FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Tomáš Hlaváček

Routing policies and real paths in the Internet

Department of Software Engineering

Supervisor of the master thesis:	RNDr. Ing. Jiří Peterka
Study programme:	Computer Science
Study branch:	Software Systems

Prague 2016

Title: Routing policies and real paths in the Internet

Author: Tomáš Hlaváček

Department: Department of Software Engineering

Supervisor: RNDr. Ing. Jiří Peterka, Department of Software Engineering

Abstract: Routing policies are now represented by RPSL and by its evolutionary extension called RPSLng. These languages can be used for describing unique routing policy of each autonomous system. Experience shows that even though there are translation tools from RPSL and RPSLng to configuration formats of commonly used routers, the actual network configuration is rarely generated from RPSL sources and routing policy is then perceived as marginal paperwork, which often does not reflect the real network settings. There will be most likely a need for RPSL format change in order to remedy the discrepancies. To support this I present long-term measurements of inaccuracies in routing policies compared to real paths in the Internet. I also present a list of the most frequent problems, and I offer suggestions, how to reform RPSL to improve situation in the long term.

Keywords: Internet routing BGP RPSL IRR DFZ

My sincere gratitude goes to my supervisor RNDr. Ing. Jiří Peterka and external counselor Mgr. Ondřej Filip, MBA for valuable advice and to companies CZ.NIC, z.s.p.o., Ignum, s.r.o. and Trenka Informatik AG for providing data feeds and substantial computing resources.

Contents

Introduction

0.1 Thesis topic

The topic of this thesis is based on my previous Bachelor's thesis [1] and on my experience that comes from operating several Autonomous Systems. My previous work has been focused on one particular detail of Internet routing management - Routing Policy Specification Language (RPSL).

It seems that efforts that have been put to many implementations of RPSL tools are not yielding the desired outcome and there is an obvious question: *Why?*

The presented standards, software tools, technical and administrative mechanisms that are subjects to subsequent analysis in this work are part of the Internet in the broad sense. The Internet is not only the physical network but also the standards and the community that maintains them.

There are many specifics of the Internet community and the most important of all is the consensus driven decision process and the emphasis on freedom. It needs to be stressed that the technical and administrative core of the Internet has been created and is operated by the community that enjoys and protects freedom of act to a level unprecedented in any other part of IT industry or any other human endeavor.

Any prospective effort to change anything in the Internet community needs a consensus. Reaching consensus depends on an ability to prove existence of a need for change and on an ability to provide a technically superior proposal for the change.

The topic of this thesis is to provide the proof of the need for change in the Internet routing system and specifically in the high level routing control.

0.2 Hypothesis

The Internet is partitioned to entities called Autonomous Systems. The Autonomous Systems are the entities that enjoy liberty and freedom of choice in questions of technical and operational procedures with regard to interconnection among each other. One of the procedures is utilization of RPSL for capturing existence and certain technical details of the interconnections.

In order to prove the need for future change of RPSL and related standards I am going to provide arguments supporting the following hypothesis: "The utilization of RPSL in the current Internet is sub-optimal both in scale and accuracy of the information. The situation is not improving and there is no perspective of change in this trend."

There is an apparent problem with the stated hypothesis - it could not be directly tested by means of mathematical statistic. A value of the stated hypothesis lies in a fact that proving enough evidence for this hypothesis is most likely the only possible way to eventually reach consensus in the Internet community.

Formal approach is also provided for sake of explaining the problem in this thesis and avoiding any confusion.

0.3 Extended hypothesis

An obvious extension of the stated hypothesis is this question: “What is the cause of the sub-optimal utilization of existing mechanism?” or “What can be done to improve the situation?”. An educated guess of the possible answer is: “High added workload compared to relatively low benefit obtained from extensive utilization of RPSL prevents the AS operators from wider deployment.“

Immediate problem in the extended hypothesis lies in the fact that it is extrapolation of causality in an unique problem with apparently complex causes and with no counter-examples. The arguments for the extended hypothesis are therefore based on experience gained by implementing the related standards into a software and on critical analysis of the relevant standards. The software, called **bgpcrunch**, is attached to this work and will be referred to on many occasions.

0.4 Thesis application

The Internet community currently goes through difficult times caused by the IPv4 address pool exhaustion and it’s consequences: Struggle with IPv6 adoption and rapid increase of fraudulent behavior that begins to infiltrate lower networking layers.

These issues are remarkably connected with the Internet routing system and its management or lack of high-level management. Scarcity of IPv4 addresses causes many problems, most notably excessive de-aggregation of prefixes in BGP table and fraudulent announcements of prefixes, which is known as *IP space hijacking*. Both problems can be avoided by proper use of routing policies that would filter out fraudulent prefixes and would provide more control on de-aggregations and centralize data to a readable structure. IPv6 adoption might be easier with well defined high-level description of existing IPv4 routing. IPv6 could benefit from better control and secure way of the Internet routing management from the beginning of the new protocol deployment. The infiltration of lower levels by sophisticated attacks on lower levels of networking stack is a difficult issue to analyze and prevent. There are certain situations that would benefit from utilization of information in routing policies in order to construct filters or provide way to define an automatic filtering mechanisms to stop these attacks.

The current framework with RPSL, BGP and all the parties that form the Internet routing system are theoretically capable of doing all these things. However, only few Autonomous Systems use RPSL in real operation of their network on daily basis and the public RPSL databases are both inaccurate and incomplete. These problems severely limit outlined potential and reduce interest in the current system.

Objective of the research presented in this thesis is to provide necessary arguments to the community for justifying a change in standards and in operational procedures and for starting a discussion in this direction.

1. Internet routing

1.1 Internet resources

Internet routing is a massively distributed cooperative process that supports data transmission by means of IP (Internet Protocol). The IP needs resources for moving packets from one point to another using certain pre-computed paths. Most notably it needs IP addresses, which are in fact integers combined with rules for their utilization. It also needs Autonomous System numbers (ASN), which are simple integers. Apart from these virtual resources the protocol needs physical resources - processing power, memory for packet buffers, configuration and the pre-computed routing tables that direct packets along their way towards the destination.

1.1.1 Coordinators

There are organizations that coordinate use of the number resources in the Internet, most notably IP addresses and ASNs. These resources that are being used in Default Free Zone (DFZ), which is a synonym for the Internet, are subject of policies. The policies for the Internet are created and subsequently maintained in an open community which consist of smaller and more focused communities, like RIPE community, organizations that perform certain technical or administrative tasks and large network operators.

There is neither technical nor formal obstacle for anybody to configure anything that is technically possible on any router or host in the network, regardless of policies and resolutions of any organization. The only measure of adherence to the rules created by the community is technical possibility of establishing network interconnection.

When a host or a router connects to the Internet it is expected to follow rules established by the Internet community that actually provide technical framework for the host to be connected and start using certain addresses. Acceptance of routing information and packets from or for the particular host in the Internet depends on free will of any Autonomous System along the path.

The coordinating role performed by service organizations that follow policies created by the community is possible only because all community members adhere to core rules they created for themselves. The coordination bodies do not have any direct authority over the resources in any enforcement sense. Since it is not possible for any single organization or company to enforce even the most fundamental policies, it is up to the community to self-regulate and self-policy. Policies are expected to become reality automatically, because there has to be a consensus on them in the policy development process prior to their acceptance. Moreover, the policy development process involves the parties that are expected to implement the resulting policies.

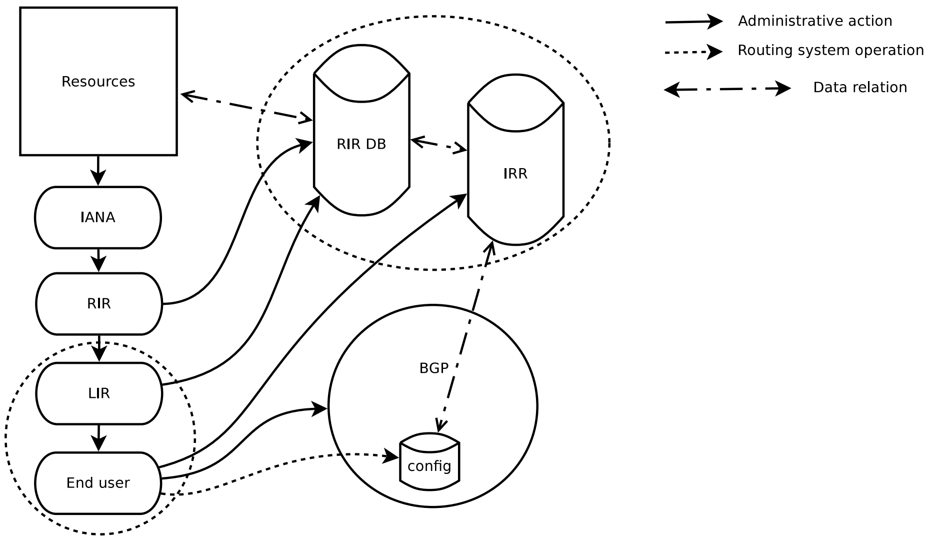

Figure 1.1: Internet coordination hierarchy

1.1.2 Consensus based rules

The self-enforcement of the rules depends on the fact that majority of the Internet, or at least a majority of the local network environment implement a policy, it becomes mandatory for everybody in their network-wise proximity. An alternative is to cease the interconnection and effectively stop the data flow for those parties, which decided not to accept the policy.

It is technically possible to disobey community rules and ignore any authority and coordination bodies to certain degree. Some people actually do that in order to gain some extra profit and with limited damage to others, so nobody notices. Even if somebody learns about the wrongdoing, they will not be most likely in a position to do anything about it.

1.2 Hierarchy of coordinators

1.2.1 Coordination bodies

A huge network, like the today's Internet, needs coordination to operate safely and efficiently. The Internet community has established a hierarchy of coordination bodies that support the community in creating the policies for operating the network and executing the policies. Figure 1.1 shows the simplified flow of resources from the resource pool through the coordinators to the end user. Coordinators and users should use RIR database and Internet Routing Registry for registering their resources. The registration should be completed before setting the resources to the BGP configuration.

The coordination of the most important resources - IP addresses and AS numbers is being executed at the top level in IANA (Internet Assigned Numbers Authority), which is a department of ICANN (Internet Corporation For Assigned Names and Numbers), a non-profit private American corporation that oversees not only IP address and AS number allocation but also different IP constants and DNS resources. IANA operates according to policies that are created on the basis of consensus among RIRs (Regional Internet Registry), the only direct recipients of resources that IANA hands out.

There are five RIRs, each serves its own service region that is roughly equivalent to a continent: AfriNIC for Africa, ARIN for North America, APNIC for Asia and Pacific region, LACNIC for South America and RIPE NCC for Europe, Middle East and parts of Central Asia. All RIRs operate according to policies that are being developed and agreed upon by their communities and based on consensus amongst the community members.

1.2.2 Communities and their members

In case of RIPE the community is open to everyone, regardless of network size or status of LIR. The RIPE NCC is the executive body, based in Amsterdam, Netherlands. The community meets twice a year on RIPE meetings and develop policies in several working groups and mailing lists.

Please note that even though the names of the coordination organizations have roots in acronyms, the names are now used as substantives and often serves as an official name of the legal entity.

Although the community creating the policies is open to everyone, there are on contrary formal requirements for a company to receive resources from RIR - RIPE NCC for instance. The company needs a status of LIR (Local Internet Registry) to use services of RIPE NCC. The LIR is able to obtain IP addresses either for their own networks or their customers, which they directly manage. Direct management means that the LIR physically transports packets for these IP addresses. This type of address space is called PA (Provider Aggregatable) resource. The LIR can also obtain PI (Provider Independent) resources for any other network that fulfills corresponding assignment criteria. The PI addressing resources are intended for the end network and the operator of the end network is fully responsible for routing the traffic and the LIR's role is usually limited to dealing with RIPE NCC during the allocation process. The two types of addressing resources, PA and PI, are not technically distinct, however there is a substantial difference in their formal handling in policies.

There is another level of authority transfer allowed in case of PA allocations: LIR can make sub-allocation to another party and the recipient of the sub-allocation does not need to have any formal relationship with RIPE NCC. In this case the LIR is still responsible for the sub-allocated IP addressing resources even though the right to make assignments for the end users has been transferred.

The same process is in effect for ASNs (Autonomous System numbers) with the difference that an ASN can not be partitioned. IANA allocates ASN blocks to RIRs and RIRs allocate a single ASNs to LIRs or their customers on request. ASN allocation process closely resembles the PI allocation. It applies even in case the final recipient of the resource is the requesting LIR.

Described basics of the resource administration process in RIPE NCC is almost the same in all RIRs. Even though there are differences among RIRs in the particular rules, fees, requirements, time frames, in terminology and evaluation procedures to certain extent, the basic ideas described in this chapter stay the same.

1.3 Network operation

The most important resources for operating IP protocol are IP addresses and ASNs. Each party that connects to an IP network needs IP addresses for addressing the devices and a way to transport traffic from and to the devices.

Apart from physical interconnection the connected parties require routing protocol to generate routing tables in the network. Today's standard protocol for routing traffic in the Internet is BGP (Border Gateway Protocol). It is a path-vector EGP (Exterior Gateway Protocol), that is being used for exchanging routing information among autonomous systems. In terminology of BGP the Internet is an interconnection of large amount of Autonomous Systems that use common routing infrastructure called DFZ (Default Free Zone).

Despite of extra costs that bring operation of private networks isolated from the Internet, there have been several large-scale deployments and these networks might still persist. These networks usually operated IP with various routing protocols, including BGP and historically co-operated with the Internet coordinators to a certain degree, mainly to avoid addressing collisions in case of eventual interconnection. Existence of these networks with large scale (nonetheless, it is still incomparable with the Internet) shows that there is a complete freedom of act in networking area. And there are legitimate reasons for ignoring Internet community and its rules and operating own network in parallel or co-operate with the Internet only partially. Following discussion is not going to take into consideration these private or partially-cooperating networks and it will be focused solely on the Internet.

1.4 Internet resources

Addressing resource has to be announced by an autonomous system to DFZ to be actually usable in the Internet. BGP is employed for transmitting announcements as well as forwarding announcements of other parties learned from BGP neighbors. This process creates DFZ routing table cooperatively. The DFZ table contains in theory all information needed to route traffic to any possible destination in the Internet. In fact there is a minority of addressing resources that fluctuates in DFZ and some are even unreachable due to technical, political or economic reasons.

There is no right of Internet connectivity and nobody could claim transportation of packets. Internet connectivity in its technical core depends on good will and spirit of cooperation that precede formation of contractual relationships that concerns accepting and forwarding BGP announcements and transportation of packets.

1.5 Internet standards

Technical standards that concern protocols operated in the Internet as well as operational requirements and procedures are described in several types of documents. Internet Engineering Task Force (IETF) and the Internet Society (ISOC), are considered to be the principal technical development and standards-setting bodies. Their outputs that take form of Request For Comments (RFC), Best Current Practice (BCP) and Internet Standards (STD) play prominent role among others. These documents are widely respected and considered as the main building block of the Internet.

The standards might be more or less successful. Since no standard is obligatory by itself, the only way of making a standard obligatory for some party is to put it as a requirement into a contract with that party.

I am not aware of any rigorous methodology that would define a procedure for measuring success of a standard. But simple interpretation is that successful standards are actively developed, have more implementations and have greater user base than less successful standards.

1.6 Records and public databases

IANA, RIRs and LIRs allocate Internet addressing resources to the end users. The whole allocation and assignment process is documented. There are publicly available records about each step in the process. The records contain basic information that is needed to operate IP correctly and to troubleshoot the routing system or services running above the network layer. Validity and accuracy of the allocation and assignment records are subjects of shared and transferred responsibility among the parties that allocate the resources and the parties that use them.

1.6.1 Top level

IANA keeps lists of IP address blocks and ASN blocks that have been allocated to RIRs or directly to end users before the RIR system was invented in 1996. Today's Internet coordination and registration system is described in RFC 2050/BCP 12 [2]. IANA lists are publicly available on the web in form of plain text file and CSV [4]. It is actually a remarkably short list considering the fact that it describes usage of 2^{32} IPv4 addresses, up to 2^{32} ASNs and certain portion of IPv6 addresses, which may top 2^{125} of individual addresses under current policies.

1.6.2 RIR level

RIR actions and corresponding records are much more interesting compared to IANA level. RIRs have to deal with substantially more information that capture resource usage in much more detail and that are changing much faster than the top-level allocations. Problem is that records of RIRs might contain personal and confidential information. The RIR allocation data has to be structured and indexed in order to allow searching for records, using certain criteria for them and to support operation and troubleshooting of IP in large scale. The detail level and volume of data as well as the way of storing records is based on policy created by the particular RIR.

Remarkably, the unifying element of RIR databases is the access protocol *whois* [3]. The protocol is simple and open to interpretation but it also imposes some important requirements on the data. The most important requirement is that the queries and answers have to be in human-readable text format.

Despite of the name *database*, RIR databases are only text files. Basic access method for these files is publishing them on a FTP server or via *whois* protocol. Updates were originally directed to a maintainer of the files via e-mail but today there are e-mail robots and recently also web services for processing update requests on the objects automatically and with more security. The text files contain *objects*. The objects are text blocks that contain keys and values. These keys are called *attributes*. There is a remote similarity with databases and objects in their ordinary meaning but the Internet community and the following parts of this thesis use these words in this particular meaning.

