UNIVERSITY OF AMSTERDAM

Faculty of Physics, Mathematics and Informatics
Graduate School of Informatics
System and Network Engineering MSc

Automated configuration of BGP on edge routers

Research Project

Stella Vouteva
stella.vouteva@os3.nl

Tarcan Turgut
tarcan.turgut@os3.nl

August 14, 2015

Abstract

The growth of the Internet not only enables many new services and allows access by an increasing number of people, it also put us to new challenges. Today, Internet Service Providers need to deal with very large routing tables and complex router configurations, resulting in a higher chance of misconfiguration caused by human errors. To allow for future growth and correct inter-domain routing configuration, the de-facto Internet routing protocol BGP needs tools for automated configuration and management.

In this report, we first studied the current BGP automation strategies and analyzed the existing tools. Our analysis showed that current solutions are either over complex or outdated. Some lack IPv6 or 32-bit ASN support, but most importantly, none of these tools take security into consideration. In the light of current best practices defined in RFCs, we specified the requirements of a secure end-to-end BGP automation tool. After identifying the required features and functions, we developed the BGPWizard to satisfy the current needs of the Internet. We propose the complete design of the tool and an implementation as a proof of concept. Several test scenarios are designed to observe the effectiveness of the tool and future improvements are discussed accordingly.

Contents

1 Introduction

The Internet is an ever-growing and evolving system. The billions of devices interconnect and form the big Internet we currently know and use. The size of the Internet and amount of devices connected have led to the exhaustion of IPv4 address pool in recent years. In 2011, IANA announced the depletion of their IPv4 address space by allocating the remaining five /8 to the Regional Internet Registries (RIRs). The RIRs also allocated the IPv4 address space in recent years, APNIC in 2011, LACNIC in 2014, ARIN in 2015, AFRINIC in upcoming years [1]. RIPE NCC still assigns /22 to ISPs. [2]

De-aggregation of networks is basically spreading small subnets across different locations over countries and continents [3]. As a result, today there are 563,435 globally routable prefixes [4] and longer routing tables on routers, even exceeding the default forwarding table size of 512,000 in many older routers [5]. Consequently, longer and more complex prefix-lists have to be maintained. This made the Internet a more vulnerable system and more prone to human errors causing misconfiguration of Border Gateway Protocol (BGP), the de-facto standard of routing in the Internet.

There are many public incidents related to BGP misconfiguration. For example, on February 2008 Pakistan Telekom accidentally hijacked a /24 portion of YouTube's address space which redirected YouTube's traffic for a few hours [6]. Another high-profile incident occurred on March 2010, when China Telekom hijacked 15% of all the prefixes in the Internet [7]. Those incidents could be mitigated by using a security measure or a proper route filtering in their neighbor ASes.

These issues show that human errors need to be mitigated by automation, therefore, we came up with several research questions (proposed below).

1.1 Research questions

The main research question for this project is:

To what extent current technologies can be used to efficiently automate the configuration of BGP?

To provide an answer to the main research question, the following sub-questions are taken into account:

- *What are the existing public tools used to collect BGP policy information?*
- *Are those tools reliable enough to provide the necessary information*
- *Do current technologies adapt to the security trends in BGP?*
- *What are the limitations of automatic BGP configuration?*

1.2 Outline

This document includes background information with basic Internet information and security practices for BGP in section 2. In section 3 we look at the current tools for automation, analyze and classify them. Later, in section 4 we propose a solution that covers the limitations of the current tools. Section 5 shows the proposed design and a proof of concept of our solution. Test of the solution and the results are shown in section 6. We finalize this report with the conclusion and future work in sections 7 and 8.

2 Background

This section provides an overview of the current technologies that are involved around the Internet, including BGP, the Internet Routing Registries (IRR), Routing Policy Specification Language (RPSL) and current security practices for BGP.

2.1 Internet technologies

In order to have a better understanding in the rest of this report, we describe the main Internet technologies that we mention throughout this document.

2.1.1 BGP

The Internet consists of a number of routing domains, called Autonomous Systems (ASes). Each AS is described by an Autonomous System Number (ASN) and includes a collection of IP prefixes. An AS uses different mechanisms to propagate network information internally - either statically defined or dynamic using Internal Gateway Protocols (IGP). In order to exchange information with other ASes, however, an External Gateway routing protocol (EGP) is used.

BGP is described in RFC 4271 [8] and is currently the only EGP protocol in use. For two BGP routers (called speakers) to exchange reachability information, they must first form a neighbor relationship (also called peering relationship). There are two types of BGP peering: Internal BGP (IBGP) and External BGP (EBGP). Figure 1 shows an example topology where AS1 and AS2 are neighbor ASes. Inside AS1, BGP routers establish IBGP and edge routers in AS1 and AS2 establish EBGP relationship.

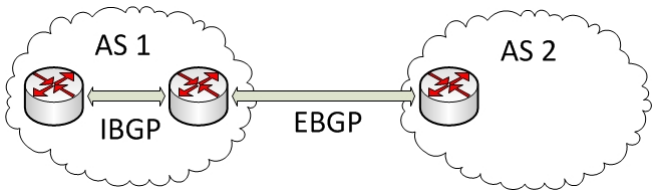

Figure 1: BGP Relationships

BGP information is propagated using the protocol's UPDATE message. This message contains the prefix information, along with BGP path attributes (attributes are defined in RFC 4271 [8]). The BGP path attributes are used for

control over the selection of the best BGP routes, which are then added to the routing table (if no better path exist).

What information is exchanged and which extra (non-mandatory) path attributes are set is described in policies that vary per organisation. These policies can also be included in public Internet Routing Registry (IRR).

2.1.2 IRR

An IRR is a distributed database system which contains routing polices of ASes [9]. Its purpose is to ensure stability and consistency of the Internet-wide routing by sharing information between network operators [10]. The IRR can be used by operators to look up peering agreements, to study optimal policies, and to (possibly automatically) configure routers.

IRR is publicly available and consists of several registries (databases) that are maintained on a voluntary basis. There are currently 34 operational registries [9]. Each RIR has its own registry and there are also independent registries such as RADB. The terms database and registry will be used interchangeably.

In order to design a reliable BGP automation, the source of information must be reliable. RIPE database is the only reliable source because the information can only be registered and edited by the associated AS owners. Thus, RIPE database provides authentic data that can be used for router configuration.

However, there can still be the chance that the owners may register wrong information in their database objects that would effect its neighbor ASes. In such a case, a neighbor would update its BGP configuration depending on the data provided in IRR, that would accidentally cause prefix hijacking or route leaks. To overcome such an accident, we propose an extra verification step before creation of router configuration by using RPKI (Resource Public Key Infrastructure) which will be discussed in 4.2.

The routing policies in IRR consists of objects, whose content is described in Routing Policy Specification Language (RPSL). RPSL is described in the next section.

2.1.3 RPSL

RPSL is a language, used to describe organisational routing policies. It is defined in RFC 2622 [11] and extended to support IPv6 and multicasting in RFC 4012 [12] as RPSLng. Two RPSL objects are important for the scope of this project - the AUT-NUM and the ROUTE object. Descriptions of the two objects can be found in the RIPE Database documentation [1] [2]. What is important to mention here is that the AUT-NUM object holds the policies in the import and export attributes and the ROUTE object contains the prefixes, owned by an AS.

The usage of RPSL is described in RFC 2650 [13]. An overview of this RFC can be found in Appendix A.

[1] http://tinyurl.com/ripe-aut-num-doc
[2] http://tinyurl.com/ripe-route-doc

2.2 Security practices

With all the malicious or accidental issues with BGP, security has become an important part of the Internet. This section considers several security practices for BGP. Other security mechanisms, such as authentication of the BGP session (as BGP runs on top of TCP), rate-limiting, Generalized TTL Security Mechanisms (GTSM) and Unicast Reverse Path Forwarding (uRPF) are outside the scope of this project and information about them can be found in RFC 7454 [14]. There are several Best Current Practices (BCP) that describe recommended BGP filtering mechanisms.

2.2.1 Prefix filtering

BCP 38 [15], also known as RFC 2827, describes inbound and outbound prefix filtering, applied as a basic DoS prevention when spoofed IP addresses are used. Basically, providers should include in their policies to export only prefixes that are allocated to them, otherwise they risk accidentally hijacking someone else's prefix. This practice prevents accidental spoofing of addresses outside of the AS that announces them, but it does not prevent spoofed addresses within the network itself. In case of an attack when the real address is used, the source can be located. The malicious source can then be blocked or be filtered using remote triggered black holing (RTBH) [16].

It also prevents the leakage of local prefixes into BGP. Filtering using the IRR is also useful for peers, connected to an Internet Exchange (IX). Here, it is important to make sure that no other peer announces more specific prefix than the IX prefix, or a case of a black hole can occur. The information in the IRR can change, so such prefix filters need to be refreshed. RFC 7454 proposes a refresh interval of one day.

Other than control over announced public prefixes, there are several types of filtering - martian address, bogon and default route filtering. The first two address types should be discarded when seen in a BGP peer session. [14]

BCP 38 is updated by BCP 84 [17] for multi-homing.

Martian address filtering
A martian includes IP addresses that are reserved by the Internet Assigned Numbers Authority (IANA) and have a special meaning. [18] Such addresses are private IP ranges, loopback and multicast addresses, etc. [19] The complete list special addresses can be found at [20] for IPv4 and at [21] for IPv6 addresses.

These addresses should not be included in the BGP routing table and have to be discarded.

Bogon filtering
Unlike martian addresses, bogon prefix space consists of the IP addresses that are unallocated by the IANA or the RIRs. These addresses are often the source of an attack with a spoofed IP address. Since 2011, all /8 IPv4 addresses are assigned. [22] IPv6 space, however, still requires filtering. As address space can be allocated, such filters must be refreshed often or a risk arises that routable prefixes are discarded and have to be de-bogonized. [23]

Default route filtering

The 0.0.0.0/0 and ::/0 prefixes should only be accepted or advertised in specific customer/provider setup. [14]

Limitations of specific prefix filtering
An issue with today's Internet is not that administrators cannot filter properly, but it is that they are often not concerned with it, as it mostly benefits other peers. [24] This filtering, however, can be applied easily if the policy is defined in RPSL. RPSL is used to describe import and export policies of an organisation in the IRR database. These policies can be translated into ingress or egress filtering within a router. In other words, an AS can query the route object for a neighbor to get the prefixes that neighbor announces, generate a prefix-list (or other form of prefix filtering, such as a route-filter) and apply it in such a way that only those prefixes are accepted from the neighbor.

The issue raises when the relationship between two peers is transit and the policy states that both ASes import and export everything between each other (they import/export ANY). In such cases, filtering AS prefixes becomes very difficult or impossible. In such case, filtering of martians, bogons and default route is not sufficient.

Prefix length filtering
It is a common practise for ISPs to filter prefixes that are more specific than a certain threshold (usually /24 for IPv4 and /48 for IPv6). This is used as a control filter that assures that the routing table cannot be filled with long prefixes, leading to bad performance of the router or even completely filling up the routing table.

Prefix filter application recommendations
RFC 7454 [14] describes that simplifying the announced prefixes can be done with the usage of prefix filters, generated from the IRRs. Several types of inbound and outbound filtering are defined. It is important to note that there are two options for inbound filtering - loose and strict. The only difference between the two is that in the strict option, the announcements are checked against and conform to the routing registry data. Here, we describe the inbound and outbound filters as suggested in the RFC.

- **Filtering with external peers**

Inbound	If strict filtering is used, the data in the routing registry should be checked for consistency (missing prefixes, etc.) and, if none are found, the filters can be applied. If loose filtering is applied, the following received prefixes are rejected: • Martian addresses • Bogons (IPv6) • Prefixes with more specific length than a chosen threshold (usually /24 for IPv4) • Local AS prefixes • IX point LAN prefixes • Default route
Outbound	There are two options for outbound filtering - either to allow a list of prefixes to be advertised or, if that is not possible, to make the following filters: • Martian addresses • Prefixes with more specific length than a chosen threshold (usually /24 for IPv4) • IX point LAN prefixes • Default route

- **Filtering with customers**

Inbound	All customer prefixes should be accepted and the rest is rejected. The only exception to this rule is if strict filtering is applied and the list of prefixes is too long. Then the prefix filter for the loose option, described in the inbound filtering for external peers, is used.
Outbound	Here, the customer can choose only to receive a default route or the full routing table. In case of the latter, the following prefixes are filtered: • Martian addresses • Prefixes with more specific length than a chosen threshold (usually /24 for IPv4) • Default route (unless the customer wants it included)

- **Filtering with upstream providers**

Inbound	There are several options here. Either the provider can send a default route or the full routing table. In the latter situation, the prefix filter for the loose option, described in the inbound filtering for external peers, is used. The default route in this filter is optional, as it may be required along with the full routing table.
Outbound	Same as outbound for external peers, unless extra tuning is needed.

2.2.2 Other recommended filtering

RFC 7454 [14] describes other methods to apply control over BGP routes. These strategies are route flap damping, maximum accepted prefixes from a peer, AS-path filtering, next-hop filtering and BGP communities scrubbing. The last three filters are considered, but not described in this report.

Route flap damping
Route flap damping is a mechanism, defined in RFC 2439 [25], that penalizes a route if that route changes in the routing table more times than a certain threshold. Route flap damping was considered bad practice. Since 2013, however, [26] and [27] provide new recommendations on how to use route flap damping.

Maximum prefixes from peers
Maximum prefixes from peers represents the maximum number of prefixes that a neighbor is allowed to announce. RFC 7454 [14] recommends to set the maximum prefixes lower than the size of the full BGP routing table, so that no peer advertises the full routing table. When the peer is an upstream, then the limit is set to the memory limit of the router that receives the routes.

2.2.3 Secure Inter-Domain Routing (SIDR)

SIDR is an infrastructure, aimed at the security of BGP advertisements and currently includes two services [14]:

- Origin validation using RPKI, specified in [28], is used to verify whether an AS is allowed to announce a route
- Path validation using BGPsec, described in [29]

It is important to note that BGPsec is still being standardized. It also requires changes in the BGP messages and it is outside the scope of this project. For more information, see [30].

RPKI Validation
Resource Public Key Infrastructure (RPKI) is a public key infrastructure framework designed to validate the origin of routes using X.509 Certificates. It is a distributed repository system that holds the digitally signed routing objects. Those objects are called Route Origin Authorizations (ROAs) and they define associations between the authorized ASes and IP prefixes [31]. ROAs also include the prefix length to determine the maximum length of a prefix that can be announced by the origin AS.

RPKI validation process is based on the validity state of a ROA. A BGP route can have one of the three states [28]:

- Valid: There is at least one ROA covering the route announcement.
- Invalid: Either the route is announced by an unauthorized AS (AS-mismatch), or the route announcement has a more specific prefix length than the allowed maximum prefix length (Prefix length mismatch).
- Not Found: There is no existing ROA covering the route announcement.

