are two types of BGP peers, internal peering where internal BGP (iBGP) is used to share information within an AS about direct connections. Additionally, there are external peers that apply external BGP (eBGP) for sharing reachability information with other peers residing in other domains [10].

The information shared among BGP peers is used to construct a local graph of AS paths of the Internet represented in the form of routing table entries. These entries contain the prefixes announced by remote domains and the AS paths to traverse to reach those domains. An AS path is defined as an ordered set of domains that on traversal leads to the specified set of origin networks (first AS in the path) that own the prefix. The routing table may include multiple possible routes to the origin networks, however, only the optimal one is broadcasted to BGP peers [10], [11].

BGP utilizes a finite-state machine (FSM) model for determining its behavior and keeping track of its BGP peers' status. The model consists of six states each having specific conditions for when to transition to the next state:

- Idle
- Connect
- Active
- OpenSent
- OpenConfirm
- Established

In addition to this, BGP has four different messages for communicating with its peers. These messages are:

- OPEN
- KEEPALIVE
- UPDATE
- NOTIFICATION

The OPEN message establishes a BGP peering and contains operational parameters such as its ASN, BGP identifier, and a Hold Time for ensuring that a peer is up and running correctly. If the Hold Time reaches zero the connection is torn down and routes going through the peer are removed from the routing table. Furthermore, even thus BGP uses TCP for communication, it does not rely on it for ensuring that neighbors are still up. Instead, the KEEPALIVE message is sent out every one-third of the agreed-upon Hold Time. For exchanging reachability information, the UPDATE message is used and acts as an alternative KEEPALIVE message to reduce traffic. Lastly, the NOTIFICATION message is for distributing error messages and always leads to the connection being torn down [10].

2.3.1 BGP path selection

BGP utilizes a special algorithm to determine the optimal route for reaching each prefix in the BGP table. The chosen route is installed into the routing table and advertised to peers. The complete algorithm used for path selection is too extensive to cover in this section, however, some of the algorithm's checkpoints include the following:

- Verify that the next hop can be reached.
- Paths having the highest "local preference" are chosen first. Local preference is used in order to configure an AS for, example, selecting a certain exit point. This metric is not advertised to BGP peers.
- Prefer the route having the shortest AS-path, *i.e.* what set of domains that has to be traversed in order to get to the desired destination.
- Lower origin codes are rewarded, an origin code determines if the routes were advertised through IGP or EGP where the former has a lower value meaning it is preferred to keep the traffic inside the domain.
- Strictly internal routes are rewarded, this includes local, static, and direct routes.
- The first received path is preferred.

The steps listed above are a selection of all the criterion's used in the algorithm [12].

2.3.2 BGP Filtering

BGP is a vital part of the Internet's core infrastructure, yet back in the day BGP was not designed with security in mind. To ensure that the routes announced through BGP are neither accidentally nor maliciously malformed in an obvious way, a technique called BGP filtering is used. This technique makes sure that only what are considered as correct IP prefixes are accepted from peers by utilizing inbound and outbound filters that match on AS paths, prefixes, and other useful attributes such as community lists. Reliable information about network resources can be obtained from the Internet Routing Registry (IRR) that stores the configurations from different network operators.

Multiple attack cases made possible through BGP have been documented since the beginning of the Internet. Man-in-the-middle and denial of service (DoS) attacks are such examples [13]. If an AS is able to hijack certain important IP prefixes, it can cause severe damage to large parts of the Internet. In 2020 there were over 17 BGP hijacking incidents per day, targeting high profile incidents involving companies such as MasterCard, Amazon, Google, and national telecom operators [14], [15]. Moreover, in December of 2017 an unknown Russian AS hijacked 80 high-traffic prefixes usually used by companies such as Google, Apple, Facebook, Microsoft, TwitchTV, and Riot Games. The attack resulted in that user credentials including passwords were compromised [14], [15]. Attacks towards these entities are especially harmful since much of the Internet relies on cloud services offered by Amazon and Google [16]–[20]. The reason for attacks like these is that BGP is not yet capable of protecting itself against these types of attacks, it solely relies on trust between peers. The inbound and outbound filters mentioned above are not capable of protecting users from these types of hijacking attacks. To solve this, network administrators may utilize the Public Key Infrastructure (PKI) to authenticate that the prefixes and routes exchanged by BGP are valid. The simplest form of this authentication, the Resource Public Key Infrastructure (RPKI), validates whether an AS number claiming control over certain IP prefixes is authorized or not. This is known as origin validation and is specified in RFC 6811 [21]. In this implementation approach, a BGP speaker gets a dictionary from a remote distributed RPKI repository containing the valid prefixes owned by each ASN. For each BGP advertisement that the speaker receives then from its peers, it validates the origin making sure to filter out unverified advertisements with an incorrect origin. This technique does not require any changes to BGP to function properly, which makes it one of the more popular approaches for securing BGP [22].

2.4 Internet Exchange Points

To facilitate inter-domain routing, ASes can communicate over Internet eXchange Points (IXPs). IXPs offer a shared switching fabric where ASes can connect to exchange internet traffic with other ASes connected over the same IXP [23]. As of 2021, there exists over 400 IXP's worldwide where the largest players have peerings with 300-500 of the Internet marketplace's most prominent networks [24], [25]. Since many different ASes are interconnected at a single IXP, this results to multiple benefits for all participating networks such as increased fault tolerance, better bandwidth utilization, decreased latency for communication, and reduced costs leading to higher efficiency of inter-domain routing [2], [25]. Examples of some of Europe's largest IXPs that offer RS services are DE-CIX, AMS-IX, and Netnod [26]–[28]. In 2021, Netnod reported to have an average incoming internet traffic of 1.148 Tb/s, which accumulates to 12.4 PB per day [29], [30].

2.4.1 Route Servers

IXPs serve an important role in sharing reachability information between the ASes that are connected to the IXP infrastructure. By establishing a Route Server (RS) at the IXP, its members can replace their separate BGP peerings with a single BGP session towards the RS located at the IXP. This greatly simplifies inter-domain routing letting each AS peer with up to hundreds of different networks through a single BGP session. This decreases the burden on the network administrator as well as the border router itself [31], [32]. Moreover, the IXP's RS costs are often included in the membership fee required for peering to the IXP allowing member to use the RS service at no additional charge. However, there exists many networks that prefer to arrange private agreements with other networks located at the IXP, which means that these networks are not relying on routes announced by the RS unless as a backup. This means that, in general, the RS service offered by an IXP is highly desirable for many ASes since it offers a large set of available routes through a single BGP session instead of the AS having to maintain such a session for each network connected to the IXP. This results in a growing number of members at an IXP leading to a positive network effect.

An RS makes use of an External Gateway Protocol for coordinating reachability information. An example of such a protocol is BGP [23], [32], described in Section 2.3. Additionally, there exist different implementations of RSes where each implementation brings different functionality. For example, a Software-Defined Internet Exchange (SDX) utilizes Software-Defined Networking (SDN) to significantly increase the flexibility and functionality of inter-domain routing. SDN gives fine-grained control over packet-processing rules matching on multiple header fields. Furthermore, SDX enables application-specific peering that lets two networks peer for specific purposes like video streaming [33]. This requires new programming abstractions that allow participating networks to create and run these applications and a run-time that both behave correctly when interacting with BGP [34]. However, Siem Hermans et al. state in their work "*On the feasibility of converting AMS-IX to an Industrial-Scale Software-Defined Internet Exchange Point*" that today's SDX infrastructure does not achieve the scalability needed for the most demanding IXP's since the "*currently employed switch platform is incapable of supporting iSDX due to the lack of sufficient flow tables*". The paper concludes that although there has been improvements regarding the policy compression algorithm used by the iSDX, there still exists scalability constraints due to the capabilities of current hardware [35], [36]. Consequently, this work is only going to focus on traditional RSes. In addition to SDX, there are privacy-preserving RSes that aim to solve the privacy concerns that arise when ASes are required to disclose their routing policies to the IXP's RS. SIXPACK is an RS-service that uses Secure Multi-Party Computation (SMPC) to keep peering policies confidential [2].

2.5 Free Range Routing

There exist several open-source packages that are capable of running BGP such as GNU Zebra, Quagga, OpenBGPD, XORP, BIRD, and Vyatta [37]. The Free Range Routing (FRR) software suite used in this work is a fork of Quagga that is aimed to serve as an open-source alternative to the commercial implementations from companies such as Juniper and Cisco. FRR is supported for the Linux, OpenBSD, FreeBSD, NetBSD, and Solaris operating systems and is distributed under the GNU General Public License, version 2 (GPL2). Furthermore, FRR is part of the Linux

Foundation making it a suitable alternative for many of the worlds biggest and most influential companies, universities, and research labs in the field [38].

2.5.1 System Architecture

The FRR code-base is mostly written in the C programming language resulting in high performance and low overhead. One of the benefits of FRR compared to other routing software is the use of several daemons for handling different routing protocols. Instead of relying on the use of a single process containing all different protocols and additional functionality, FRR separates each routing protocol in its own process. For coordinating routing decisions each daemon communicates with the Zebra routing manager responsible for updating the kernel routing table, performing interface lookup, talk to the data-plane, and redistribute announced routes between the daemons. This type of architecture gives high fault tolerance since a crash in one of the daemons does not affect the other daemons resulting in high reliability which is highly valuable in the type of environments where FRR is mostly used. In addition to this, the modular design choice facilitates the implementation of new protocols as well as making it possible to re-configure one or more protocols without having to stop other parts of the routing suite.

The configuration of the daemons and the routing protocols is done in the *daemons* and *frr.conf* files residing in /etc/frr/ on Linux, examples of configurations for *frr.conf* are available in Appendix A.2-A.4. The interaction with the daemons is done through the virtual teletype integrated user interface shell (Vtysh) which is part of the FRR software suite [38]. Lastly, FRR utilizes a PTrie data structure to store the prefix filters to ensure fast retrievals. By using a Trie, FRR can search the key in $O(M)$ time where M corresponds to maximum string length. The PTrie data strucure used in FRR is developed by Jensen et.al. and is further explained in their paper "*PTrie: Data Structure for Compressing and Storing Sets via Prefix Sharing*". The PTrie data structure distinguishes itself by compressing the stored elements while sharing the desirable key characteristics with conventional hash-based implementations, namely fast insertion and lookup operations [39].

2.6 Related work

Since IXPs, RSes, BGP, and inter-domain routing are critical components of the Internet's infrastructure, there has been extensive research performed in this area where the focus is on improving performance, scalability, and security. For example, Gupta et al. have in their study "*SDX: A Software Defined Internet Exchange*" researched how SDN could improve wide-area traffic delivery with direct control over packet-processing. The authors also envision how BGP could evolve to support more advanced routing policies beyond destination prefixes and single next-hop [34]. Furthermore, it has been shown that SDXs generally do not scale well and are struggling to meet the demands of the world's largest IXP's. Therefore, Gupta et al. have in a later study developed an industrial-scale SDX (iSDX) that aims to be able to operate at the scale of the largest IXPs by addressing the fundamental scalability challenges of traditional SDX design. This is partly achieved by re-designing the data plane that exploits the fact that each participant expresses its SDN policy independently, which implies that each participant can also compile its SDN policies independently [35]. However, Hermans et al. reveals in their work "*On the feasibility of converting AMS-IX to an Industrial-Scale Software-Defined Internet Exchange Point*" that the currently employed switch platform is incapable of supporting iSDX due to the lack of sufficient flow tables, thus making it insufficient to use in large scale IXPs [36]. Additionally, Taylor et al. have developed an enhanced and unique test framework for RS benchmarking. The framework makes it possible to create a realistic one-to-one copy of a live IXP network in order to estimate RS capacity limits and test new features or configurations in large-scale deployments. Potential measurements include the number of announced prefixes in a given time frame as well as CPU and memory utilization [40].

Chapter 3

Methodology

In this chapter, we explain the different parts of the theoretical method used to understand the performance limits of FRR. Specifically, we begin with Section 3.1 that presents the metrics used to evaluate the performance of FRR. Section 3.2 continues by describing the benchmarking environment including the configuration of the network and FRR. Finally, Section 3.3 explains the structure of our benchmarking experiments along with how the data gathered is processed.

3.1 Benchmarking metrics

To analyze the performance of FRR, we use the following metrics. First, we measure the rate at which FRR announces prefixes to its BGP peers. This is referred to as the number of *announced prefixes*. Second, by utilizing Vtysh, we obtain the number of prefixes that has been received by FRR and installed in the routing table of each peer. We refer to this metric as the number of *installed prefixes*. Since prefixes are first received and installed into the corresponding routing table at the RS before being announced to the peers, we compare those two metrics in order to quantify how FRR performs in regards to the announced prefixes to its peers at a network level during the given time span. For example, if the number of announced prefixes is considerably less than the number of installed prefixes, it could potentially reveal that FRR suffers from a performance bottleneck.

3.2 Benchmarking environment

The chosen benchmark environment for performing the experiments is illustrated in Figure 3.1 and consists of two servers, named *S1* and *S2*. The first one, *S1*, is configured as an RS and runs the FRR software suite including the BGP daemon that we benchmark. The second one, *S2*, emulates a set of virtual BGP peers that are to announce prefixes towards the RS running at *S1*. We use multiple servers to achieve a more realistic environment and to make sure that the BGP peers are not consuming computing resources needed to run FRR at its full capacity. Furthermore, the hardware of *S1* consists of two 8-core Intel Xeon E5-2667 running at 3.2 GHz respectively along with 128 GB of RAM. *S2*, on the other hand, is equipped with two 12-core Intel Xeon CPU E5-2690 CPUs clocked at 2.6 GHz as well as 256 GB of RAM. In addition to this, the two servers are interconnected with a 100 Gigabit Ethernet (100GbE) interface through an OpenFlow switch. Lastly, both servers run the Ubuntu 18.04.5 LTS Linux distribution for their operating system.