1.6.3 LIR level

Record keeping on the LIR level is mandatory because of rules imposed by the community through the coordination bodies. These rules are being enforced as a part of the contractual relationship among the RIRs and the LIRs. There are two types of mandatory records that LIRs are obliged to keep: Public records and mandatory private records that the LIRs have to keep internally and provide upon request. Mandatory internal records apply mainly to IPv6 protocol family, because prefix boundaries are too small compared to the allocated address space to the LIRs. It would cause huge administrative burden and it would be technically challenging if the RIRs had to keep these records in the public databases operated by the RIRs themselves to the level of detail that might reach vast amount of subnet allocations for the end networks.

On the sub-allocation level there are generally no additional rules and the overall responsibility for keeping records up to date stays with the LIR. Despite that there are mechanisms that make it possible to transfer some authority and some responsibilities towards the customer or downstream network in general.

1.7 Routing registry databases

1.7.1 Resource assignment records

Records describing resource assignments and linking them to their users have to be kept up to date through a mandatory process in all RIR regions. However, there are differences in policies of RIRs. The records might differ in level of detail, format, access methods and in many minor constants. The obligation to keep records up to date is based on contractual relationship among RIR and LIR and among LIR and the end user.

1.7.2 Routing databases

On the contrary, records describing technical details of resource usage and most importantly routing, are not mandatory but recommended. The resource usage on the Internet can be represented in routing policies that network operators might publish in routing registry databases. There are several routing registry databases, some of them are accessible to the general public and some require certain form of membership (both paid and formal) in order to publish policies, but are freely accessible for reading.

Certainly some private databases exist. These databases are used for internal processes inside network operators and for keeping records of customers in transit networks.

1.7.3 Public routing databases

Public databases can be commonly referred to as IRR (Internet Routing Registry).

The reason for having one common name for all public routing databases is that these databases were able to mirror each others' data at a certain point. IRR is still a sensible name for the whole system even though the mirroring of data is rather complicated and incomplete. Reason for inconsistencies lies in the fact that different databases keep data in slightly different formats and it is difficult to create and maintain a general conversion algorithm.

Another problem is that even though network operators are strongly discouraged from registering their routing policies in more than one routing registry database, it might still happen and in fact it happens. In that case the result, when the same routing policy is registered in two different places, is not defined. Document RFC 2650 [12] explicitly states that this situation may lead to confusion of tools and therefore mislead anybody who tries to verify the routing policy.

1.7.4 Content of routing databases

A common core of all routing databases is Routing policy Specification Language (RPSL). It has been defined in documents RFC 2622 [5] and RFC 4012 [6]. The document RFC 2650 was created in 1999 with intent to unify fragmented database formats, however since then the differences among routing databases deepened. Some databases even ceased to operate and new ones have been created.

1.7.5 Relations among public routing databases

Nowadays (May 2016) there are at least 34 known routing databases. Comprehensive but not authoritative list is published on the web by Merit Network, Inc. [7]. Relations among the databases, methods of replication, rules and guidelines for each database are generally unknown. The Merit's website contains majority of publicly known information on this topic.

The most important routing registries are the databases operated by RIRs, and the most important for Europe is RIPE DB.

1.8 Network operation process

1.8.1 Resource setup

IP network operation depends on many resources discussed in previous sections. The number resources play extremely important role. Some resources have to be obtained prior to physical network setup and another can be added in increments to meet the demand of the network's users. The operator of the network is responsible for obtaining the resources from RIR that serves his region either through a LIR or directly. Operators generally need a contractual relationship with the LIR or RIR for this reason. Purpose of the LIR is to provide help to the network operator with requesting resources and with the administrative part of network setup. The LIR have to keep track of PA resources and remains responsible for keeping the records in public databases up to date. In case of PI resources, the responsibility stays with the resource holder and therefore it is responsibility of the network operator, rather than the LIR, which has provided help only with the set up of the resource. The network operator may start using the resources in DFZ after allocation. Using resources generally means announcing the IP address blocks to DFZ or using the AS number as an originator of the IP address announcements. The LIR in cooperation with the operator (or the operator alone) is supposed to create the objects that describe assignment and purpose of the resources and upload the objects into the RIR database. Subsequently the operator is supposed to create and publish routing policy in one of the public routing databases. In addition both the LIR and the operator should keep the public information up to date.

1.8.2 Role of a LIR

Reason for having LIRs in the loop is that the LIR is supposed to be well equipped with knowledge of RIR policies and certain experience that makes the RIR's processes more efficient. LIR should be able to help network operator to deal with many different situations concerning number resources and act according to current policies and best practices in the community. LIR is even supposed to speak on behalf of the network operator in resource management procedures, and most importantly, in resource allocation process.

1.8.3 Resources managed by the end users

In case of PI resources, a bond between the resource holder and the LIR is deliberately loose. This option allows IP resource holders to remain technically independent on the LIR. The LIR is nonetheless needed in the process of resource set up.

In case of RIPE NCC service region, there is also a contractual relationship requirement that demands PI resource holders to keep a contractual relationship with any LIR of their choice and pay small annual fee to RIPE NCC. This requirement has been imposed in RIPE NCC region in early stage or IPv4 address pool exhaustion. The main reason was a concern about abandoned PI space that had been virtually irreclaimable. The contractual relationship serves as an assurance of the continuing need for the PI space and as an instrument of keeping contact with the PI space holder. Before that the contact information in the RIPE DB sometimes proved to be invalid and there were neither formal nor practical means of finding the real resource holder.

1.8.4 Resource registration

Network operator and the supporting LIR are obliged to keep records about the resources. The records usually contain:

1. Resource specification,
2. holder of the resource,
3. reason for the assignment,
4. authentication keys for writing.

Network operators might publish much more information in routing registries. Some of them are strongly recommended. List of the most important ones follows:

1. BGP routing origin for addressing resources,
2. list of connections to another Autonomous Systems (peerings),
3. routing filters,
4. routing metric details,
5. memberships in Internet Exchange Points (IXP) and routing details,
6. traffic filters.

These information are the most useful and best known examples of routing policies.

2. Routing and routing policies

Routing policy in general is a set of high level instruments for describing configuration of an element in routing system. The element is usually a router or set of routers that execute certain routing protocol.

The only existing standard for representing routing policies is *Routing Policy Specification Language* (RPSL). Even though the concept of routing policies is broader and can cover things like SDN controller description, RPSL takes into account only BGP.

The relationship between BGP and RPSL is close because the RPSL specification documents RFC 2622 [5] and RFC 4012 [6] repeatedly refer to specific BGP attributes, best-path selection algorithm and BGP specific mechanisms.

2.1 BGP

BGP stands for *Border Gateway Protocol.* It is a path-vector routing protocol, that essentially executes distributed version of a shortest-path graph algorithm with certain unique features that reach well beyond simple graph model.

2.1.1 BGP overview

The graph model of BGP routing consist of vertices, edges and the shortest path algorithm.

A vertex in BGP protocol is the Autonomous System. The AS is identified by an integer, called *Autonomous System Number* (ASN). The autonomous system might consist of several routers that runs BGP with the same ASN. In that case these routers cooperate with each other to form a single and consistent entity.

An edge in BGP is called a session. The BGP session is internally a TCP connection that the router uses to exchange routing information with a neighbor. There are two different modes of BGP sessions: Internal BGP session is a session that links two routers with the same ASN and it is hidden inside the AS, which externally acts as one vertex while the internal structure of the AS remains unexposed. On the contrary, external BGP session has to be set between two routers with different ASNs. Only this type of BGP session is the edge in the graph model.

2.1.2 Sessions

Each BGP router in the network needs its own specific configuration. BGP configuration reflects point-to-point model of BGP relationships. Point-to-point is also the most common physical topology in the Internet backbone - direct physical links between two routers. The BGP session represents this adjacency between two Autonomous Systems and it is informally called “peering”.

When the BGP session is set on both sides, the routers connect over TCP, start-up procedure is executed and then routing information can be exchanged. The most important BGP message in the routing information exchange phase is *update*. It contains either a new routing information, update of any existing information or routing information withdrawal.

2.1.3 Path vectors

A routing information in BGP is called a path vector, or more formally Network Layer Reachability Information (NLRI). It contains mandatory elements and might contain certain optional elements as well. The elements are called BGP attributes. The most important attributes in the path vector are:

1. *Prefix* - a network subnet that is a subject of the update,
2. *AS_path* - a list of all ASes that the path vector traversed from the originator,
3. *Next Hop* - IP address of the next hop for the prefix.

Originator is the first ASN in *AS_path* of the source of the BGP announcement.

Routers utilize *AS_path* as a loop prevention mechanism. Each path vector entering the AS is checked for occurrence of the local ASN in *AS_path*. Path vectors that contains local ASN in *AS_path* are discarded because these path vectors already traversed local AS and therefore form a loop.

Length of *AS_path* is also a default metric for BGP best-path selection algorithm. Shorter *AS_path* (the *AS_path* with lower number of ASNs) is considered to be better route than a path vector with longer path.

A router has to append its own ASN to the *AS_path* in each path vector before it can be transmitted over external BGP session to another AS.

2.1.4 Attributes

There are basically four types of BGP attributes:

1. *Well-known, mandatory* - have to be supported by all BGP implementations and have to be contained in each *update* message,
2. *well-known, discretionary* - have to be supported, but usage in the *update* messages is not mandatory,
3. *optional, transitive* - may not be supported and unrecognized attributes will be passed on without changes,
4. *optional, non-transitive* - may not be supported and will not be passed on.

The attributes allow administrators to define complex rules that modify the best path selection process either on the local routers or in remote Autonomous Systems.

2.1.5 Best path selection

BGP has been defined in RFC 4271 [8] and in subsequent amendments. Despite the fact that variants and non-standard features exist in certain implementations, the basic operation principle is always the same - received path vectors are directed through series of following actions: Path vectors that would close routing loop and routes with unreachable next-hop are removed. Then user-defined input filters are evaluated. After that the remaining path-vectors are passed to user-defined rules for matching arbitrary path-vectors and modifying some of the attributes. The resulting path-vectors are stored in input tables (known as Adj-RIB-In).

All available path vectors in the input tables are processed by the best-path selection algorithm. Main purpose of the best-path selection algorithm is to decide on concurrent and distinct paths to the same destination. The algorithm executes following steps from the beginning until some rule selects the best path vector:

1. Path with the highest *LOCAL_PREF* attribute is selected.
2. Path with the shortest *AS_path* is preferred.
3. Path with the best *ORIGIN* attribute is selected. (*internal* is better than *external* and *external* is better than *incomplete*).
4. Path with the lowest *MED* is preferred.
5. External BGP paths are preferred over internal BGP.
6. There are two last resort rules in the end of the paths selection algorithm to get a resolution even in case of a tie among the paths up to this point.

The resulting path vectors are transformed to routes and added to Routing Information Base (RIB). RIB is a source for construction of Forwarding Information Base (FIB), which then directs actual packet flow in the router.

The best-path selection process can be altered by two major mechanisms:

1. Filtering out path vectors,
2. changing the metric attributes.

Apart from the *AS_path* length, there are two major metrics: Local Preference and Multi-Exit Discriminator (MED). These names are often used in parallel with the BGP attribute names *LOCAL_PREF* and *MED*.

2.1.6 Metrics

The traffic flows in the Internet according to the routes in routing tables. The direction of the traffic flow is the opposite of the BGP path vector transmission, because path vectors represent paths towards a destination.

Network operators can use additional metrics in path vectors to modify the path selection results and therefore the route that the corresponding traffic takes. Each metric fits specific direction and common requirements.

For instance Local Preference can be assigned to incoming path vectors. It can change priority of incoming routes and therefore it may change direction of the outgoing traffic.

MED can be assigned to the outgoing routes and therefore it can modify path selection process in the neighboring AS. This metric can be utilized only for path vectors transmitted to the same AS over multiple distinct routes. Having multiple links to the same AS is in fact rather common situation.

2.1.7 Communities

Apart from these metrics BGP can carry *community* attributes. The *community* attribute is an integer that can be added to a path vector. Meaning of the particular number in the attribute can be defined by the user. The *communities* are often used for signalization both inside the AS among internal BGP routers or between Autonomous Systems that have direct or even indirect connection.

The *community* attributes can be added, removed and matched by filters. It brings general signalization capability to BGP and network operators use it to implement complex setups like Remote Triggered Blackholing (RTBH) or remote path filtering or remote metric selection.

2.1.8 Transit AS

The process of sending out a path vector to the BGP neighbor consist of:

1. The path vector is picked from RIB.
2. It goes through series of user-defined rules and filters.
3. Local ASN has to be appended to *AS_path*
4. Then it can be transmitted over the BGP session.

Ordinary BGP implementations in default configuration usually forward all best paths to each neighbor.

2.1.9 Prefix origination

Besides forwarding existing path vectors, new path vectors can be injected (originated). Originating the path vector means that the AS injects a routing record into DFZ, provided the route is not filtered out soon enough along its way.

The announcement summons traffic destined to a subnet that is a subject of the new announcement. Any router can send out any prefix and it would eventually spread out through all BGP inter-connected nodes to each router in the network.

This basic operation mode of BGP is the reason why sending announcements and originating new prefixes is considered to be a delicate operation. Erroneous announcements might affect not only local traffic but also third parties, when there is a routing conflict.

It is a common practice to set outgoing filters in order to manually restrict or modify announcements. In general the level of control over the outgoing path vectors is greater than control over the incoming ones.

2.1.10 Instrumentation and data

BGP has comprehensive instrumentation and debugging tools. The path vectors carry a lot of interesting information. Data from BGP reflect current operational status of the network and in case of DFZ the BGP data equals to the current state of the Internet.

Each path vector contains ASN of its originator and a list of all Autonomous Systems it traversed. It means that the path vector won in the best path selection process.

Preliminary analysis of networking relationships is needed before any reasoning about the BGP data can be done. The reason for the preliminary analysis is that each AS in the Internet has its own version of DFZ table - its own unique view on the Internet.

Moreover, DFZ is a massive distributed list of path vectors that controls routing of packets over the Internet. Even small changes in DFZ can be monetized - either by respectable ways because optimization pays off, or because there is wide range of possible malicious tampering with BGP. Unfortunately, the malicious activity regularly happens in the Internet.

2.2 Routing Policy Specification Language

2.2.1 History of the standard

The first definition of a language for describing routing policies was the document ripe-81 [10]. It was published in 1993 and it subsequently evolved to the current RPSL. The definition in ripe-81 refers to the document ripe-60 [9] from 1992. This document contains description of routing and associated issues. Both documents reflects early phase of the Internet development in Europe. From today's point of view the documents are partially based on obsolete ideas and wrong assumptions about the future requirements.

It is difficult to interpret the outdated standard from 1993, but it is obvious that the basic view of the Internet was different in many aspects at that time. The standard was focused on control of BGP routing in the Internet in open and public manner. This objective was the primary reason for developing the language.

The language that can capture routing policies in an unified format could serve as a public statement of intentions, supplement to a documentation and records in RIR databases and as a primary data source for configuration of the Autonomous System Border Routers (ASBR). The document states that the main requirements for the new RPSL are:

1. Clarity,
2. translatability,
3. checkability,
4. applicability,
5. generality.

The document defines these features in detail.

RPSL development continued and the result was the document ripe-181 [11] from 1994. This document brought more features than ripe-81 and more closely resembled today's standard RFC 2622, except for several constructs that have shifted its meaning, have been renamed or have been replaced since then. The differences between ripe-81 and ripe-181 are too substantial to summarize them here but the trend was: Complexity of the language increased to allow creating compact descriptions of huge networks. Main tools for that are recursion and `-set` objects. These two tools allow administrators to logically partition the network and use chains of symbolic names to construct filters level by level.

2.2.2 Current standard

RFC 2622 [5] is the current standard for RPSL that has been amended by RFC 4012 [6]. New elements from RFC 4012 are sometimes called RPSLng (RPSL new generation). RPSLng has brought support for IPv6 and generalized support for future protocols or address families. However, the basic principle stays the same in both of these documents and it still derives from ripe-181.

2.3 RPSL elements

According to the current standards RFC 2622 and RFC 4012, the basic unit of RPSL information is called an *object*. The object is a text fragment that has:

- Object type,
- primary identifier that is often used for searching and referring to the object in a *whois* database,
- optional secondary identifiers that might be also used as lookup keys,
- set of mandatory or optional attributes that carry additional information.

Objects are represented in text format that is supposed to be human-readable as well as easy to parse for machines. The text format consists of:

- Inactive lines (comments),
- object separators - empty lines,
- lines that carry the text of the objects.

Object line contain either an attribute and a value or it might be a continuation of a multi-line value. Objects has to be separated by one or more empty lines.

Set of allowed attributes and mandatory attributes is determined by the object type. Object type is the name of the first attribute. Its value is also the primary identifier of the object. Exceptions from these rules exist only in objects derived from RPSL format, but not conforming to the current RPSL itself.

For instance, a simple RPSL object is:

```
route:          217.31.48.0/20
descr:          Network of Ignum s.r.o.
descr:          Czech Republic
descr:          http://www.ignum.cz/
origin:         AS29134
```

```
mnt-by:         IGNUM-MNT
created:        2003-06-12T11:37:52Z
last-modified:  2008-04-16T21:25:49Z
source:         RIPE # Filtered
```

The example shows an object of type `route`. Its primary identifier is **217.31.48.0/20**. The second most important attribute in this object type is `origin`. The high-level meaning of the object in our example is that the prefix **217.31.48.0/20** can be announced to DFZ from **AS29134**.

The `descr` attributes is an unstructured human-readable description and the attributes `created`, `last-modified` and `source` are service information created or required by the RIPE database.

The objects that contain sensitive information, like personal data, might get *filtered* by the database output front-end, which is usually a *whois* server or any other RPC server. The purpose of this is to remove private data from the objects, unless privileged access to the database is granted.