In this project, RPKI is considered as a security mechanism to configure BGP routers.

3 Current tools

Manual BGP configuration, including proper filtering mechanisms, is no longer an easy task for ISPs, as it is most likely to lead to misconfigurations. Therefore, it is important to analyze the existing public tools for BGP automation to see how far they can actually automate and if that is enough for the current needs. In this section, we provide an insight on the tools, used to get the policy data from the databases. We also look into strategies to push a complete BGP configuration to a device. In the end, we classify these mechanisms to see what limitations do they have and whether these tools can really be used for complete automation.

3.1 Policy-based tools

There are several ways to extract policies from the IRR databases, mentioned in the RIPE documentation [32]. The most famous one is the Whois service. Other command-line tools that can be used to extract RIPE data are also Netcat and Telnet. They have the same functionality as the Whois service and will not be described in this report. There are also more advanced tools that provide more than query functions.

3.1.1 Whois

The IRRs also provide a lookup service with access to the publicly available part of the database, called Whois [33]. The protocol is defined in RFC 3912 [34] and is basically a client/server TCP-based protocol where the client sends a query to a server, and the server sends a response and terminates the connection. The connection concept is shown in figure 2.

Whois covers data about IPv4, IPv6 addresses, DNS and autonomous systems that are defined in RPSL. In the scope of this project, only the RPSL data is considered. RIPE also provides a Web query form that can be used to search the database [35]. The output is basically the same as with the Whois service, but the advantage here is that it does not require installation of the client. The IRR data can also be accessed using the Whois REST API [36].

This tool has an informational goal, rather than to be used for extracting configuration data. RFC 3912 [34] states that the Whois protocol is not secure and should therefore not be used for sensitive information, as it provides no confidentiality, integrity or access control.

3.1.2 IRRToolSet

The Internet Registry Toolset (IRRToolset)[37] is developed initially by the Information Sciences Institute at the University of Southern California as a part of a project, called the Routing Arbiter ToolSet (RAToolSet) project. During the years, several parties have held the copyright. In 2001, the project is migrated to the RIPE NCC, where it gets the name IRRToolSet. In 2004, the project is maintained by the Internet Systems Consortium (ISC) and in 2014, the code is

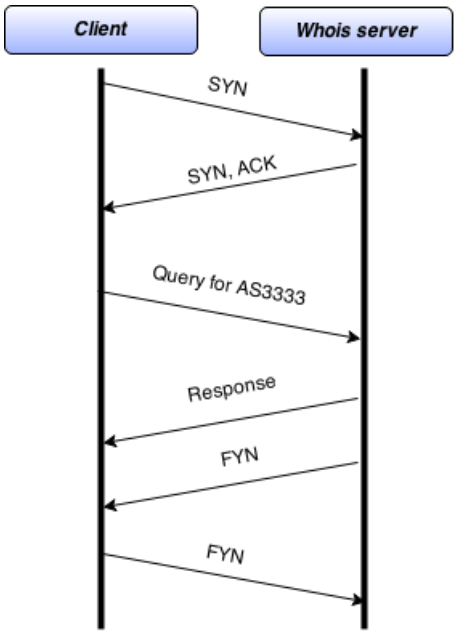

Figure 2: Whois as specified in RFC 3912

moved to Github. The software consists of three tools - RtConfig [3], peval [4] and rpslcheck [5]:

- RtConfig - This tool extracts policies from the IRRs using a whois (see 3.1.1) connection and produces router configurations.
- peval - low level policy evaluation tool.
- rpslcheck - used to verify the syntax of RPSL objects.

The complete structure of the toolset is shown in Figure 3.

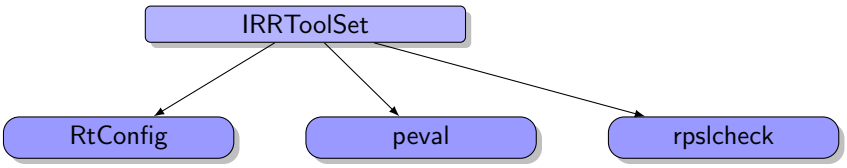

Figure 3: IRRToolSet

Its primary goal is to automate router configuration. This is very convenient for network administrators to generate neighbor, network statements, prefix lists, etc for both Cisco and Juniper routers. It supports RPSL and RPSLng. It also works with 32-bit ASNs. It does not, however, support queries for an AS-SET. The tool is described as "extremely complex and implements rarely

[3] http://irrtoolset.isc.org/wiki/RtConfig
[4] `http://irrtoolset.isc.org/wiki/peval`
[5] `http://irrtoolset.isc.org/wiki/rpslcheck`

used features"[38]. The code from the currently official website [37] does not compile for the newest versions due to a wrong code statement. The code is also found in a Github[39] repository [40], maintained by Nick Hilliard. This version can be compiled and run.

The IRRToolSet can produce complete BGP router configuration. This process is, however, not automated. An administrator has to manually add the neighbor ASN and IP address, and then the toolset can generate peering configuration, based on the policies in the IRR.

Configurations can be generated for a Cisco and a Juniper router (see Appendix B for a Juniper example and Appendix C for a Cisco example). There is an issue that arises when using the import(/export) function of the tool. The AS-numbers and IP addresses have to be known by the administrator (or manually extracted from RPSL, if they are present).

There are multiple issues with IRRToolSet:

- Newest versions from the ISC have issues when they have to be compiled and installed
- The code for the tool is overly complex and currently there are only attempts to maintain it, but not develop it further.

3.1.3 IRR Power Tools

IRR Power Tools is a collection of tools which aims at automatic prefix-list generation by using IRR[41]. The file structure of the tool can be seen in figure 4.

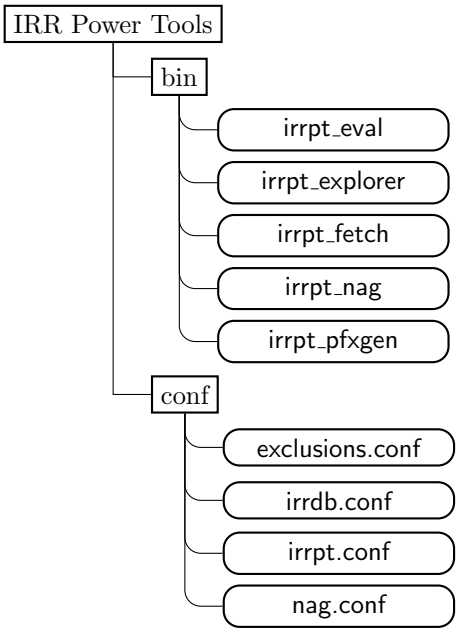

Figure 4: IRR Power Tools structure

The operation of the tool is divided into three sub-operations; *fetch*, *nag* and *pfxgen* as shown in Figure 5. An example of irrpt_pfxgen script, used for prefix

list generation as a part of the Pfxgen operation, can be found in Appendix D. Fetch operation is initiated by the irrpt_fetch script with following functions:

- queries IRR objects (aut-num and as-set)
- fetches prefixes (in Cisco, Juniper, Extreme or Force10 format)
- excludes the prefixes defined in exclusions.conf file

Power Tools can also be used in order to track changes in IRR. The Nag process is used for mail notification in case of any changes. The general configurations of the tool (such as IRR database address, local AS numbers) is stored in irrpt.conf and irrdb.conf files.

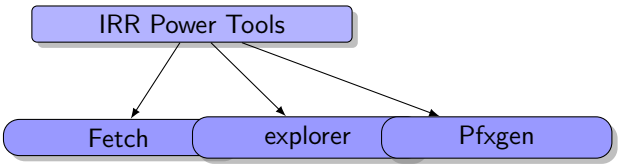

Figure 5: IRR Power Tools

The issue with this tool is that it does not support neither RPSLng nor 32-bit ASN.

3.1.4 BGPq3

BGPq3 is a tool used to generate prefix-lists, extended access lists, etc. for Cisco and Juniper routers based on data, taken from the IRRs. It makes use only of the ROUTE object and is not concerned with import or export policies. It supports RPSL, RPSLng and AS-SET queries. An example of the tool can be seen in Appendix E.

Unlike IRRToolSet, BGPq3 cannot generate complete BGP router configuration for a network. It cannot extract peering information, create neighbor or network statements, so basically it uses the RPSL database information only partially.

3.1.5 RPSLtool

Rpsltool is another BGP configuration automation tool that consists of a set of libraries and programs mainly based on the Perl Template Toolkit[42]. Rpsltool was developed in order to overcome the complexity of RPSL[38]. This is due to the fact that sometimes configuring an import or export attribute in an aut-num object requires many directives. This includes the usage of complicated regular expressions, in turn, requiring difficult parsing mechanisms. In order to circumvent this complexity, Rpsltool make use of local configuration files in YAML (Yet Another Markup Language) format which contains, for instance, peer information (ASN and peer IP) and BGP attributes.

An example of configuration for a certain neighbor AS is shown in figure 6. The only data that Rpsltool needs from an IRR database is the route objects associated with the peer AS, which is defined in import attribute. For this reason, only route and route6 objects are needed to be kept up to date in

IRR database by the owners. However, the import key can also be defined manually without needing any IRR database. Thus, Rpsltool can also be used as a standalone tool which only relies on the local configuration.

```
- as: 12654
  description: RIPE RRC10
  ip: 2001:7F8:B:100:1D1:A5D1:2654:6
  import: AS12654:RS-RIS
```

Figure 6: RPSL neighbor config

This tool supports 32-bit ASN, but it does not support RPSLng. In a sense, this tool is not so much different from the IRRToolSet. It moves the complexity from writing RPSL to writing local files.

3.1.6 Net::IRR

Net::IRR is a library that provides an interface to the Internet Route Registry Daemon [43], written in Perl by Todd Caine. It is used to extract IPv4 and IPv6 routes for an originating autonomous system, get a list of the contents of the AS-SET and ROUTE-SET objects. The advantage of this library is that it is not TCP-intensive, as only one TCP connection is established for multiple queries.

The library does not have enough functionality to be used as a configuration tool. The tool can also be considered outdated, as the last version is from 2010.

3.1.7 Netconfigs

Netconfigs is an online web service[44] that contains several tools related to BGP configuration. Cisco-BGP Config Tool can generate a BGP configuration by entering local and peer AS numbers and peer IP address as input. The tool uses data from RADB registry to create prefix lists and route maps. Here, we can see the issue that queries are made only to the RADB registry, so the policy data needs to exist in that particular database. The tool can only generate peerings if the peer IP addresses are defined in import and export attributes in IRR.

The tool is vendor specific, can only be used for Cisco devices. In addition, it is not an open source tool and a subscription is required for unlimited access. Another drawback of the tool is that the information source is RADB, which may contain unreliable data, as no authentication process exists in RADB, in contrast with RIPE IRR.

3.1.8 Policy-based tools summary

In order to give a general overview of what the current tools are capable of, we summarize the advantages and disadvantages of the advanced tools (see table 1). We look for features, such as IPv6, 32-bit AS-number, AS-SET query support and how far these tools can reach when it comes to actual router configuration. We want to see the common features and functionality of all tools, as well as their limitations.

Tool	Advantages	Disadvantages
IRRToolSet	- Full RPSL support - RPSLng support - 32-bit ASN support - Full BGP config generation	- No AS-SET query support - Manual peering configuration on the fly - Official latest release does not compile - Difficult to understand
IRR Power Tools	- Route aggregation - AS-SET queries	- No RPSLng support - No 32-bit ASN support
BGPq3	- RPSL support - RPSLng support - 32-bit ASN - AS-SET queries - Easy to use	- Only partial BGP configuration. Cannot extract policies from the IRR.
RPSLtool	- 32-bit ASN - AS-set queries	- No RPSLng support - Dependency on local configuration files
Net::IRR	- RPSL and RPSLng support - One TCP connection for multiple queries	- Outdated - Does not support the community attribute from RPSL data - No AS-SET queries - Not supported, only maintained
netconfigs	- Provides peering analysis - Can generate full configuration, based on peering relationship	- Does not support RPSLng - No command line query - Vendor dependent (Cisco)

Table 1: Policy-based tools - comparison summary

The table shows several common disadvantages for all the tools when it comes to automation. Some of them can create complete BGP configuration, but they do not take into consideration the current router situation (neighbor groups, policy names, etc.), security policies, nor they actually push that configuration to the device. The one thing they have in common is the ability to generate IPv4 prefix lists for an AS-number.

3.2 Router configuration tools

As mentioned in the previous section, the policy-based tools are not enough to provide complete automation. Here, we look into some of the tools that can be used to push configuration to the router.

3.2.1 OpenNaaS

OpenNaaS [6] is an open source Network-as-a-Service software stack for provisioning network devices. It is used for deployment and automated configuration with a vendor independent interface. It was developed as a virtualization tool for the National research and education networks (NRENs) and operators, and it promotes a trusted platform by providing continuity, transparency and adaptability. OpenNaaS makes use of the NETCONF protocol (which we describe later in 4.3). What is important to say here about NETCONF is that it describes the means for establishing a secure remote connection to a networking device and the configuration commits are executed as atomic transactions. When it comes to routers, OpenNaaS supports the following features:

- Interfaces and routing instances virtualization
- static and OSPF routing
- GRE tunneling

It does not, however, support IPv6 and firewalling yet and can be used only on Juniper devices. OpenNaaS has too much functionality for the simple task to push BGP configuration to a router.

3.2.2 ConfD

ConfD [45] is a network management agent, developed by Tail-f (now part of Cisco). It provides a lot of functionality, among which atomic transactions, SDN support, scalability and vendor independence. Like the previously discussed tools, it also uses NETCONF in its underlying structure. One of its advantages over the mentioned tools is that it uses YANG - a data modeling language created especially for NETCONF and defined in RFC 6020 [46].

The tool has a free and a commercial version, the differences of which are not relevant to this project. Both versions are closed source. Just like OpenNaaS, ConfD (both versions) has too much complexity.

3.2.3 Network Control System

Soon to be renamed to Network Service Orchestrator (NSO), NCS [47] was also developed by Tail-f. It provides advanced provision services for configuration of multiple devices at once. It can be used for central management or for automation purposes.

The beauty of NCS is that it makes use of a publish/subscribe model for programmability as a feature. It also uses YANG and NETCONF. The only downside from the point of view of this project is that NCS, just like OpenNaaS and ConfD is unnecessarily complex for only BGP configuration.

[6] http://opennaas.org/

3.2.4 PyEZ

PyEZ [48] is a microframework for Python that allows remote management and automation of JunOS devices. Its main goal is to provide the means to manage a router remotely, without requirements, such as programming skills or deep understanding of how JunOS works. Here, Python is used mostly for the shell environment, rather than as a programming language. The basic capabilities of PyEZ are as follows:

- retrieving configuration and operational information
- configuration changes
- software updates

PyEZ also makes use of the NETCONF protocol for remote connectivity by including a library, called NCClient [7]. NCClient is a Python library, used for scripting around the NETCONF protocol.