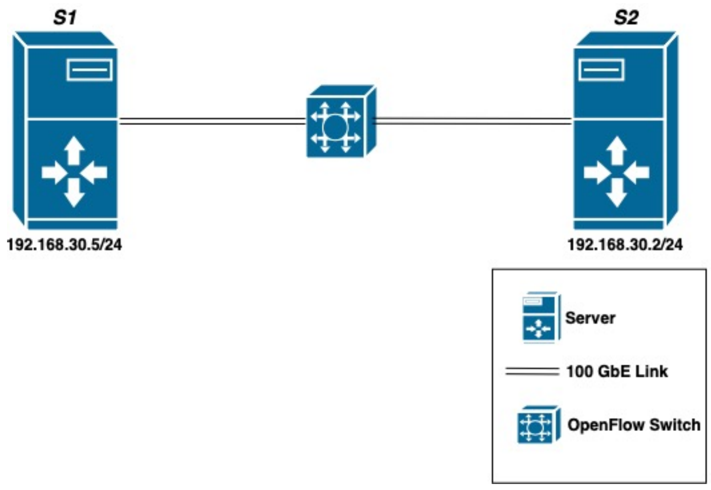

Figure 3.1: Benchmarking environment.

3.2.1 BGP peers

To establish a set of BGP peers that are to announce routes towards the FRR instance, running at *S1*, the route announcer ExaBGP is used [41]. ExaBGP is written in the Python programming language and provides a convenient way to interact with emulated BGP peers by transforming BGP messages into friendly plain text or JSON. By using ExaBGP, we are able to establish many different BGP peers without the need to run multiple instances of FRR since ExaBGP handles all BGP communication with its configured peers [41]. Figure 3.2 illustrates each virtual BGP peer, made up of a single process ExaBGP instance, running within *S2*. Each virtual BGP peer connects to a virtual Open vSwitch (OVS), named V_1, with an IP address of 10.0.0.200/24 which in turn is connected to a network interface on *S2*. Additionally, each BGP peer is given a unique IP address on the 10.0.0.0/24 subnet, which lets them communicate with *S2* as well as *S1* as seen in Figure 3.1.

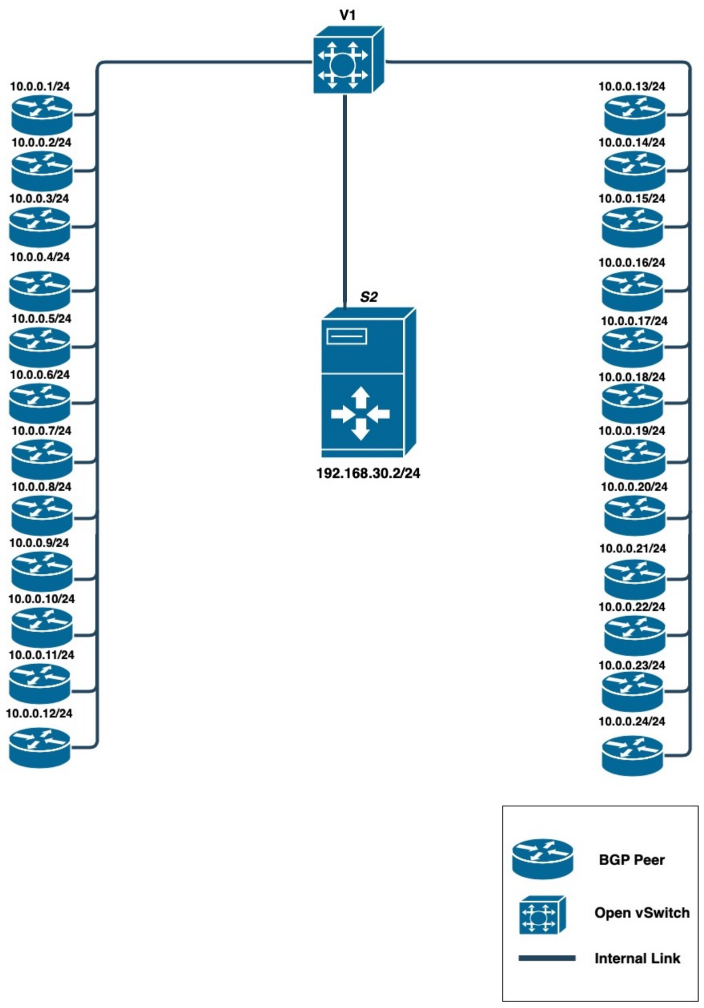

Figure 3.2: The complete set of virtual BGP peers running within *S2* for the purpose of documenting the load. Each peer possesses a different ASN and establishes a BGP peering session with the RS situated in *S1*.

Due to the single-threaded nature of Python, 24 ExaBGP instances are established on *S2*. Each ExaBGP process runs on its own core which minimizes the overhead that comes with interrupts and swapping CPU contexts [42], [43]. As seen in Figure 3.2, every peer $P_n \in \mathrm{P} = \{P_1, \ldots, P_{24}\}$ is given an IP address on the subnet 10.0.0.n/24, but since every virtual peer operates under the same server, *S2*, we use namespaces to separate them, which makes it possible to assign the peers different and separate instances of network interfaces as well as routing tables that operate independently from each other [44]–[46]. The process for configuring these BGP peers, together with the necessary configuration files, is shown in Appendix A.1.

3.2.2 Prefix announcements

To achieve a realistic environment, each peer is configured to announce distinct IP addresses by letting the forth octet of the IPv4 address correspond to the index of the peer announcing the route. An arbitrary address announced by a peer would therefore take the format of $a.b.b.n/32$ where $a \in [1, 223]$, $b \in [0, 255]$, $n \in [1, 24]$, and $a,\ b,\ n \in \mathbb{Z}$. For example, the first prefix announced by P_1 is 1.0.0.1/32 while the second is 1.0.1.1/32. The first prefix announced by P_2 is 1.0.0.2/32 while the second is 1.0.1.2/32. However, it should be noted here that /32 prefixes do not commonly propagate the Internet, yet by utilizing more of the IPv4 range we are able to announce a total of 350,748,672 distinct prefixes and run the benchmark for a longer period of time before the peers start to repeat their already announced prefixes.

3.2.3 Route Server configuration

The FRR instance running at *S1* takes the role of an RS responsible for receiving, processing, and announcing routes to all of its peers. We configure the BGP daemon running at the RS to apply certain prefix filters for each peer. These filters decide which received prefixes to advertise back to a specific peer. Figure 3.3 shows the announcement processing model of these filters. There are two types of prefix filters used by the RS, the Inbound filter (In-filter) and the Outbound filter (Out-filter). The In-filter controls which received prefixes that are allowed to participate in the best path selection process of BGP. The resulting best paths are inserted into the local routing information base (LoC-RIB) of the RS of each peer. The next filter, the Out-filter, checks which prefixes in the LoC-RIB that are permitted to be announced to each peer. These filters can reject the announcement, accept it unmodified, or accept it with some of its attributes modified.

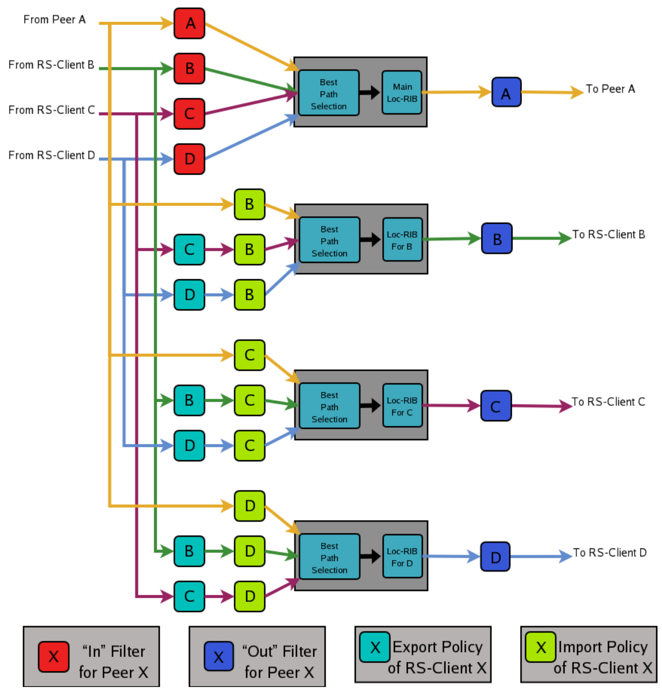

Figure 3.3: Announcement processing model implemented by the RS [47].

It follows a description of each prefix filter configuration that is going to be used in the benchmark. In this work, we create and apply prefix filters to a peer by configuring the `frr.conf` file accordingly based on the below syntax:

```
neighbor <peer-ip> prefix-list <prefix-list-name> in/out
```

```
ip prefix-list <prefix-list-name> permit / deny <network/mask>
```

A. First configuration: Allow everything

The purpose of our first configuration is to establish a base case where no filtering of prefixes is applied. Later in Chapter 4, we compare the benchmarks of this configuration with benchmarks where the RS is configured to filter certain prefixes. In this configuration, we neither apply In-filters nor Out-filters. By passing the argument "permit any" to the In-bound and Out-bound prefix-lists, every prefix reaching the RS is allowed through and announced to each BGP peer. For a complete version of this configuration, we refer our readers to the Appendix A.2.

B. Second configuration: Outbound filtering

With our second configuration of the RS, we seek to configure the Out-filters of the RS to enable outbound filtering of prefixes and hence measure how our benchmark metrics differ from the first configuration. Here, we allow all prefixes to pass through the In-filter. The Out-filter is configured to reject a set of routes that are not announced by any of the BGP peers. This is achieved by passing the argument "deny", followed by the IP prefix to be rejected, to the Out-bound filter. This means that the peers always announces valid prefixes to the RS resulting in the same number of announced prefixes as in case A regardless of the RS filters. However, FRR are still required to perform filtering operations to verify that every prefix is valid. The set of prefixes that are included in the Out-filter is $a.b.b.25/32$ where $\{a \in \mathbb{Z} \,|\, 0 < a < 6\}$ and $\{b \in \mathbb{Z} \,|\, 0 < b < 11\}$. This results in that a total of 500 prefixes are included in the Out-filter. Since there are no BGP peers announcing prefixes with *.25* as the forth octet, the Out-filter permits every route it encounters. The complete configuration file is available in the Appendix A.3.

C. Third configuration: Outbound filtering of announced prefixes

With our third configuration of the RS, we examine how the benchmark metrics differ when the RS is actually configured to filter prefixes announced by its peers. This is achieved by letting the Out-filter reject the set of 500 prefixes $a.b.c.d/32$ where $\{a \in \mathbb{Z} \,|\, 0 < a < 5\}$, $\{b \in \mathbb{Z} \,|\, 0 < b < 6\}$, $\{c \in \mathbb{Z} \,|\, 0 < c < 6\}$, and $\{d \in \mathbb{Z} \,|\, 0 < d < 26\}$. The complete configuration file is available in the Appendix A.4.

3.2.4 eBGP multihop

Since the BGP peers are not directly connected to the RS, each peer is configured with *ebgp-multihop* enabled. As can be seen in Figure 3.2, for the BGP peers to reach the RS located at *S1*, they must first go through a routing table at *S2* resulting in the standard TTL of 1 not been sufficient. By enabling *ebgp-multihop*, FRR allows eBGP neighbors to establish BGP sessions even if they are multiple L3 hops away [38].

3.2.5 Network analyzer

To capture the number of announced prefixes traversing the wire between *S1* and *S2*, as defined in Section 3.1, we use the terminal-based version of the Wireshark network protocol analyser named tshark. To get the number of prefixes that the RS announces back to its peers, tshark is configured to capture traffic on TCP port 179 originating from the 100 GbE interface on *S1*. When the network traffic has been captured, we analyze the resulting PCAP file with a BGP Parser developed by DE-CIX, named pbgpp [48], to obtain an easy to read format of the captured network traffic.

The parser offers multiple modes to convert BGP messages to a human readable format. We use two readable format modes: first, the *HUMAN_READABLE* mode for debugging showing information such as MAC and IP addresses, timestamps, message type, and path attributes including the announced prefixes. Second, the *LINE* mode for obtaining a file containing the announced prefixes from all of the captured BGP UPDATE messages [48]. To get the number of prefixes in the file parsed using the *LINE* mode, we search it for how many number of times the network mask (/32) occurs, where one occurrence corresponds to one prefix.

3.2.6 Vtysh parser

The second performance metric are the installed prefixes and has been defined in Section 3.1 as the number of installed prefixes obtained by querying Vtysh included in the FRR software suite.

Unlike tshark that is capturing network traffic, Vtysh allows direct interaction with the FRR daemons in a single combined session, providing useful metrics about the current state of the BGP daemon. Examples of such useful metrics are the number of prefixes that the RS has received as well as what prefixes that has been installed in the LoC-RIB of each peer at the RS. Vtysh is capable of exporting this data in JSON format allowing for further processing using a Python script shown in Appendix B.1 and B.2. As partly explained in Section 3.1, the reason for using Vtysh, in conjunction with tshark, is to distinguish the number of prefixes our RS has received and installed in its routing tables but has not yet send out to the wire. The method described in Section 3.2.5 and 3.2.6 is automated using a bash script shown in Appendix A.5.