The `mnt-by` attribute contains the name of the `mnter` (from *maintainer*) object, which declares the write access control rules for the object. The `mnter` object determines also the write authentication mechanism and contains required authentication data.

2.4 RPSL object types

The basic types of RPSL objects that form the core of routing policy expressions are `route` objects and `aut-num` objects. Supplemental `inet-rtr` and `-set` objects are utilized to group data and provide additional information.

2.4.1 route object

The `route` object contains information about the BGP announcement of a prefix into the Internet routing system (into DFZ) and states which AS can originate the announcement of the particular prefix.

Example of an route object is in the previous section.

There might be two or more route objects with different `origin` attributes for one particular route. The interpretation of these objects is `OR` operator: Any matching *originator* might occur.

2.4.2 aut-num object

The `aut-num` object serves three main purposes:

1. It is a record of an autonomous system number administrative assignment to a specific organization.

2. It links the ASN with proper contact information.

3. It is a starting point for the definition of the autonomous system's routing policy.

The routing policy can be described in `import`, `export` and `default` attributes. These attributes are sometimes called "peering expressions". The reason for using that summary name is that `import` and `export` attributes, when combined together, might contain complete information that is sufficient to set up the BGP sessions (peerings) for each BGP neighbor of the AS.

The data in these attributes contain filters, that can be translated to *route-maps* for BGP. The route-maps are the low-level tool for altering various metrics and attributes in path vectors. Therefore, the route-maps represent the prevalent mechanism for modifying BGP operation.

2.4.3 inet-rtr

The `inet-rtr` object is a container for information about a BGP router in Internet routing system. It could help describing external topology of an Autonomous System and it might serve as a data source for automated router configuration. There are three main data attributes:

1. `local-as` that contains ASN of the router,
2. `ifaddr` that contains interface address configuration,
3. `peer` that contains information about routing protocol, direction, neighboring ASN and optional information about the protocol configuration.

2.4.4 -set objects

The peering expressions in the `aut-num` object might contain either constants or symbolic names that need to be resolved to objects carrying the referred information. These symbolic names might be references `-set` objects.

A `-set` object might contain one or more constant or further references to another `-set` objects of a compatible type. The `-set` objects could form a graph that has to be traversed to collect required information from the nodes.

Possible loops in these object graphs are not addressed in the standard. However, even if the loop is not a syntax error, it is almost certainly a semantic error. Despite that, there are several loops in the RIPE DB data and therefore parsing software has to be able to deal with the loops.

The `-set` object types are:

- `as-set` that might contain a list of *ASN* and references to another `as-set` objects,
- `route-set` that might contain a list of IP prefixes and references to another `route-set` objects,
- `filter-set` that might contain a list of filter elements and references to another `filter-set` objects,
- `rtr-set` that might contain a list of references to `inet-rtr` objects and references to another `rtr-set` objects,
- `peering-set` that might contain multiple peering definitions.

2.5 References in RPSL

The `-set` objects might directly reference specific objects of proper type or another `-set` object of the compatible type. These objects form a graph that has to be resolved to gather the referenced information and create flat[1] lists that are usually needed to match the filters or to create a router configuration.

Another method of connecting objects into the `-set` object is a *back-reference*. The attribute for specifying *back-references* is `member-of` and it might contain one or more `-set` object identifiers. Following example demonstrates this concept:

```
route:          194.113.52.0/23
descr:          BEUMER
descr:          BEUMER
origin:         AS702
member-of:      AS702:RS-DE,
                AS702:RS-DE-PI
mnt-by:         WCOM-EMEA-RICE-MNT
```

The `member-of` attribute adds this route to the two following objects:

```
route-set:      AS702:RS-DE
descr:          AS702:RS-DE route-set
members:        192.109.206.0/24,
                192.109.207.0/24,
                192.44.36.0/24,
                192.44.37.0/24,
                193.101.167.0/24,
                194.55.166.0/24,
                198.36.86.0/24,
                198.36.87.0/24
mbrs-by-ref:    WCOM-EMEA-RICE-MNT
```

and

```
route-set:      AS702:RS-DE-PI
descr:          AS702:RS-DE-PI route-set
mbrs-by-ref:    WCOM-EMEA-RICE-MNT
```

The *back-references* create a possible security problem which could allow anybody to add a member to any `-set` objects. To prevent creation of an unauthorized *back-reference* the database requires that the maintainer of the new back-referencing member object is listed in the `mbrs-by-ref` attribute of the `-set` object. Because adding `mnt-by` attribute is a subject of authentication by the database, it provides protection of the *back-references*.

The `mbrs-by-ref` attribute might contain keyword `ANY` that allows any object to back-reference to this `-set` object. Missing `mbrs-by-ref` attribute in the `-set` object causes that no object is allowed to create any *back-reference* to the object.

[1] Flat data structure is a set of unstructured elements. Flat list is usually contained in an array or in a lookup table. The terminology is being used in the same meaning and context as in PERL.

2.6 Filters in RPSL

2.6.1 Filter attributes

The routing policy of an Autonomous System is a set of peerings and corresponding input and output filters for each of these peerings. The filters in RPSL are captured in `import` and `export` multi-value attributes. Multi-value attribute is also an attribute type that can occur in an object multiple times.

These attributes have to be part of `aut-num` object that corresponds with the AS in question.

Both `import` and `export` attributes comprise of three basic parts: Peering selector, action and filter. There can be multiple selectors and actions bound to the same filter in a single attribute.

The peering selector is a name of source in `import` lines or name of destination in case of `export`. This name can be a particular ASN, a reference to a `-set` object or an expression that can be expanded to list of ASNs or keyword `AS-ANY` that matches all ASNs.

The action is an optional part of the attribute and it can contain one or more rules for modifying BGP path vector contents. Possible actions are:

- Setting of local preference (`LOCAL_PREF`).
- Setting of Multi-Exit Discriminator (MED).
- Setting or modification of BGP communities.
- Prepending to AS_path BGP attribute.

The filter is the most complex part of these RPSL attributes. It can comprise of basic filter elements, logical operators and set operators. The basic filter elements are list of prefixes, ASNs or `as-set`, that both have to be resolved through `route` objects to list of prefixes, or `AS_path` filters. The set operators are `refine` and `except`. These operators make it possible to apply different actions to a subset of the filter on the left side of the operator. The `refine` is inclusive and `except` is exclusive with respect to the right portion of the filter. Moreover, these operators can be used recursively.

2.6.2 Example aut-num object

```
aut-num:   AS29134
as-name:   IGNUM-AS
descr:     Czech Republic
...
export:    to AS6939 announce AS-IGNUM-OUT
import:    from AS6939 action pref=384; accept ANY AND
 NOT fltr-bogons
export:    to AS5580 announce AS-IGNUM-OUT
import:    from AS5580 action pref=384; accept ANY AND
 NOT fltr-bogons
```

The first line with `aut-num` attribute is the key of this object. The following line with the `as-name` attribute is only a symbolic name. The `descr` attribute contains unstructured text.

The purpose of the first `export` attribute is to define a BGP filter in the outgoing direction from the **AS29134** to the **AS6939** (it is the selector part). It states that the **AS29134** announces or intends to announce prefixes that match the filter defined by the reference to `as-set` object **AS-IGNUM-OUT**. This object has to be further resolved in order to obtain list of the allowed prefixes.

The first `import` attribute says that **AS29134** accepts or would accept routes from **AS6939** that match the filter **ANY AND NOT fltr-bogons**. There is also an action that sets local preference to the value **384**.

The filter expression comprises of keyword `ANY`, logical operators `AND` and `NOT` and a reference to a `filter-set` object **fltr-bogons** that has to be resolved to obtain the remaining part of the filter.

2.6.3 Peering definitions

Both `import` and `export` attributes in the `aut-num` object can contain information about peering configuration. The following example shows extension of the the previous example. In this example the ASBR of **AS6939** is **216.66.80.24** and the local address of **AS29134** is **216.66.80.242**.

```
export:  to AS6939 216.66.80.241 at 216.66.80.242 announce
 AS-IGNUM-OUT
import:  from AS6939 216.66.80.241 at 216.66.80.242 action
 pref=384; accept ANY AND NOT fltr-bogons
```

Peerings might use logical operators in definitions. And more complex peerings can be defined in a `peering-set` object that could be linked to the `aut-num` object in place of peering selector. The `peering-set` objects might use recursive references and logical operators as well.

Omission of the peering specification in `import` or `export` attribute means that the attribute is valid for any peering between the corresponding AS of the `aut-num` object and any Autonomous System that matches the peering selector.

Peering multiplicity

The selectors can be repeated multiple times in one attribute. The reason is to share the rest of the attribute content among two or more peerings. This notation is often used to visually group two peerings between different routers within the same pair of neighboring Autonomous Systems. The following example shows this notation:

```
export: to AS6939 216.66.80.241 at 216.66.80.242;
        to AS6939 216.66.81.129 at 216.66.81.130;
        announce AS-IGNUM-OUT
import: from AS6939 216.66.80.241 at 216.66.80.242 action pref=384;
        from AS6939 216.66.81.129 at 216.66.81.130 action pref=256;
        accept ANY AND NOT fltr-bogons
```

The peering selectors can be encapsulated into `peering-set` objects and referenced from the `aut-num` attribute. This object type brings a possibility of `aut-num` object content simplification and grouping of many peering definitions. The following example shows the same filter as in the previous example with different notation using two objects:

```
aut-num:  AS29134
...
export:   to PRNG-IGNUM-UP announce AS-IGNUM-OUT
import:   from PRNG-IGNUM-UP accept ANY AND NOT fltr-bogons

peering-set: PRNG-IGNUM-UP
peering:     AS6939 216.66.80.241 at 216.66.80.242
peering:     AS6939 216.66.81.129 at 216.66.81.130
```

Further peering abstraction

The IP address of the router in the peering statement or in the `peering-set` object can be either a constant or a reference encapsulated in an `inet-rtr` object. These objects can contain more detailed definition of a peering router.

There are `rtr-set` objects for listing multiple router references. These lists might be used in previous constructions where router definition is expected. This adds one more level of abstraction that has to be resolved in order to create the filter and it brings a possibility of further multiplication of the simple records expressed by this compound filter.

Examples of more complex peering constructs are in the RFC 2622, section 5.6. These notations are not the point of interest in this work, because we can see the exact identity only of the direct neighbors in BGP and remote peerings can not be tracked. It can be done in even more extensive experiment with IP *traceroute* probes, but meaningfulness of that experiment is questionable in light of the following findings that come from simpler and more fundamental verification.

2.6.4 Filter section

There are four basic filtering mechanisms that can be used in filter statement:

1. Origin AS match,
2. IP prefix filter,
3. `AS_path` regular expression match,
4. BGP *community* match.

These components can be directly used in the `import`, `export` and `default` attributes or composed into compound expressions with logical and set operators. The expressions can be further recursively encapsulated into `filter-set` objects.

Origin AS matches can be grouped into `as-set` objects and these `-set` objects can be used in filters as one element that performs multiple matches internally.

Origin AS match

The following example illustrate the process of resolving origin AS match.

```
aut-num:        AS29140
export:         to AS6939 announce AS29134

route:          217.31.48.0/20
origin:         AS29134

route:          62.109.128.0/19
origin:         AS29134

route:          188.227.128.0/19
origin:         AS29134
```

The filter "`announce AS29134`" on the `export` line in the `aut-num` object has to be resolved to all prefixes that any `route` objects allow to originate from the ASN in question. In our example the listed `route` objects have the `origin` attribute *AS29134* and thus the filter `AS29134` is equivalent to the explicit IP prefix filter {`217.31.48.0/20, 62.109.128.0/19, 188.227.128.0/19`}.

IP prefix filter

The constant IP prefix filter is a list of prefix expressions.

The prefix expression is either a prefix to match or a prefix and a specification of allowed subnets. Details of the syntax are in the RFC 2622, section 5.4. The idea is to allow announcements of more specific prefixes that are contained by a less specific prefix in the filter. It is a desirable feature in certain practical situations and it can save a lot of space that the explicit notation of the equivalent filter would take.

The following example shows a simple prefix filter with both simple prefixes and prefixes with subnet specifications:

```
{ 0.0.0.0/0, 10.0.0.0/8^+, 100.64.0.0/10^+, 127.0.0.0/8^+,
  169.254.0.0/16^+, 172.16.0.0/12^+, 192.0.0.0/24^+,
  192.0.2.0/24^+ , 192.168.0.0/16^+, 198.18.0.0/15^+,
  198.51.100.0/24^+, 203.0.113.0/24^+, 224.0.0.0/3^+ }
```

The first prefix in the list is the default route and since it does not have any prefix subnet specifier it is the exact match expression - only the prefix `0.0.0.0/0` would match this filter.

The following prefix `10.0.0.0/8^+` has the prefix specifier `^+` which means that any subnet of the prefix is allowed. This filter would match `10.0.0.0/8` itself, then `10.0.0.0/9` and `10.128.0.0/9` and any other more specific subnet up to any single IP addresses - `/32` prefixes.

Another possibilities are the exclusive prefix specifier `^-`, the exact-length specifier `^n` and the range specifier `^n-m`. The exclusive specifier matches more specific subnets but not the subnet in the specifier. In our example `10.0.0.0/8^-` would match the same prefixes as `10.0.0.0/8^+`, except `10.0.0.0/8`, which is excluded. The exact match `10.0.0.0/8^16` would match any more specific subnet of `10.0.0.0/8` that has prefix length 16. The range operator matches any more specific subnet that has a prefix length that falls into the interval, including the endpoints of the interval.

AS_path regexp

The AS_path regular expressions consist of extended regular expressions subset. Basically, there are no back-references and named groups. Details are in RFC 2622 in section 5.4. The following example shows basic syntax:

```
<^AS1 .* AS2$>
```

It matches only the prefixes that come directly from `AS1` and originate in `AS2` and the ASNs that the matching prefix traversed between these two ASNs are ignored. It also allows the list of ASNs between `AS1` and `AS2` to be empty.

Logical operators

There are three logical operators that might be used in the filters:

1. `AND`
2. `OR`
3. `NOT`

The meaning and precedence of these key words is the same as in most programming languages. The only problem is that the operators can be omitted when two filters with the same selectors are defined either in the different attributes or within one attribute in different expressions delimited by semicolon. In these cases the meaning is the same as `OR` operator.

Precedence of the operators can be modified by parentheses.

filter-set object

Following example shows compound filters that refers to a recursive `filter-set` object:

```
aut-num:  AS29134
...
export:   to PRNG-IGNUM-UP announce AS-IGNUM-OUT
import:   from PRNG-IGNUM-UP accept ANY AND NOT fltr-bogons

as-set:   AS-IGNUM-OUT
members:  AS29134
members:  AS51278
...

filter-set:  fltr-bogons
filter:      fltr-unallocated OR fltr-martian
```

```
filter-set:  fltr-unallocated
filter:      {}

filter-set:  fltr-martian
filter:      { 0.0.0.0/8^+, 10.0.0.0/8^+, 100.64.0.0/10^+,
             127.0.0.0/8^+, 169.254.0.0/16^+, 172.16.0.0/12^+,
             192.0.0.0/24^+, 192.0.2.0/24^+ , 192.168.0.0/16^+,
             198.18.0.0/15^+, 198.51.100.0/24^+, 203.0.113.0/24^+,
             224.0.0.0/3^+ }
```

In this example the `export` attribute uses filter **AS-IGNUM-OUT**. This filter is a reference to `as-set` object of the corresponding name. The `as-set` object contains `member` attributes, which can be either ASNs or further references to different `as-set` objects. In our example there are only two ASNs - **AS29134** and **AS51278**. The parser has to find all `route` objects with `origin` attribute that equals one of these two ASNs. Set of the prefixes specified by the matching `route` object is the actual list of the prefixes allowed by this filter.

The second part of this example is the line `import` that contains compound filter that utilizes two logical expressions `AND` and `NOT`. These logical expressions connect the keyword `ANY` and the `filter-set` object **fltr-bogons**. The meaning of the `ANY` keyword is self-explanatory. The `filter-set` object name has to be resolved: The corresponding object contains an attribute `filter` with the filter expression. The content of this particular filter in our example refers two more `filter-set` objects that has to be further resolved in order to evaluate the filter. The two final objects in the filter evaluation contain empty filter in case of **fltr-unallocated**, which equals to constant negative result. The **fltr-martian** contains a list of prefix expressions to match.

Set operators

There is also a possibility to define refinement and exception to the filters and to nest these constructs. It is therefore possible to define refinements to refinements, exceptions to exceptions, exceptions to refinements and so on. In principle, `refine` statement means that both filters - left and right side of the `refine` keyword have to match in order to execute action in the refinement. An example from RFC 2622 shows simple use of refinement:

```
import: ... { from AS-ANY action pref = 1; accept community(3560:10);
              from AS-ANY action pref = 2; accept community(3560:20);
            } refine {
              from AS1 accept AS1;
              from AS2 accept AS2;
              from AS3 accept AS3;
            }
```

This is a brief version of the following equivalent filter:

```
import: ... {
   from AS1 action pref = 1; accept community(3560:10) AND AS1;
   from AS1 action pref = 2; accept community(3560:20) AND AS1;
   from AS2 action pref = 1; accept community(3560:10) AND AS2;
   from AS2 action pref = 2; accept community(3560:20) AND AS2;
   from AS3 action pref = 1; accept community(3560:10) AND AS3;
   from AS3 action pref = 2; accept community(3560:20) AND AS3;
}
```

Please note that the filters use yet not discussed filter element `community(XX:YY)`, which simply matches BGP community in the path vectors in question.