The only advantage that PyEZ has over a regular SSH command-line configuration (other than an intended simplicity) is that it can still be used as a tool by programmers that can push a complete configuration as an atomic transaction (apply all or apply nothing).

Unlike the previously described tools, PyEZ has the opposite issue - it is too simple and limited to one vendor.

3.2.5 Router configuration tools summary

Here, we compare the advantages and disadvantages of the router configuration tools to see if any of those tools are suitable for pushing router configuration in a simple, atomic, scalable, flexible and secure manner without providing unnecessary overhead (see table 2).
The advantages of these tools are obvious - complete configurations, automation, SDN, etc. and all can push BGP configuration to a router. A disadvantage of all the tools is that they are over complex and over functional (for OpenNaaS, ConfD and NCS) or oversimplified (PyEZ) to be used as a simple BGP configuration tool.

3.3 Classification of current tools

Based on our analysis, the current tools can be classified under two categories, namely functionality (policy extraction or routing configuration) and profit (open source or commercial) of the tools. Figure 7 represents a diagram showing the position of each tool in this classification.

IRRToolSet, Rpsltool, BGPq3, PowerTools and Net::IRR are both open source tools (and library) designed to extract policy information from the IRRs. Netconfigs is the only commercial source used for policy extraction in our analysis. OpenNaaS and PyEZ are the open source tools used to interact with routers to push configuration. ConfD is a commercial software, also used in order to remotely configure routers, but it includes a free closed source version as well. NCS, on the other hand, is completely commercial and closed source.

[7] https://github.com/leopoul/ncclient

Tool	Advantages	Disadvantages
OpenNaaS	Network virtualization Secure connection	- No IPv6 support - Too much unneeded functionality for the goal of this project
ConfD	- Atomic - Scalable - SDN support - Vendor independent	- Closed source - Also has too much unnecessary functionality for this project
PyEZ	- Simplicity - Security - Remote configuration and automation - Open source - Code can be extended	- Intended non-programmer simplicity can lead to administrators with poor JunOS knowledge - Works only for Juniper devices
NCS	- Scalability - SDN support - Complete configuration automation	Too much functionality

Table 2: Router configuration tools - comparison summary

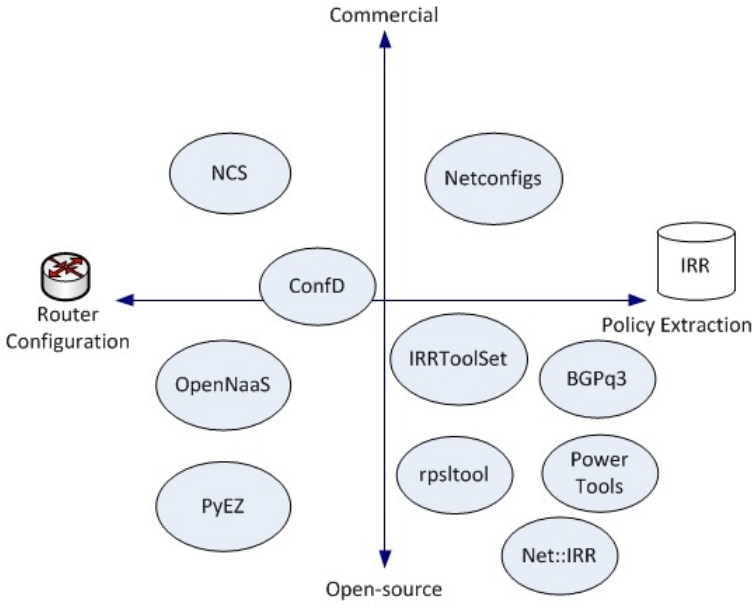

Figure 7: Tools classification

Our analysis showed that there is no publicly available tool that can both extract policies and configure routers accordingly, thus providing end-to-end automation. This gap between IRR and routers needs to be filled with such a tool that is able to perform this task. In addition, none of the tools has a set of security features and does not cover private information, such as private peering.

4 Automating BGP configuration

As discussed in the previous section, the currently existing tools still leave a serious gap that needs to be filled. This section describes the requirements, that were considered necessary for the design of a new tool. This section also discusses what are the expected input and output of the tool, and what obstacles have to be considered when developing such a tool.

4.1 Defining requirements

In order to build a general image of the capabilities of the new tool, two sets of requirements need to be defined - features and functionalities.

The analysis of the current tools, used to extract policies from IRR, brought to the attention that several features must be present if the solution we propose is to work with the Internet as it is today. With the exhaustion of IPv4 addresses, IPv6 support becomes a must. The same is the case with 16 and 32-bit AS-numbers. For ease of management, many import and export attributes contain AS-SETs, therefore increasing the necessity for the support of the sets. These AS-SETs can produce quite a long list of prefixes that need to be filtered. Route aggregation, in this case, would help reduce the size of an prefix-list. Last, but not least, there are many devices from various vendors that form the Internet, therefore vendor independence is an important factor that must be considered.

Complete automation of BGP cannot be achieved without having a public source of information. Therefore the new tool still needs to make use of the IRR as a source. To do this, the RPSL data has to be extracted and parsed. However, not everything is stored in the IRR databases for security reasons, such as confidential information; router credentials, etc. are not found in IRR. There can also be situations where an organization does not want to include all peering information in the IRR.

Summary of the desired features and functionalities can be found in tables 3 and 4.

Features
IPv6 support
32-bit ASN support
AS-SET query support
Route aggregation
Vendor independent

Table 3: Features of the proposed tool

Functionality
Query IRR and parse RPSL
Use a local file for additional information
Extract peer and policy configuration
Apply security
Push configuration to the router

Table 4: Functionalities of the proposed tool

Now that the requirements are defined, we need to make choices on the input and output sources of the proposed tool.

4.2 Input collection

Several of the functionalities we outlined need data input. The sources we used to get this information are discussed here. For policy data, we need input that comes from the IRR. For security, we considered Team Cymru's bogon lists. We decided to include anything that is missing in the public tools in a private local file. Finally, for RPKI validation, we looked into the RIR ROAs.

- IRR - The tool will use policy data from the RIPE registry. RIPE allows only authorized administrators to make changes to the database and that makes it a good choice for this project. Here, we make use of the REST Whois interface of RIPE, so no other IRR can be used. It is not impossible to change the code to include data from other IRRs. Our recommendation is, however, to use the RIPE database.

- Team Cymru - Team Cymru is a non-profit organisation that collects information about bogon prefixes for both IPv4 and IPv6 and updates the list every four hours. What they consider as 'Full bogon' lists includes martians, prefixes that are not allocated by IANA and prefixes that are allocated to the RIRs, but not assigned to ISPs. We consider bogon filtering as a very important criteria for this project, as it is used for DoS attack mitigation. It is crucial that it should be applied for security and updated as often as possible, or else de-bogonising might be needed. (see RFC 7454 [14] for this recommendation).

- Local file - As mentioned earlier, we need a mechanism to include private information. This is where the local file comes to play. It holds data, such as management IP address, user name, password for a secure connection, names of the access lists, extra filtering choices and some fine-tuning. The architecture of this local file is discussed later in 5.2.

- ROA from the RIRs - Even though RPKI's deployment is still very limited and slow, we consider that it is important to use it for this project. This is because of the fact that RPSL objects are maintained by administrators and is still susceptible to human error. The usage of RPKI validation in the new tool could help mitigate human mistakes when it comes to entering incorrect data in the IRR route object. It would not, however, solve the problem that the human error can be made in the ROA itself (by signing wrong or incomplete information, such as missing a prefix range).

These are the input sources we chose for the new tool. Next, we discuss the decisions we made on how to handle the output.

4.3 Output handling decisions

The output of the proposed tool is a BGP configuration of a router. Here, we consider several ways to structure the output and to send it to a device remotely:

- NETCONF - The Network Configuration protocol (NETCONF) is specified in RFC 6241 [49]. It defines mechanisms to extract, manipulate and send configurations to network devices. Remote Procedure Call (RPC) requests and replies are encoded in XML and sent using a secure transport mechanism. The structure of the protocol is shown in figure 8.

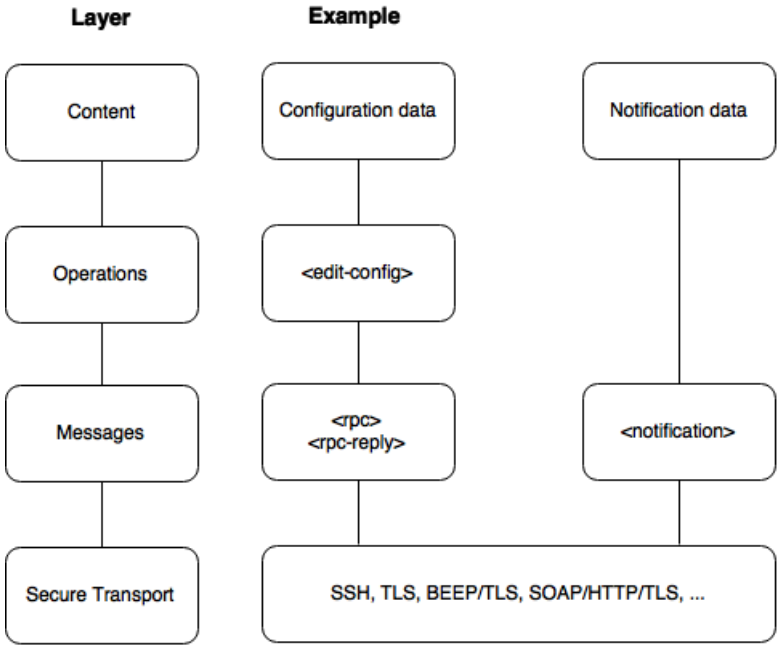

Figure 8: NETCONF Architecture, as specified in RFC 6241

Figure 8 shows that NETCONF consists of four layers - Secure Transport, Messages, Operations and Content. Examples of those layers are provided on the right side of figure 8. The secure transport is achieved in a connection-oriented way and provides authentication, integrity, confidentiality and defense against replay attacks. For example, the connection can be encrypted using TLS.

The Content that is meant to be included in the device is added as a Operation that is to be executed by editing the current device configuration. This is then wrapped in an RPC request and securely sent to the device. The device then responds with an RPC reply.

All the tools, mentioned in 3.2 make use of NETCONF at a basic level.

The beauty of NETCONF lies in the fact that it is designed as a unified strategy for remote device configuration, vendor independent, includes security and atomic transactions (all is committed or nothing is committed).

- SNMP - The Simple Network Management Protocol (SNMP) can be used to push network configuration to a device. Its main use if for performance and monitoring purposes. It uses UDP, so no connection is established with the router. There are models for SNMP that use TLS/TCP and DTLS/UDP [50]. SNMPv3 provides authentication, but does not provide atomic transactions.

- CLI - All major vendors (such as Cisco and Juniper) provide CLI APIs for router configuration. Both vendors, for example, provide a Python API [51] [52]. Many issues can arise with the use of CLI libraries:

  - Transactions are not atomic. If something goes wrong, the error is limited to the current command.
  - Maintainability. Commands can change which can lead to errors
  - Vendor specific. The APIs are not an unified way to configure multi-vendor devices.

After comparing the possible router configuration choices, we decided that NETCONF has the most necessary features and will therefore be the mechanism we select.

4.4 Development constraints

Above, we defined the expected input and output of the tool we propose in this report. It is important to note that these selections have limitations of their own that can become an issue for the new tool. These constraints can be classified as external to the proposed solution and are described here.

In order to get public information, we need to query the RIPE database. Parsing of nested elements, such as as-sets within as-sets means a lot of requests sent to the RIPE server(s). This issue that arises here is that RIPE has set a limit of how many times the database can be accessed per day. Once this threshold is passed, the IP address that is used for the queries is restricted for one day. This is probably RIPE's strategy for mitigating DoS attacks, but it also caused one of the important functionality in the proposed solution to be dropped. This functionality will be explained later in this report.

Another constraint comes from the Team Cymru side. They update their bogon lists every four hours. Which means that if filters need to be updated, one either has to wait for the next update in list, or to temporarily apply manual changes.

A notable confusion is created by RPSL itself. RPSL defines an attribute 'pref' that plays the role of inverse local-preference. The local-preference can then be calculated as 65535-pref. That means that the maximum local preference is 65535. There are two issues in this situation. First, nowadays routers support a maximum local-preference of 2^{32}-1 (4294967295) (for Juniper version 11.4 onwards [53], Cisco IOS [54] and Cisco NX-OS [55]) and that value can never be reached if the RPSL standard (which is from 1999) is followed.

The other issue is that IRRToolSet, which was maintained by RIPE until 2004, also does not comply with the standard and uses another formula 1000-pref. It is possible that the idea is to conform with legacy devices that only support maximum local-preference of 1000. This may, however, not be useful for newer devices. In this project, we follow the RPSL standard. However, we consider that this structure is outdated and the pref should reflect the real value of the local-preference attribute, not the inverse. This way all computational confusion will be cleared.

The proposed tool will not attempt to fix or overcome these issues as they are outside the scope of this project.

5 BGPWizard

In the previous section, we defined the requirements for the solution we propose in this report. In subsection 5.1, we provide the desired architecture of the tool, which we called BGPWizard. We also offer a structure of the local file, pointed as necessary input of the tool in 4.2. Note that the idea of this local file is that it can be extended if new functionality that needs new data comes along. Later on in this section, we introduce the proof of concept that was developed during the course of this project.

5.1 Tool architecture

The desired architecture for the tool is shown in figure 9.

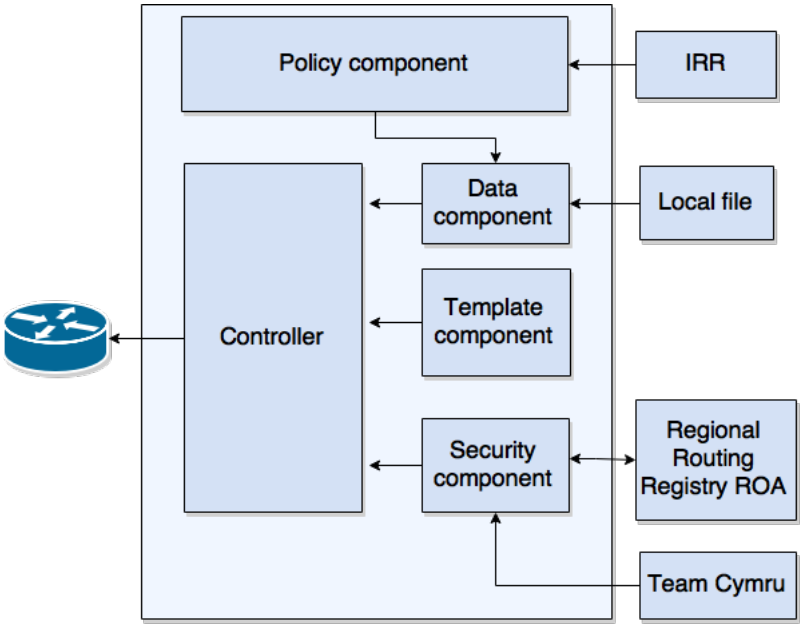

Figure 9: BGPWizard architecture

After formulating the input and output of BGPWizard in sections 4.2 and 4.3, now we describe the components of the tool:

- Policy component. The policy component is split in two parts, as shown in figure 10 - extraction and parsing.