3.3 Benchmarking methodology

To achieve a realistic benchmarking methodology, we let our BGP peers announce prefixes towards the RS during a 180 second time-span, this is referred to as one *benchmarking round*. Since scientific accuracy is of uttermost importance, an average of ten benchmarking rounds is calculated to counteract any potential unexpected or unknown parameters that, momentarily, could impact the performance of FRR. The average is calculated with a Python script shown in Appendix B.3. Moreover, by performing the benchmark with a wide variety of connected BGP peers, we obtain data showing how FRR performs under different loads and configurations which could potentially expose a bottleneck or threshold where FRR start performing worse. The chosen numbers of BGP peers with which we perform benchmarks are 4, 8, 12, 16, 20, and 24. Additionally, we monitor the outgoing network traffic at the RS during the entirety of each benchmarking round. Last but not least, we query Vtysh for data only once, after the 180 seconds has elapsed. This is because we want to avoid introducing any additional load on FRR, something that could potentially impact the benchmarking in a negative way.

3.3.1 Received prefixes

As part of our methodology is based on measuring the number of prefixes that the RS announces in a given time span, we need to take into consideration the rate at which our BGP peers announce prefixes towards the RS. For example, if the rate at which the RS receives prefixes from its peers fluctuates in-between benchmarking rounds, it may affect the RS's rate of announcing prefixes. This fluctuation could mislead our benchmarks to show the performance of the BGP peers rather than the one we desire; the performance of the RS. To avoid this, we measure the rate at which prefixes arrive at the RS by capturing network traffic originating from S1 as well as querying Vtysh for received prefixes using the same time span and number of rounds as when benchmarking the RS. Moreover, apart from giving us information about the stability of our BGP peers, we can use this data to determine how the RS is performing. Specifically, if we notice a difference in the number of received and announced prefixes by the RS, it could potentially show that it is struggling to keep up with the rate at which it receives prefixes. Likewise, by comparing the fluctuations of received prefixes with the RS benchmark, we obtain another dimension of data to analyze.

3.3.2 Preprocessing of data

Since the RS is announcing the received prefixes back to all of its BGP peers, multiple duplicates of every prefix are captured on the network by tshark. However, the data given by Vtysh only includes unique prefixes, this results in that we are not able to compare the number of announced prefixes with the number of installed prefixes. To compensate for this, we multiply the data obtained from Vtysh by the number of BGP peers used in the corresponding benchmarking round. This lets us accurately compare the two measurements while preserving the real rate of which the RS is capable of announcing prefixes. Finally, it should be noted that we only perform this preprocessing of data when working with data originating from the RS, meaning that no preprocessing is made for determining the base case in section 3.3.1 that measures the number prefixes that the RS has received from its peers. The preprocessing is automated using a Python script shown in Appendix B.5.

3.3.3 Network bandwidth

To make sure that the network between *S1* and *S2* is not being bottlenecked by our benchmark, we perform a simple iPerf test between the two servers. Additionally, we monitor the average bits per second as reported by tshark.

Chapter 4

Results

This chapter presents the results of the performed benchmarks. Section 4.1 starts by presenting the data gathered in our experiments. We continue with Section 4.2, which visualizes the differences in the results between our filter configurations as well as how the number of announced prefixes differ from the number of received and installed prefixes.

4.1 Tables and data

This section documents the results of the performed benchmarks. First, we present the data from our tests of the network bandwidth between our two servers *S1* and *S2*. We continue with Section 4.1.2 where we present benchmarks of received prefixes, which documents the rate at which our BGP peers are capable of announcing prefixes towards the RS. Next, in Section 4.1.3 and 4.1.4, we present the recorded number of installed and announced prefixes for our first, second, and third filter configuration defined in Section 3.2.3. Finally, Section 4.1.5 presents the average number of received, installed, and announced prefixes as well as how the data fluctuates.

4.1.1 Utilized network bandwidth

The PCAP files, created by the network analyzer in our performed benchmarks, reported that the network between our two servers, *S1* and *S2*, on average utilized a maximum network bandwidth of 27 Mb/s [49]. In addition to this, from the iPerf test that measured the available network bandwidth between our two servers, *S1* and *S2*, we know that an optimized process such as iPerf is only capable of utilizing 22 Gb/s out of the 100 Gb/s that is available. What this means is that since iPerf is written in the C programming language, which is faster than Python that ExaBGP is implemented in [50], [51], the network between the two servers is more than sufficient to handle our benchmarks without being bottlenecked.

4.1.2 Received prefixes

In this section, we present data from our benchmark experiment regarding the number of prefixes our RS has received from its peers within 180 seconds. Two tables are used, first, Table 4.1 shows the number of received prefixes when using Vtysh as our benchmarking tool. Second, Table 4.2 deals with the data recorded when capturing received prefixes on the wire. All PCAP files containing the captured network traffic are available in [49]. The first column of the two tables specifies which benchmarking round the number of prefixes refers to. Here, a benchmarking round is defined as the number of prefixes the RS has received after 180 seconds of prefix announcing by its peers. Moreover, the first row of Table 4.1-4.2 specifies the number of peers used in the benchmark. The data presented in the tables has not been preprocessed, meaning that the data presented is equivalent to the data obtained from Vtysh. When examining the tables, we notice that the number of prefixes are slightly higher in Table 4.1 when compared to Table 4.2. The data from these tables are later used in Section 4.1.5 as well as Section 4.2.

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	1,686,005	2,685,074	3,483,607	3,937,874	4,312,962	4,492,011
2	1,680,925	2,792,339	3,517,421	4,040,698	4,295,218	4,398,821
3	1,670,062	2,880,580	3,485,413	4,075,761	4,355,882	4,478,622
4	1,575,140	2,796,262	3,467,030	3,978,662	4,337,659	4,495,173
5	1,661,779	2,523,688	3,453,320	3,954,223	4,319,648	4,479,468
6	1,674,588	2,790,158	3,582,440	3,964,204	4,037,483	4,496,862
7	1,644,145	2,744,608	3,510,301	4,061,132	4,141,536	4,507,699
8	1,660,756	2,751,049	3,461,215	4,033,728	4,269,746	4,513,442
9	1,635,249	2,709,729	3,509,757	4,018,933	4,322,443	4,466,575
10	1,686,025	2,675,834	3,476,856	4,015,721	4,430,809	4,501,039

Table 4.1: Prefixes received in 180 seconds as reported by Vtysh

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	1,562,472	2,470,464	3,179,076	3,570,602	3,953,775	4,315,693
2	1,545,729	2,574,661	3,178,659	3,703,350	3,979,248	4,263,773
3	1,549,235	2,537,118	3,198,488	3,717,681	3,941,419	4,295,607
4	1,467,065	2,529,772	3,152,519	3,618,640	4,054,344	4,309,473
5	1,541,560	2,320,826	3,165,560	3,625,680	3,965,909	4,296,778
6	1,545,743	2,572,409	3,245,058	3,565,940	3,754,717	4,314,956
7	1,525,997	2,523,803	3,185,917	3,709,300	3,799,224	4,317,742
8	1,532,413	2,496,434	3,160,314	3,699,560	3,926,528	4,319,657
9	1,519,882	2,493,288	3,157,165	3,661,118	3,990,962	4,289,569
10	1,565,124	2,438,068	3,178,900	3,671,660	3,971,086	4,315,308

Table 4.2: Prefixes received in 180 seconds as reported by captured network traffic

4.1.3 Installed prefixes

Here, we present data showing the number of installed prefixes for each filter configuration. Table 4.3 shows the number of installed prefixes with the first filter configuration applied, defined in Section 3.2.3, which allows all received prefixes to pass through the Out-filter at the RS. Likewise, Table 4.4-4.5 presents the data gathered when applying the second and third filter configuration. As defined in Section 3.2.3, our second filter configuration rejects prefixes that are not announced by any of the peers while the third configuration rejects some of the prefixes that actually are announced by the peers. Tables 4.3-4.5 are structured as follows; the first row specifies the number of peers used in the corresponding benchmarking round specified in the first column of the table. A benchmarking round is here defined as the number of prefixes the RS has installed in its routing table after 180 seconds of prefix announcing by the peers. The remaining columns holds the number of prefixes the RS has received in a specific benchmarking round. The data presented in the tables has been preprocessed as described in Section 3.3.2, meaning that the data presented has been multiplied by the number of peers used in the benchmark. This enables us to directly compare this data with the number of announced prefixes presented in Section 4.1.4.

Moreover, when examining the data present in Table 4.3-4.5, it can be seen that the number of installed prefixes in a specific benchmarking round increases when applying filtering of prefixes. Specifically, Table 4.4 shows more prefixes than Table 4.3 while Table 4.5 shows more than Table 4.4. The differences in the number of installed prefixes between the three filter configurations are later visualized in Section 4.2.1.

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	5,901,804	19,360,560	34,213,632	54,798,960	76,474,040	97,031,904
2	6,020,868	19,400,464	34,947,564	54,711,952	73,254,900	96,822,048
3	5,973,676	19,032,840	36,839,652	51,086,336	77,307,660	94,557,432
4	5,448,372	19,015,880	35,565,600	51,874,656	76,795,960	97,073,784
5	5,577,016	19,142,784	35,351,340	49,637,296	76,965,040	96,995,520
6	5,827,556	19,524,408	35,126,304	54,891,184	70,784,660	96,950,856
7	5,949,460	18,798,440	35,235,276	57,667,840	76,647,260	96,951,408
8	5,757,092	18,536,664	35,154,492	52,611,360	76,920,140	96,699,648
9	5,869,544	19,556,896	34,347,144	53,134,928	76,412,080	96,769,344
10	5,959,460	19,567,768	36,803,628	53,913,600	72,750,520	92,721,072

Table 4.3: Number of installed prefixes after 180 seconds with first filter configuration

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	5,630,988	18,935,640	37,594,428	57,161,040	75,656,400	96,478,632
2	5,984,676	19,491,400	37,714,272	56,607,648	76,285,740	97,130,256
3	5,751,656	19,648,784	37,171,584	57,138,224	77,264,420	96,975,984
4	5,875,244	19,083,408	37,384,164	57,886,912	77,294,180	96,276,264
5	5,800,368	19,434,552	35,271,900	56,493,248	76,535,880	96,381,576
6	5,988,788	19,654,104	37,490,184	57,851,072	77,103,080	96,532,080
7	5,528,284	17,357,000	37,576,092	57,056,336	76,836,740	96,403,176
8	5,795,748	19,577,320	37,333,512	57,218,816	76,637,340	97,232,664
9	5,594,160	18,606,912	37,613,208	57,458,688	77,349,840	95,297,328
10	5,697,332	19,821,688	36,596,244	57,531,536	76,363,560	95,733,408

Table 4.4: Number of installed prefixes after 180 seconds with second filter configuration

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	6,219,156	17,177,336	39,942,972	59,633,696	82,050,760	102,612,600
2	5,136,320	19,734,512	41,801,724	60,873,488	79,925,600	101,863,032
3	6,287,968	20,637,184	38,838,456	59,336,736	80,168,260	102,834,240
4	6,184,684	19,877,272	41,527,740	58,692,096	81,231,640	103,202,904
5	6,207,644	20,781,872	41,759,028	57,796,512	81,204,540	103,480,512
6	6,235,992	19,590,840	42,149,400	64,964,944	80,772,720	100,657,176
7	6,252,784	20,208,504	39,288,216	63,726,480	81,010,960	101,205,528
8	5,988,840	20,563,176	41,488,092	60,662,240	82,281,540	100,491,264
9	5,923,712	20,225,984	40,143,900	61,354,656	79,667,700	100,661,736
10	6,199,752	21,169,544	38,966,604	61,075,792	79,468,880	103,036,224

Table 4.5: Number of installed prefixes after 180 seconds with third filter configuration

4.1.4 Announced prefixes

In this section, we present data showing the number of announced prefixes for each filter configuration. All PCAP files containing the captured network traffic are available in [49]. Table 4.6 illustrates our benchmark experiment regarding the number of prefixes the RS has announced with the first filter configuration applied, defined in Section 3.2.3, this configuration allows all received prefixes to be announced back to RS's peers. Likewise, Tables 4.7-4.8 present data gathered when applying the second and third filter configuration. As defined in Section 3.2.3, our second filter configuration rejects prefixes that are not announced by any of the peers while the third configuration rejects prefixes that actually are announced by the peers. Table 4.6-4.8 are structured as follows; the first row specifies the number of peers used in the corresponding benchmarking round that is specified in the first column of the table. A benchmarking round is here defined as the number of prefixes that the RS has announced on the wire after 180 seconds of prefix announcing by the peers. The remaining columns hold the number of prefixes the RS has announced in a specific benchmarking round. The data presented in the tables has not been preprocessed, meaning that the data presented is equivalent to the number of prefixes captured by the network analyzer.

When examining the data in Table 4.6-4.8, it can be seen that although the number of prefixes increases from Table 4.6 to Table 4.8, the trend does not continue as we compare Table 4.7 and Table 4.8. This means that there is a discrepancy when comparing these results with those presented in section 4.1.3. Additionally, when comparing these results with those of the previous section, it can be seen that the differences in-between benchmarking rounds are considerably greater than in Section 4.1.3. The difference in the number of announced prefixes is further examined in Section 4.1.5.