The function of `except` operator is to selectively change the result of more general filter on the left of the keyword and add a finer filter that succeeds the general one in evaluation order. In order to set relations among the filter expressions there exist considerably complex rules that define how to apply multiple levels of nested `refine` and `except` operators. General rule is that `except` statement has the same precedence as `AND`, because it is semantic equivalent of `AND NOT` expression.

Pre-defined keywords

The last important group of elements in filters are pre-defined words. The most important filter is `ANY`. Another important and widely used keyword is `PeerAS`, which matches only the same origin AS as the AS in peering selector. But there are more additional actions and matches that are not essential for the language itself. We are going to discuss some of them later, the rest is described in RFC 2622.

3. Related work on routing management

3.1 Internet routing research

3.1.1 Data analysis

BGP routing characteristics

The Internet routing system contains a lot of data in the BGP tables and in many supporting data sources as well. There are lots of Internet routing analyses that refer to routing policies or work with available data. The most famous long-term data analysis effort is the work by Geoff Huston of APNIC with his series of papers, for instance [13], [14] and numerous presentations on networking conferences. Most of his results are also available on the web site [15].

Moreover, there is a paper [16] that analyze the most fundamental question: “Do routing policies have any effect on routing in the Internet?”. The asnswer is “Yes.”.

One more summarizing paper on correlating BGP data is [17]. Even though this paper’s purpose is to present a method for finding autonomous systems that generate instability by repeatedly sending BGP updates, it is also interesting because of the methods it devised and used. The most interesting part is the inferring of AS topologies and correlation of different BGP update sources. This work may serve as an inspiration for further work on routing policies validation: More BGP update sources might be connected to the system in order to obtain multi-lateral results that might be subsequently correlated.

Effects of routing policies

The paper [18] shows fundamental relation between routing policies and data flows on data obtained from academic networks. The method is based on investigation, how routing policies for both intra- and inter-domain routing can give rise to violations of the triangle inequality with respect to RTT. Unfortunately, this work is limited to studying of the implementation of routing policies only in academic and research networks. These networks are known for clean design and exceptionally good operation practices that are scarcely matched in the rest of the commercial Internet. This fact leaves space for doubts about applicability of the results to the ordinary networking environment.

Another related paper [19] published in 2000 captures the effects of routing policies on convergence. The most interesting part of this paper is a formal model of Internet route propagation that allows modeling of BGP convergence. The model provides insight into the dynamics of route propagation which might help with understanding the analysis of routing policies that will be presented in this thesis. The paper concluded that the Internet convergence characteristics are insufficient for deployment of real-time technologies and calls for better route validation and authentication mechanisms. Interestingly, not much changed since 2000 in terms of route validation and authentication, yet the real-time services like VoIP became a reality and convergence times in BGP have improved. The reason is most likely deployment of faster CPUs in routers and greater density of the Internet itself.

Inferring of routing policies

Another related work attempts to infer routing policies form BGP. The method and results are described in paper [20] from 2003. It proves that routing policies are widely deployed in the Internet and autonomous system operators use them mainly to achieve load balancing or direct the traffic to the preferred path. The presented method for inferring more information on routing policy of remote AS relies on both BGP and RPSL in IRRs. The problem of this work is the speculative nature of the method and small scale of the presented results.

Moreover, the paper [21] shows BGP configuration analysis compared to high-level policy description. Even though the paper focuses on local misconfigurations that might occur in an autonomous system and affects usually the directly-connected neighbors, it also provides analysis of common BGP configuration patterns and most frequent errors in BGP configuration. The conclusion of the paper calls for more centralized and higher-level BGP management system based on policy specification language that could be directly applied to the router.

Another method for detecting possible routing misconfigurations and attacks on BGP is presented in paper [22]. The method is based on correlating known data on distance between autonomous systems and validation of the geographical location of the prefix originators from RIR databases. The method shows that correlation among physical locations and span of the network exists and it can be used to detect anomalies.

3.1.2 Routing description languages

Practical approach - RPSL

More general and higher-level research of routing and routing description exists in greater extent. There are supporting papers that have been written prior to the creation of the current set of standards or approximately at the same time.

Most notably, there is the paper [23] that was co-authored by the RFC 2622 authors. This paper provides a summary of the IRR system outline and reasoning for some choices made.

Several papers [24], [25], [26] had been written in succession to endorse this idea and to find new use cases for the RPSL or to analyze possible effects of routing policies. The most representative example of this period is [27]. Authors of this paper described the most common and widely used design patterns in the Internet routing system and subjected them to critical analysis with respect to common errors, failure modes, scalability issues and security concerns.

3.1.3 Theoretical approach

The paper [28] provides high-level view on routing policies and possible languages and tools for capturing them. The most important contribution of this paper is a creation of “Formal Definition of Path-Vector Systems”, which is an algebraic description of the routing system. This served as the basis for the following algebraic description of the routing policy and relation to the BGP routing in the next chapter.

Another paper [29] shows different way of thinking about computer networking from service oriented point of view and suggests employing more sophisticated automation.

3.1.4 Trends and current research

S-BGP

The first related subgroup of current research that has certain overlaps with RPSL comprises of Resource Public Key Infrastructure (RPKI) [31], [32] and Secure BGP (S-BGP). These interconnected standards have been defined in RFC 6480 [30] and following series of RFCs 6481 - 6493. Even though the standards intend to build an independent and cryptographically-secure delegation tree of resource certificates that should coexist with both BGP and RPSL routing policies, it might eventually supplement or replace the function of `route` objects in RPSL.

The papers [33] and [34] provide insight into the standard definition and into intended operational practices and possible deployment procedures.

Implementation report [35] from 2014 describes current software and deployment status at the top levels of the resource delegation hierarchy. The paper [36] shows that utilization of RPKI remains low.

BGP prefix hijacking analysis

The long history of malicious or accidental BGP hijacking has been described on numerous occasions, namely in articles [37], [38], [39].

There is also a new (January 2016) research described in paper [40], which use a method that is similar to this thesis - comparison of BGP data and a database of routing policies constructed partially from RIPE DB. However, the intent and focus of this work is different from mine.

Another direction to address this issue is a research of specialized automata [43] that can model Internet routing system and could detect anomalies and attacks on BGP.

Reactive languages

Rise of Software Defined Networking (SDN) has created an interest in reactive languages[1]. Some projects use general purpose programming languages for definition of SDN controller functions and some have decided to research into special purpose-oriented languages [41].

Example of such a language is Procera [42].

The most practically important and novel domain-specific languages for SDN are *Frenetic* [44] and *Pyretic* [45].

[1]Reactive language is a programming language that facilitate functional reactive programming - paradigm oriented toward data flows and the propagation of changes.

SDN management

NetIDE is one notable project that has set an objective to create an implementation of vendor-independent system with its own network description language for SDN configuration. The project has created an outline of the new networking management platform in papers [46] and [47]. Even though this project is not focused on BGP management, it has great potential to extend in that direction because of its emphasis on scalability and universality.

Route servers

Operation of route servers is a specialized task and the Internet Draft [48] summarizes the main operational aspects that are related to multi-lateral peerings. The document describes the special relationships among ASes that are difficult to capture by the current RPSL.

NETCONF

Another notable effort that is related to management of networks is *NETCONF*. It has been defined in RFC 6241 [67] and in a series of following documents. The scope and focus of NETCONF is different than in RPSL. RPSL is a high-level configuration of the Autonomous System and NETCONF is a management protocol for configuring and operating the routers.

There is obviously a missing component that would interconnect both technologies and transform RPSL data to NETCONF and apply the results to the routers. A new initiative [54] attempts to fill this gap. This project is going to be described in more detail in later sections that analyze the existing software.

3.2 RPSL development

A brief history of the documents RFC 2622 [5] and RFC 4012 [6] has been described in the section 2.2.1.

There is also a supplemental document RFC 2650 [12] called “Using RPSL in Practice”. Nonetheless, many aspects of RPSL are established only by the reference implementation. The current de-facto standard and reference implementation of RPSL-related tools is **IRRToolset** [49].

Multiple parties participated in RPSL standard development and in development of related tools as well. Even though the technology exists for more than 20 years, the development is rather slow.

RPSL development efforts can be partitioned into three groups:

1. Tools development,
2. data analysis,
3. standard amendments and development.

Problem of the RPSL standard is that it is maintained solely by the community. There is only very limited involvement of networking hardware vendors, researchers in the networking field and standardization consortia that are usually formed from the two previous groups. These companies and consortia were the most productive authors of the new networking standards in the last decade. It seems that revision of RPSL has been left behind even during the peak of research effort directed into reactive languages and network descriptions, which had been motivated by SDN research.

The most active, though still relatively very limited development and research effort, that concerns RPSL, takes place in RIPE community. This community is the RPSL's place of origin and it is active in many related topics. The community uses specific form of communication that is based on mailing lists and rather informal meetings on conferences that steer the formal community bodies called *working groups*. The basic difference in delivering results to the community lies in the specific point of view: The community members are usually interested in raw data and simple data interpretation. However, the point of interest has to be kept strictly practical for operation of the Internet and no space is left for speculations. The basic and most respected documents are only RFCs.

Direct responsibility of the RIPE community is limited to RIPE database. Development of the standards in form of RFC documents is off scope and should be done within IETF, but few minor amendments to the existing documents have been created and implemented within RIPE community.

The community is therefore the obvious starting point for research of the RPSL-related topic.

3.2.1 Tools development

IRRToolset

The most important tool for RPSL is the reference implementation - IRRToolset package [49]. It offers feature-complete and practically usable software tools that can interpret RPSL objects, help configure and operate Autonomous System border routers and check portions of RPSL code.

IRRToolset features

The IRRToolSet has been written in C++ and it consists of several tools and common libraries shared among them. There is a basic low-level tool intended for reuse in scripting and for debugging called `peval`. There is also syntax checking tool `rpslcheck`. The most important tool in the suite is `RtConfig` that translates routing policies into configuration formats for several router operating systems, most notably Cisco IOS and Juniper JunOS. There are also tools that can be used to compare routing paths in BGP with RPSL policies. Most notably `prtraceroute` that validates actual routing in an IP network discovered by a method derived from standard `traceroute` tool. Another tool is `prpath`, which lists available paths that can be matched in BGP tables.

IRRToolset issues

The problem of IRRToolset lies in the maintenance. The project is Open Source, however it does not have active community of users and developers. The maintenance responsibilities have been transferred from one maintainer to another several times and there is not much progress in the project, according to the presentation [62].

It has been originally created at the Information Sciences Institute at the University of Southern California as part of the Routing Arbiter project [50]. Responsibility for the IRRToolSet project passed to the RIPE NCC in 2001 in order to add multi-protocol support, which is basically the support for IPv6. Then the Internet Systems Consortium, took over the project in 2004 in order to provide long term support of the stable organization.

Libraries and other projects

Apart from IRRToolSet there is a PERL module called RPSL::Parser [51], which is unfortunately only a RPSL syntax parser. It does not have a mechanism for full interpretation of the filters or any mechanism for recursive resolutions of the filter elements on its own.

Another attempt to create comprehensive library and tool set for parsing and transforming RPSL into router configurations in PERL was my own project called BGF [52]. The project is now abandoned despite the fact that it reached fairly mature status at its time. It has almost complete parser, interpreter and a communication engine. Therefore, it can resolve the RPSL objects and query for missing parts autonomously.

There is also (presumably incomplete) support for RPSL build in NOC project [53]. Moreover, there is a myriad of small tools, written for single purposes, put to GitHub or kept without proper licensing somewhere on the web. These tools can parse RPSL, reuse any mentioned general tools or do something else that concerns RPSL.

There is low-traffic, nevertheless still existing discussion on RIPE mailing lists about RPSL tools development. The prevailing attitude is that a redesign of the tools and of the standard is needed, but no complete proposal has ever been introduced. One example of the most recent (May 2016) discussion on this topic is a thread about future of RIPE DB [56]. The interesting point in this discussion is the fact that for the first time there is someone seriously calling for complete abolition of the current tools and data models. The most important argument for that seems to be the limitation of the current plain-text based data model as well as obsolete software and operational procedures that are bound to the current data model. The following quote from Denis Walker's e-mail sent on May 23, 2016 provides a summary of the proposal:

> What I am suggesting is a serious review of the data model and the database design to identify areas that could be improved in an organized, backwards compatible, step by step process.

mantaBGP

The missing component between RPSL and NETCONF from the previous NETCONF section is currently under development. The project [54] has been recently renamed from *ENGRIT* to *mantaBGP*. The purpose of the project is to create an open-source, modern and comprehensive network management tool in Python.

It should use RPSL as one of the inputs and NETCONF as an output.

ENGRIT/mantaBGP component called *LibRPSL* should become a complete Python RPSL library, parser and interpreter. The project is currently (May 2016) developed by NLNet Labs - one of the most respected European not-for-profit Internet technology research companies.

This project is particularly interesting because it re-uses parts of RPSL parser from **bgpcrunch** software code that has been created for this thesis.

The source repository of the *mantaBGP* project is accessible on GitHub [55].

3.2.2 Data analysis

RIR data analysis efforts

In 2001, RIPE Routing Working Group asked RIPE NCC and any interested party in general to participate in an effort outlined in the document ripe-201 [57]. This document calls for crosschecking European Internet routing against data registered in the RIPE Routing Registry. Both motivation and the outlined crosschecking process are thoroughly described and goals of the process are set in the document.

Even though the timeline has been discussed in the last section of the ripe-201 document, no final date has been set and I am not aware of any formal paper on results of this initiative. However, there is a web page [58] that refers to this document, which summarizes particular additions to different software tools and services of RIPE NCC.

Another comparable effort that intended to measure accuracy of IRRs in the past has been represented by the tool called *Nemecis*. The paper [59] describes this tool and brings preliminary results, however I was not able to find any other data related to this project that I could study or compare with my own observations. The source code of the tool has not been made public to the paper publication date and the authors have declined my recent request (May 2016) for access to their old code.

3.2.3 Standard amendments and development

The documents RFC 2622 [5], RFC 4012 [6], RFC 2650 [12], the IRRToolset software [49] and another libraries and documents that concern RPSL form an impressive ecosystem. However, there are quite severe problems: The basic idea, the standards and the software have been created in 1990's with respect to the Internet of that time. The only update of the RPSL since then was RFC 4012 that added support for IPv6, multicast and a new level of extensibility that was actually never used. The fundamental approach, basic syntax and semantics of RPSL stays almost the same as in the ripe-181 [11] that came out even before RFC 2622. Nevertheless the Internet, routing and associated technologies have evolved rapidly since then. Therefore operation principles and best practices as well as business environment nowadays are completely different from the mid-1990's, when the RPSL was created.

Amendments reflecting operators needs

Most of the amendments to the standard are only small changes in the database operational procedures, done by the proper working groups or IRR management departments and the form of changes varies, from full RIPE document to simple software changes.

Apart from major amendment of RFC 4012 that introduced multi-protocol support in 2005, there has been only one known effort to amend RPSL standard [60]. The purpose of the amendment is to provide an effective tool for defining routing policies for paths that traverse route servers in Internet Exchange Points (IXP).

A route server is a BGP hub in IXPs, it usually facilitates exchange of routes but does not carry the traffic, which is routed directly over the L2 infrastructure of the IXP. The basic challenge of route server operation lies in limiting or selectively allowing routing in a certain direction while other directions remain in the opposite state. The current solution to this problem is based on exploitation of signaling based on BGP communities, which is unnecessarily complex and difficult to manage. In addition, the proposal can alleviate another problem with length of the BGP communities that has been created by adoption of RFC 6793 [61].

Standard revision efforts

The community members have known for a long time that the data in Internet Routing Registries are doubtful both in correctness and extent. Even though the overall level of the data quality is difficult to measure, people devised a hypothesis that the technology needs an update to have better chance of capturing real internet routing and be more useful for operation of the Internet.

As a consequence of these opinions there has been the proposal [62], which has been oriented towards the community and called for the start of discussion about the future evolution of the Internet routing system.

There are even proposals for rebuilding the entire routing system including both BGP and current network design patterns. One possible approach is based on Locator/Identifier Separation Protocol (LISP) [63]. Work on integration of LISP with current Internet is summarized in papers [64] and [65].

Another inter-AS routing management standard is now (May 2016) being developed by NLnet Labs under name *Routing Documentation Language* (RDL). The project is now in early draft [66] phase.

3.3 Other relevant resources

3.3.1 RPSL-related reports

The report [68] has been presented on RIPE 64 Toolset BoF[2]. It described the utilization of RPSL and IRRToolSet in large scale at DeutscheTelecom. Interest-

[2] *Birds of Feather* (BoF) is an informal meeting on RIPE conference that might have full charter, but the topic is too broad or experimental to fit any existing working group, but too focused to be presented on a plenary session.

ing point of this presentation is not only the scale of use but also the great depth of intergration among RPSL and network operation and provisioning tools.

On the contrary, RPSL has been criticized shortly after publication of the specification. In the discussion in mailing list thread [69] there are serious questions about the IRR system and RPSL. Many of them are still not answered today.

3.3.2 Supporting technical standards

Process of creation and amending Internet standards is described in [70] and [71]. These standards are relevant for conclusion phase of this thesis and for setting future work outline.

4. Evaluation of routing policies in the Internet

4.1 Use cases for routing policies

4.1.1 Usage of RPSL in RIPE region

There are three basic use cases for routing policies captured in RPSL based on the network type:

1. End (*stub*) network needs to announce its resources,
2. Transit network needs to validate announcements of its customers,
3. Route servers need to validate all passing announcements.

The publication part of the RPSL utilization process is usually a manual work. Creation of RPSL requires proficiency in RPSL, understanding of the network being described and a considerable effort.