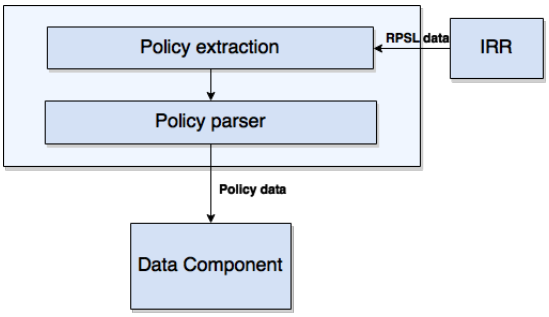

Figure 10: Policy component

The extraction is achieved by using the RIPE IRR as an input, querying the data and retrieving RPSL, enclosed in XML. The parser is necessary, because RPSL is returned in lines, where each (of the attributes we need) line represents an import or export policy. The parser then takes the individual elements of a policy, such as a neighbor AS-number, action, accept or announce, and sends them to the data component.

- Data component. The data component acts as a data handler, as it receives the elements from the IRR and the local file, and constructs XML documents that can be parsed easier and faster than the RPSL/YAML data. The XML data is then sent to the controller when requested by the controller. We note that the local file is parsed using a YAML parser, which is omitted from the general architecture picture, as YAML is not the mandatory choice for the local file and may, in the future, be replaced.

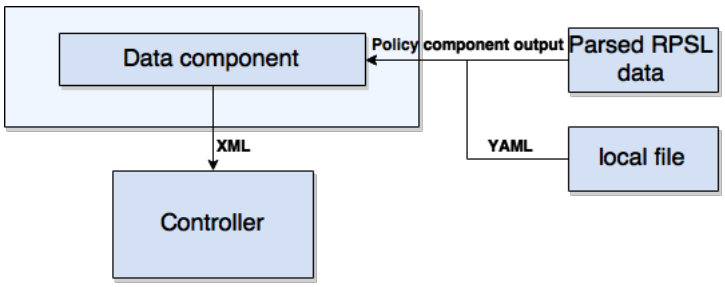

Figure 11: Policy component

- Template component. The template component is responsible for generating templates for different vendor-specific configurations. Figure 12 shows that the template component is called by the controller that passes a specific data and a certain template is returned (a neighbor template, for example).

  The templates generation is split into functional parts - access control part, neighbor configuration part, etc. This is done for several reasons:

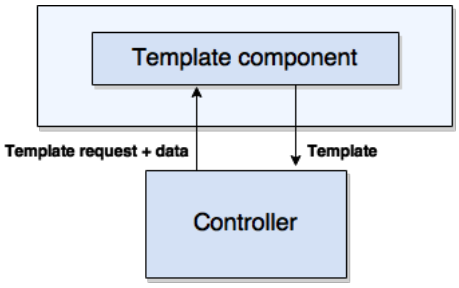

Figure 12: Template component

  - Code re-use - The code for creating one function can be re-used to create the same function for a different policy.
  - Flexibility - The use of generated templates facilitates the configuration of many devices without the need to change the code. This is also one of the reasons NETCONF was chosen as an output. Templates for NETCONF can be used well on one or on hundred devices without issues. If new functions have to be implemented, then they can easily be extended in the templates and integrated in the main code.
  - Vendor independence - Different vendors have different syntax and various criteria on how configuration should be structured in order to be pushed to the device. This means that the templates for each vendor should be separated and called upon depending on the choice of equipment.

- Security component. Many security practices were discussed in 2.2. Those practices were created to solve security related issues with BGP configurations. We consider that automated configuration is not complete without these practices. Therefore, we included one mandatory and one optional filter:

  - Martian/Bogon filtering - We take the full bogon prefixes (as described by Team Cymru). Even though bogon filtering can be expressed in RPSL, we consider that it should be mandatory in any case and that including it in the IRR would only complicate the policy.
  - (Optional) RPKI validation - Currently, RPKI is not included in RPSL. There is an IETF Internet Draft for the integration of RPKI in RPSL [56], but this is not implemented by the IRRs yet. There is a possibility to perform validation of an AS-prefix pair using the RIPE RPKI validator [57]. We defined filtering of RPKI invalid routes as optional, because we consider the implementation of RPKI validation using the validator would require a query for every received prefix and cause overhead that will not be justified, when only a very small amount of the prefixes on the Internet are actually signed.

  The architecture of the Security component is shown in figure 13.

  The Security component takes the list of prefixes, returned either by Team Cymru or by the ROA (if RPKI validation is used) and passes them to

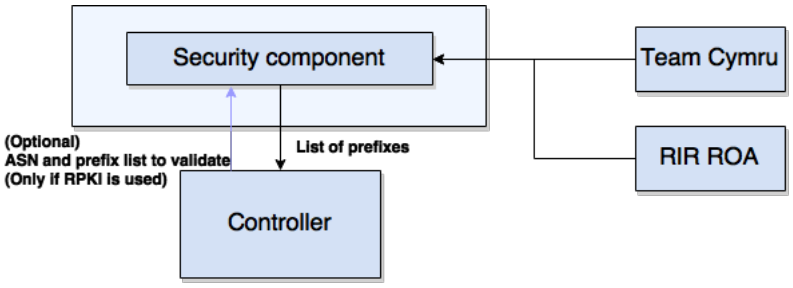

Figure 13: Security component

the controller, when requested. The RPKI validation would require extra data (ASN and prefix for validation) to be sent from the controller first.

- Controller. As the name implies, this is the component that merges all the components. We describe how the controller operates in a step-by-step fashion in figure 14.

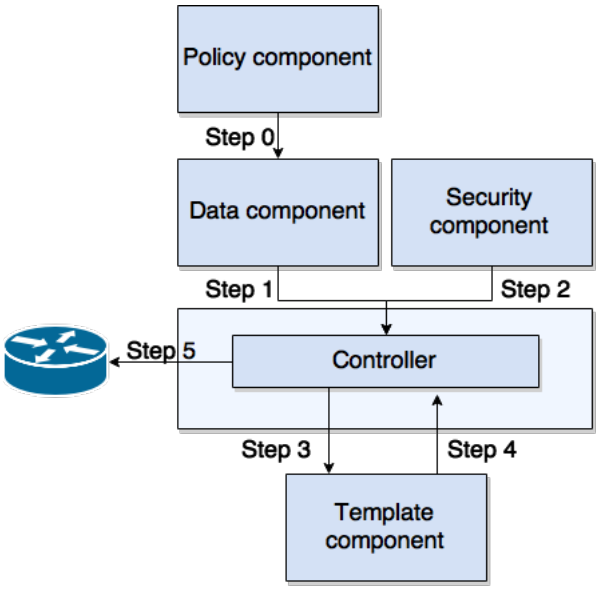

Figure 14: Controller

In figure 14, the following steps are defined:

  - Step 0 - This step was described in the Data component part and is not directly done by the controller, but it is added in the figure for completeness.
  - Step 1 - The controller calls the Data component and receives XML data for the IRR and local file data.
  - Step 2 - The controller then asks the Security component for the bogon list and, if specified in the local file information from step 1,

RPKI information, based on neighbor AS-prefix pairs, extracted from the information in step 1.

- Step 3 - The controller asks the Template component to return templates for a certain data. An example of this can be a policy creation request.
- Step 4 - The Template component returns the requested template to the Controller.
- Step 5 - The controller merges all templates in the order of configuration, required by the concrete router vendor [8] and sends them to the router.

The tool is only concerned with BGP configuration. Setting up Autonomous System numbers, interface addresses and all other forms of router basic configuration is outside the scope of this project.

5.2 Local file architecture

In order to achieve a secure end-to-end automation, the data in IRR is essentially not sufficient. For instance, the automation tool needs to be authenticated by the router, and the tool needs to get credentials externally. Several security features are not supported by the IRR, such as RPKI, maximum prefix filtering, etc. but should be taken into consideration during the automation process. RPSL does support several filters, such as default route, bogon and martian filtering, but writing such policies can end up being a very complex task that produces hardly readable RPSL statements. In addition, some AS owners might not want to publish their private peering information in IRRs. Also, network operators may need to overwrite their policies stored in RPSL for security or simplicity. Those limitations can be overcome by enriching the RPSL data with a local configuration file. Table 5 shows the design of the local file that we propose.

The mandatory attributes of such a file are used as a minimum requirement to perform full automation for a single router along with RPSL data (assuming router IP addresses are included in the IRR). In the case where addresses are excluded, the minimum local file must include the neighbor mandatory requirements as well.

As mentioned in the beginning of this section, we provided a proof of concept implementation for BGPWizard, so we created a partial version of the local file. The greyed attributes in the table were not included in the implementation we created for this project. The only mandatory attribute we omitted was the router vendor, because the proof of concept is created only for Juniper routers. A router can be configured completely without the rest of the greyed options, therefore resulting in their optional status. We do, however, consider that these attributes will come in handy and should be included in a full BGPWizard implementation.

[8] A Juniper router, for example, has to receive neighbor configuration before policy configuration

Attribute	Config type	Appearance	Description
Public IP	Basic	Mandatory	Router interface IP for peering. Can be nested, if multiple routers are configured.
Management IP	Basic	Mandatory	Used for actual connection to router.
(Local) ASN	Basic	Mandatory	AS number of the router.
Router name	Basic	Mandatory	Router hostname.
Router vendor	Basic	Mandatory	Router vendor.
Username and password	Basic	Mandatory	Router credentials used by NETCONF.
Neighbor IP	Neighbor	Mandatory if not in IRR	Peer IP address. Can be nested for multiple neighbors or one neighbor with multiple peering addresses.
(Neighbor) ASN	Neighbor	Mandatory if neighbor IP defined	Neighbor's AS number.
Peering group	Neighbor	Optional	BGP group that the neighbor belongs to. If not specified, a general name is used by the tool.
Logical system	Neighbor	Optional	If the router uses logical systems for multiple BGP instances.
Local Preference	Policy	Optional	Applied to inbound external routes, dictating the best outbound path. Has priority if the same information is stored in the IRR.
MED	Policy	Optional	Applied to outbound routes, dictating the best inbound path into the AS. Has priority if the same information is stored in the IRR.
Community	Policy	Optional	Allows routes to be tagged into certain communities.
No import	Policy	Optional	List of excluded inbound routes.
No export	Policy	Optional	List of excluded outbound routes.
RPKI	Security	Optional	RPKI validation.
RPKI action	Security	Optional	Discards or sets local pref for invalid routes.
Max-prefix	Security	Optional	Maximum number of prefixes accepted from a neighbor.
Default route	Security	Optional	Default route filtering.

Table 5: Attributes of local configuration file

The local file of BGPWizard has a parent-child structure, an example of which is shown in Figure 15. The figure contains the mandatory router and neighbor attributes, along with several optional attributes. BGPWizard uses a YAML format for human readability and ease of use. As mentioned earlier, we also provide a proof of concept implementation for BGPWizard, which will be shown and used in section 6.

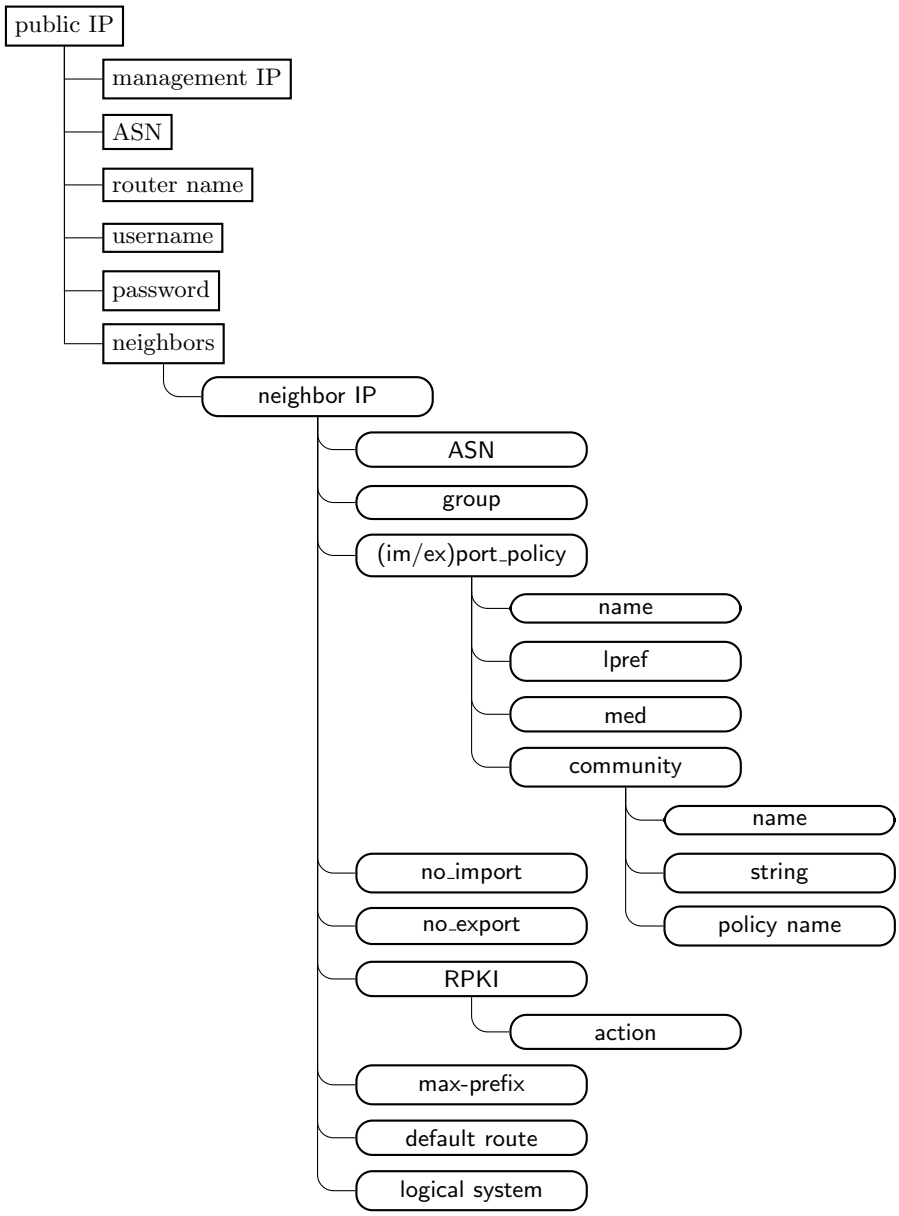

Figure 15: Local file parent-child structure

5.3 Proof of Concept

We developed an implementation of BGPWizard as a proof of concept. The code itself is written in Python and the rest of the implementation is discussed in this sub-section.