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	20,552	204,636	489,279	2,210,042	6,357,673	15,009,400
2	14,564	217,139	676,577	2,125,387	6,716,304	13,445,970
3	14,480	139,767	525,942	2,281,784	6,845,945	13,261,194
4	26,953	282,243	537,506	2,037,925	6,402,018	14,637,308
5	16,476	174,449	642,035	1,915,071	7,191,992	14,520,070
6	17,190	162,053	538,652	2,079,356	7,243,258	14,825,627
7	21,520	165,033	865,353	2,143,054	7,362,352	13,883,276
8	27,789	224,178	968,523	2,437,251	6,421,320	13,731,680
9	42,755	307,853	1,400,629	2,162,811	6,371,791	14,194,059
10	55,739	230,929	771,626	2,098,896	6,745,281	14,148,780

Table 4.6: Prefixes announced in 180 seconds with first filter configuration

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	10,304	582,366	1,010,734	4,933,370	7,474,857	15,399,010
2	27,934	1,151,552	3,113,119	6,109,504	7,752,584	13,541,130
3	25,085	398,754	3,243,862	3,315,030	7,771,180	13,152,813
4	59,916	227,019	1,562,839	3,417,871	6,658,452	14,716,664
5	32,875	659,562	2,921,677	3,889,478	6,863,172	14,850,583
6	32,522	184,127	1,056,643	3,338,421	6,768,061	15,360,370
7	27,383	417,085	4,064,204	3,613,570	7,788,926	18,845,915
8	14,238	204,463	903,537	3,190,332	8,197,863	16,370,122
9	62,910	719,228	2,596,477	4,369,462	7,603,319	17,342,969
10	52,858	453,048	1,255,726	3,338,027	8,006,477	16,834,501

Table 4.7: Prefixes announced in 180 seconds with second filter configuration

Round	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
1	23,465	258,637	658,094	2,416,181	6,951,442	15,207,703
2	306,487	485,531	725,823	1,625,529	7,384,440	14,106,294
3	14,146	166,884	782,729	3,201,285	6,718,481	13,979,150
4	15,384	459,311	556,645	3,048,046	7,677,886	15,102,952
5	29,853	456,389	680,304	3,248,277	7,131,864	14,227,153
6	14,913	405,476	769,064	3,298,110	6,693,493	13,201,754
7	26,636	323,653	841,609	3,549,418	7,575,584	13,945,621
8	81,374	132,448	780,497	2,360,920	6,677,934	15,835,574
9	100,802	292,666	1,126,662	2,382,295	7,953,766	13,840,586
10	16,156	227,628	809,356	2,428,576	8,017,979	14,770,893

Table 4.8: Prefixes announced in 180 seconds with third filter configuration

4.1.5 Averages and fluctuations

This section focuses on showing the average number of received, installed, and announced prefixes for each filter configuration and set of peers. We also present the fluctuations in-between all of the recorded benchmarking rounds. Table 4.9-4.11 shows the average number of received, installed, and announced prefixes. The first column of Table 4.9 specifies if the average was recorded by using Vtysh or by capturing network traffic. In contrast to Table 4.9, the first column of Table 4.10-4.11 specifies which of our three filter configurations the average refers to. As defined in Section 3.2.3, the first filter configuration is set to not apply any filtering of prefixes the RS has received from its peers. Our second filter configuration, on the contrary, rejects prefixes that are not announced by any of the peers while the third configuration rejects prefixes that actually are announced by the peers. The first row of Table 4.9-4.11 specifies the number of peers used in the benchmarking round. A benchmarking round is here defined as the number of prefixes that the RS has either received, installed, or announced within 180 seconds. Table 4.9 presents the average number of prefixes that the RS has received from its peers. Moreover, the data presented in Table 4.9-4.10 has been preprocessed as described in Section 3.3.2, meaning that the data presented has been multiplied by the number of peers used in the benchmark. This lets us directly compare the data in Table 4.9-4.10 with the non preprocessed data in Table 4.11. When comparing the data in Table 4.9 with the average number of installed prefixes in Table 4.10 as well as the average number of announced prefixes in Table 4.11, it can be seen that the number of received prefixes is higher than both the number of installed and announced prefixes. Additionally, all data presented in Table 4.9 and Table 4.10 are mostly the same when compared to the data in Table 4.11, meaning that the average number of announced prefixes is considerably less than what has been received and installed at the RS. This discrepancy is noteworthy since it shows that our RS is not capable of announcing prefixes at the same pace as it receives them from its peers.

Metric	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
Vtysh	6,629,869.6	21,879,456.8	41,936,832	64,129,497.6	85,646,772	107,591,308.8
Network	6,142,087.5	19,965,473.5	38,161,986.8	58,469,651.7	85,646,772	103,292,531.8

Table 4.9: Average number of received prefixes after 180 seconds

Configuration	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
First	5,828,484.8	19,193,670.4	35,358,463.2	53,432,811.2	75,431,226	96,257,301.6
Second	5,764,724.4	19,161,080.8	37,174,558.8	57,240,352	76,732,718	96,444,136.8
Third	6,063,685.2	19,996,622.4	40,590,613.2	60,811,664	80,778,260	102,004,521.6

Table 4.10: Average number of installed prefixes after 180 seconds

Configuration	**4 peers**	**8 peers**	**12 peers**	**16 peers**	**20 peers**	**24 peers**
First	25,801.8	210,828	741,612.2	2,149,157.7	6,765,793.4	14,165,736.4
Second	34,602.5	499,720.4	2,172,881.8	3,951,506.5	7,488,489.1	15,641,407.7
Third	62,921.6	320,862.3	773,078.3	2,755,863.7	7,278,286.9	14,421,768

Table 4.11: Average number of announced prefixes after 180 seconds

Now we show the level of fluctuation for the number of received, installed, and announced prefixes. The fluctuation is calculated by using a Python script shown in Appendix B.4. We define fluctuation as the percentage difference between the benchmarking round with the most and least number of recorded prefixes for each filter configuration and set of peers. For example, when examining the first column of Table 4.1, holding the data from our benchmark of received prefixes using Vtysh, we see that the benchmarking round with the least amount of prefixes is round four with 1,575,140 prefixes while round ten recorded the most with 1,686,025 prefixes, resulting in a fluctuation of 7.039%. These numbers, together with the corresponding numbers for all of the other benchmarking rounds, are presented in Table 4.12-4.14. Specifically, Table 4.12 deals with the fluctuation of received prefixes while Tables 4.13-4.14 show how the number of installed and announced prefixes fluctuates in-between benchmarking rounds. The first row of Tables 4.12-4.14 show the number of peers used in the benchmark. The first column of Table 4.12 is structured as follows; it first presents the values used for calculating the fluctuation, called *min* and *max* and are preceded by the corresponding metric used to record the number of prefixes which is either Vtysh or captured announcements at the network-level (*i.e.*, Network). However, for Tables 4.13-4.14, the min and max are instead preceded by the corresponding filter configuration used in the benchmark. Below the min and max values, the table presents the actual fluctuation these two values results in and is preceded by either benchmarking metric, as in Table 4.12, or filter configuration as in Table 4.13-4.14.

Furthermore, when examining the fluctuations in Table 4.12-4.14, we notice that our benchmarks of received prefixes resulted in the lowest fluctuation rate. Announced prefixes, on the contrary, shows a considerably higher rate of fluctuation when compared to both received and installed prefixes. The fluctuation of announced prefixes is at the most 345 times higher when comparing it with the fluctuation of received prefixes with the same number of peers used in the benchmark. The considerably higher fluctuation of announced prefixes further strengthens the results of the average number of prefixes shown in Table 4.9-4.11 that showed that our RS seems to not be able to announce prefixes at the same rate as it receives these prefixes from its peers.

Metric	4 peers	8 peers	12 peers	16 peers	20 peers	24 peers
Vtysh max	1,686,025	2,880,580	3,585,440	4,075,761	4,430,809	4,513,442
Vtysh min	1,575,140	2,523,688	3,453,320	3,937,874	4,037,483	4,398,821
Vtysh fluctuation	7.039%	14.142%	3.739%	3.502%	9.742%	2.606%
Network max	1,565,124	2,574,661	3,245,058	3,717,681	4,054,344	4,319,657
Network min	1,467,065	2,320,826	3,152,519	3,565,940	3,754,717	4,263,773
Network fluctuation	6.684%	10.937%	2.935%	4.255%	7.980%	1.311%

Table 4.12: Fluctuation of received prefixes

Configuration	4 peers	8 peers	12 peers	16 peers	20 peers	24 peers
First max	1,505,217	2,445,971	3,069,971	3,604,240	3,865,383	4,044,741
First min	1,362,093	2,317,083	2,851,136	3,102,331	3,539,233	3,863,378
First fluctuation	10.508%	5.563%	7.675%	16.178%	9.215%	4.694%
Second max	1,497,197	2,477,711	3,142,856	3,617,932	3,867,492	4,051,361
Second min	1,382,071	2,169,625	2,939,325	3,530,828	3,782,820	3,970,722
Second fluctuation	8.329%	14.199%	6.924%	2.467%	2.238%	2.031%
Third max	1,571,992	2,646,193	3,512,450	3,612,282	3,973,444	4,187,136
Third min	1,284,080	2,147,167	3,236,538	4,060,309	4,114,077	4,187,136
Third fluctuation	22.422%	23.241%	8.525%	12.403%	3.539%	2.975%

Table 4.13: Fluctuation of installed prefixes

Configuration	4 peers	8 peers	12 peers	16 peers	20 peers	24 peers
First max	55,739	307,853	1,400,629	2,437,251	7,362,352	15,009,400
First min	14,480	139,767	489,279	1,915,071	6,357,673	13,261,194
First fluctuation	284.938%	120.262%	186.264%	27.267%	15.803%	13.183%
Second max	62,910	1,151,552	4,064,204	6,109,504	8,197,863	18,845,915
Second min	10,304	184,127	903,537	3,190,332	6,658,452	13,152,813
Second fluctuation	510.540%	525.412%	349.810%	91.501%	23.120%	43.280%
Third max	306,487	485,531	1,126,662	3,549,418	8,017,979	15,835,574
Third min	14,186	132,448	556,645	1,625,529	6,677,934	13,201,754
Third fluctuation	2066.599%	266.582%	102.402%	118.355%	20.067%	19.951%

Table 4.14: Fluctuation of announced prefixes

4.2 Graphs

In this section, we visualize our data with a set of graphs. Section 4.2.1 starts by presenting the difference between received, installed, and announced prefixes. Next, in Section 4.2.2, we focus on showing the difference in-between filter configurations for installed as well as announced prefixes. This is followed by Section 4.2.3 that aims to illustrate the difference in fluctuation between the three filter configurations for received, installed, and announced prefixes.

4.2.1 Received, installed, and announced prefixes

The data presented in Sections 4.1.2-4.1.4 is in this section visualized in Figure 4.1-4.2. Figure 4.1 and 4.2 illustrate how the average number of installed and announced prefixes for each of the three filter configurations compares to the average number of received prefixes as reported by Vtysh. As defined in Section 3.2.3, our first filter configuration allows all received prefixes to pass-through the Out-filter at the RS while the second filter configuration rejects prefixes that are not announced by any of the peers while the third configuration rejects prefixes that actually are announced by the peers. Furthermore, the Y-axis of Figure 4.1 and 4.2 specifies the average number of prefixes recorded while the X-axis illustrates the number of peers used. The number of received and installed prefixes shown in the two table have been preprocessed as described in Section 3.3.2, this enables us to directly compare Figure 4.1 and 4.2. When examining the two tables, it can be seen that the four graphs in Figure 4.1 are relatively similar when compared with the corresponding four graphs in Figure 4.2 that all have deviating traits. Specifically, Figure 4.1 shows that the average number of installed prefixes is close to the average number of received prefixes regardless of filter configuration. However, Figure 4.2 clearly shows that non of our filter configurations are capable of announcing prefixes at the same rate as it receives them from its peers. This shows that our RS is bottlenecked by the number of prefixes it is capable of announcing to its peers.

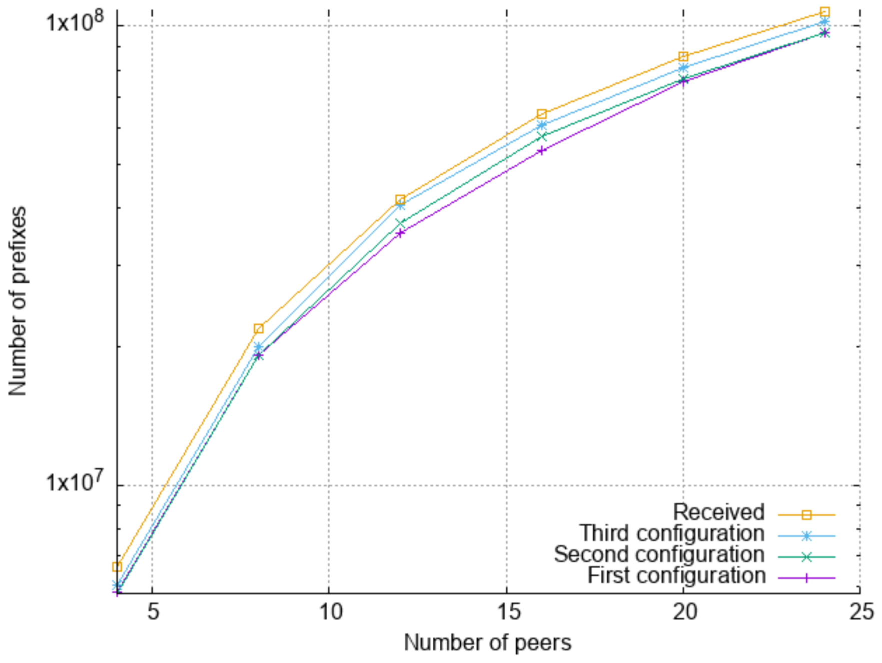

Figure 4.1: The average number of installed prefixes for each filter configuration compared to the average number of received prefixes