The validation part might be automated to a certain extent and might use IRR as the only data source or on the contrary it might completely ignore public RPSL sources and use its own data in arbitrary format.

End network

The end network needs to describe the following facts:

- For each ASN there has to be exactly one *aut-num* object,
- For each IPv4 prefix there has to be at least one *route* object that captures the binding between the resource and the originating AS that announces the prefix.
- For each IPv6 prefix there has to be at least one *route6* object that has the same function as its IPv4 counterpart.
- Each eBGP connection has to be a part of a selector in at least one pair of *import* and *export* records that captures filters for the eBGP connection.

In the ideal case, the configuration of AS border routers should exactly correspond to the RPSL routing policies. In addition to it, the filters should exactly correspond to the filters defined on the routers and describe all changes and modifications that the routers apply on the prefixes in both directions.

The ideal case is however seldom achieved in reality. The difference is usually in filters, because the level of detail that is needed to successfully announce an IP prefix to the Internet is much lower than the common complexity of the corresponding BGP configuration. Many Autonomous System operators prefer to arrange details of the eBGP connection and filters directly with the neighboring AS operators without use of IRR and RPSL. In that case they may fill basic filters that allow all prefixes in all directions to avoid any future conflict of the restrictive filters and any announcement.

Too restrictive filters that conflict with certain announced prefix can be detected by comparing the announcements and the filters. On the contrary, the case of too permissive filters is very difficult to detect without detailed information about the remote networks.

Transit networks

Transit networks usually announce their own prefixes and the process is the same as in the end network. In addition the transit networks also accept and forward prefixes of other parties. Since the default configuration of BGP propagates all learned prefixes to all directions the transit AS actually need less restrictive filters in certain directions to allow propagation of more prefixes.

Moreover the transit AS should filter input prefixes to prevent any malicious or accidental prefix import and subsequent leakage.

Route servers

Operating a route server means interconnecting many autonomous systems and interchanging routing information among them according to given rules. Even though the route server does not carry traffic its role as decision-making party is very important. Route servers has great potential for leaking unwanted routes. However the operators are aware of that and in order to protect trustworthiness of the route server and the company operating it, they usually introduce strict and more comprehensive filters.

4.1.2 Prefix leakage

Prefix leakage or more formally path vector leakage is a condition when a router announces certain path vectors that it is not supposed to announce even though those path vectors are in its routing table.

The prefix leakage has a potential to cause three types of problems:

1. Injecting new prefixes into DFZ despite the fact that their originator is not entitled to announce them.
2. Redirecting traffic to wrong destination.
3. Redirecting traffic via unexpected route.

The first possibility creates a potential for malicious actions that involve the leaked prefix. The prefix might be used as a temporary source for spam or different kinds of attacks.

The last two problems are the most common cause of traffic *blackholing* - announcing routes but not delivering *all* packets to the destination of the route. The blackholing condition might be caused by different sources of redirection:

- Designed malevolent announcement of a prefix.
- Accidental manual announcement of a prefix.

- Leakage of BGP routes.
- Leakage of internal routing protocol routes.

All these conditions could direct the traffic to local blackholing route - `Null0` interface for instance, or it can route the traffic towards an unexpected path that is not capable of carrying the packets to their destination - either because of capacity or configuration problems along the path.

4.2 Scale and accuracy hypothesis

4.2.1 Hypothesis statement

Statement interpretation

The previously stated hypothesis *"The utilization of RPSL in the current Internet is sub-optimal both in scale and accuracy of the information. The situation is not improving and there is no perspective of change in this trend."* is too general to be directly proven.

The possible interpretation with respect to the problems described above is that the *consumers* of routing policies, namely transit autonomous systems and route servers, cannot verify the incoming prefixes from their neighbors solely on RPSL evidence basis.

This statement is better, however it still concerns actions of many unknown parties. We need to set a threshold from which the information in IRR is not useful anymore. Then we can measure accuracy of the data in IRR with respect to the data in DFZ and compare the results with the threshold.

Accuracy threshold

The threshold is obviously individual for each type of network and application. Despite the differences among networks and users we can surmise that Pareto principle, also known as "80-20" rule, applies to RPSL as to any other complex software system. There is enough evidence in [72] for claiming relation of project success and Pareto principle in software engineering. Pareto principle has been studied in other fields in [73] and using it as an argument for distinguishing majority seems reasonable. However, the more general question of measuring success of an unique project remains open and falls out of scope of this thesis.

For declaring RPSL standard successful according to Pareto principle, it would have to cover 80% of cases with the RPSL data. Thus at least 80% of prefixes in the Internet routing system should pass validation. The rest 20% of cases might be too complex to be accurately captured in IRR. Therefore minor validation failures might happen in these 20% and that failures are deemed harmless for the entire system.

Cause and effect

There are two possible outlooks on the validation failures:

1. The data in IRR are too unreliable. Because of that, an individual AS operator decides not to use the data for BGP filtering. It causes discrepancies between IRR and BGP.

2. The AS operators do not use IRR data for BGP filtering. Therefore an individual AS operator does not care about accuracy and reliability of IRR data that he is supposed to maintain. That causes discrepancies between IRR and BGP.

The most important problem of the analysis is the fact that we can only measure the number of differences in expected paths according to routing policies and the actual path vector propagation in DFZ, but we can not decide on the root cause. It has a practical implication in form of a speculative question: "Who is to blame for the differences between IRR and BGP?" Unfortunately, this question remains unanswered. However, related topic will be discussed in chapter 6.

4.2.2 Definitions

Regardless of the culprit and the root cause that we can only speculate on, we can provide the formal definition of the planned measurements.

Path vector

Let $\vec{\rho} = (p, \vec{\gamma}, M)$ be the path vector that consist of prefix p, `AS_path` $\vec{\gamma}$ and metrics M. From $\vec{\gamma} = (a_0, a_1, ..., a_n)$, where a_j are autonomous system numbers, we can construct an expanded path:

$$\vec{P} \equiv ((a_0, a_1), (a_1, a_2), ..., (a_{n-1}, a_n))$$

It contains all the autonomous systems the prefix has traversed to get from its source to the observation point. The path is captured in the usual order: On the left side is the observation point a_0 and on the right side there is the originator's AS a_n.

Hop

It proves useful to name the transition from one AS to another a *hop*. Let h_j be the hop from AS a_{j+1} to a_j:

$$h_j \equiv (a_j, a_{j+1})$$

IRR data coverage

To express the fact that we have data for an Autonomous System a so we can look up related `route` and `aut-num` object in IRR, we define the following function:

$$d(a) \equiv \begin{cases} 1 & \text{if we have authoritative data for the ASN } a \\ 0 & \text{otherwise} \end{cases}$$

RIPE DB data coverage

In our case, this function is equivalent to the function $R(a)$ that indicates the AS a belongs to RIPE NCC service region:

$$d(a) = R(a) \equiv \begin{cases} 1 & \text{if ASN } a \text{ operates withing RIPE NCC service region} \\ 0 & \text{otherwise} \end{cases}$$

Route object validation

To check `route` object validity we define the function:

$$r(p, o) \equiv \begin{cases} 1 & \text{if } d(o) = 1 \bigwedge \text{ there is a } \texttt{route} \text{ object with prefix } p \text{ and origin } o \\ 0 & \text{otherwise} \end{cases}$$

Hop validation

Let e_{h_j} be the export filter in AS a_{j+1} towards AS a_j and let i_{h_j} be the import filter in AS a_j from AS a_{j+1}.

For validating hops we define functions:

$$v^e(p, j, (a_j, a_{j+1}, ..., a_n)) \equiv \begin{cases} 1 & \text{if } d(a_{j+1}) = 1 \bigwedge \text{ there is } \texttt{aut-num} \text{ object} \\ & \quad \text{for AS } a_{j+1} \text{ with matching export line} \\ & \quad \text{that defines the filter } e_{h_j} \text{ and} \\ & \quad \text{the filter matches the prefix } p \\ 0 & \text{otherwise} \end{cases}$$

$$v^i(p, j, (a_j, a_{j+1}, ..., a_n)) \equiv \begin{cases} 1 & \text{if } d(a_j) = 1 \bigwedge \text{ there is } \texttt{aut-num} \text{ object} \\ & \quad \text{for AS } a_j \text{ with matching import line} \\ & \quad \text{that defines the filter } i_{h_j} \text{ and} \\ & \quad \text{the filter matches the prefix } p \\ 0 & \text{otherwise} \end{cases}$$

Path decidability

The path decidability is defined by the following function:

$$D(p, \vec{\gamma}) = \prod_{a_j \in \vec{\gamma}} d(a_j)$$

Path validation

A complete validation of j-th hop is described by the following function:

$$v(p, j, (a_j, a_{j+1}, ..., a_n)) = v^e(p, j, (a_j, a_{j+1}, ..., a_n)) \cdot v^i(p, j, (a_j, a_{j+1}, ..., a_n))$$

Therefore path validity can be defined by this function:

$$V(p, \vec{\gamma}) \equiv r(p, a_{|\vec{\gamma}|}) \cdot \prod_{j=0}^{|\vec{\gamma}|-1} v(p, j, (a_j, a_{j+1}, ..., a_{|\vec{\gamma}|})) \cdot v^e(p, |\vec{\gamma}|, (a_{|\vec{\gamma}|}))$$

4.2.3 Failure set indicator functions

We can evaluate a "validity indicator" V for each prefix p and associated `AS_path` $\vec{\gamma}$ in DFZ. Moreover, we can define indicator functions for any type of errors based on recognition functions that can analyze each hop.

Route validation failure

For instance we can define a function:

$$\bar{r}(p,o) = \neg r(p,o) \wedge d(p,o) = 1$$

This is an indicator of `route` validation failure. There are two major types of errors concerning `route` objects: $\bar{r_1}$, $\bar{r_2}$. The semantics of the two error types will be explained later.

Route validation completeness

Let Δ be the set of all path vectors $\vec{\gamma}$ in DFZ, then following obvious equations hold:

$$|\{(p,\vec{\gamma})|\bar{r}(p,o)=1, \vec{\gamma}=(a_1,a_2,...,a_{n-1},o),(p,\vec{\gamma})\in\Delta\}| =$$
$$= |\{(p,\vec{\gamma})|\bar{r_1}(p,o)=1, \vec{\gamma}=(a_1,a_2,...,a_{n-1},o),(p,\vec{\gamma})\in\Delta\}| +$$
$$+|\{(p,\vec{\gamma})|\bar{r_2}(p,o)=1, \vec{\gamma}=(a_1,a_2,...,a_{n-1},o),(p,\vec{\gamma})\in\Delta\}|$$

$$|\Delta| = |\{(p,(a_1,a_2,...,a_{n-1},o))|r(p,o)=1,(p,(a_1,a_2,...,a_{n-1},o))\in\Delta\}| +$$
$$+|\{(p,(a_1,a_2,...,a_{n-1},o))|\bar{r_1}(p,o)=1,(p,(a_1,a_2,...,a_{n-1},o))\in\Delta\}| +$$
$$+|\{(p,(a_1,a_2,...,a_{n-1},o))|\bar{r_2}(p,o)=1,(p,(a_1,a_2,...,a_{n-1},o))\in\Delta\}| +$$
$$+|\{(p,(a_1,a_2,...,a_{n-1},o))|d(o)=0,(p,(a_1,a_2,...,a_{n-1},o))\in\Delta\}|$$

Hop validation errors

The same inversion function $\bar{V}$ for decidable path vectors can be defined for the V function that validates paths and similar sub-types of $\bar{V}$: $\bar{V_1}, \bar{V_2}, ..., \bar{V_t}$ can be defined analogically. The analogous error indicators might be extended to hop validation, where the definition of inversion function for j-th hop depends on decidability of a_j and a_{j+1} in `AS_path`:

$$\bar{v}(p,j,(a_j,a_{j+1}...,a_n)) = \neg v(p,j,(a_j,a_{j+1}...,a_n)) \cdot d(a_j) \cdot d(a_{j+1})$$

The semantics of these error indicators will be described in greater detail later in this thesis.

4.2.4 DFZ subsets

The data available in form of daily BGP dumps and RPSL data archive allows us to evaluate the indicator functions for each path vector and construct subsets of DFZ. The table 4.1 contains a brief list of fundamental subsets.

Indicator function	Routing semantic
$d(a)$	data are available for AS a
$\bar{d}(a)$	AS a can not be resolved in RIPE DB
$r(p, o)$	prefix p can be originated by AS o
$\bar{r}(p, o)$	prefix p should not be originated by AS o
$v^i(p, j, ...)$	prefix p matches input filter in AS a_j from AS a_{j+1}
$\bar{v^i}(p, j, ...)$	prefix p does not match input filter in AS a_j from AS a_{j+1}
$v^e(p, j, ...)$	prefix p matches export filter in AS a_{j+1} towards AS a_j
$\bar{v^e}(p, j, ...)$	prefix p does not match export filter in AS a_{j+1} towards AS a_j
$v(p, j, ...)$	prefix p passes filters on hop between AS a_j and AS a_{j+1}
$\bar{v}(p, j, ...)$	prefix p fails at least one filter on hop (a_j, a_{j+1})
$V(p, \vec{\gamma})$	path vector with prefix p and `AS_path` $\vec{\gamma}$ is valid
$\bar{V}(p, \vec{\gamma})$	prefix p with `AS_path` $\vec{\gamma}$ fails at least one hop filters

Table 4.1: Subset indicator functions

The most interesting subgroups of sets derived from $\bar{r}(p, o)$ function are the two `route` object failure types:

1. No matching `route` object exists for the prefix in question.
2. At least one matching `route` object for the prefix exists but none of the objects has a matching `origin` attribute.

Obviously the first type of error is either a result of negligence or product of wide-spread misconception about the validity of `route` objects: People tend to think that a less specific `route` object covers all more specific announcements, which is not true.

The second type of error represents a real conflict that might be either the result of much more serious negligence or the indicator of IP prefix hijacking.

The failure types derived from $\bar{v}(p, j, \vec{\gamma})$ are the following:

- The `aut-num` object for AS a_j in `AS_path` does not exists, even though the AS belongs to RIPE NCC service region, and therefore the object should exist.
- The `aut-num` object for AS a_{j+1} in `AS_path` does not exists, even though the AS belongs to RIPE NCC service region, and therefore the object should exist.
- Missing a matching `import` line in `aut-num` object for AS a_j that allows importing routes from a_{j+1} to a_j.
- Missing a matching `export` line in `aut-num` object for AS a_{j+1} that allows exporting routes from a_{j+1} to a_j.

- The filter for exporting routes from a_{j+1} to a_j exists but does not match the prefix.
- The filter for importing routes to a_j from a_{j+1} exists but does not match the prefix.

4.2.5 Evidence for the hypotheses

To collect evidence supporting the hypothesis we need to measure the size of the subsets of DFZ, especially the subsets that represent errors of various kinds and show that it consistently falls outside of limits we set.

The analysis is the topic of the following sections as well as more detailed analysis is added to the thesis as appendix A. However, high-level results show that paths compared to filters in `aut-num` objects by far do not reach the limit of 80% correct paths. Surprisingly even the trivial relation between originators and `route` objects is showing unfavorable ratio of errors.

IPv4 route objects

The figure 4.1 shows that the percentage of `origin` validation failures in `route` objects is consistently over 20% in the entire analysis period - starting 2012 and spanning to the end of 2015. The chart in figure 4.2 shows absolute value time series of the validation results in this period. The result for each day is equal to the size of sets

$$\{(p,o)|r(p,o)=1,(p,(a_0,...,a_{n-1},o))\in\vec{\gamma},(p,\vec{\gamma})\in\Delta\}$$

for line marked "*OK*" and

$$\{(p,o)|\bar{r}(p,o)=1,(p,(a_0,...,a_{n-1},o))\in\vec{\gamma},(p,\vec{\gamma})\in\Delta\}$$

for "*validation failure*" line. The Δ is DFZ snapshot of a day within the observation period.

IPv6 route objects

The figure 4.3 shows the percentage of `origin` validation failures in `route6` objects for IPv6. The result is slightly better than the IPv4 case, however there are extended periods when error ratio exceeds 20%. The chart in figure 4.4 shows comparison of absolute numbers of the validation results, which are naturally lower for IPv6 than for IPv4.

Current deployment of IPv6 in the Internet concerns mainly technical leaders in the industry. It is sensible to estimate that the quality of routing paths and data in IRR are most likely going to deteriorate when less technically-capable networks finally connects to IPv6 Internet. Despite the prediction the trend is not yet visible. Perhaps it is too soon to determine the trend, because IPv6 is still not an absolute necessity (in May 2016), so the less-capable network operators might still be avoiding it.