5.3.1 Description of the functions

Up until now, we described the desired architecture and functionality. Here, we will show our implementation and the functionality we managed to cover for the duration of this project.

XML libraries
The policy component is responsible for extracting RPSL data using HTTP and RIPE's REST WHOIS API, and parsing the collected data. We decided to write our own parser for this. It returns XML structured information with all the information, needed from RPSL for router configuration. The returned XML has the following structure, where the values are shown between [ ] and the optional attributes are shown in red in figure 16.

```
<root>
  <imports>
    <import from="[ASN]" local_ip="[IP]" remote_ip="[IP]">
      <action>
            <pref></pref>
            <med></med>
            <community></community>
            </action>
      <accept> </accept>
    </import>
  </imports>
  <v6imports>
    <import from="[ASN]" local_ip="[IP]" remote_ip="[IP]">
      <action>
            <pref></pref>
            <med></med>
            <community></community>
            </action>
      <accept> </accept>
    </import>
  </v6import>
  <exports>
    <export for="[ASN]" local_ip="[IP]" remote_ip="[IP]">
      <action>
            <pref></pref>
            <med></med>
            <community></community>
            </action>
      <announce> </announce>
    </export>
  </exports>
  <v6exports>
    <export for="[ASN]" local_ip="[IP]" remote_ip="[IP]">
      <action>
            <pref></pref>
            <med></med>
            <community></community>
            </action>
      <announce> </announce>
    </export>
  </v6exports>
</root>
```

Figure 16: XML structure of parsed RPSL data

Table 6 shows the XML structure that was implemented for this project. The areas in grey indicate the tags that can be used only once.

Tag	Parent(s)	Appearance	Description
imports	root	Mandatory	Holds all IPv4 import policies. If none are found in RPSL, it stays empty.
v6imports	root	Mandatory	Holds all IPv6 import policies. If none are found in RPSL, it stays empty.
exports	root	Mandatory	Holds all IPv4 export policies. If none are found in RPSL, it stays empty.
v6exports	root	Mandatory	Holds all IPv6 export policies. If none are found in RPSL, it stays empty.
import	imports v6imports	Optional	Holds one import policy with the neighbor ASN as an attribute. If router IP addresses are included in RPSL, they are added as attributes.
export	exports v6exports	Optional	Holds one export policy with the neighbor ASN as an attribute. If router IP addresses are included in RPSL, they are added as attributes.
accept	import	Mandatory for import	Holds one import policy.
announce	export	Mandatory for export	Holds one export policy.
action	import export	Optional	Holds one import policy. Must have at least one child - either pref, med or community
pref	action	Optional	Inverse local-preference, stored in the action attribute.
med	action	Optional	MED, stored in the action attribute.
community	action	Optional	Community in the action attribute.

Table 6: XML components of parsed RPSL data

The XML handler is also used to parse the local file and return XML for ease of search.

Templates generation library
Our implementation was created for a Juniper router, but can easily be extended to other vendors. We decided that we want flexibility without unnecessary functionality, so we used NCClient (which we mentioned in 3.2.4). We developed a template generator for JunOS that consists of multiple functions - for neighbor

creation, for policy generation, etc. What it also provides is the function to query the route set for an AS, AS-SET or a ROUTE-SET using BGPq3. Here, we chose not to parse the route object ourselves, as BGPq3 was already doing a great job at it.

Security data
The bogon prefix list from Team Cymru is downloaded, parsed and returned as a list of route filters. Each route filter contains one prefix and the statement that it should match the prefix or longer prefixes from the same network.

Controller
The controller parses the XML data from the policy parser and an XML-generated version of the local file for router information. Then it extracts peering relations. It creates a full bogon filter from the list, returned as security data, and applies that filter for all import/exports per neighbor. It is important to mention again that Team Cymru's full bogon filter includes both martians and bogons.

If the accept/announce statements contain an AS, as-set or a route-set, it uses the BGPq3 generation option from the template generator to create aggregated route filters. If the output exceeds the amount of prefixes allowed by the router (which for JunOS 11.4 onwards is 85,325 [58]), then we partially follow the procedure for inbound filtering, as specified in RFC 7454 [14] and include only full bogon filtering and maximum prefix length filtering.

When the complete configuration of a router is generated, NCClient connects to the management IP, as specified in the local file, and pushes the configuration.

5.3.2 Limitations

While designing and testing BGPWizard, several limitations were found. We split those limitations in two parts - design (what is missing in the original plan) and implementation (what we wanted to include, but did not).

Design limitations

The local file does not have a separate method for IPv6 peering. What we could do is add the IPv6 neighbors anyway and just check what version the address is when configuring peers.

This design is made for a router, configured with only one AS-number. Using logical systems, multiple ASes can reside in one device. We consider this is not often the case, but the configuration option should be present as an optional attribute of the router.

The router configuration is, by design, achieved with NETCONF. The issue with the protocol is that currently not all vendors incorporate the standard in their equipment or part of their equipment.

Implementation limitations
Our BGPWizard implementation demonstrates the basic functionality of BGP-Wizard as a proof of concept and as such it does not incorporate the full BGP-Wizard architecture. It can be used for end-to-end automation on simple RPSL

policies. The current parser does not cover the complete RPSL. Statements, such as the RPSL refine, are ignored. The accept and announce attributes are only parsed if they contain one or more AS, as-set or a route-set, or if the value is ANY.

Two of the design features were also left out of the proof of concept, namely IPv6 support and vendor independence. IPv6 is not difficult for integration with the tool and is left as future work.

Earlier, in the local file design in table 5, we greyed out the areas that we could not include in our implementation. There are currently no syntax or structure checks for the local file.

Since we chose to use BGPq3 for our implementation, we inherited one of the tool's limitations, namely the size of the output. When long lists are generated, the output may not fit and the TCP window size has to be increased in the OS. Such a 'tweak' is also recommended for our implementation of BGPWizard and is described in the BGPq3 README information [9]. In case of a long output from BGPq3, as mentioned earlier, we only add full bogon and maximum length filtering. Additionally, local AS prefixes must be filtered as well. This can be considered both design and implementation limitation and we recommend that local AS prefix filtering is included in future work on the tool.

We also aimed for atomic transaction level of configuration. We managed to implement it on a per router basis. A scenario we would ideally like to have would be to configure all routers atomically, so if one configuration fails, all configurations are rolled back.

For security, we wanted to integrate RPKI validation. We installed RIPE's RPKI validator on an external server. We wanted to verify that when BGPq3 returns a list of prefixes, those prefixes are valid for the AS that announces them. We designed a recursive function for parsing as-sets to get all the members. As described in section 4.4, querying the database has a limit. During the tests we performed, we reached this limit several times just by running a recursion on nested as-sets to get all the AS members and their prefixes. Therefore, we did not include validation in the tool. We do consider that this form of validation is still possible for small ASes that mostly accept ANY and do not require many queries of the IRR database. However, we preferred avoiding the risk of a bigger AS attempting to configure a router and ending up not only without configuration, but locked out of the IRR for a day.

6 Experiments and Results

To test our implementation of BGPWizard, we created a testbed and ran several tests for two scenarios.

6.1 The testing environment

To test BGPWizard, we came up with the following setup:

- JunOS Olive routers - JunOS Olive is an open source software that runs on top of FreeBSD and is used to emulate a Juniper router. It is mostly used with the purpose of learning JunOS configurations. The tests for

[9] https://github.com/snar/bgpq3

this project used several VirtualBox VMs with JunOS Olive on top of FreeBSD to play the role of real routers.

- Ubuntu Desktop 14.04 - A Ubuntu VirtualBox VM was used as a control server that runs BGPWizard.
- GNS3 - Used to setup a topology and connect the routers and the control server.
- Lenovo Z50-70 laptop with Intel(R) Core(TM) i7-4510U CPU 2.00GHz, RAM 8GB, 64-bit Windows 8.1 - Used as a VirtualBox host.

6.2 Configuration scenarios

After creating the test environment, we designed several tests for our implementation. All peering relationships, used in the following scenarios, are extracted from RPSL. All addresses are real and are used only for example and testing purposes. This is a closed environment with no access from the Internet. In every test, we show partial local file configuration. It is important to note that there is one local file that contains all the configurations.

6.2.1 Scenario 1

Scenario 1 involves one empty router that will act as a RIPE NCC router. The goal of this scenario is to present a router configuration that includes only the mandatory local file attributes. The rest of the data is included in the IRR. The router has no policy or BGP neighbor configuration. It does, however, have the AS number set to 3333 and the IP addresses of the local interfaces. The topology of the scenario is shown in figure 17.

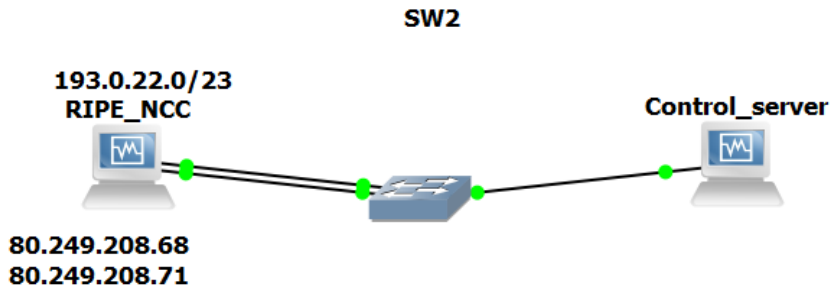

Figure 17: One router

Figure 17 shows that RIPE uses two public IP addresses for peering. Additionally, we added one of RIPE's networks as a loopback interface. The router IP and the neighbor IP addresses were included in RIPE's aut-num object import and export attributes so we used them for this scenario. So the only information we had to include in the local file was the router management IP address and authentication information, see figure 18. Since we use only one router with two

interfaces, we added only one management interface (not shown in the topology for simplicity purposes).

```
- public_ip: 80.249.208.71
  mgmt_ip: 10.0.0.1
  asn: 3333
  name: RIPE
  username: root
  password: password.1

- public_ip: 80.249.208.68
  mgmt_ip: 10.0.0.1
  asn: 3333
  name: RIPE
  username: root
  password: password.1
```

Figure 18: Minimum local file example

Before we run the tool, it is important to note that this is a single router, so no actual neighbor relationships will be established. Also, no BGP groups are specified. The tool automatically adds a single BGP group in such a case. In a Juniper router, only one local address can belong to one group. Therefore, here only one address will be added. If two routers were used, then both would be configured. Since this is only a proof of concept, written for a short period of time, we did not attempt to change this. The issue can be solved by including different group names in the local file (group names example is included in the next scenario).

We ran the tool and verified that the neighbor configuration is added in figure 19.

```
root# show protocols bgp
group qgroup {
    type external;
    local-address 80.249.208.71;
    neighbor 80.249.209.47 {
        import [ BOGON_FILTER4 8218_import_80.249.209.47 ];
        export [ BOGON_FILTER4 AS-RIPENCC ];
        peer-as 8218;
    }
    neighbor 80.249.208.200 {
        import [ BOGON_FILTER4 12859_import_80.249.208.200 ];
        export [ BOGON_FILTER4 AS-RIPENCC ];
        peer-as 12859;
    }
    neighbor 80.249.208.90 {
        import [ BOGON_FILTER4 8608_import_80.249.208.90 ];
        export [ BOGON_FILTER4 AS-RIPENCC ];
        peer-as 8608;
    }
    neighbor 80.249.208.198 {
    import [ BOGON_FILTER4 28878_import_80.249.208.198 ];
    export [ BOGON_FILTER4 AS-RIPENCC ];
    peer-as 28878;
    }
    [....]
```

Figure 19: Result RIPE neighbor addition (partial output)

To verify the result, we looked into the output, produced by Whois, as shown in figure 20, where some of the output is omitted due to the fact that RIPE has over a hundred neighbors. Only the records that correspond with the result in figure 19 are shown. Note, the order here is different, because of the sorting in the router and the IRR.

On the output of figure 19 we see that a bogon filter is added inbound and outbound for all neighbors. The other import filters contain a local preference, as it is specified in the RPSL import statement. RIPE announces the as-set AS-RIPENCC to its neighbors. Two of these filters can be seen in figure 21.

This scenario showed that when most information is included in RPSL, it is possible to configure a router with minimum manual efforts.

```
    import:          from AS8218
                     80.249.209.47 at 80.249.208.71
                     action pref=100; accept ANY
    export:          to AS8218
                     80.249.209.47 at 80.249.208.71
                     announce AS-RIPENCC
    [...]
    import:          from AS12859
                     80.249.208.200 at 80.249.208.71
                     action pref=100; accept ANY
    export:          to AS12859
                     80.249.208.200 at 80.249.208.71
                     announce AS-RIPENCC
    [...]
    import:          from AS8608
                     80.249.208.90 at 80.249.208.71
                     action pref=100; accept ANY
    export:          to AS8608
                     80.249.208.90 at 80.249.208.71
                     announce AS-RIPENCC
    [...]
    import:          from AS28878
                     80.249.208.198 at 80.249.208.71
                     action pref=100; accept ANY
    export:          to AS28878
                     80.249.208.198 at 80.249.208.71
                     announce AS-RIPENCC

    [....]
```

Figure 20: RIPE Whois result (partial output)

```
root# show policy-options policy-statement AS-RIPENCC
from {
    route-filter 193.0.0.0/21 exact;
    route-filter 193.0.10.0/23 exact;
    route-filter 193.0.12.0/23 exact;
    route-filter 193.0.18.0/23 exact;
    route-filter 193.0.20.0/22 prefix-length-range /23-/23;
    route-filter 193.0.24.0/21 exact;
}
then accept;
[edit]
root# show policy-options policy-statement 8218_import_80
    .249.209.47
then {
    local-preference 65435;
}
[edit]
```

Figure 21: Result RIPE filter generation

6.2.2 Scenario 2

Often, we see that not all information (especially IP addreses of the routers) is included in the IRR. Scenario 2 adds two more routers to the topology that act as a KPN (AS286) and a SURFnet (AS1103) router. The purpose of this scenario is to show a slightly extended version of the local file that now includes basic neighbor configuration. It is important to note again that there is one local file that contains all the configurations. For simplicity, here we show only the relevant parts of the file. This is basically the case when all the policy is expressed in RPSL, but the local and neighbor router IP addresses are missing. Both routers have no prior policy or BGP neighbor configurations, but have AS and interface configurations. Management interfaces are omitted. The extended topology can be seen in figure 22.