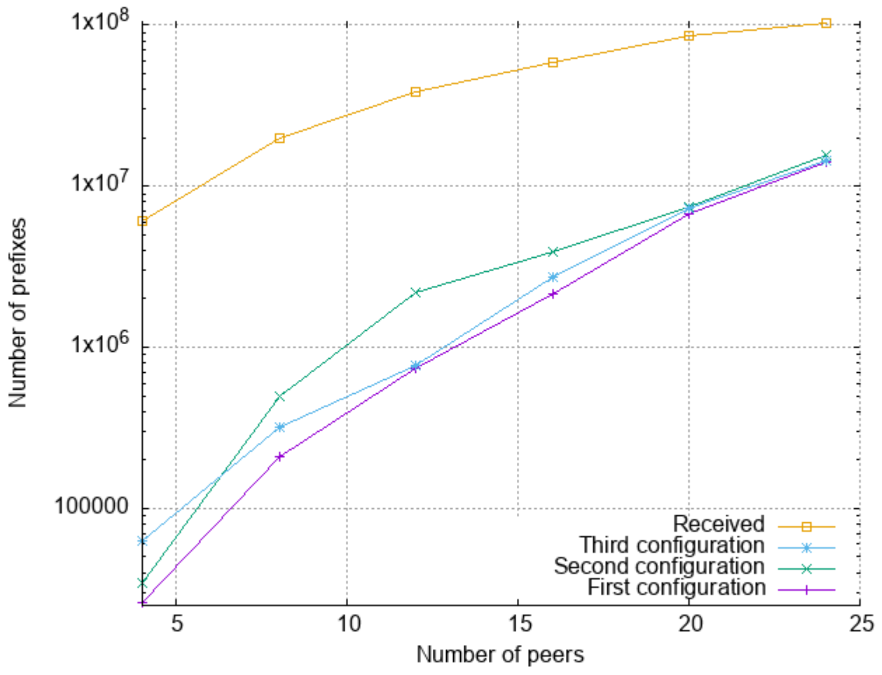

Figure 4.2: The average number of announced prefixes for each filter configuration compared to the average number of received prefixes

4.2.2 Filter comparison

In this section, we aim to illustrate in one coherent figure the difference between the three filter configurations for installed and announced prefixes. As defined in Section 3.2.3, the first filter configuration is set to not apply any filtering of prefixes the RS has received from its peers. Our second filter configuration, on the contrary, rejects prefixes that are not announced by any of the peers while the third configuration rejects prefixes that actually are announced by the peers. In Figure 4.3, the Y-axis illustrates the average number of recorded prefixes while the X-axis specifies the number of peers used in the corresponding benchmark. There are a total of six lines in the figure, first we have the three lines illustrating the number of announced prefixes for each of the three filter configurations. Next, we have the three lines positioned at the top of the figure illustrating the number of installed prefixes for each filter configuration. The number of installed prefixes shown in Figure 4.3 have been preprocessed as described in Section 3.3.2, this means that the number of installed prefixes has been multiplied by the number of peers used in the benchmark to enable a direct comparison with the number of announced prefixes that is shown in the same figure. When examining the figure, we notice that all of the graphs representing the average number of installed prefixes show roughly the same number of recorded prefixes when compared to the lines illustrating announced prefixes. For the announced prefixes, we see a more noticeable difference between the three filter configurations. The best performing configuration, in terms of announced prefixes, is the second configuration which with 12 active peers was capable of announcing three times more prefixes than the two other configurations. However, the trend does not hold true for four peers where the third configuration performed slightly better. Lastly, with 24 active peers, all three configurations seems to have announced roughly the same number of prefixes. Figure 4.3 shows that the number of prefixes our RS is capable of announcing depends on the filter configuration applied while the number of prefixes the RS is capable of installing is less dependent on filter configuration.

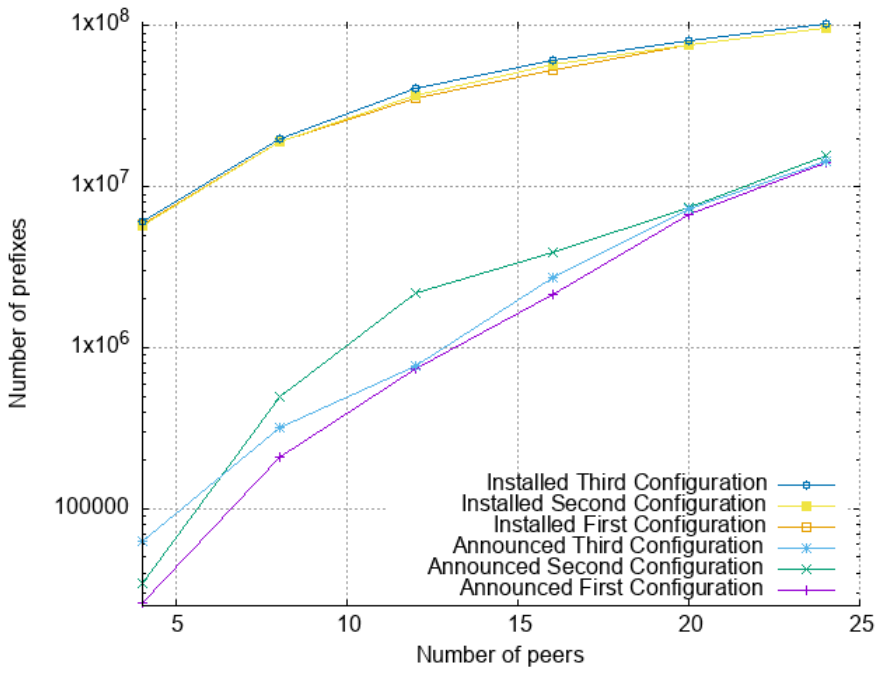

Figure 4.3: Comparison of the three filter configurations regarding the number of installed and announced prefixes

4.2.3 Fluctuations

This section aims to illustrate the rate of fluctuation for each filter configuration and benchmarking metric. We define fluctuation as the percentage difference between the benchmarking round with the most and least number of recorded prefixes for each filter configuration and set of peers. A benchmarking round is here defined as the number of prefixes that the RS has either received, installed, or announced within 180 seconds.

Figures 4.4 and 4.5 illustrate how the rate of fluctuation for each filter configuration relates to the fluctuation of received prefixes. As defined in Section 3.2.3, our first filter configuration allows all prefixes received by our RS to be announced back to our peers. The second filter configuration applies active filtering of prefixes that are not announced by any peer while the third filter configuration rejects prefixes that indeed are announced by our peers. Figure 4.4 aims to illustrate how the installed number of prefixes fluctuates when compared to the fluctuation of received prefixes. Likewise, Figure 4.5 illustrates the difference in fluctuation between announced and received prefixes for each of the three filter configurations and set of peers. Figures 4.4-4.5 are structured as follows, the Y-axis illustrates the percentage of fluctuation while the X-axis specifies the number of peers used in the benchmark. When examining the two tables, it is seen that the rate at which the number of announced prefixes fluctuates is considerably higher than the rate at which received and installed prefixes fluctuates. For example, from Figure 4.5 it is seen that the fluctuation, for the number of announced prefixes, ranges from around 10% to well over 1,000%. However, in Figure 4.4 we notice that the fluctuation ranges from a few percentages to a bit over 20%. The considerably higher fluctuations in the number of announced prefixes show that the RS is struggling to announce prefixes at the same rate as it receives them from its peers.

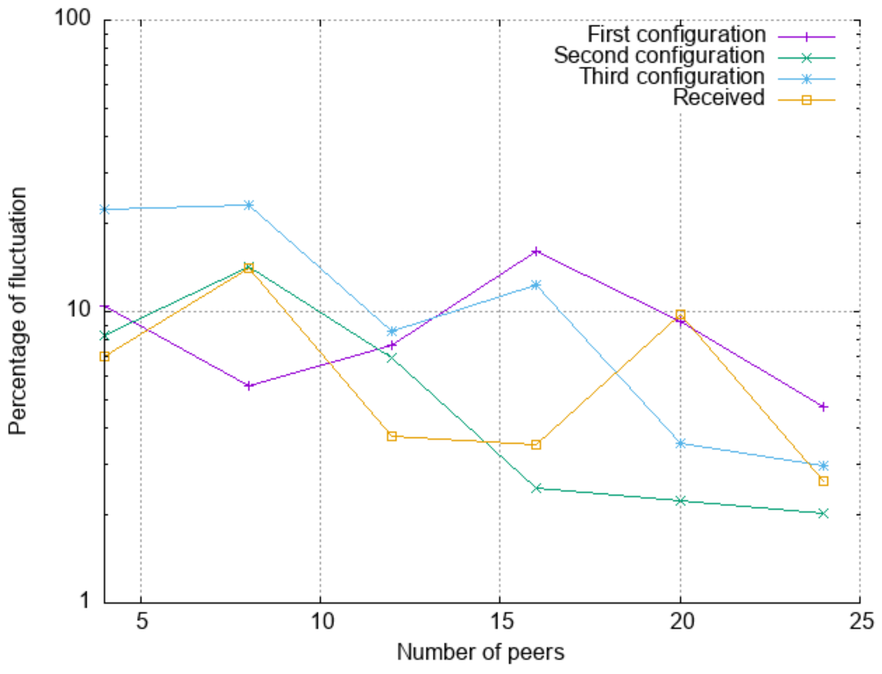

Figure 4.4: The fluctuation of installed prefixes compared with the fluctuation for received prefixes

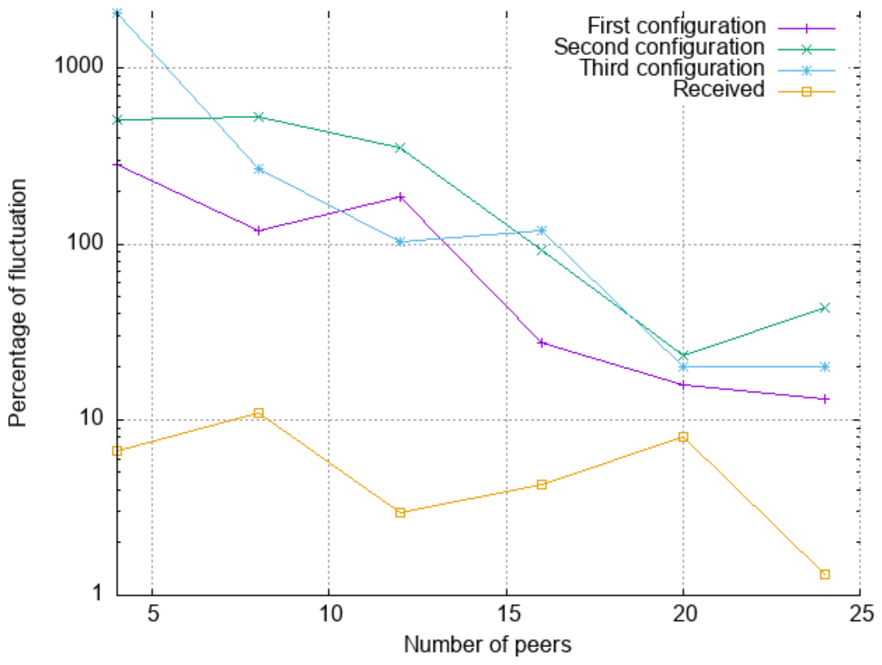

Figure 4.5: The fluctuation of announced prefixes compared with the fluctuation for received prefixes

Chapter 5

Discussion

This chapter consists of two major sections that provide an analysis of the results presented in Chapter 4 and respectively, an analysis regarding the different parts of the project as well as how they may have impacted the results. Specifically, in Section 5.1, we analyse the meaning of the results presented, potential reasons, how filtering impacts the number of announced prefixes, as well as what real-world limitations FRR might have. Further, in Section 5.2, we discuss how our benchmarking environment and methodology could be improved as well as alternative benchmarking metrics that might give a better representation of the performance of FRR. Lastly, we shortly touch upon the potential sustainability and ethical impacts of our findings.

5.1 Analysis of results

This section presents an analysis of the results from Chapter 4. We begin by analysing the cause of the fluctuations between different benchmarks. Next, the impact of the different filter configurations on the results is analysed. Finally, we relate the results to some real-world usage and requirements.

5.1.1 Fluctuation

The data presented in Chapter 4 clearly shows that our RS is not capable of announcing prefixes at the same rate as it receives these prefixes from its peers. The number of prefixes the RS announces within 180 seconds is consistently several magnitudes less than the number of prefixes it has received from the peers as well as installed in its routing tables. In addition to this, the results are further strengthened when examining the fluctuation data for announced prefixes, here we notice a considerably higher fluctuation rate across all benchmarking rounds and set of peers when we compare these results with the fluctuation for received and installed prefixes. Additionally, the fluctuation rate for announced prefixes increases when applying filtering of prefixes, meaning that the second and third filter configuration, that apply filtering of prefixes, shows a higher fluctuation rate compared with the first filter configuration that does not apply any filtering of prefixes. This can potentially show that the filtering of prefixes at our RS leads to an increased load on the FRR software suite since it must examine each prefix it receives and take action according to the rules specified in the filter configuration. For example, when looking at the CPU utilization on *S1* during the performed benchmarks we notice that the BGP daemon is utilizing around 40% of the CPU when the first filter configuration is applied. However, as for the two other filter configurations that do apply filtering of prefixes the CPU utilization increases to around 150% meaning that filtering prefixes is more computationally heavy for FRR to perform compared with not filtering prefixes. Moreover, since the BGP daemon makes use of a single thread for filtering prefixes as well as announcing prefixes on the network [52], [53], it means that our RS cannot perform filtering of prefixes and making announcements on the network in parallel leading to that one of the two tasks has to be suspended while the other is running. Since the filtering take computing resources as well as time to complete, it may result in that the RS is forced to build up a large queue of received prefixes that is to be filtered before they can be announced to a specific peer. Exactly how FRR decides to distribute this work may be non-deterministic, meaning that the RS decides to send out this set of prefixes on the wire in a non-deterministic fashion whenever appropriate according to some unspecified metrics.

However, although the fluctuation increases when filtering of prefixes is applied, the trend does not hold true when the number of peers increases. Specifically, the fluctuation rate is at its maximum when 4 peers are used in the benchmark regardless of filter configuration. Since 4 peers result in the least amount of combined prefixes the RS receives from the peers, this brings questions regarding if the fluctuations really are a consequence of the large number of prefixes the RS receives from its peers or merely how the FRR software suite is implemented to function.