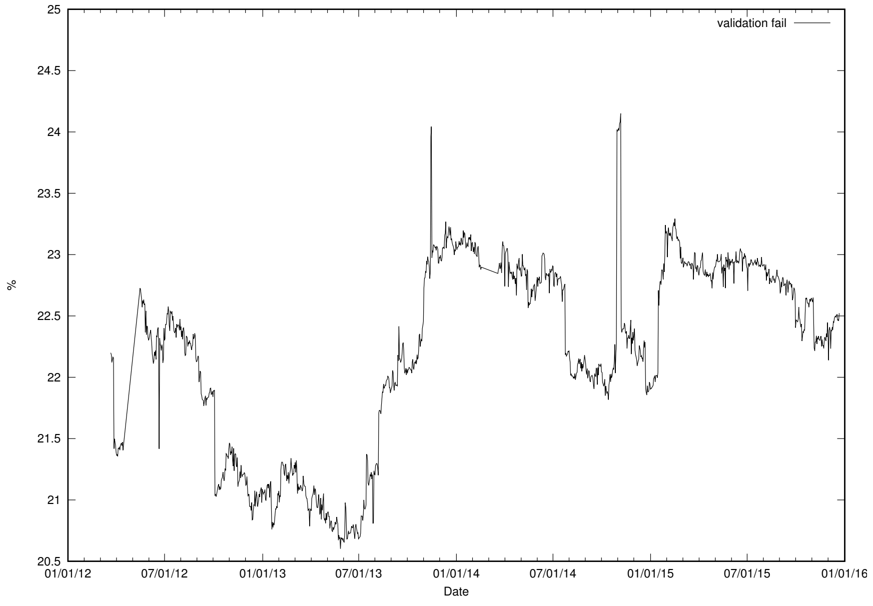

Figure 4.1: IPv4 route origin validation percentage

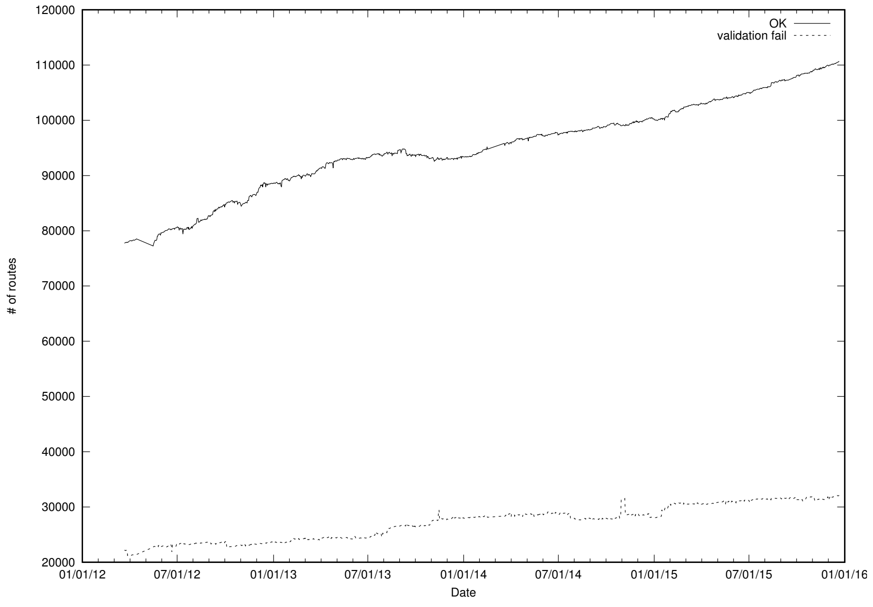

Figure 4.2: IPv4 route origin validation results

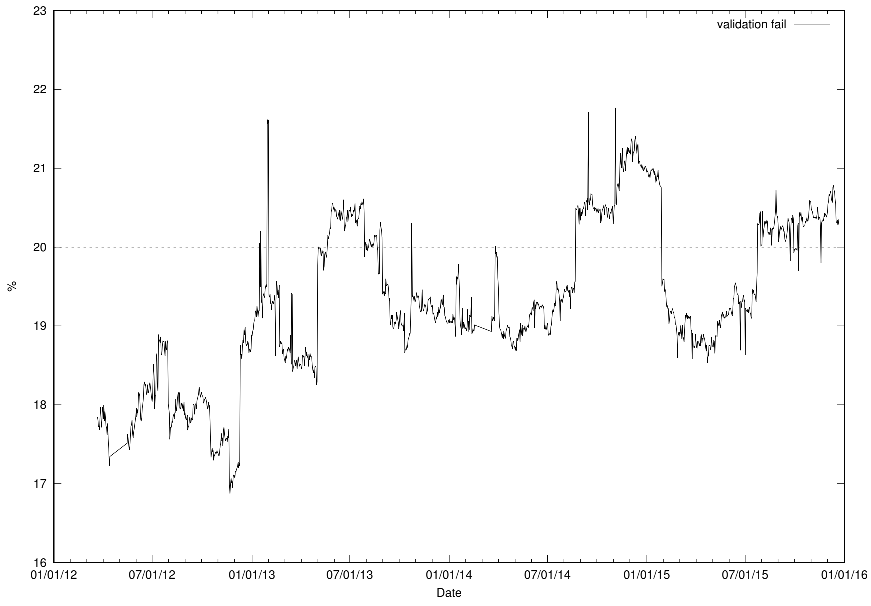

Figure 4.3: IPv6 route origin validation percentage

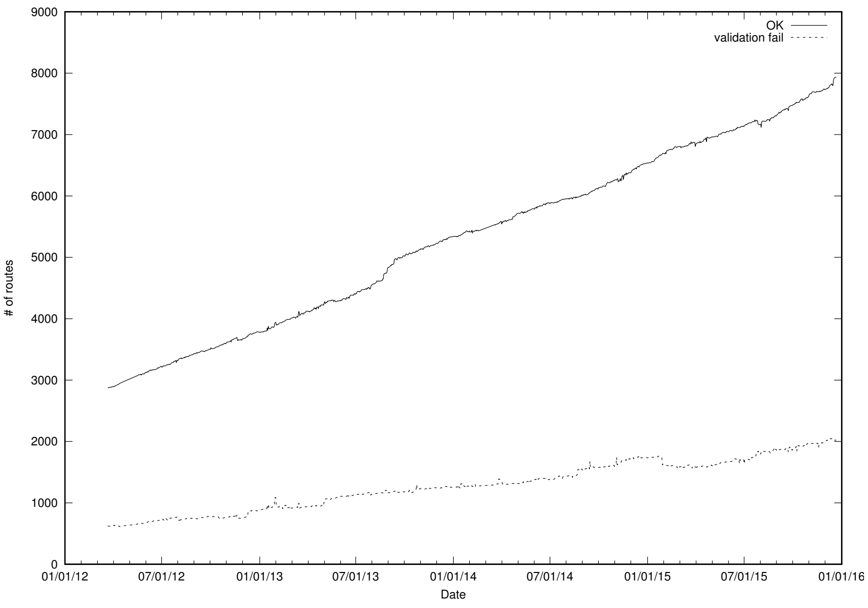

Figure 4.4: IPv6 route origin validation results

IPv4 paths

The paths are more difficult to validate from the particular data that I have. The main problem is that my observation point (**AS29134**) uses a network as the primary upstream provider, which primarily operates outside of RIPE NCC service region. The consequence is that RIPE DB does not contain authoritative data for this AS, which is equivalent to $d(a_1) = 0$. Thus $D(p, \vec{\gamma}) = 0$ for each path that traverses this AS holds.

Even though the majority of paths are undecidable under the strict rules, we can still decide on each *hop* in the path and count hops independently. The idea is that we do not need to validate the entire path to measure the number of detectable failures in hops along the path as well as to count the valid hops. Formally, we count size of the set

$$\{(p,j) | v(p,j,(a_j,...,a_{|\vec{\gamma}|})) = 1 \land r(p,a_{|\vec{\gamma}|}) = 1, a_j \in \vec{\gamma}, (p,\vec{\gamma}) \in \Delta\}$$

for the positive ("*OK*") set and

$$\{(p,j) | \bar{v}(p,j,(a_j,...,a_{|\vec{\gamma}|})) = 1, a_j \in \vec{\gamma}, (p,\vec{\gamma}) \in \Delta\}$$

for the negative ("*failure*") set.

The figure 4.5 shows time series of valid and failed hops in absolute numbers and the figure 4.6 shows percentage of errors. All types of hop filter failures are summarized for better visibility.

The percentage time series shows that the minimum error ratio in the measurement period is 26.7% and maximum is 67.4%.

The reason for this high variability is that the observation point has changed the main upstream networks several times: The first change occurred gradually from September to November 2012. Second change occurred in May 2014, third in January 2015 and then in March 2015 and July 2015. The first upstream hop has a great potential to change the statistics since average path length to all IPv4 prefixes in DFZ, from my observation point, is approximately 3.8.

IPv6 paths

The figure 4.7 shows the same type of results for IPv6: Time series of valid and failed hops. And the figure 4.8 shows percentage of errors. The results for IPv6 are much worse than for IPv4: Minimum failure ratio in the observation period is over 72% and maximum under 86%. Fluctuations are similar and the reason for them is analogous as in IPv4 case.

Summary

It would be too early to conclude the analysis before gaining deeper understanding of the relations among data and the RPSL. At this point we can say that the evidence supporting the hypothesis concerning scale and accuracy of IRR is strong enough not to reject it. Deeper analysis of the IRR and DFZ data will provide enough evidence for conclusion in favor of the hypothesis later in following chapters.

Finally, the detailed analysis of the available data is provided in the appendix A.

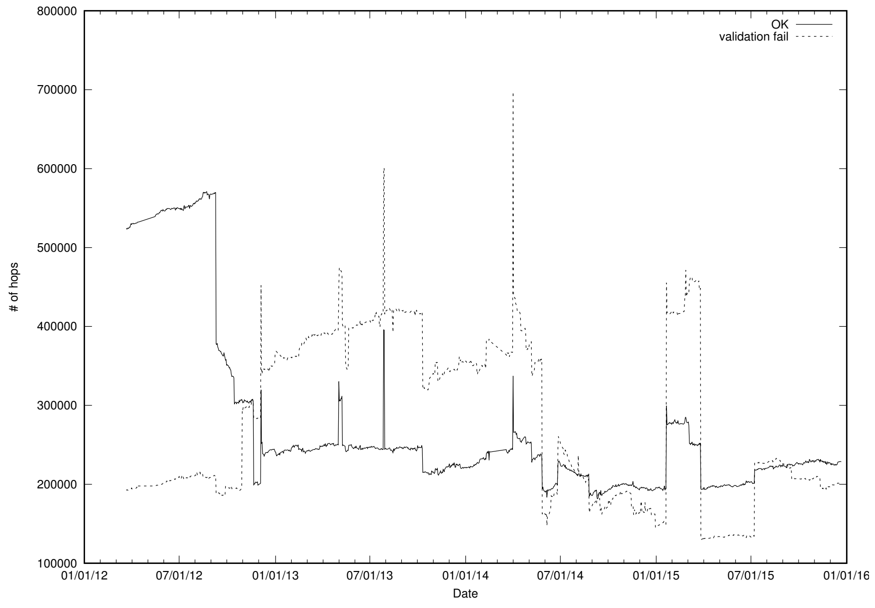

Figure 4.5: IPv4 hop validation results

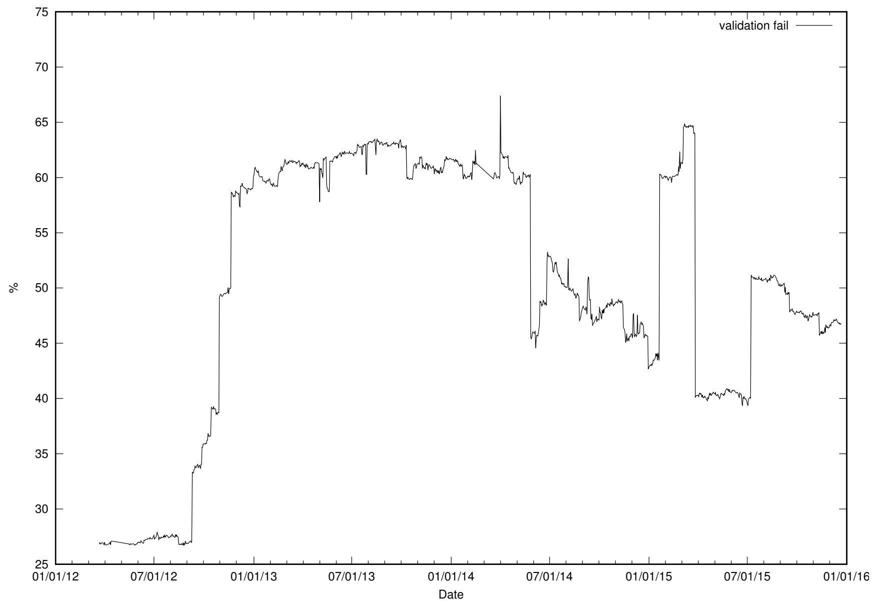

Figure 4.6: IPv4 hop validation percentage

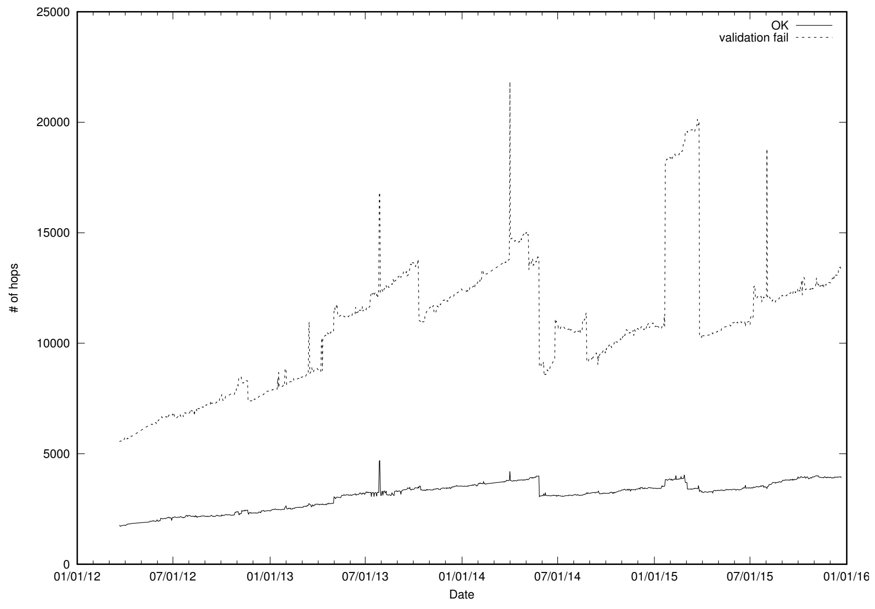

Figure 4.7: IPv6 hop validation results

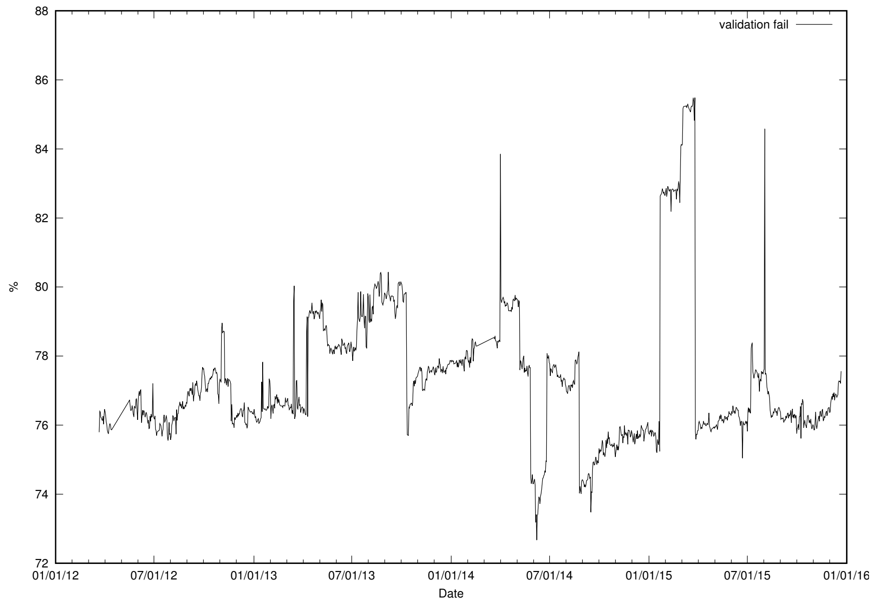

Figure 4.8: IPv6 hop validation percentage

Operational experience shows that information in `route` and `route6` objects is considered more useful than `import` and `export` filters in `aut-num` objects. This observation is in accordance with the error ratios: The percentage of errors in `route` and `route6` objects is closer to the suggested 20% limit than the hop validation results that give little hope for end-to-end path validation.

Another argument in favor of the hypothesis is that if we marked the prefixes that failed either origin filter or path verification as route leaks and decided to filter them out, we would disconnect major portion of the networks and render the filtered Internet connection most likely unusable.

5. RPSL

The previous chapter focused on analysis of BGP paths with respect to RPSL data. This chapter is going to analyze RPSL itself. RPSL is a language and therefore it is only a tool for describing BGP configuration. Instead of creating a meta-language and attempting to analyze the language from theoretical point of view, we will explore tools and steps needed for practical utilization of RPSL: Creation of routing policies in RPSL, publishing them and using our own or others' routing policies for configuring routers.

5.1 RPSL decomposition and semantization

The theoretical analysis of the routing languages has been provided by [28]. The reasoning about the language might occasionally refer to theory of automata and theory of computational complexity.

5.1.1 RPSL language categorization

Even though I am not aware of any formal proof of RPSL language categorization into Chomsky hierarchy, we can assume for this chapter that since we need context sensitive parser to parse certain RPSL statements, it has to be context-sensitive language (type-1 grammar). The common consensus, based on reasoning in [74], is in agreement with this categorization.

5.1.2 IRR components

Apart from the language specification there are important software features that affect use of RPSL in practice. The most important ones are the IRR *databases*[1]. The reason for the tight bond with the IRR database software is that there are three basic operation with RPSL:

1. Searching objects: This task is usually performed by the IRR database or by third-party software, but always with IRR DB original data.
2. Parsing objects and collecting data: This is usually a responsibility of the client that connects to IRR DB. The most important part of this phase is recursive resolution of the referenced object elements.
3. Semantic analysis of the obtained RPSL data and their translation into internal data representation: This phase is clearly application specific task.

5.1.3 RPSL data sources

The RPSL specification defines an *object* as a block of text. Objects are separated by blank lines. The *attribute* is defined as a line or lines that contain attribute name and data. Each object type has a primary identifier, which is an attribute that has the same name as the object type in most cases.