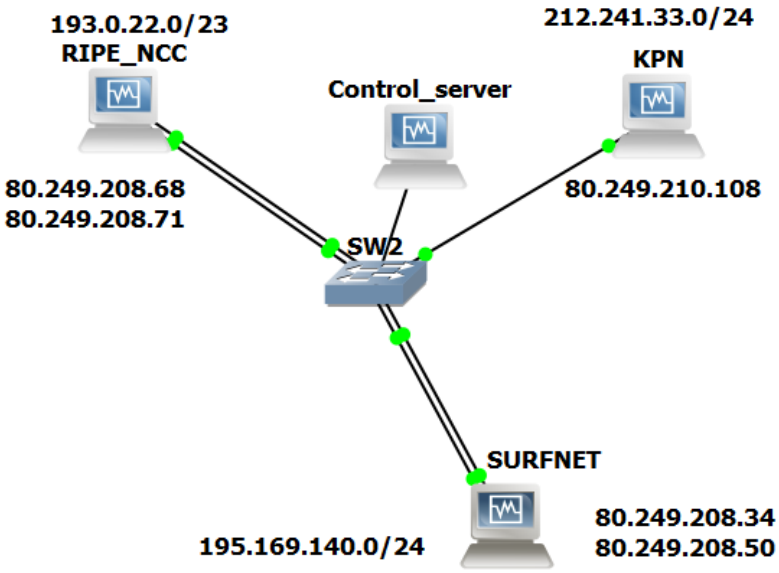

Figure 22: Three routers

On the figure we see that the RIPE NCC router configuration has not changed since scenario 1. The KPN router has one interface configured for BGP, and that is 80.249.210.108. It also has a loopback interface that holds the 212.241.33.0/24 network. The SURFNET router has two interfaces that participate in BGP, namely 80.249.208.34 and 80.249.208.50, and a loopback with network 195.169.140.0/24.

All the loopback networks are mapped to the real networks, announced by the organizations and are used for verification whether routes are received. As this setup uses only RPSL peering, the following relationships were extracted:

- RIPE's policy defines a peering relationship with SURFnet and with KPN, where RIPE accepts ANY and announces AS-RIPENCC set.
- SURFnet defines a peering relationship with RIPE and KPN. To RIPE, it announces ANY and accepts AS3333 AS2121. From KPN, it accepts AS-KPN and announces AS-SURFNET.

- KPN's policy, however, defines only IPv6 peering with RIPE (so no peering on IPv4 is configured here). To SURFnet, it announces ANY and accepts AS-SURFNET.

For this scenario, we will make two tests.

- Configure the KPN and SURFnet routers, and check whether the routes are properly announced and accepted.
- Simulate KPN attempt to announce a martian prefix and verify that the filters work as planned.

Test 1
Unlike the RIPE RPSL policy, KPN and SURFnet's RPSL policies do not contain neighbor IP addresses. Therefore, we introduce the following local file for both organisations in figure 23.

```
- public_ip: 80.249.210.108
  mgmt_ip: 10.0.0.2
  asn: 286
  name: KPN
  username: root
  password: password.1
  neighbors:
          - 80.249.208.34:
                as: 1103
          - 80.249.208.50:
                as: 1103
- public_ip: 80.249.208.34
  mgmt_ip: 10.0.0.3
  asn: 1103
  name: SURFnet
  username: root
  password: password.1
  neighbors:
          - 80.249.208.71:
                as: 3333
          - 80.249.208.68:
                as: 3333
          - 80.249.210.108:
                as: 286
- public_ip: 80.249.208.50
  mgmt_ip: 10.0.0.3
  asn: 1103
  name: SURFnet
  username: root
  password: password.1
  neighbors:
          - 80.249.208.71:
                as: 3333
          - 80.249.208.68:
                as: 3333
          - 80.249.210.108:
                as: 286
```

Figure 23: Minimum local file neighbor example

The IP addresses of the routers were specified in the RIPE RPSL data, so we used them as router IP addresses within the local file. We ran the tool to configure the two routers.

On the SURFnet router we verified that the policies and neighbors have been added as shown in figure 24. We can also see that the bogon filter is also included.

```
root# show policy-options policy-statement ?
Possible completions:
  <policy_name>          Name to identify a policy filter
  ANNOUNCE_ANY           Name to identify a policy filter
  AS-KPN                 Name to identify a policy filter
  AS-SURFNET             Name to identify a policy filter
  AS2121                 Name to identify a policy filter
  AS3333                 Name to identify a policy filter
  BOGON_FILTER4          Name to identify a policy filter
[edit]
root# show protocols bgp
group qgroup {
    type external;
    local-address 80.249.208.34;
    neighbor 80.249.208.68 {
        import [ BOGON_FILTER4 AS3333 AS2121 ];
        export [ BOGON_FILTER4 ANNOUNCE_ANY ];
        peer-as 3333;
    }
    neighbor 80.249.208.71 {
        import [ BOGON_FILTER4 AS3333 AS2121 ];
        export [ BOGON_FILTER4 ANNOUNCE_ANY ];
        peer-as 3333;
    }
    neighbor 80.249.210.108 {
        import [ BOGON_FILTER4 AS-KPN ];
        export [ BOGON_FILTER4 AS-SURFNET ];
        peer-as 286;
    }
}
```

Figure 24: SURFnet router result

The routing table of the SURFnet router is shown in figure 25 and we can see that it has received all the routes from RIPE and KPN.

```
root> show route protocol bgp

inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0
    hidden)
+ = Active Route, - = Last Active, * = Both

80.249.0.0/16       [BGP/170] 02:11:56, localpref 100
                      AS path: 286 I
                    > to 80.249.210.108 via em1.0
193.0.22.0/23       *[BGP/170] 02:30:52, localpref 100
                      AS path: 3333 I
                    > to 80.249.208.71 via em1.0
212.241.33.0/24     *[BGP/170] 02:11:56, localpref 100
                      AS path: 286 I
                    > to 80.249.210.108 via em1.0
212.241.33.1/32     *[BGP/170] 02:11:56, localpref 100
                      AS path: 286 I
                    > to 80.249.210.108 via em1.0
```

Figure 25: SURFnet router routing table

The results of the RIPE and KPN router are omitted here, but can be found in Appendix F.

The results of Test 1 show that the configuration is successfully added and works. Now, we test if the bogon filter does its job.

Test 2
In Test 2, KPN accidentally announces 192.168.1.0/24 network, which is a martian network and should not be accepted by SURFnet. The network is added as a loopback interface in the KPN router as shown in figure 26.

```
root# show interfaces lo0
unit 0 {
    family inet {
        address 212.241.33.1/24;
        address 192.168.1.1/24;
    }
}
```

Figure 26: Example misconfiguration setup

Since the bogon list is also applied when exporting prefixes, we verify that KPN does not announce the private network to its neighbor SURFnet in figure 27.

```
root> show route advertising-protocol bgp 80.249.208.34

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0
    hidden)
  Prefix                  Nexthop              MED     Lclpref
        AS path
* 80.249.0.0/16           Self
                                        I
* 212.241.33.0/24         Self
                                        I
* 212.241.33.1/32         Self
                                        I
```

Figure 27: Outbound bogon filter result

Now, if we consider that KPN did not configure their network using BGP-Wizard and forgot to add the outbound filter (which we simulated by removing the filter from the import/export neighbor statements. We verified again if the prefix is indeed announced by KPN (figure 28). Even worse, now a /32 is announced, because of the loopback address.

```
root> show route advertising-protocol bgp 80.249.208.34

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0
    hidden)
  Prefix                  Nexthop              MED     Lclpref
        AS path
* 80.249.0.0/16           Self
                                        I
* 192.168.1.0/24          Self
                                        I
* 192.168.1.1/32          Self
                                        I
* 212.241.33.0/24         Self
                                        I
* 212.241.33.1/32         Self
                                        I
```

Figure 28: Outbound misconfiguration example

We checked if the import bogon filter works on SURFnet's side and placed the result in figure 29.

```
root> show route protocol bgp

inet.0: 10 destinations, 12 routes (8 active, 0 holddown, 2
    hidden)
+ = Active Route, - = Last Active, * = Both

80.249.0.0/16       [BGP/170] 02:27:39, localpref 100
                      AS path: 286 I
                    > to 80.249.210.108 via em1.0
193.0.22.0/23       *[BGP/170] 02:46:35, localpref 100
                      AS path: 3333 I
                    > to 80.249.208.71 via em1.0
212.241.33.0/24     *[BGP/170] 02:27:39, localpref 100
                      AS path: 286 I
                    > to 80.249.210.108 via em1.0
212.241.33.1/32     *[BGP/170] 02:27:39, localpref 100
                      AS path: 286 I
                    > to 80.249.210.108 via em1.0
```

Figure 29: Inbound bogon filter result

Indeed, the private network was filtered and SURFnet did not accept the bogus announcement. The story in the last scenario is, of course, just an example and it is not involved with KPN's filtering practices.

The goal of these two experiments was to show that the proof of concept for BGPWizard really works. We demonstrated that an implementation of BGPWizard works as intended. However, results of the tool cannot be drawn solely by this criteria. We also measured the performance of the tool for the setup in scenario 2.

6.3 Performance analysis

We ran four tests on two different routers (for one public IP address per router). The tests were conducted on the first N neighbors in the RPSL policies, where N is specified in every test. Each test used timestamps to measure the duration and was executed 20 times with a 10Mbit/s Internet speed. The speed here is considered, because of the necessary queries for the RIPE database. The results are presented in the graph where the duration of a run (in seconds) and the probability density function represent the x and y axis values, respectively. We also provide the mean and standard deviation of the results in each test.

6.3.1 Test one

Test one was conducted on the RIPE router we configured in scenario 1. The purpose of this test is to measure the speed of the BGPWizard implementation using public RPSL data and minimum local file intervention. This test is run only on for peering on one of the local router IP address.

Included in this test are:

- one bogon filter, generated by the tool

- one route filter, generated for AS-RIPENCC by BGPq3, as this is the as-set that RIPE announces to the neighbor
- one local-preference policy, as defined for all RIPE's peers
- one neighbor creation

The result of the first test can be seen in figure 30:

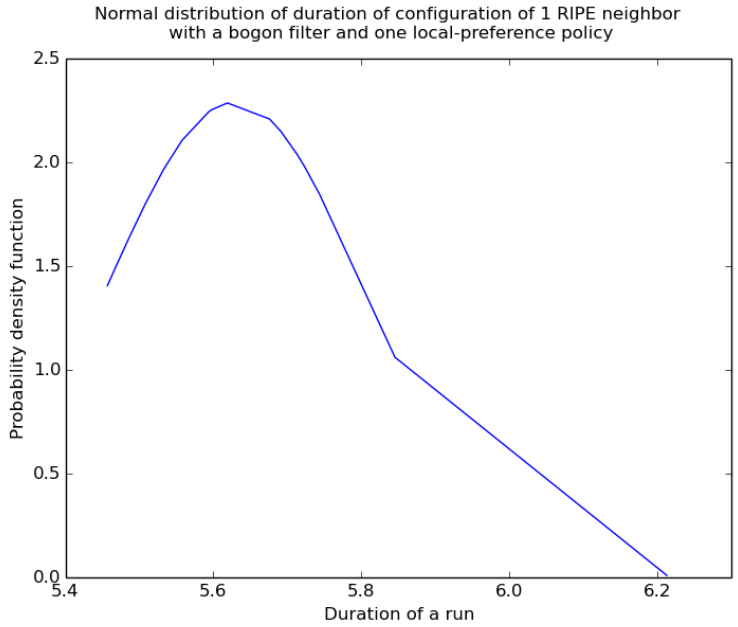

Figure 30: 1 RIPE neighbor

Where the mean time (in seconds) and standard deviation were:

```
mean time = 5.6289134
standard deviation = 0.17431099999
```

We see that the mean run time is approximately 5.6 seconds for one neighbor. Important thing we must consider here is that this mean time will be the same per neighbor if multiple neighbors are configured. The reason for this is that only one REST WHOIS query is required for the AS-number, independent of the amount of neighbors. The local file and the parsed RPSL file are also loaded only once. Policy-wise, an AS usually announces the same prefixes to all of its neighbors. This means that the filter for AS-RIPENCC can be created only once and used many times. The same holds for the bogon filter.

To see how these two filters influence the run time, we ran the test only for them and got the following mean duration in seconds:

```
BGPq3 query for AS-RIPENCC - 0.65459025 seconds
Bogon filter query - 0.96464545 seconds
```

We also ran a separate Whois query for RIPE's AS 3333 using HTTPS and the REST API, as in the tool and got a mean query time of 0.87022815 seconds.

As described, many phases will be used once, independent of the amount of neighbors that are configured. We conclude that test one is not sufficient as a performance test. Therefore, to get a more realistic view of the performance, we ran the same test with 172 neighbors in test two.

6.3.2 Test two

We performed the same test as in test one, but this time we ran it for 172 neighbors. This means that the test includes:

- one bogon filter, generated by the tool
- one route filter, generated for AS-RIPENCC by BGPq3, as this is the as-set that RIPE announces to the neighbors
- 172 local-preference policies, as defined for all RIPE's peers
- all 172 IPv4 peers for address 80.249.208.71 (as defined in RIPE IRR)

The test result is shown in figure 31.

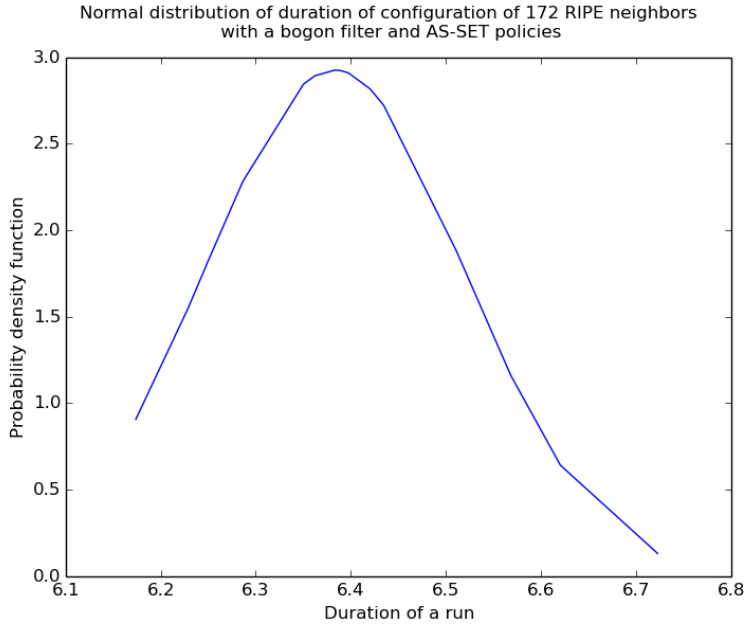

Figure 31: 172 RIPE neighbors

The mean run time in seconds and the standard deviation for this test were:

```
mean time = 6.3826165
standard deviation = 0.136329732634
```

What this test shows is that the mean time has increased by approximately 0.75 seconds. This result is more realistic in terms of neighbor configuration time, as it shows that if we exclude the code that is run only once per router, we end up with a very fast neighbor setup.

What this test does not consider is when imports include AS or as-set prefixes. Therefore, we ran two more tests.