5.1.2 Filtering

As discussed in the previous section, the rate of fluctuation for announced prefixes increases when the RS is configured to apply filtering of prefixes. However, when examining data from Section 4.1, as well as the figures in Section 4.2, it can be seen that the first filter configuration, which does not apply any prefix filtering, resulted in the least amount of installed and announced prefixes. This is noteworthy since we in Section 5.1.1 discussed why filtering of prefixes resulted in a higher fluctuation rate, one could therefore think that the increased fluctuation rate would impact the number of prefixes the RS is capable of announcing, but apparently it does not. From Table 4.11 we see that when 12 peers are used in the benchmark, the second filter configuration, which applies filtering of prefixes that are not announced by any peer, resulted in almost 3 times more announced prefixes when compared with the first filter configuration that does not apply filtering of prefixes. Additionally, the trend holds true even for the number of prefixes that has been installed at the RS, where the third configuration consistently reported to have installed more prefixes than the first and second filter configuration. However, the reason for this behaviour remains unknown, but it can potentially be a consequence of the increased fluctuations for the second and third filter configuration where the RS needs to queue up a bigger set of prefixes to announce, leading to that more prefixes reaches the peers in each BGP update.

5.1.3 Real-world limitations

In our results shown in Table 4.11, it is reported that a maximum of around 15 million prefixes was announced by our RS. On a peer by peer basis, this results in that each peer received 625.000 prefixes from our RS. Moreover, in January of 2021, it was reported that the complete IPv4 BGP routing table grew by 6%, year over year, to a total of 860,000 prefixes [1]. This means that to announce the complete BGP routing table in our benchmarking environment, FRR would require more than 180 seconds to complete this task. However, according to the Amsterdam Internet Exchange (AMS-IX), a neutral member-based association that operates multiple interconnection platforms around the world, their RSs' hold around 255,000 IPv4 prefixes in the master routing table. These prefixes are the best routes that the BGP path selection algorithm has selected among all received routes from all the established BGP feeds [27]. Additionally, AMS-IX advises its members to configure a max-prefix of 320,000 for IPv4. What this means is that far from every RS advertises the complete BGP routing table of 860,000 prefixes. According to the information given by AMS-IX, our research shows that FRR is more than capable of meeting the requirements that some of the world's IXP's may have. However, it should be noted that real-world RSes may configure their BGP implementation differently than in this work, meaning that there is a possibility that real-world configurations of FRR are even more computational intensive compared with our filter configurations. This may impact real-world limitations of FRR and could preferably be further researched in future work.

5.2 Discussion

In this section, the impact on our results is discussed from a perspective of the benchmarking environment as well as the method used which are described in Section 3.3. Further, possible solutions to the problems of these impacts are also suggested. Finally, we discuss potential sustainability and ethical impacts that our work may result in.

5.2.1 Alternative to ExaBGP

As seen in Table 4.1, when making use of 4 active peers in our benchmark, it resulted in that each peer was capable of advertising around 400,000 prefixes towards the RS within 180 seconds. In addition to this, as stated in Section 4.1.1, our two servers only utilized 27 Mb/s out of the 100 Gb/s available network capacity. This may not be enough prefixes and bandwidth needed to put maximum load on FRR and our RS. Additionally, the number of prefixes does not scale well since 24 peers only resulted in that around 4,500,000 prefixes reached the RS in the specified time span of 180 seconds. If each peer is capable of announcing 400,000 prefixes it should result in a total of 9,600,000 prefixes reaches the RS when 24 peers are in use. The reasons for this are unknown but could potentially be because ExaBGP is implemented in the Python programming language which is known to be one of the slower programming languages in terms of execution speed [50], [51]. Additionally, since we make use of a virtual network switch (OVS) to connect our ExaBGP instances within *S2*, this can result in that our network becomes bottlenecked. Virtual switching is known to be limited by the number of packets it can forward, applications that require a large number of small packets, as in our benchmarks, are thus more difficult for a virtual switch than applications relying on bulk data transfer, e.g., file servers [54]. Therefore, a potential superior, as well as more realistic, benchmarking environment could be to use multiple instances of FRR itself instead of relying on ExaBGP and virtual switching. However, this would require a more complex and expensive environment where each FRR instance either would have to be run on its own server or make use of containers for running multiple FRR instances on a single machine.

5.2.2 Tracing

As described in Section 3.3, we have in this work relied on the tshark network protocol analyzer for capturing what prefixes propagate the network as well as the Virtual Teletype Shell (Vtysh) for obtaining data about the BGP daemon. However, this might not be the optimal tool for the purpose of researching the performance of FRR since these methods only report the number of prefixes that has been received, installed, or announced. Additionally, since we run the network analyzer on *S1* it may have caused bottlenecks on the machine where FRR runs, leading to inaccurate results of the performance of FRR. As tshark consistently writes the captured network traffic to disk, which is a slow operation to perform, it is possible that tshark consumes resources that FRR needs in order to run at its maximum capacity. However, tshark is a single threaded application and the BGP daemon of FRR only utilizes 3 threads out of the 16 threads that are available on *S1*. This means that tshark in theory should not bottleneck the CPU on *S1*, but it does not rule out other potential bottlenecks such as mutual exclusions or systems calls that requires the kernel of the operating system to intervene. To get more in-depth information about how FRR functions as well as minimize potential bottlenecks that might affect the performance of FRR, one may make use of software tracing. With tracing, it is possible to record information about a program's execution. FRR has a small but growing number of static tracepoints available for use with various tracing systems. These tracepoints can assist with debugging, performance analysis, and to help understand program flow while maintaining extremely low overhead [55]. The reason why this work made use of tools other than tracing is that in order to utilize tracing, one must perform more complicated configurations such as compiling FRR from source as well as configure the tracepoints accordingly.

5.2.3 Further dimensions

In this work, we chose to only cover the performance of FRR based on the number of prefixes the RS is capable of announcing on the network to its peers. However, there exists a wide range of other dimensions that may be more impacted when an RS is put under intense load. For example, it may be more computational heavy for FRR to perform AS-path filtering which is a method used to choose routes based on the ASes an IP-packet has to traverse to reach a specific destination prefix. Therefore, an alternative methodology would be to let our peers advertise the same set of prefixes but with varying AS-paths. This would force FRR to look at each AS-path and make decisions according to the criteria specified in Section 2.3.1. Since FRR would have to compare all of the AS-paths for each prefix, this may be more computational heavy and time consuming for FRR, leading to that a real bottleneck easier can be identified.

5.2.4 Sustainability and Ethics

Since our work shows that announcing hundreds of thousands of prefixes with FRR may take longer than 180 seconds, this may spark an interest in minimizing the need for the Internet's RS's to regularly announce their complete routing table which potentially consists of hundreds of thousands of destination prefixes. This could potentially save on expensive energy usage as well as ensure that the optimal route towards a destination on the internet always is known by the ones wishing to reach it. This, as a whole, could in a positive way lead to a more sustainable society [4]. Regarding the ethics of our work, we have considered the potential ethical impacts that it may result in. For example, showing limitations in how FRR performs and function might cause harm to the people that have developed the software, leading to a deteriorated reputation as well as deter individuals, companies, and organizations of choosing the FRR software suite. However, we argue that this simply is not the case, benchmarking a system is something that is widely accepted in the software community and is often welcomed by the developers. Additionally, our work does not show any directly negative aspects related to how FRR has been implemented. Moreover, other ethical aspects may be related to that the limitations of FRR can result in a new vector of potential attacks against RS's running the FRR software suite. We argue that this is not the case, it is obvious that the system scales up to a certain level and any attacker could simply run the same experiments if they wanted to. With this in mind, we argue that our work does not result in any ethical problems.

Chapter 6

Research conclusion

The BGP protocol is a critical component of the Internet infrastructure for exchanging routing information, consisting of IP-prefixes, between distinct independent entities. These entities often make use of an RS located at an IXP's to facilitate this exchange of information. Network devices processing BGP information must be highly optimized performance wise in order to be able to process any incoming information in real time. One of the leading open-source alternatives that implements the BGP protocol is Free Range Routing (FRR), resulting in that it is of interest of understanding the potential limitations of FRR to ensure a well functioning Internet.

This research has performed a set of benchmarks with the purpose of revealing potential limitations in how FRR performs regarding the number of prefixes it is capable of processing in a time span of 180 seconds. The benchmarking methodology consisted of configuring FRR as an RS where it received prefixes from a set of peers each of which simulating a distinct entity. Simultaneously, FRR processed the set of received prefixes in order to announce them on the network to all of its connected peers. The results shows that FRR has limitations when it comes to announcing prefixes on the network. Specifically, the amount of received prefixes, during the benchmark, was consistently several magnitudes higher than the amount of prefixes that FRR was capable of announcing on the network to its peers. This shows that there exists a bottleneck in the performance of FRR. However, one can question what potential consequences this limitation may result in since our research shows that FRR, in our benchmarks, announced more prefixes than what an average IXP's RS holds in its master routing table [27]. This means that, although FRR may not be able to announce prefixes at the same rate as it receives them from its peers, there should not be any problems regarding announcing an average sized routing table in a few minutes time.

For future research purposes, one should consider developing a more realistic benchmarking environment that, instead of relying on emulated peers and virtual switches, makes use of several FRR instances. Another direction is to improve the benchmarking methodology regarding how the number of received and announced prefixes is recorded. Here we suggest to make use of techniques such as software tracing which offers deeper insight regarding the program flow of the FRR software suite. Lastly, there exists a wide variety of potential configurations that may be more challenging for FRR to handle, one such dimension is to configure FRR to examine the AS-path of each prefix received and take action accordingly to a set of rules.

Bibliography

[1] G. Huston and 2. a. 1. Am, *BGP in 2020 – The BGP Table*, en-US, Section: Tech matters, Jan. 2021. [Online]. Available: https://blog.apnic.net/2021/01/05/bgp-in-2020-the-bgp-table/ (visited on 03/01/2021).

[2] M. Chiesa, D. Demmler, M. Canini, M. Schapira, and T. Schneider, "SIXPACK: Securing Internet eXchange Points Against Curious onlooKers", Nov. 2017, pp. 120–133. DOI: 10.1145/3143361.3143362.

[3] R. De', N. Pandey, and A. Pal, "Impact of digital surge during Covid-19 pandemic: A viewpoint on research and practice", *International Journal of Information Management*, vol. 55, p. 102 171, Dec. 2020, ISSN: 0268-4012. DOI: 10.1016/j.ijinfomgt.2020.102171. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7280123/ (visited on 05/25/2021).

[4] M. Bexell and K. Jönsson, "Responsibility and the United Nations' Sustainable Development Goals", *Forum for Development Studies*, vol. 44, no. 1, pp. 13–29, Jan. 2017, Publisher: Routledge _eprint: https://doi.org/10.1080/08039410.2016.1252424, ISSN: 0803-9410. DOI: 10.1080/08039410.2016.1252424. [Online]. Available: https://doi.org/10.1080/08039410.2016.1252424 (visited on 02/25/2021).

[5] A. Håkansson, "Portal of Research Methods and Methodologies for Research Projects and Degree Projects", eng, CSREA Press U.S.A, 2013, pp. 67–73. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-136960 (visited on 05/25/2021).

[6] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz, "Characterizing the Internet hierarchy from multiple vantage points", vol. 2, Feb. 2002, 618–627 vol.2, ISBN: 978-0-7803-7476-8.

[7] D. Estrin, Y. Rekhter, and S. Hotz, *RFC 1322: A Unified Approach to Inter-Domain Routing*, 1992. [Online]. Available: https://www.hjp.at/doc/rfc/rfc1322.html (visited on 02/25/2021).

[8] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing", *IEEE/ACM Transactions on Networking*, vol. 10, no. 2, pp. 232–243, Apr. 2002, Conference Name: IEEE/ACM Transactions on Networking, ISSN: 1558-2566. DOI: 10.1109/90.993304.

[9] Y. Rekhter and P. Gross, *RFC 1772 - Application of the Border Gateway Protocol in the Internet*, EN, Mar. 1995. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1772 (visited on 05/26/2021).

[10] *RFC 4271: A Border Gateway Protocol 4 (BGP-4)*, RFC_4271. [Online]. Available: https://www.hjp.at/doc/rfc/rfc4271.html (visited on 01/27/2021).

[11] M. N. Gobrial, "Evaluation of border gateway protocol (BGP) version 4 (V4) in the tactical environment", in *Proceedings of MILCOM '96 IEEE Military Communications Conference*, vol. 2, Oct. 1996, 490–495 vol.2. DOI: 10.1109/MILCOM.1996.569372.

[12] *Understanding BGP Path Selection - TechLibrary - Juniper Networks*. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/reference/general/routing-protocols-address-representation.html (visited on 03/02/2021).

[13] *What is BGP prefix hijacking? (Part 1)*, en-US, Sep. 2020. [Online]. Available: https://www.manrs.org/2020/09/what-is-bgp-prefix-hijacking-part-1/ (visited on 05/26/2021).

[14] Anapaya, *Border Gateway Protocol Hijacking - Examples and Solutions*, en. [Online]. Available: https://www.anapaya.net/blog/border-gateway-protocol-hijacking-examples-and-solutions (visited on 05/26/2021).

[15] *Blog - Radar by Qrator*. [Online]. Available: https://radar.qrator.net/blog/born-to-hijack (visited on 05/26/2021).

[16] J. Peters, *Prolonged AWS outage takes down a big chunk of the internet*, en, Nov. 2020. [Online]. Available: https://www.theverge.com/2020/11/25/21719396/amazon-web-services-aws-outage-down-internet (visited on 05/26/2021).

[17] *Internet and cloud services - statistics on the use by individuals*, Dec. 2014. [Online]. Available: https://www.cloudwatchhub.eu/internet-and-cloud-services-statistics-use-individuals (visited on 05/26/2021).