[1]IRR database is a collection of text files that contains RPSL objects.

The identifiers and certain selected attributes are *class keys*. It means that these attributes are designated to serve as lookup keys. The details of searching the RPSL objects in the database are left for further specification as an implementation-specific detail by the RPSL standards.

Nonetheless, utilization of RPSL data depends on the way of requesting and obtaining the data and on ability to search them. The traditional way is based on publishing the *database* over FTP in the form of text files and providing interactive *whois* service. Unfortunately, these two access methods does not seem to facilitate the common use cases well.

Whois protocol

In case of the *whois* access, the standardized protocol is only an envelope for transmitting arbitrary data, and therefore the IRR database has to define and implement its own data access protocol. In case of RIPE NCC, the protocol is bound to **RIPE Database** software and is defined in the official documentation [75]. Other RIRs have either their own database software and their own protocol or use the open-source, the most popular and the most successful software **RIPE Database**.

The most severe problems of this access method are:

- Fragmentation of the access protocol and lack of standardization,
- limited support for bulk queries,
- *round trip times* that slow down recursion,
- limitations of the *whois* protocol that impose restrictions on data content,
- lack of data authentication,
- lack of central registry,
- lack of transactions and susceptibility to race conditions when data are modified by third-party.

There are certain methods for overcoming or alleviating some of the listed problems. For instance RIPE DB supports "*pipelining*" method for issuing multiple requests in one session to reduce problems related with round trip time.

A new IETF committee was formed in 2003 in order to lift the limitations on the content that results from *whois* protocol and to create a new standard for looking up information on domain names and network numbers. Numerous RFC documents were created to define a new protocol called Internet Registry Information Service (IRIS). In 2013, the IETF acknowledged thet IRIS had not been a successful replacement for *whois* and it rendered the new protocol irrelevant. The primary technical reason for that appears to be the complexity of IRIS.

Text files

The supplemental method of publishing data in the form of text files is even more difficult to use because it requires the client application to implement searching in the IRR data. Searching in the IRR data is a complex task because the application needs to partially parse the files and objects in files and to build index tables to search in the IRR data. The data in files are usually provided in random order and *back-references* in objects require complete indexes and creation of complex meta-data.

The problem is not only the added complexity of searching that is a prerequisite for any simple task that needs to issue bulk queries to any IRR. Another problem lies in the scale: IRR is a data source that may contain information about any possible route in the Internet. Describing not only the existing routes but any potential routing path is much more space consuming. In fact, the current (May 2016) unpacked RIPE DB text dump is approximately 5 gigabytes.

RESTful API

Even though the standardization effort that created a successor for *whois* failed, RIPE NCC and ARIN started to provide the IRR data through *RESTful web services*. A new effort to standardize this type of service started in 2012 and several RFC documents [76], [77], [78], [79], [80] have been produced. This new type of service lifts many limitation of *whois*, the downside however is that it adds more complex and demanding HTTP and HTTPS transport protocols.

5.1.4 Parsing RPSL objects

There are two known general approaches that can be applied to parsing RPSL objects:

1. Using Extended Backus–Naur Form metasyntax to describe the context-free envelope of the object and then decompose the contextual part.

2. Create a data model in a particular programming language and use *lazy* decomposition approach, which is straight-forward in programming languages that support *lazy* evaluation.

EBNF parsers

Both approaches have advantages and disadvantages: The Extended Backus–Naur Form (EBNF) description is relatively simple, the EBNF descriptions for RPSL objects exist and existing EBNF parser libraries can be used.

The disadvantage is an addition of an unnecessary step between the RPSL and internal representation, addition of an overhead of the EBNF parser and addition of certain restrictions that depend on the parser implementation. The most important restriction is that the parser have to parse the entire object at once. This does not make sense in case of bulk operations when we need to extract only small portion of the objects.

In this case it proves to be faster and simpler to parse the textual representations of the objects by regular expressions or to filter the objects before using general-purpose parser. This combined approach might be the best way when we need complete coverage of RPSL and reliable syntax checking, it is however potentially both complex and slow because it might have to duplicate work.

Lazy data model

Creation of *lazy* data model is more a work-flow management method and it would need some parser or parsing engine in any case. The difference is that we can resort to simpler parsers like regular expressions and we can capture the context in the object model that hold links to both processed data and unprocessed textual fragments that wait for the explicit requirement to parse them due to the *lazy* evaluation principle.

I decided to use this model in my RPSL library **bgpcrunch** after evaluating the previous approach and after making a conclusion that it would take too much overhead to completely parse all the objects in the daily database dumps in order to be able to search in them.

The *lazy* data model provides more insight into the parsing process and allows to skip unnecessary information and parsing steps to speed up the process. Another advantage is that this process can directly construct the resulting flat data structures on demand, because it is able to immediately lookup any identifier and replace it with the resolved information.

An obvious disadvantage of this approach is greater complexity that is directly dependent on complexity of RPSL and on the need for deep understanding of both the programming language and the RPSL and their overlaps and differences.

5.1.5 Semantic analysis and translation

Known applications that use RPSL falls into following categories:

- Syntax checkers
- Router or route server configurators
- Path resolution tools
- Statistic collection tools
- General purpose libraries

Apart from syntax checker, all other types of software need semantic engine that is able to interpret information in RPSL.

Issues of RPSL semantics

The objects described above carry different types of data. By connecting the data together it is theoretically possible to construct configuration of routers in the chosen AS. It means that RPSL might contain information like IP addresses or routers, configuration of BGP peers and BGP filters. The problem of RPSL is that the information is stored in an unusual format and in different linked objects.

There are major issues of the RPSL parsing:

- RPSL objects can contain comments and unstructured fields that can be ignored.
- RPSL objects can repeat themselves with minor differences. Repetition rules are different for each object type.

- RPSL objects can be searched either by the identifier or by any key attribute.
- `-set` objects might refer to another `-set` objects.
- `-set` object members might be appended by *back-references* from the objects of the compatible type.
- Selectors on `import` or `export` lines in `aut-num` object can be sets.
- Selectors on `import` or `export` lines in `aut-num` object might be repeated or overlap with previous selectors.
- Overlapping or repeating selectors in `aut-num` object might use different filters. In this case the connecting operation is *OR*.
- Filters can be compound expressions that contain both logical and set operators and might be recursively nested via `-set` objects.

These problems are a direct result of the existing RPSL standards that are unusually permissive in matters of formatting the RPSL data in databases and are broad in the syntax specification. The meanings of particular RPSL constructs are minimally described and the descriptions often depend on examples in the RFC standards.

Translating RPSL

One of the use cases for RPSL is creating router configurations out of RPSL prescriptions. This task requires a capability of translating certain RPSL constructs into the configuration language of the particular router. It is easy for basic elements like IP addresses, BGP peers and basic lists. The potential problem lies in two aspects of RPSL:

1. Potentially *unlimited scale*[2].
2. Use of advanced elements in RPSL: Regular expressions or complicated filters for instance.

It is interesting that these two aspects are interconnected to a certain degree. The first problem is derived from the fact that routers usually have strictly limited resources in terms of memory and CPU power in comparison with the scale of RPSL data.

The second problem lies in the fact that routers usually have limited configuration language and certain RPSL constructs might be too complex for direct translation. Possible solutions are either dropping part of the RPSL information or expanding the RPSL elements to simpler ones. Expanding the RPSL elements might therefore cause creation of many filtering rules that would cause the previous type of problem.

[2]Scale of the RPSL information is much greater than the practical limits on filter length in majority of, if not all, current routers. Even though general purpose computes can store and recursively resolve RPSL data, the results can be too long for direct use in routers.

The translation is theoretically a less complex task because the work is split between the translator and the router that has to interpret the resulting rules. This balance is of course fragile and depends on compatibility of RPSL and the filtering mechanisms in router operating systems. Luckily, RPSL has been specified with respect to existing routers and the regular expressions allowed in RPSL matches capabilities of common routers.

5.2 RPSL production

5.2.1 Object creation

The RPSL standards do not explicitly specify any preferred method for creating RPSL statements. The obvious method of choice is manual data entry. It was also the first and perhaps one of the most widely used methods.

It is obviously possible to manage Autonomous Systems of moderate size manually. This approach has an advantage in the fact that there is a possibility of double-checking router configuration and manually created RPSL data against each other.

Larger deployments often use automation for generation and maintenance of RPSL descriptions. In this case, the RPSL is usually considered to be *low-level* language that has to be handled by specialized back-end of the network management application. This is the case of the previously mentioned NOC project [53].

5.2.2 IRR update mechanisms

The traditional update mechanism for IRR databases was an e-mail. Initially, it was simply a matter of sending the changes to the designated hostmaster who manually verified the changes, committed them into the database files and sent the response to the requester.

Later, an automation took over these tasks on both sides: The IRR databases use e-mail *bots*[3] to process the incoming e-mails. And a communication protocol, processing rules and authentication method have been created for the communication between the client and the IRR e-mail bot.

On the client side, people wrote a lot of scripts to help format RPSL and the resulting e-mail to avoid sending improperly formatted message.

Another problem with this method is that it originally used plain text passwords for authentication over the untrusted channel like e-mail. More to it e-mail can be misdirected or bounced to third-party by accident, disclosing the password to random people.

Later updates to this mechanisms introduced web services that allow sending updates over a web form and via simple machine-to-machine interface over HTTP or HTTPS protocols. In case of RIPE Database software, the mechanisms are called *Webupdates* and *Syncupdates*. The advantage of using web forms for users lies in the possibility of instant error checking. And HTTP or HTTPS is obviously

[3]This terminology comes from Internet Relay Chat. The IRC bot is a set of scripts or a program that connects to Internet Relay Chat as a client and appears to other IRC users as another user.

much better for scripting than e-mails because it is much more difficult to interact over e-mail than over HTTP from a scripting language.

The latest development in European IRR brought a new *RIPE DB API*, which is a RESTful API that conveys RIPE DB objects in XML or JSON formats. Details of the API are described in the documentation on GitHub [83].

5.3 Extended hypothesis

The previously stated second hypothesis was an answer to the question: "*What is the cause of the sub-optimal utilization of existing mechanism?*" or "*What can be done to improve the situation?*". The possible answer is: "*High added workload compared to relatively low benefit obtained from extensive utilization of RPSL prevents the AS operators from wider deployment.*"

The fact that the RPSL routing policies and IRR system are sub-optimally used is the consequence of data accuracy problems which was the subject of the previous chapter. That hypothesis can be considered as a valid fact because of measurement results presented in the previous chapter and because of the analysis protocol in the appendix A.

The answer to the question "*What can be done to improve the situation?*" is going to be the topic of the last two chapters.

Therefore, the remaining part of this chapter has to provide evidence for the "workload-benefit disbalance" hypothesis, more specifically it has to describe possible benefit of RPSL deployment and the extra workload it would add.

5.3.1 Possible benefits of RPSL deployment

Automated configuration

The idea of documents mentioned above that serve as RPSL standards is to use RPSL as a primary configuration source of all external relations in an AS. AS operators are supposed to create their routing policies by filling in their `aut-num` objects and creating proper connected `-set` objects that might reference `-set` objects of other parties. Then they should create records for the routes to be originated in their AS to `route` and `route6` objects. The final step should be checking of the RPSL configuration and publication of the complete data in IRR database.

With this data in place the AS can generate configuration files for their ASBRs using IRR as a data source.

Issues in this use case are connected with the complexity of stating the routing policies in RPSL, because it is unique and rare skill. Complexity of learning RPSL is, in my opinion, comparable to learning a new programming language.

Another problem with this scenario is confidentiality of configuration data that describes the AS internals and general low trust in the data held by third-party. These security concerns may prevent the AS operator from publishing the complete data externally. This would add even more workload for the AS operator to keep data internally and filter the outputs that can be published.

Even though it is complex task, there are Autonomous System operators that are doing this. The best known, and perhaps the only publicly known AS that operates internal RPSL-based management system, is Deutsche Telecom. Deutsche Telecom provided the report on their operational experience with RPSL in [68].

Automated peering in IXPs

Another use case for RPSL is configuration of route server in internet exchange points. The configuration is usually too extensive to be managed by hand since it should accommodate to needs of n parties that might form $\frac{(n-1)^2}{2}$ connections, and therefore the route server might need special routing policy for each interconnected pair. The methods for configuring the route server vary from using web-based tools, complex BGP communities, to RPSL-based configuration with extensive scriping.

The latest development in RPSL that intends to make use of RPSL for configuring either route servers in IXPs or for border routers of transit ISPs is the draft [60]. The draft attempts to solve a more general issue of routing policy configuration in an adjacent AS in the case that the adjacent AS has to act on behalf of its customer. In this scenario the adjacent AS should apply the routing policy according to wishes of the customer. This clearly generates a need for a method that would express the routing policy in the unique, clear and machine-readable format.

This is particularly interesting for route servers in IXPs. Customers or route servers have lost their ability to filter outgoing routes and differentiate among peering partners by sending their complete announcement to the route server. The route server is expected to apply filtering rules according to needs of the client to send out the learned path vectors only to the proper peering partners. Nowadays this filtering is usually configured by complex BGP communities. The communities encode what to send and what to filter out in certain directions. Downside of that is that there is no standard for the encoding and therefore each IXP may come up with their own BGP community assignment and evaluation rules.

Moreover, there is a problem with BGP community length, which has originally been 32 bits. The BGP community signalization works only for 16 bits long ASN because RS operators and customers are able to use the high 16 bits for destination ASN and low 16 bits for encoding an action or any other additional information. The solution is specified in documents [81] and [82] that adds longer BGP communities. However, there are still (mid-2016) important Autonomous Systems in the Internet that operate software, which is not fully implementing this standard.

Routing anomalies prevention

Data from IRR can be possibly used to validate the BGP announcements at virtually any router. The problem is that the validation comes with inherent cost in terms of CPU time, power and manual work needed to generate consistent results.

The current situation, when more than 20% of validations fail with RIPE DB data, does not allow to use the validation result to automatically prevent BGP routes to be propagated in all directions. There are well known exceptions that may afford blocking prefixes that do not pass the IRR filters: The route servers in the Internet exchange points, namely the European neutral IXPs. The reason for doing that is the emphasis on security, the Internet stewardship and acceptance of these values by the local community and ultimately by the parties connected to these IXPs. Another reason is that IXP is usually in position of short-cut for Internet traffic. When a path to certain destination gets blocked in the IXP the data can still flow through upstream providers, though the path would be slower and it would generate extra transportation cost.

Routing anomalies detection

The current data can be used for detecting anomalies. This thesis is evaluating routing anomalies with respect to IRR data. Or on the contrary, it evaluates anomalies in IRR data with respect to the Internet routing system. This reasoning might lead to the question what data source is more authoritative and which source should we trust. Even though this question is complicated and we will not try to answer it, there is an important lead: The Internet transports data very successfully according to the Internet routing system despite the discrepancies between IRRs and DFZ.

The IRR data are being used as a supplemental data source for certain services to detect anomalies in the Internet with varying success. One example of an extensive utilization of IRR data to detect anomalies is RIPE NCC **Routing Information Service (RIS)**.

5.3.2 Increased workload

IRR side

It is difficult to estimate the cumulative workload that the IRR system generates now. In RIPE NCC service region there are two RIPE-sponsored working groups that take care of RIPE DB standard and operational procedures connected with the IRR. The RIPE Database software itself has been created and is maintained by the RIPE NCC staff.

In addition, there is the IRR mirroring service that requires considerable effort unevenly spread over all IRR operators.

Client side

From the client side, the most work in relation with RPSL databases is mastering of RPSL and related procedures in the first place. The subsequent maintenance of the records should not cause problems and add much work load. The problem is that when the AS do not use the RPSL-based configuration and when there is not any formal requirement for the AS that would mandate the RPSL usage, the obvious question arises: "Is it really necessary?"

The same question has been repeatedly asked by the involved individuals on numerous occasions: Nick Hilliard presented his opinion [62] on RPSL usefulness and expressed his wish to start a discussion about a possible reform of RPSL on RIPE 61 in 2010.

Almost the same concerns about usefulness of RPSL and IRR system have arisen a long time ago. In 2000, there was a discussion that took place in NANOG mailing list [69]. It started with a simple questions “Who should use Routing Registries? Why?” and “Is it worth the time?”.

The responses to these questions came directly from people involved in Routing Registry operation. They were correct, extensive and could serve as a source of education, though they also acknowledged that IRRs do not have enough data and from the practical point of view the accuracy is low. The replies were stated in such a manner that gives hope for the future when the data would be complete and accurate enough to control the Internet routing. Unfortunately, this hope has not been fulfilled so far.

Software support

The software is an integral part of the AS operational procedures and RPSL would bring little benefits without the compatible software. Unfortunately, the existing software is either old and difficult to use or incomplete and not well tested.

The oldest, most complete and most complex software is the **IRRToolset**. The main problem of this software is that it has been designed as a compact package. The design work began in 1990’s and it is written in C++. Nowadays the members of the networking community generally prefer using scripting languages like Python and need easy-to-use libraries more than compact packages. From my experience it seems much easier to pre-process text inputs and post-process text outputs for programs that belong to **IRRToolset** than attempting to modify or extend the parts of **IRRToolset** itself when additional functionality is needed. This programming pattern is obviously wrong. Calls for creating a new and modern RPSL library occasionally appears over time [62], [1] and few libraries have been created [52], [54]. However, none of these projects have overcome the prototyping stage so far.