6.3.3 Test three

Test three is basically the same as test one, but this time on the KPN router. Unlike RIPE, KPN accepts only specific prefixes, resulting in the extra querying from BGPq3. Here, we used the peering relationship between KPN and AS5400 (British Telecommunications), because their corresponding as-sets (namely AS-KPN and AS-BT-EU) contain a lot of prefixes. Basically, we demonstrate a bad-case scenario, where two queries have to be made for quite big as-sets. The result is shown in figure 32:

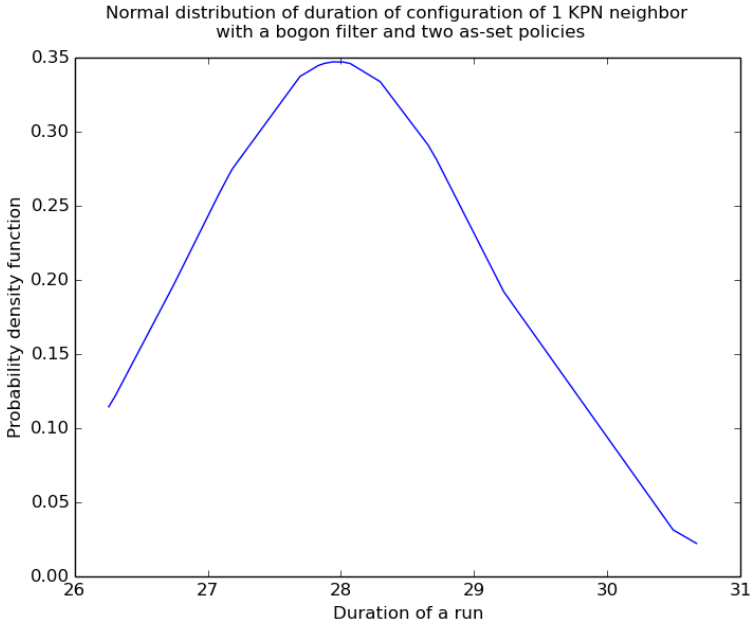

Figure 32: 1 KPN neighbor + import/export AS-SET policies

Here, we see quite different mean value:

```
mean time = 27.97105855
standard deviation = 1.14942173312
```

There are two reasons for this. First, a BGPq3 query to AS-KPN returns significantly longer output than AS-RIPENCC, and second, an extra query is made for the neighbor prefixes. It is also important to mention that the local file now contains more information, than with the RIPE configuration.

6.3.4 Test four

This test we performed is the same as test two, but this time with the KPN router. Now we have 50 neighbor configurations that have 50 additional and unrepeated BGPq3 queries (one for each neighbor). Since the addition of a

neighbor here requires quite a large local file, we limited the test to 50 peers. The results of this test are shown in figure 33:

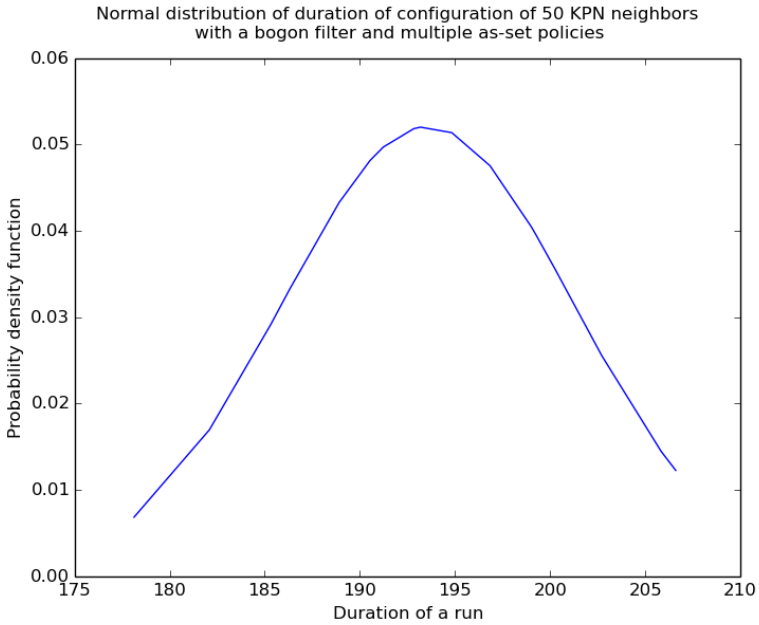

Figure 33: 50 KPN neighbors with bogon filtering and AS-SET policies

The mean time and standard deviation of the test:

```
mean time = 193.56562835
standard deviation = 7.66015881826
```

The mean time for 50 peer configurations is approximately 3,14 minutes. It has increased significantly, compared to the increase in test two (which also contained more peers), but this is still a very acceptable duration. After this test we conclude that running the implementation of BGPWizard is feasible for real-live scenarios, but we recommend that the usage includes higher bandwidth than 10Mbits/s, so that queries to the IRR database will be sent faster.

7 Conclusion

We collected and analysed the currently existing tools, such as IRRToolSet, IRR Power Tools, etc. The result of the analysis was that most of these tools are currently unreliable as they cannot keep up with the latest BGP practices, let alone with the security trends.

The current technologies can be used for BGP automation only to the extent of configuring a BGP router info from scratch (IRRToolSet), adding (partial) filters as done by BGPq3 or a simple visual comparison. In this report, we proposed several ideas on how to overcome some of the limitations.

Our conclusion is that BGP can only be completely automated if all the information is public and routers are configured from scratch with the public

data. This is, however, difficult, because there are organizations that do not want to make all their peering publicly known. Even if the peering information is public, other issues can arise. For example, some organisations receive their routers pre-configured. Automating the update process of such a router using public information cannot occur without prior knowledge of the running configuration. In other words, predicting the router configuration will be very difficult (maybe even impossible), as one configuration can have different setup (for example, a filter can be created with a prefix-list, route-filter, etc.). Therefore, intervention from administrators will be required as long as BGP is used as the Internet routing protocol.

BGP is widely deployed at the moment, so it will be hard to introduce a new protocol. However, we assume that in the future, a better (or less prone to errors) protocols will be used.

8 Future work

We have several ideas of future extension of the local file. The local file was originally designed to include private peering information. This can be achieved by adding a few more attributes to the file, such as accept and announce, and an attribute that indicates that RPSL data should not be used. We would also like to see other filters included in the local file, such as as-path filtering, flap damping, etc., but we have not researched these filtering mechanisms any further.

Several parts of this report discussed limitations involved in some way around the tool. Whether they are external or design/implementation issues, some can be overcome with some time and effort. We would like to suggest that such a tool is integrated into a big management system, such as NCS (mentioned in 3.2.3, where we discussed the feasibility of integrating NCS to BGPWizard). The tool can then not only help automate the configuration process, but also make use of the vendor-independent configuration options from the management system.

Appendix A RPSL usage

The usage of RPSL is described in RFC 2650 [13]. This section provides a short overview of the strategies, mentioned in the RFC. In RPSL, the BGP peering policies are presented in the import/export attributes of the AUT-NUM object. The policies, described in [13] are shown in table 7.

Policy	Accepts	Announces
Provider-Customer	Only announcements, originated from the customer AS/prefix	Full routing table
Provider-Transit	Full routing table	Only own and customer prefixes

Table 7: Policies from RFC 2650

Table 7 shows two policies - provider to customer and provider to transit.

An example of the provider-customer policy is shown in figure 34, where AS1 is the provider and AS2 is the customer.

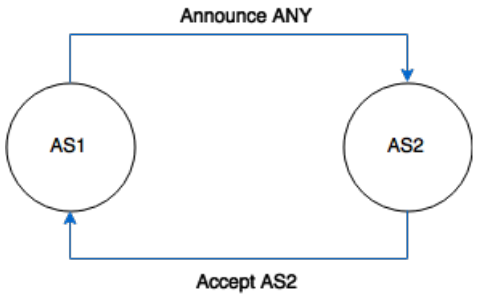

Figure 34: Provider-Customer policy example

The example from figure 34 can be translated into the following RPSL policy for AS1 (provider):

```
import:     from AS2 accept AS2
export:     to AS2 announce ANY
```

Here, the import attribute describes what AS1 accepts from its customer AS2. In other words, AS2's announcements are approved only if they originate from AS2 itself. To AS2, AS1 exports its full routing table. Since AS2 uses AS1 to connect to the rest of the Internet (hence, the customer-provider relationship), AS2 accepts all the routes from AS1. This example is only a simple scenario of provider-customer policy.

Figure 35 extends figure 34 to accommodate a provider to transit relationship. AS3 is added as a transit peer to AS1. AS1 accepts all routes, sent from AS3, but it announces only routes that originate from itself or its customers.

The RPSL translation of this scenario is shown in the snippet below:

```
import:     from AS3 accept ANY
export:     to AS3 announce AS1 AS2
```

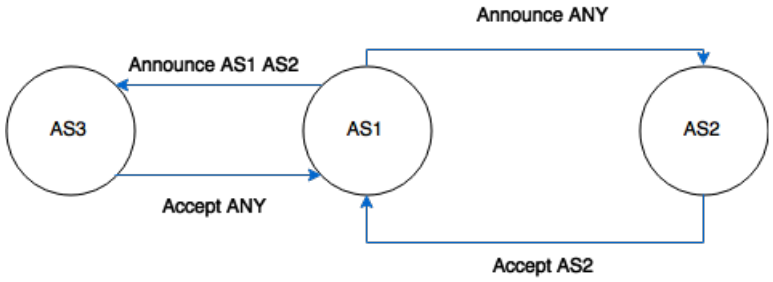

Figure 35: Provider-Transit policy example

These two examples show that there are several difficulties that need to be dealt with when using RPSL. The first issue is scalability. In the examples above, if AS1 adds one more customer, both AS1 and AS3 have to update their records. Instead, an *as-set* object can be created. An AS-SET has the following structure, taken from [59]:

Attribute Name	Presence	Repeat	Indexed
as-set:	mandatory	single	primary/lookup key
descr:	mandatory	multiple	
members:	optional	multiple	
mbrs-by-ref:	optional	multiple	inverse key
remarks:	optional	multiple	
org:	optional	multiple	inverse key
tech-c:	mandatory	multiple	inverse key
admin-c:	mandatory	multiple	inverse key
notify:	optional	multiple	inverse key
mnt-by:	mandatory	multiple	inverse key
mnt-lower:	optional	multiple	inverse key
changed:	mandatory	multiple	
source:	mandatory	single	

The AS-SET attribute name starts with "AS-" and includes the name of the set, which is usually also a description of the set. An example can be AS-CUSTOMERS. The autonomous systems, included in the set, are stored in the *members* field. Following the example from figure 35, if AS1 has AS2, AS4 and AS5 as customers, the AS-SET would look like this:

```
as-set:      AS-CUSTOMERS
members:     AS2 AS4 AS5
```

The set can then be recursively parsed to get al the members. The policy in RPSL can now become more flexible, as only the AS-SET needs to be specified:

```
import:      from AS3 accept ANY
export:      to AS3 announce AS-CUSTOMERS
```

AS1 can also choose to prefer only one prefix, instead of all prefixes (ANY) from an AS or AS-SET. In this case, the accept statement will have a prefix or a set of prefixes, called a *route-set*. In this sense, RPSL is very flexible. Regular expressions can be used to formulate even more concrete accept policies. Regular

expressions in RPSL are beyond the scope of this project and will therefore not be discussed here.

For more fine-grain policy, the *action* attribute is used. It contains information about the local preference, med and community BGP attributes. The local preference is described in the *pref* attribute. Pref is, however, the inverse of the local preference - lower values are preferred:

```
local-preference = 65535 - pref
```

The action RPSL attribute can be used in case of backup connections. The *from* and *to* statements in the import and export attributes correspondingly can also accommodate IP addresses. The following example shows the extended RPSL import record for AS1:

```
import:        from AS3 <AS3-router-IP> at <AS1-router-IP>
    action pref=10; accept ANY
```

This can be useful in multiple cases. First, if different routers are used for the same neighboring relationship, load balancing can be achieved or one link can be used as a backup (in this case by using local preference).

RPSL also allows to describe multi-homing relationships and other access control policies using the *refine* attribute. An example is shown below, where AS-ANY means all autonomous systems:

```
refine: from AS-ANY action pref=40;
                accept community(community_value)
```

Here, for any AS-number the local preference is set if a certain community string is seen.

Appendix B RtConfig Juniper import configuration example

Import statement of RIPE (AS3333) for SURFnet (AS1103) translated to Juniper configuration:

```
root@ubuntu:/home/ubuntu# rtconfig -config junos
rtconfig> @rtconfig import AS3333 80.249.208.71 AS1103
    80.249.208.34
policy-options {
Warning: filter "policy_1103_1" matches ANY/NOT ANY
  policy-statement policy_1103_1 {
      term policy_1103_1-term-1 {
          from {
          }
          then {
              local-preference 900;
              accept;
          }
      }
   }


   policy-statement policy_1103_1 {
      term policy_1103_1-catch-rest {
          then reject;
      }
   }
}

protocols {
   bgp {
      group peer-80.249.208.34 {
          type external;
          peer-as 1103;
          neighbor 80.249.208.34 {
              import policy_1103_1 ;
              family inet {
                  unicast;
              }
          }
      }
   }
}
```

Appendix C RtConfig Cisco import configuration example

Import statement of RIPE (AS3333) for SURFnet (AS1103) translated to Cisco configuration:

```
root@ubuntu:/home/ubuntu# rtconfig
rtconfig> @rtconfig import AS3333 80.249.208.71 AS1103
    80.249.208.34
Warning: filter "MyMap_1103_1" matches ANY/NOT ANY
!
no route-map MyMap_1103_1
!
route-map MyMap_1103_1 permit 1
 set local-preference 900
exit

!
router bgp 3333
!
 neighbor 80.249.208.34 remote-as 1103
 neighbor 80.249.208.34 route-map MyMap_1103_1 in
!
exit
rtconfig> @rtconfig export AS3333 80.249.208.71 AS1103
    80.249.208.34
!
no access-list 150
access-list 150 permit ip 193.0.0.0    0.0.0.0    255.255.248.0
      0.0.0.0
access-list 150 permit ip 193.0.10.0    0.0.0.0    255.255.254.0
       0.0.0.0
access-list 150 permit ip 193.0.12.0    0.0.0.0    255.255.254.0
       0.0.0.0
access-list 150 permit ip 193.0.18.0    0.0.0.0    255.255.254.0
       0.0.0.0
access-list 150 permit ip 193.0.20.0    0.0.2.0    255.255.254.0
       0.0.0.0
access-list 150 permit ip 193.0.24.0    0.0.0.0    255.255.248.0
       0.0.0.0
access-list 150 deny ip 0.0.0.0 255.255.255.255 0.0.0.0
    255.255.255.255
!
no route-map MyMap_1103_1
!
route-map MyMap_1103_1 permit 1
 match ip address 150
exit

!
router bgp 3333
!
 neighbor 80.249.208.34 remote-as 1103
 neighbor 80.249.208.34 route-map MyMap_1103_1 out
```

```
!
exit
```

Appendix D IRR Power Tools example

```
root@IRR:~/irrpt-1.27/bin# ./irrpt_pfxgen 1103
conf t
no ip prefix-list CUSTOMER:1103
ip prefix-list CUSTOMER:1103 permit 129.125.0.0/16 le 24
ip prefix-list CUSTOMER:1103 permit 130.37.0.0/16 le 24
ip prefix-list CUSTOMER:1103 permit 130.89.0.0/16 le 24
[output omitted]
```

Appendix E BGPq3 example

An example prefix-list generation for the 32-bit ASN of NLNetLabs and the AS-SET of SURFnet is shown below:

```
root@ubuntu:/home/ubuntu# bgpq3 AS199664
no ip prefix-list NN
ip prefix-list NN permit 185.49.140.0/22
root@ubuntu:/home/ubuntu# bgpq3 AS-SURFnet
no ip prefix-list NN
[output omitted]
ip prefix-list NN permit 145.100.0.0/15
ip prefix-list NN permit 145.100.64.0/20
ip prefix-list NN permit 145.102.0.0/16
ip prefix-list NN permit 145.102.0.0/23
ip prefix-list NN permit 145.102.2.0/24
ip prefix-list NN permit 145.102.4.0/23
ip prefix-list NN permit 145.102.8.0/22
[output omitted]
```

Appendix F Scenario 2 additional results

The snippets below show the policies and BGP neighbor statements in the KPN router after the configuration with BGPWizard.