[18] R. Brandom, *Using the internet without the Amazon Cloud*, en, Jul. 2018. [Online]. Available: https://www.theverge.com/2018/7/28/17622792/plugin-use-the-internet-without-the-amazon-cloud (visited on 05/26/2021).

[19] *What is AWS? Does Amazon control the modern day internet?*, en-US, Feb. 2020. [Online]. Available: https://augustafreepress.com/what-is-aws-does-amazon-control-the-modern-day-internet/ (visited on 05/26/2021).

[20] *Search Engine Market Share Worldwide*, en. [Online]. Available: https://gs.statcounter.com/search-engine-market-share (visited on 05/26/2021).

[21] J. Scudder, R. Bush, P. Mohapatra, D. Ward, and R. Austein, *BGP Prefix Origin Validation*, en. [Online]. Available: https://tools.ietf.org/html/rfc6811 (visited on 02/25/2021).

[22] R. Kuhn, K. Sriram, and D. Montgomery, "Border Gateway Protocol Security", en, National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-54 (Withdrawn), Jul. 2007. DOI: https://doi.org/10.6028/NIST.SP.800-54. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-54/archive/2007-07-17 (visited on 02/25/2021).

[23] P. Richter, G. Smaragdakis, A. Feldmann, N. Chatzis, J. Boettger, and W. Willinger, "Peering at Peerings: On the Role of IXP Route Servers", *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 31–44, Nov. 2014. DOI: 10.1145/2663716.2663757.

[24] M. Masoud, Y. Jaradat, and I. Jannoud, "A measurement study of Internet Exchange Points (IXPs): history and future prediction", *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 25, Dec. 2015. DOI: 10.3906/elk-1412-23.

[25] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger, "Anatomy of a large european IXP", *ACM SIGCOMM Computer Communication Review*, vol. 42, Sep. 2012. DOI: 10.1145/2342356.2342393.

[26] *DE-CIX - Where networks meet - DE-CIX*. [Online]. Available: https://www.de-cix.net/ (visited on 05/25/2021).

[27] O. A. www.oberon.nl, *AMS-IX Route Servers / AMS-IX Amsterdam*. [Online]. Available: https://www.ams-ix.net/ams/documentation/ams-ix-route-servers (visited on 05/20/2021).

[28] *Start / Netnod*. [Online]. Available: https://www.netnod.se/?gclid=Cj0KCQjwwLKFBhDPARIsAPzPi-Luf3UqOEn8AyGTui-sD5Xr1c8UP5SPMgZwvjxa6a5CyYCI66Y7_1EaAt1CEALw_wcB (visited on 05/25/2021).

[29] J. Fredin, *Average data flow*, Sv, Mar. 2021.

[30] *IX Statistics / Netnod*. [Online]. Available: https://www.netnod.se/ix/statistics (visited on 05/25/2021).

[31] I. Dolnák, "The purpose of creating BGP route servers at Internet Exchange points", in *2016 International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Nov. 2016, pp. 59–62. DOI: 10.1109/ICETA.2016.7802068.

[32] R. Govindan, C. Alaettinoğlu, K. Varadhan, and D. Estrin, "Route servers for inter-domain routing", en, *Computer Networks and ISDN Systems*, vol. 30, no. 12, pp. 1157–1174, Jul. 1998, ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00008-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169755298000087 (visited on 02/25/2021).

[33] *SDX - A Software Defined Internet Exchange Point*, 2015. [Online]. Available: https://sdx.cs.princeton.edu/ (visited on 03/17/2021).

[34] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: a software defined internet exchange", en, in *Proceedings of the 2014 ACM conference on SIGCOMM*, Chicago Illinois USA: ACM, Aug. 2014, pp. 551–562, ISBN: 978-1-4503-2836-4. DOI: 10.1145/2619239.2626300. [Online]. Available: https://dl.acm.org/doi/10.1145/2619239.2626300 (visited on 03/17/2021).

[35] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An Industrial-Scale Software Defined Internet Exchange Point", en, p. 15,

[36] S. Hermans, J. Schutrup, A. Vijn, and J. Claassen, "On the feasibility of converting AMS-IX to an Industrial-Scale Software Defined Internet Exchange Point", en, p. 19,

[37] M. Winter, "Open Source Software for Routing", en, *OpenSourceRouting.org*, p. 31,

[38] *FRRouting*. [Online]. Available: https://frrouting.org/ (visited on 02/25/2021).

[39] P. G. Jensen, K. G. Larsen, and J. Srba, "PTrie: Data Structure for Compressing and Storing Sets via Prefix Sharing", en, in *Theoretical Aspects of Computing – ICTAC 2017*, D. V. Hung and D. Kapur, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 248–265, ISBN: 978-3-319-67729-3. DOI: 10.1007/978-3-319-67729-3_15.

[40] A. Taylor, B. Rudolph, D. Spierling, and J. Moos, *An IXP Route Server Test Framework*, EN, Barcelona, Apr. 2017. [Online]. Available: https://www.de-cix.net/_Resources/Persistent/fba89bc19381b6784df99d2a78d4a11ebb7583c2/DE-CIX-route-server-testframework.pdf.

[41] T. Mangin, *ExaBGP*, original-date: 2013-08-27T13:56:08Z, Mar. 2021. [Online]. Available: https://github.com/Exa-Networks/exabgp (visited on 03/19/2021).

[42] D. Pruthvi, "Reducing Context Switching Overhead by Processor Architecture Modification", en, p. 8,

[43] D. Tsafrir, "The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops)", Jun. 2007, p. 4. DOI: 10.1145/1281700.1281704.

[44] S. M. Jain, "Namespaces", en, in *Linux Containers and Virtualization: A Kernel Perspective*, S. M. Jain, Ed., Berkeley, CA: Apress, 2020, pp. 31–43, ISBN: 978-1-4842-6283-2. DOI: 10.1007/978-1-4842-6283-2_3. [Online]. Available: https://doi.org/10.1007/978-1-4842-6283-2_3 (visited on 04/27/2021).

[45] D. Schubert, "Network Emulation using Linux Network Namespaces", en, 2019, Medium: PDF Publisher: Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich. DOI: 10.2313/NET-2019-10-1_11. [Online]. Available: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2019-10-1/NET-2019-10-1_11.pdf (visited on 04/27/2021).

[46] M. Kerrisk, *Network Namespaces - Linux manual page*. [Online]. Available: https://man7.org/linux/man-pages/man7/network_namespaces.7.html (visited on 04/27/2021).

[47] *BGP — FRR latest documentation*, EN. [Online]. Available: https://docs.frrouting.org/en/latest/bgp.html#description-of-the-route-server-model (visited on 04/16/2021).

[48] P. Carlo Chiodi, DE-CIX-Role, and B. Rudolph, *DE-CIX/pbgp-parser*, original-date: 2016-09-16T09:36:30Z, Mar. 2021. [Online]. Available: https://github.com/DE-CIX/pbgp-parser (visited on 04/06/2021).

[49] E. Ståhl, *emilstahl97/Performance-analysis-of-the-FRRouting-Route-Server*, original-date: 2021-06-30T11:59:38Z, Jun. 2021. [Online]. Available: https://github.com/emilstahl97/Performance-analysis-of-the-FRRouting-Route-Server (visited on 07/01/2021).

[50] *Why Python is Slow: Looking Under the Hood | Pythonic Perambulations*, May 2014. [Online]. Available: http://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/ (visited on 05/19/2021).

[51] K. R. Srinath, "Python – The Fastest Growing Programming Language", en, *International Research Journal of Engineering and Technology*, vol. 04, no. 12, p. 4, Dec. 2017.

[52] *FRRouting/frr*, original-date: 2016-12-15T21:56:38Z, May 2021. [Online]. Available: https://github.com/FRRouting/frr (visited on 05/20/2021).

[53] D. Sharp, *FRR applies prefix filtering and announcements in the same thread*, en, May 2021.

[54] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis", *Journal of Network and Systems Management*, vol. 26, Apr. 2018. DOI: 10.1007/s10922-017-9417-0.

[55] *Tracing — FRR latest documentation.* [Online]. Available: http://docs.frrouting.org/projects/dev-guide/en/latest/tracing.html (visited on 05/20/2021).

Appendices

Appendix A

Bash scripts

A.1 Create benchmarking environment

```
#!/bin/bash

cat << EOF

Following is a script asking for your network
parameters in order to create the necessary configuration files:

EOF

cwd=$(pwd)

read -p "Enter the IP-adress of the host where bgpd is running: " FRRIP
echo " "
echo "IP of host where FRR is running: $FRRIP"
echo "Current directory: $cwd"
echo " "

mkdir exabgp
cd exabgp
mkdir prefixes route-batches instances

for i in {1..24}
do
echo "Creating instance$i.ini, prefixes$i.txt and route-batches$i.py"

rm instance$i.py && touch instance$i.py

cat > instance$i.py << EOF
#!/usr/bin/env python3

import sys
import time
from time import sleep

def route_smash():
    for first in range(1,223):
        for second in range(0,255):
            for third in range(0,255):
                sys.stdout.write('announce route %d.%d.%d.$i/32 next-
                   hop 10.0.0.$i\n' % (first, second, third))
                sys.stdout.flush()
```

```
if __name__ == '__main__':
    try:
            route_smash()
    except (BrokenPipeError, IOError):
        sleep(10)
        route_smash()
EOF

cd instances
rm instance$i.py && touch instance$i.py

cat > instance$i.ini << EOF
process announce-routes {
    run /usr/bin/python3 ${cwd}/exabgp/route-batches/route-batches$i.py
        ;
    encoder json;
}

neighbor $FRRIP {                      # Remote neighbor to peer with
    router-id 10.0.0.$i;               # Our local router-id
    local-address 10.0.0.$i;           # Our local update-source
    local-as 6500$i;                    # Our local AS
    peer-as 65000;                     # Peer's AS
    group-updates true;

    api {
        processes [announce-routes];
    }
}
EOF

cd ..
cd prefixes
rm prefixes$i.txt && touch prefixes$i.txt
echo "Creating prefixes$i.txt"
python3 ${cwd}/exabgp/instance$i.py >> prefixes$i.txt
cd ..
rm instance$i.py

cd route-batches
rm route-batches$i.py && touch route-batches$i.py

cat > route-batches$i.py << EOF
#!/usr/bin/env python3

import sys
import time
import random
from time import sleep

def readFile():
    path = "${cwd}/exabgp/prefixes/prefixes$i.txt"
    with open(path) as fp:
        line = fp.readline()
        lines = []
        while line:
            lines.append(line)
            line = fp.readline()


    route_smash(lines)


def route_smash(lines):
    for i in range(len(lines)):
        sys.stdout.write(lines[i])
        sys.stdout.flush()


if __name__ == '__main__':
        try:
            readFile()
        except (BrokenPipeError, IOError):
                sleep(10)
                readFile()
EOF
cd ..
done

cd ..
cat > start_exabgp.sh << EOF
cd exabgp/instances
for j in $(eval echo {1..24});
do
    echo "Starting␣instance␣$j"
    sudo ip netns exec exa$j exabgp instance$j.ini &
done
EOF
```

A.2 First filter configuration

```
#!/bin/bash

rm frr.conf && touch frr.conf

cat > frr.conf << EOF
! -- bgp --
!
! BGPd sample configuration file
!
!
hostname RS
password zebra
enable password zebra
!
!
router bgp 65000 view RS
bgp router-id 192.168.30.5
no bgp ebgp-requires-policy
EOF
for i in {1..24}
do
echo "neighbor 10.0.0.$i remote-as 6500$i" >> frr.conf
echo "neighbor 10.0.0.$i update-source cx5n0if0" >> frr.conf
echo "neighbor 10.0.0.$i ebgp-multihop" >> frr.conf
echo "neighbor 10.0.0.$i prefix-list anyIn in" >> frr.conf
echo "neighbor 10.0.0.$i prefix-list anyOut out" >>  frr.conf
done

echo "!" >> frr.conf
echo "address-family ipv4 unicast" >> frr.conf

for i in {1..24}
do
echo "neighbor 10.0.0.$i activate" >> frr.conf
echo "neighbor 10.0.0.$i route-server-client" >> frr.conf
echo "neighbor 10.0.0.$i soft-reconfiguration inbound" >> frr.conf
done

echo "exit-address-family" >> frr.conf
echo "!" >> frr.conf

echo "ip prefix-list anyIn permit any" >> frr.conf
echo "ip prefix-list anyOut permit any" >> frr.conf

echo "!" >> frr.conf
echo "log file /var/log/frr/frr.log" >> frr.conf
```

A.3 Second filter configuration

```
#!/bin/bash

rm frr.conf && touch frr.conf

cat > frr.conf << EOF
! -- bgp --
!
! BGPd sample configuration file
!
!
hostname RS
password zebra
enable password zebra
!
!
router bgp 65000 view RS
bgp router-id 192.168.30.5
no bgp ebgp-requires-policy
EOF
for i in {1..24}
do
echo "neighbor 10.0.0.$i remote-as 6500$i" >> frr.conf
echo "neighbor 10.0.0.$i update-source cx5n0if0" >> frr.conf
echo "neighbor 10.0.0.$i ebgp-multihop" >> frr.conf
echo "neighbor 10.0.0.$i prefix-list partialIn in" >> frr.conf
echo "neighbor 10.0.0.$i prefix-list partialOut out" >>  frr.conf
done

echo "!" >> frr.conf
echo "address-family ipv4 unicast" >> frr.conf

for i in {1..24}
do
echo "neighbor 10.0.0.$i activate" >> frr.conf
echo "neighbor 10.0.0.$i route-server-client" >> frr.conf
echo "neighbor 10.0.0.$i soft-reconfiguration inbound" >> frr.conf
done

echo "exit-address-family" >> frr.conf
echo "!" >> frr.conf

echo "ip prefix-list partialIn permit any" >> frr.conf
for i in {1..5}
do
for j in {1..10}
do
for k in {1..10}
do
echo "ip prefix-list partialOut deny $i.$k.$j.25/32" >> frr.conf
done
done
done
echo "ip prefix-list partialOut permit any" >> frr.conf

echo "!" >> frr.conf
echo "log file /var/log/frr/frr.log" >> frr.conf
```

A.4 Third filter configuration

```
#!/bin/bash

rm frr.conf && touch frr.conf

cat > frr.conf << EOF
! -- bgp --
!
! BGPd sample configuration file
!
!
hostname RS
password zebra
enable password zebra
!
!
router bgp 65000 view RS
bgp router-id 192.168.30.5
no bgp ebgp-requires-policy
EOF
for i in {1..24}
do
echo "neighbor 10.0.0.$i remote-as 6500$i" >> frr.conf
echo "neighbor 10.0.0.$i update-source cx5n0if0" >> frr.conf
echo "neighbor 10.0.0.$i ebgp-multihop" >> frr.conf
echo "neighbor 10.0.0.$i prefix-list partialIn in" >> frr.conf
echo "neighbor 10.0.0.$i prefix-list partialOut out" >>  frr.conf
done

echo "!" >> frr.conf
echo "address-family ipv4 unicast" >> frr.conf

for i in {1..24}
do
echo "neighbor 10.0.0.$i activate" >> frr.conf
echo "neighbor 10.0.0.$i route-server-client" >> frr.conf
echo "neighbor 10.0.0.$i soft-reconfiguration inbound" >> frr.conf
done

echo "exit-address-family" >> frr.conf
echo "!" >> frr.conf

echo "ip prefix-list partialIn permit any" >> frr.conf
for i in {1..4}
do
for j in {1..5}
do
a=$((j*50))
b=$((j*40))
for k in {1..25}
do
echo "ip prefix-list partialOut deny $i.$a.$b.$k/32" >> frr.conf
done
done
done
echo "ip prefix-list partialOut permit any" >> frr.conf
echo "!" >> frr.conf
echo "log file /var/log/frr/frr.log" >> frr.conf
```

A.5 Capture prefixes on network

```
#!/bin/bash

#Remove old files
touch captured-results-$2peers-$1.txt

#Create directories
mkdir $2peers-$1
mkdir $2peers-$1/pcaps
mkdir -p Results/$1

if [[ $1 == "help" ]]
then
  echo "filter-version,␣#peers,␣round"
  exit 0
fi

read -p "Enter␣the␣network␣interface␣to␣capture␣on:␣" interface
read -p "Enter␣the␣MAC-address␣of␣the␣network␣interface:␣" MAC

bash ../restartFRR.sh

echo "---------------------------------------------"
echo "Started:␣$(date)"
echo "---------------------------------------------"

rm capture$3-$2peers-$1.pcap parsed.txt capture.json
touch capture$3-$2peers-$1.pcap parsed.txt capture.json vtysh-pfxsnt-
    $2peers-$1.txt vtyshresults-$2peers-$1.txt

tshark -i $interface -f "ether␣src␣host␣$MAC␣and␣tcp␣port␣179" --
    autostop duration:180 -w capture$3-$2peers-$1.pcap

vtysh -c 'show␣bgp␣view␣RS␣summary␣json' > capture.json
python3 parseJson.py $1 $2 $3

echo "Parsing␣file:␣capture$3-$2peers-$1.pcap"
cat capture$3-$2peers-$1.pcap | pbgpp --fields prefixes -f LINE -p FILE
    -o parsed.txt -
numberOfPrefixes=$(grep -o -i /32 parsed.txt | wc -l)
echo "$numberOfPrefixes" >> captured-results-$2peers-$1.txt
mv capture$3-$2peers-$1.pcap $2peers-$1/pcaps

sed -i 's/parsed.txt/␣/g' captured-results-$2peers-$1.txt
echo "Results␣after␣$3␣rounds:"
cat captured-results-$2peers-$1.txt

if [[ $3 == 10 ]]
then
average=$(python3 mean.py captured-results-$2peers-$1.txt)
delta=$(python3 diff.py captured-results-$2peers-$1.txt)
echo "$average" >> captured-results-$2peers-$1.txt
echo "Delta:␣$delta" >> captured-results-$2peers-$1.txt
cat captured-results-$2peers-$1.txt

average=$(python3 mean.py vtysh-pfxsnt-$2peers-$1.txt)
delta=$(python3 diff.py vtysh-pfxsnt-$2peers-$1.txt)
```

```
echo "$average" >> vtysh-pfxsnt-$2peers-$1.txt
echo "Delta: $delta" >> vtysh-pfxsnt-$2peers-$1.txt

sudo mv captured-results-$2peers-$1.txt $2peers-$1
sudo mv vtysh-pfxsnt-$2peers-$1.txt $2peers-$1
sudo mv vtyshresults-$2peers-$1.txt $2peers-$1
sudo mv $2peers-$1 Results/$1/

echo "Done"
fi
```

Appendix B

Python scripts

B.1 Parse Vtysh data - Installed prefixes

```
import json
import sys

def parse():

    with open('capture.json') as json_file:
        data = json.load(json_file)

        pfxRcd = 0
        pfxSnt = 0
        msgSent = 0
        inQ = 0
        outQ = 0
        peersEstablished = 0
        receivedPrefixes = []
        for value in data['ipv4Unicast']['peers'].values():
            pfxRcd += int(value["pfxRcd"])
            msgSent += int(value["msgSent"])
            inQ += int(value["inq"])
            outQ += int(value["outq"])
            if (value["state"] == "Established"):
                pfxSnt = int(value["pfxSnt"])
                receivedPrefixes.append(int(value["pfxRcd"]))
                peersEstablished += 1


    receivedPrefixes = sorted(receivedPrefixes)
    res = ((receivedPrefixes[-1] - receivedPrefixes[0])/
        receivedPrefixes[0])*100
    print(f"Results␣for␣{sys.argv[1]}␣peers␣and␣round␣{sys.argv[3]}:\n"
        )
    print("Number␣of␣peers␣established:␣" + str(peersEstablished))
    print("Prefixes␣in␣InQ:␣" + str(inQ))
    print("Prefixes␣in␣OutQ:␣" + str(outQ))
    print("Total␣message␣sent:␣" + str(msgSent))
    print("Total␣prefixes␣received:␣" + str(pfxRcd))
    print("Total␣prefixes␣sent:␣" + str(pfxSnt))
    print("Difference␣between␣received␣and␣sent␣prefixes:␣" + str(int(
        pfxRcd-pfxSnt)))
    print("Difference␣between␣most␣and␣least␣received␣prefixes:␣" + str
        (res) + "%")
    filename = f"vtyshresults-{sys.argv[2]}peers-{sys.argv[1]}.txt"
    filename2 = f"vtysh-pfxsnt-{sys.argv[2]}peers-{sys.argv[1]}.txt"
    results = open(filename, "a")
    pfxSntResults = open(filename2, "a")

    results.write(f"Results␣for␣{sys.argv[2]}␣peers␣and␣round␣{sys.argv
        [3]}:\n")
    results.write("Number␣of␣peers␣established:␣" + str(
        peersEstablished) + "\n")
    results.write("Prefixes␣in␣InQ:␣" + str(inQ) + "\n")
    results.write("Prefixes␣in␣OutQ:␣" + str(outQ) + "\n")
    results.write("Total␣message␣sent:␣" + str(msgSent) + "\n")
    results.write("Total␣prefixes␣received:␣" + str(pfxRcd) + "\n")
    results.write("Total␣prefixes␣sent:␣" + str(pfxSnt) + "\n")
    results.write("Difference␣between␣received␣and␣sent␣prefixes:␣" +
        str(int(pfxRcd-pfxSnt)) + "\n")
    results.write("Delta␣received␣prefixes:␣" + str(res) + "%\n")
    results.write("\n")

    pfxSntResults.write(str(pfxSnt) + "\n")


if __name__ == "__main__":
    parse()
```

B.2 Parse Vtysh data - Received prefixes

```
import json
import sys

def parse():

    with open('capture.json') as json_file:
        data = json.load(json_file)

        pfxRcd = 0
        pfxSnt = 0
        msgSent = 0
        inQ = 0
        outQ = 0
        peersEstablished = 0
        receivedPrefixes = []
        for value in data['ipv4Unicast']['peers'].values():
            pfxRcd += int(value["pfxRcd"])
            msgSent += int(value["msgSent"])
            inQ += int(value["inq"])
            outQ += int(value["outq"])
            if (value["state"] == "Established"):
                pfxSnt = int(value["pfxSnt"])
                receivedPrefixes.append(int(value["pfxRcd"]))
                peersEstablished += 1


    receivedPrefixes = sorted(receivedPrefixes)
    res = ((receivedPrefixes[-1] - receivedPrefixes[0])/
        receivedPrefixes[0])*100
    print(f"Results␣for␣{sys.argv[1]}␣peers␣and␣round␣{sys.argv[2]}:\n"
        )
    print("Number␣of␣peers␣established:␣" + str(peersEstablished))
    print("Prefixes␣in␣InQ:␣" + str(inQ))
    print("Prefixes␣in␣OutQ:␣" + str(outQ))
    print("Total␣message␣sent:␣" + str(msgSent))
    print("Total␣prefixes␣received:␣" + str(pfxRcd))
    print("Total␣prefixes␣sent:␣" + str(pfxSnt))
    print("Difference␣between␣received␣and␣sent␣prefixes:␣" + str(int(
        pfxRcd-pfxSnt)))
    print("Difference␣between␣most␣and␣least␣received␣prefixes:␣" + str
        (res) + "%")

    filename = f"vtyshresults-{sys.argv[1]}peers.txt"
    filename2 = f"vtysh-pfxrcd-{sys.argv[1]}peers.txt"
    results = open(filename, "a")
    pfxRcdResults = open(filename2, "a")

    results.write(f"Results␣for␣{sys.argv[1]}␣peers␣and␣round␣{sys.argv
        [2]}:\n")
    results.write("Number␣of␣peers␣established:␣" + str(
        peersEstablished) + "\n")
    results.write("Prefixes␣in␣InQ:␣" + str(inQ) + "\n")
    results.write("Prefixes␣in␣OutQ:␣" + str(outQ) + "\n")
    results.write("Total␣message␣sent:␣" + str(msgSent) + "\n")
    results.write("Total␣prefixes␣received:␣" + str(pfxRcd) + "\n")
    results.write("Total␣prefixes␣sent:␣" + str(pfxSnt) + "\n")
    results.write("Difference between received and sent prefixes: " +
        str(int(pfxRcd-pfxSnt)) + "\n")
    results.write("Delta received prefixes: " + str(res) + "%\n")
    results.write("\n")

    pfxRcdResults.write(str(pfxRcd) + "\n")


if __name__ == "__main__":
    parse()
```

B.3 Calculate average prefixes

```
from statistics import mean
import sys

lst = []

results = open(sys.argv[1], "r")
for line in results:
    lst.append(int(line))

print("\nAverage:␣" + str(mean(lst)))
```

B.4 Calculate fluctuation in-between benchmarking rounds

```
import sys

def diff(lst):
    numbers = sorted(lst)
    res = ((numbers[-1] - numbers[0])/numbers[0])
    return res*100


lst = []
results = open(sys.argv[1], "r")
for line in results:
    lst.append(int(line))
print(str(diff(lst))+'%')
```

B.5 Preprocess

```
from statistics import mean
import sys

def diff(lst):
    numbers = sorted(lst)
    res = ((numbers[-1] - numbers[0])/numbers[0])
    return res*100


lst = []
results = open(sys.argv[1], "r")
new = open("multiplied-"+sys.argv[1], "w+")
for line in results:
    if (line.strip("\n")):
        if not(line.startswith("A") or line.startswith("D")):
            multiplied = round(int(line.strip())*int(sys.argv[2]))
            lst.append(int(multiplied))
            new.write(str(multiplied)+ "\n")

print(mean(lst))
new.write("\nAverage: " + str(mean(lst)))
new.write("\nDelta: " + str(diff(lst))+'%')
```

Appendix C

Acronyms

AFRINIC	African Network Information Center.
AMS-IX	Amsterdam Internet Exchange.
APNIC	Asia-Pacific Network Information Centre.
ARIN	American Registry for Internet Numbers.
AS	Autonomous System.
ASN	Autonomous System Number.
BGP	Border Gateway Protocol.
CDN	Content Delivery Network.
DE-CIX	Deutscher Commercial Internet Exchange.
EGP	External Gateway Protocol.
FRR	Free Range Routing.
FSM	Finite-State Machine.
GPL2	GNU General Public License, version 2.
IANA	Internet Assigned Numbers Authority.
IGP	Internal Gateway Protocol.
IP	Internet Protocol.
IS-IS	Intermediate System to Intermediate System.
iSDX	Industrial-Scale Software Defined Internet Exchange Point.
ISP	Internet Service Provider.
IXP	Internet Exchange Point.
JSON	JavaScript Object Notation.
LACNIC	Latin America and Caribbean Network Information Centre.
LoC-RIB	Local Routing Information Base.
OSPF	Open Shortest Path First.

OVS Open vSwitch.

pbgpp PCAP BGP Parser.

PCAP File extension for the packet capturing library libpcap.

PKI Public Key Infrastructure.

RFC Request for Comments.

RIP Routing Information Protocol.

RIPE Réseaux IP Européens Network Coordination Centre.

RIR Regional Internet Registry.

RPKI Resource Public Key Infrastructure.

RS Route Server.

S1 Server 1 running the FRR software suite.

S2 Server 2 running the ExaBGP instances.

SDN Software-Defined Networking.

SDX Software Defined Internet Exchange.

TCP Transport Communication Protocol.

TTL Time to Live.

Vtysh Virtual Teletype Shell.