KPN router policies:

```
root# show policy-options policy-statement ?
Possible completions:
  <policy_name>          Name to identify a policy filter
  ANNOUNCE_ANY           Name to identify a policy filter
  AS-SURFNET             Name to identify a policy filter
  BOGON_FILTER4          Name to identify a policy filter
[edit]
```

KPN router BGP neighbor configuration:

```
root# show protocols bgp
group qgroup {
    type external;
    local-address 80.249.210.108;
    neighbor 80.249.208.34 {
        import [ BOGON_FILTER4 AS-SURFNET ];
        export [ BOGON_FILTER4 ANNOUNCE_ANY ];
        peer-as 1103;
    }
    neighbor 80.249.208.50 {
        import [ BOGON_FILTER4 AS-SURFNET ];
        export [ BOGON_FILTER4 ANNOUNCE_ANY ];
        peer-as 1103;
    }
}
```

Next, the routing tables of the RIPE and KPN routers after the KPN and SURFnet routers were configured, are shown.

RIPE router routing table:

```
root> show route protocol bgp

inet.0: 9 destinations, 11 routes (9 active, 0 holddown, 0
    hidden)
+ = Active Route, - = Last Active, * = Both

80.249.0.0/16       [BGP/170] 02:22:25, localpref 65435
                      AS path: 1103 I
                    > to 80.249.208.34 via em1.0
195.169.140.0/24    *[BGP/170] 02:22:25, localpref 65435
                      AS path: 1103 I
                    > to 80.249.208.34 via em1.0
195.169.140.1/32    *[BGP/170] 02:22:25, localpref 65435
                      AS path: 1103 I
                    > to 80.249.208.34 via em1.0
212.241.33.0/24     *[BGP/170] 02:03:36, localpref 65435, from
    80.249.208.34
                      AS path: 1103 286 I
                    > to 80.249.210.108 via em1.0
```

```
212.241.33.1/32     *[BGP/170] 02:03:36, localpref 65435, from
    80.249.208.34
                      AS path: 1103 286 I
                    > to 80.249.210.108 via em1.0
```

KPN router routing table:

```
root> show route protocol bgp

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0
    hidden)
+ = Active Route, - = Last Active, * = Both

193.0.22.0/23       *[BGP/170] 01:55:05, localpref 100, from
    80.249.208.34
                      AS path: 1103 3333 I
                    > to 80.249.208.71 via em1.0
195.169.140.0/24    *[BGP/170] 01:55:05, localpref 100
                      AS path: 1103 I
                    > to 80.249.208.34 via em1.0
```

References

[1] Ipv4 address report. http://www.potaroo.net/tools/ipv4/index.html. Accessed on 7 Jul 2015.

[2] RIPE NCC. Request an ipv4 /22. https://www.ripe.net/manage-ips-and-asns/ipv4/request-an-ipv4-22-from-the-last-8. Accessed on 05 Jul 2015.

[3] Lutu et al. - the aftermath of prefix deaggregation. http://www.iij-ii.co.jp/en/lab/researchers/cristel/publications/Lutu-deaggreg-itc25.pdf, 2012.

[4] Tony Bates, Philip Smith, Geoff Huston. Cidr report. http://www.cidr-report.org/as2.0/. Accessed on 13 Aug 2015.

[5] João Luís Sobrinho, Laurent Vanbever, Franck Le, and Jennifer Rexford. Distributed route aggregation on the global network. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 161–172, New York, NY, USA, 2014. ACM.

[6] Massachusetts Institute of Technology. - How YouTube was Hijacked. web.mit.edu/6.02/www/s2012/handouts/youtube-pt.pdf, 2009.

[7] Andree Toonk. Chinese ISP hijacks the Internet. http://www.bgpmon.net/chinese-isp-hijacked-10-of-the-internet/. Accessed on 07 Jul 2015.

[8] Y. Rekhter et al. RFC 4271 - A Border Gateway Protocol 4 (BGP-4). https://www.ietf.org/rfc/rfc4271.txt, January 2006.

[9] Merit Network. Overview of the IRR. http://www.irr.net/. Accessed on 04 Jul 2015.

[10] RIPE. Ripe database query reference manual. https://www.ripe.net/manage-ips-and-asns/db/support/documentation/ripe-database-query-reference-manual. Accessed on 05 Jul 2015.

[11] C. Alaettinoglu et al. RFC 2622 - Routing Policy Specification Language (RPSL). https://tools.ietf.org/html/rfc2622. Accessed on 02 Jun 2015.

[12] L. Blunk et al. RFC 4012 - Routing Policy Specification Language next generation (RPSLng). https://tools.ietf.org/html/rfc4012, March 2005. Accessed on 04 Jul 2015.

[13] D. Meyer et al. RFC 2650 - Using RPSL in Practice. https://tools.ietf.org/html/rfc2650, August 1999.

[14] J. Durand et al. RFC 7454 - BGP Operations and Security. https://tools.ietf.org/html/rfc7454, February 2015. Accessed on 04 Jul 2015.

[15] D. Senie P. Ferguson. RFC 2827 - Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. https://tools.ietf.org/html/bcp38, May 2000. Accessed on 04 Jul 2015.

[16] D. McPherson W. Kumari. RFC 5635 - Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). https://tools.ietf.org/html/rfc5635, August 2009. Accessed on 04 Jul 2015.

[17] P. Savola F. Baker. RFC 3704 - Ingress Filtering for Multihomed Networks. https://tools.ietf.org/html/bcp84, March 2004. Accessed on 04 Jul 2015.

[18] F. Baker. RFC 1812 - Requirements for IP Version 4 Routers. https://tools.ietf.org/html/rfc1812, June 1994. Accessed on 04 Jul 2015.

[19] M. Cotton et al. RFC 6890 - Special-Purpose IP Address Registries. https://tools.ietf.org/html/rfc6890, April 2013. Accessed on 04 Jul 2015.

[20] IANA. IANA IPv4 Special-Purpose Address Registry. http://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml, August 2009. Last updated May 2015, Accessed on 04 Jul 2015.

[21] IANA. IANA IPv6 Special-Purpose Address Registry. http://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml, January 2006. Last updated May 2015, Accessed on 04 Jul 2015.

[22] L. Vegoda. RFC 6441 - Time to Remove Filters for Previously Unallocated IPv4 /8s. https://tools.ietf.org/html/rfc6441, November 2011. Accessed on 04 Jul 2015.

[23] Daniel Karrenberg. RIPE 351 - De-Bogonising New Address Blocks. http://ftp.ripe.net/ripe/docs/ripe-351.pdf, 2005. Accessed on 04 Jul 2015.

[24] Paul Vixie. Rate-limiting state. *Queue*, 12(2):10:10–10:15, February 2014. https://queue.acm.org/detail.cfm?id=2578510, Accessed on 04 Jul 2015.

[25] S. Turner M. Lepinski. RFC 2439 - BGP Route Flap Damping. https://tools.ietf.org/html/rfc2439, November 1998. Accessed on 04 Jul 2015.

[26] Bush R. Kuhne M. Pelsser C. Maennel O. Patel K. Mohapatra P. Smith, P. and R. Evans. Ripe-580 - ripe routing working group recommendations on route flap damping. https://www.ripe.net/publications/docs/ripe-580, January 2013. Accessed on 04 Jul 2015.

[27] C. Pelsser et al. RFC 7196 - Making Route Flap Damping Usable. https://tools.ietf.org/html/rfc7196, May 2014. Accessed on 04 Jul 2015.

[28] P. Mohapatra et al. RFC 6811 - BGP Prefix Origin Validation. https://tools.ietf.org/html/rfc6811, January 2013. Accessed on 04 Jul 2015.

[29] S. Bellovin et al. RFC 7353 - Security Requirements for BGP Path Validation. https://tools.ietf.org/html/rfc7353, August 2014. Accessed on 04 Jul 2015.

[30] S. Turner M. Lepinski. Internet Draft: An Overview of BGPsec. https://tools.ietf.org/html/draft-ietf-sidr-bgpsec-overview-07, June 2015. Accessed on 04 Jul 2015.

[31] S. Kent M. Lepinski. RFC 6840 - An Infrastructure to Support Secure Internet Routing. https://tools.ietf.org/html/rfc6480, February 2012. Accessed on 04 Jul 2015.

[32] How to query the ripe database. https://www.ripe.net/manage-ips-and-asns/db/support/documentation/ripe-database-documentation-1.78/how-to-query-the-ripe-database, 2015. Accessed on 02 Jun 2015.

[33] L. Zhou et al. RFC 7485 - Inventory and Analysis of WHOIS Registration Objects. https://tools.ietf.org/html/rfc7485, March 2015. ISSN: 2070-1721.

[34] L. Daigle. RFC 3912 - WHOIS Protocol Specification. https://tools.ietf.org/html/rfc3912, September 2004.

[35] Ripe database query. https://apps.db.ripe.net/search/, 2015. Accessed on 02 Jun 2015.

[36] RIPE NCC. Whois rest api. https://github.com/RIPE-NCC/whois/wiki/WHOIS-REST-API, 2015. Accessed on 02 Jun 2015.

[37] ISC. Irrtoolset official website. http://irrtoolset.isc.org/. Accessed on 02 Jun 2015.

[38] Marco d'Itri. RPSL and rpsltool. http://www.trex.fi/2012/rpsltool-trex.pdf, 2012.

[39] Github. https://github.com/, 2015. Accessed on 02 Jun 2015.

[40] Nick Hilliard. Irrtoolset github. https://github.com/irrtoolset/irrtoolset, 2015. Accessed on 02 Jun 2015.

[41] Richard A. Steenbergen. Irrtoolset github. http://sourceforge.net/projects/irrpt/, 2015. Accessed on 02 Jun 2015.

[42] Marco d'Itri. Rpsltool github. https://github.com/rfc1036/rpsltool, 2015. Accessed on 02 Jun 2015.

[43] Todd Caine. Net::irr. http://search.cpan.org/~tcaine/Net-IRR/lib/Net/IRR.pm, 2010. Accessed on 02 Jun 2015.

[44] Netconfigs. netconfigs.com. Accessed on 04 Jul 2015.

[45] Tail f ConfD. Management agent. http://www.tail-f.com/management-agent/. Accessed on 07 Jul 2015.

[46] Ed. M. Bjorklund. Rfc 6020 - yang - a data modeling language for the network configuration protocol (netconf). https://tools.ietf.org/html/rfc6020, October 2010. Accessed on 04 Jul 2015.

[47] Tail-f. Network control system. http://www.tail-f.com/network-control-system/. Accessed on 07 Jul 2015.

[48] Juniper documentation. Junos pyez. https://techwiki.juniper.net/Automation_Scripting/010_Getting_Started_and_Reference/Junos_PyEZ. Accessed on 07 Jul 2015.

[49] R. Enns et al. RFC 6241 - Network Configuration Protocol (NETCONF). http://tools.ietf.org/html/rfc6241, June 2011.

[50] W. Hardaker. Rfc 5953 - transport layer security (tls) transport model for the simple network management protocol (snmp)transport layer security (tls) transport model for the simple network management protocol (snmp). http://www.ietf.org/rfc/rfc5953, August 2010. Accessed on 04 Jul 2015.

[51] Cisco Systems, Inc. Cisco nexus 7000 series nx-os fundamentals configuration guide, release 6.x. http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/6_x/nx-os/fundamentals/configuration/guide/b_Cisco_Nexus_7000_Series_NX-OS_Fundamentals_Configuration_Guide_Release_6-x.pdf, July 2009. Last Modified on 21 August 2013, Accessed on 05 Jul 2015.

[52] Juniper forum Jeremy Schulman. Python for non-programmers. http://forums.juniper.net/t5/Automation/Python-for-Non-Programmers-Part-1/ba-p/216449, November 2013. Accessed on 05 Jul 2015.

[53] Juniper documentation. local-preference. https://www.juniper.net/documentation/en_US/junos14.2/topics/reference/configuration-statement/local-preference-edit-protocols-bgp.html, 2015. Accessed on 05 Jul 2015.

[54] Cisco Systems, Inc. Cisco ios ip routing: Bgp command reference. http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/command/irg-cr-book.pdf, 2012. Accessed on 05 Jul 2015.

[55] Cisco Systems, Inc. set local-preference. http://www.cisco.com/web/techdoc/dc/reference/cli/nxos/commands/bgp/set_local-preference.html. Accessed on 12 Aug 2015.

[56] B. Haberman R. Kisteleki. Internet Draft: Securing RPSL Objects with RPKI Signatures. https://tools.ietf.org/html/draft-ietf-sidr-rpsl-sig-07, May 2015. Version 7, Expires 12 November 2015, Accessed on 04 Jul 2015.

[57] RIPE NCC. Tools and resources. https://www.ripe.net/manage-ips-and-asns/resource-management/certification/tools-and-resources. Accessed on 05 Jul 2015.

[58] Juniper documentation. prefix-list. http://www.juniper.net/techpubs/en_US/junos11.4/topics/reference/configuration-statement/prefix-list-edit-policy-options.html. Accessed on 07 Jul 2015.

[59] RIPE NCC. 4.2.7 description of the as-set object. https://www.ripe.net/manage-ips-and-asns/db/support/documentation/ripe-database-documentation-1.78/rpsl-object-types/4-2-descriptions-of-primary-objects/4-2-7-description-of-the-as-set-object, 2015. Accessed on 02 Jun 2015.