Even if there were a modern and easy to use RPSL library, it would still be difficult to integrate it into many network management software packages. The IRR system has been designed as the primary and authoritative data source. The network management software usually made the same and thus conflicting assumption about their data models. This is even more complex problem because RPSL imposes unusual restrictions in certain directions[4] while allowing unrestricted nesting of references.

These problems can be solved with stable funding of the RPSL library development. The current situation however shows that the RPSL software development is based on immediate needs of certain autonomous system operators. And the custom-made advanced software, that some autonomous systems use, has not been made public.

5.3.3 Conclusion on the extended hypothesis

The two previous sections provided context for the observation related to the extended hypothesis. The detailed observations collected in the process of **bg-**

[4]The unusual restrictions are the requirement of ASCII characters in data types and contextual-based format definition in attributes.

pcrunch software creation and during several years of experience with operating various Autonomous Systems in different countries are going to be presented in the following section. Though the severity of the observed problems is a subject of individual perception, their existence is unquestionable.

We can only speculate on the reasons that caused low quality of IRR data: My personal opinion is that complexity of RPSL is one of the substantial causes, if not the most substantial one.

The following section contains a brief list of issues related to RPSL parsing. It shows inherent complexity of the language and the related tools. At the same time it brings forward many arguments supporting the extended hypothesis.

5.4 RPSL parsing issues

The above mentioned problems are rooted either in RPSL format specification or in operational procedures and old software engineering decisions. The issues that originate in the RPSL format are more interesting for this thesis. The extended hypothesis mentioned *work load* and I have described several issues that need either extensive manual work or considerable amount of CPU power with relation to RPSL.

Two fundamental reasons that imposes requirements of either human or machine work are:

1. Number of elements,
2. complexity of elements.

5.4.1 Number of elements

Absolute number of elements

Absolute number of relevant elements in the reference IRR for this thesis, which is RIPE DB, on June 21, 2015 is summarized in the table 5.1.

Object type	count
`as-set`	14,665
`aut-num`	28,935
`filter-set`	127
`inet-rtr`	104
`peering-set`	214
`route`	261,187
`route6`	12,208
`route-set`	1,394
Summary	318,834

Table 5.1: RIPE DB object count on June 21, 2015

The RIPE DB dump to this date was 4.7 gigabytes and consisted of 131,602,120 lines. We have to interpret these numbers is context of complex RPSL parsing and possible recursive references. Generally speaking it takes considerable time to read and pre-process IRR data of this scale at the current state of the art computers.

Recursive nesting

The RPSL objects that allow recursive nesting are `-set` objects, namely the following types:

- `as-set`,
- `filter-set`,
- `route-set`,
- `rtr-set`,
- `peering-set`.

The most frequently used object type within this group is `as-set` and this object type also happens to contain most of the recursion chains. Table 5.2 shows the characterization of recursion withing group of `as-set` objects.

Characteristic	value
Average recursion depth	5.232
Maximum recursion depth	29

Table 5.2: RIPE DB recursion depth in `aut-num` objects on June 21, 2015

The maximum recursion depth of 29 is low number from the perspective of machine processing, but it is high for manual resolution.

Another interesting characterization is the number of elements in recursive subtrees shown in the table 5.3. The numbers prove that collecting the information manually from recursive sets in IRR is nearly impossible even in an average case. This makes the job of AS operators much more difficult and it creates a room for errors.

Characteristic	value
Average subtree size	233.940
Maximum subtree size	7, 138

Table 5.3: RIPE DB recursion subtree size in `aut-num` objects on June 21, 2015

5.4.2 Complexity

The complexity issue connected with RPSL is divided into several aspects:

- Choosing the right IRR or combining data from several IRRs might get complicated.

- Obtaining data from the IRR is either a time consuming series of network transactions or it requires considerable resources for processing the raw text IRR contents.
- Basic text format is non-standard and therefore requires a specialized parser.
- Objects might contain context-sensitive data that need contextual parser.
- Multiplicity in selectors is possible in `aut-num` objects.
- Complex filtering expressions and recursive references might occur in `aut-num` objects.

Choosing the right IRR

Event though there are widely known IRRs that corresponds approximately to RIR service regions, the right IRR is not always an obvious choice. For the RIPE NCC service region, the preferred IRR is naturally RIPE DB, and luckily there is not any other major IRR in the service region. Moreover, RIPE DB is considered to be the most accurate, the most consolidated and the most extensively populated IRR among all other IRRs, that are serving other regions.

This relatively clear situation in RIPE NCC service region allowed us to choose the RIPE DB and European BGP data feeds as the reference for the data accuracy assessment.

It might be possible to combine data of several IRRs together and get global routing data. Unfortunately, there is not any standard way of doing that and there is a major unresolved problem: How to cope with conflicts in data of different IRRs. These conflicts are highly likely to occur with several distinct types: Semantic or naming conflicts, object duplication and divergence of duplicate objects.

Obtaining IRR data

The issues connected with obtaining data from RIPE DB have been studied in the previous sections.

Other IRRs might have different rules. Most of the public IRRs offer the *whois* service and some of them have deployed the *RESTful* API. However, not all other IRRs offer the raw database content in the form of text files, either because of disclosure concerns or without any given reason.

Parsing text format

The basic text format unfortunately does not conform to any modern and widely-accepted data markup standard. Even though the RPSL format looks simple, there are many different ways of writing the semantically same object. Moreover, there is no specified canonical format.

The language is case insensitive and only ASCII characters are allowed. White spaces might be freely used to format and indent the text with exception of the first character on the line. White space or “+” character on the first position on line marks a continuation of the previous attribute.

Many attributes might be either multi-valued or repeated several times and both possibilities might be combined. For instance, the following object contains multiple multi-valued attributes:

```
as-set:         AS-WISPERICM
members:        AS6889, AS6765, AS5600
members:        AS6889, AS6765, AS5611
```

This object is semantically equivalent to the following examples:

```
as-set:         AS-WISPERICM
members:        AS6889, AS6765, AS5600, AS6889, AS6765, AS5611

as-set:         AS-WISPERICM
members:        AS6889, AS6765, AS5600
                AS6889, AS6765, AS5611

as-set:         AS-WISPERICM
members:        AS6889, AS6765, AS5600
+               AS6889, AS6765, AS5611
```

Keywords

There are reserved words and reserved identifiers that must not be used as identifiers:

1. Reserved identifiers `ANY` and `PeerAS` for use in filters.
2. Reserved words `AND`, `OR`, `NOT`, `REFINE` and `EXCEPT` for filter expression.
3. Names starting with `as-` are reserved for `as-set` object names.
4. Names starting with `rs-` are reserved for `route-set` object names.
5. Names starting with `rtrs-` are reserved for `rtr-set` object names.
6. Names starting with `fltr-` are reserved for `filter-set` object names.
7. Names starting with `prng-` are reserved for `peering-set` object names.

Unfortunately, these keywords are reserved only in the context of the identifier that has the same type. Thus a keyword might be used as a part of other identifiers. The example is the referenced identifier "`AS6774:AS-PEERS:PeerAS`" in RIPE DB (captured on June 21, 2015) that contains reserved word `PeerAS`.

Selector multiplicity

Selectors on `import` or `export` lines in `aut-num` object might contain compound expressions that could require recursion to resolve the flat list of actual selectors.

In addition, the same semantic meaning might be achieved by multiple `import` attributes with different selectors. For instance, the following rules

```
import:      from (AS42 or AS3856) accept AS-PCH
```

are equivalent to:

```
import:      from AS42 accept AS-PCH
import:      from AS3856 accept AS-PCH
```

and the same can be achieved with an `as-set` object:

```
as-set:      AS-PCH-SELECT
members:     AS42, AS3856

aut-num:     AS29134
import:      from AS-PCH-SELECT accept AS-PCH
```

The example shows that we can use algebraic expression with parentheses and `OR` operator, or we can use equivalently repetition of the `import` line. We can also use expansion of the selector from the `as-set` object that use comma as the membership multiplicity operator.

Moreover, all the possible combinations of operators, parentheses and filters are allowed.

Filter expression variants

Filters in the `aut-num` objects have to be resolved before the filters might be evaluated or translated as per the application requirements.

The resolved filter consists of an algebraic expression or a list of algebraic expressions connected by set operators. The expressions have to be decomposed and evaluated or translated and the results have to be processed through the set operators.

The basic filters are the four described types of matches:

1. ASN that represents list of possible originated prefixes,
2. IP prefix list,
3. regular expression for matching `AS_path` BGP attribute,
4. BGP community match.

The ASNs that represent prefix origins might either explicitly occur in the filter or it might be a part of an `as-set` object that can be recursively referenced from the filter.

The IP prefix list and regular expression might be either explicit or might occur in a recursively referenced `filter-set` object.

The `filter-set` object might also contain explicit ASN representing originated prefixes and another filter fragments, including operators and parentheses.

The following three filters are equivalent:

```
import:         from AS29134 accept AS29134
```

With respect to 3 existing `route` objects:

```
route:          217.31.48.0/20
origin:         AS29134

route:          62.109.128.0/19
origin:         AS29134

route:          188.227.128.0/19
origin:         AS29134
```

is equivalent to:

```
import:         from AS29134 accept {217.31.48.0/20,
                62.109.128.0/19, 188.227.128.0/19}
```

or to:

```
import:         from AS29134 accept ({217.31.48.0/20} or
                {62.109.128.0/19} or {188.227.128.0/19})
```

Parentheses in filter expressions

The parentheses might be used to enclose algebraic expressions and delimit community list. These occurrences require contextual parser to distinguish between them. An example of both usage cases in the same rule follows:

```
import:         from AS20965 accept (community.contains(20965:155,
                20965:21320) and not fltr-bogons)
```

AFI specification in RPSLng

Address Family Identifier (AFI) specification in RPSLng (according to RFC 4012) is clearly a retro-fitted feature to the existing RPSL (RFC 2622) and these two types of RPSL expressions blend together. Fortunately, the RPSLng lines are easy to distinguish because they use "`mp-`" prefix in attribute names. However the RPSLng adds yet another way of expressing IPv4 unicast policy that was previously the domain of RFC 2622 based RPSL.

With RPSLng we can use either the `import` and `export` attributes or the newly added `mp-import` and `mp-export` attributes.

The two following filters are equivalent:

```
export: to AS6939 announce AS-IGNUM-OUT
import: from AS6939 accept ANY

mp-export: afi ipv4.unicast to AS6939 announce AS-IGNUM-OUT
mp-import: afi ipv4.unicast from AS6939 accept ANY
```

Set operations

The following example is more complex and combines several techniques. It came from RIPE DB `aut-num` object **AS20535** (captured on June 21, 2015):

```
import: { from AS-ANY accept NOT { 0.0.0.0/0 }
        AND NOT { 0.0.0.0/0^25-32 };
        } refine {
        from AS-ANY action pref=40; accept community(20535:60);
        from AS-ANY action pref=30; accept community(20535:70);
        from AS-ANY action pref=0; accept ANY;
        } refine {
        from AS13099 accept AS-AET and <AS-AET$>;
        from AS28910 accept AS-INTAL AND <AS-INTAL$>;
        from AS34639 accept AS-TOTEL and <AS-TOTEL$>;
        from AS39214 accept as-comintech and <as-comintech$>;
        from AS-INSAT accept PeerAS and <PeerAS$>;
        }
```

The filter consists of three parts that are connected by the `refine` set operator. In this case the refine operators generate cartesian product of the three groups of rules. The first group limits both default route and too much specific prefixes with net mask range 25 to 32. The second group sets `LOCAL_PREF` BGP attribute based on the incoming communities. The third group filters incoming routes from each direction according to the selector and the adjacent filter.

The last line in the last sub-filter uses special keyword `PeerAS` both as a AS origin filter and as a part of regular expression. The actual value of this keyword has to be resolved from the selector, which is a group defined by the `as-set` object in this case.

Uncommon filter elements

The previous example contains two additional points of interest:

1. Curly brackets that serve as the filter delimiters, even though the same type of brackets is used for delimiting explicit IP filters.
2. The <> block contains the regular expressions that needs to be resolved to become a valid regular expression or a set of regular expressions with **OR** operator connecting them.

Another unusual feature of filters in the example is application of prefix length modifiers on abstract filters that have to be resolved beforehand. A representative example is:

```
import:     from AS29134 accept AS-IGNUM-OUT^+
```

This filter has to be resolved to members of the `as-set` object **AS-IGNUM-OUT**. After that these origin AS filters have to be resolved in order to get the list of prefixes containing {`217.31.48.0/20, 62.109.128.0/19, 188.227.128.0/19, 195.226.217.0/24`} and then this prefix list has to be modified to the resulting equivalent filter:

```
import:     from AS29134 accept { 217.31.48.0/20^+,
            62.109.128.0/19^+, 188.227.128.0/19^+,
            195.226.217.0/24^+ }
```

Readability issues and grouping

The following example taken from the RIPE DB (captured on June 21, 2015) shows that styling of filter severely affects readability. Moreover, the filter in the next example uses semicolons to put many logically distinct filter expressions that would normally belong to different `import` lines to one expression enclosed in curly brackets. The reason for connecting more expressions to a single filter is the use of `refine` operator on it.

```
import: {  from AS8395 accept AS-EAST; from AS8592
 accept AS8592; from AS8752 accept AS8752; from
 AS15672 accept AS15672; from AS16231 accept
 AS16231 OR AS28736; from AS21085 accept AS21085;
 from AS21225 accept AS-AMTKOM; from AS25032
 accept AS25032; from AS25251 accept AS-ARTCON;
 from AS25308 accept AS25308; from AS29124 accept
 AS29124; from AS29182 accept AS-ISPSYSTEM; from
 AS31494 accept AS-INFOSETI; from AS31720 accept
 AS31720; from AS33842 accept AS33842; from
 AS33902 accept AS33902; from AS34121 accept
 AS34121; from AS34211 accept AS34211; from
 AS34352 accept AS34352; from AS34682 accept
 AS34682; from AS34687 accept AS34687; from
 AS34690 accept AS34690; from AS35178 accept
 AS-TELART; from AS35374 accept AS35374; from
 AS35750 accept AS35750; from AS35755 accept
 AS35755; from AS38922 accept AS-Wiland-TP;
 from AS38964 accept AS-ADTEL; from AS39034
 accept AS39034; from AS39150 accept AS39150;
 from AS39165 accept AS39165; from AS41667
 accept AS41667; from AS41917 accept AS41917;
 from AS41947 accept AS41947; from AS42533 accept
 AS42533; from AS42569 accept AS42569; from
 AS43327 accept AS-REDLINE-NEW; from AS24758
 accept AS24758; from AS48552 accept AS48552;
 from AS43666 accept AS-CTS; from AS47711 accept
 AS47711; from AS48050 accept AS48050; from AS48946
 accept AS48946; from AS43816 accept AS43816; from
 AS16300 accept AS16300; from AS43993 accept AS43993;
 from AS43414 accept AS43414; from AS39596 accept
 AS39596; from AS47839 accept AS47839; from AS34249
 accept AS34249; from AS43221 accept AS43221; from
 AS33902 accept AS33902; from AS34123 accept AS34123;
 from AS5531 accept AS-TEZTOUR; from AS48535 accept
 AS48535; from AS49060 accept AS-UNIONLINE; from
 AS49400 accept AS49400; from AS49371 accept AS49371;
 from AS49779 accept AS49779; from AS50212 accept
 AS50212; from AS50265 accept AS-GT; from AS51410
 accept AS51410; from AS51464 accept AS51464; from
 AS51814 accept AS-KZNET; from AS48147 accept
 AS48147; from AS39272 accept AS39272; from AS42293
 accept AS42293; from AS52112 accept AS52112;} refine
 { from AS-ANY action pref=700; accept ANY; }
```

5.4.3 Expressive power of RPSL

Despite the unquestionable complexity of the RPSL language, there are still gaps in its expressive power.

AS operators need to pass certain routing parameters to the peering partners and potentially publish some of them to the broader audience. Some of these parameters have to be passed in secret. The selection of the parameters that could be possibly passed in a machine-readable format, but RPSL does not facilitate that, follows:

- MD5 password[5] for the BGP session.
- Intent to use TTL security[6] for BGP.
- Maximum prefix count that will be accepted from the particular BGP session.
- Intent to use BGP flap dampening mechanism for the particular BGP session.
- Line MTU.
- Intent to use Jumbograms[7].
- Intent to use Reverse Path Filtering[8] and the details of the RPF settings.

Moreover, there are much more information that AS operators have to agree on before the peering can be set up: MPLS parameters, interconnection of VLANs, MPLS-TE settings and QoS settings and many other parameters that do not have any standard notation and sometimes even the terminology varies according to the vendor preference.

[5]Passwords for RFC 2385 based BGP session protection.
[6]The Generalized TTL Security Mechanism according to RFC 5082.
[7]Packet exceeding the standard maximum transmission unit (MTU).
[8]BCP38.

6. Current IRR system

6.1 Current IRR system accuracy

6.1.1 Measurement method

The implementation of **bgpcrunch** that required creation of the new RPSL parser and interpreter and the subsequent large-scale IRR data analysis with this software have brought two major findings:

- The accuracy of the data is low enough not to reject the previously discussed hypotheses.
- Creating the software for interacting with RPSL is considerably difficult.

The conducted data analysis has been limited to RIPE DB data, and it therefore addresses resources utilized in the RIPE NCC service region. Extending the analysis to another IRRs is a possible step for future work, but not many people have shown interest in that measurements so far.

The findings about the data quality are based on RIPE DB data, however the reasoning about the RPSL standard, the IRR system complexity and implementation issues concern all IRR operators and databases.

6.1.2 Results

Summary

The accuracy measurement described in chapter 4 and in greater detail in appendix A shows that:

- Over $\frac{1}{5}$ of European path vectors in DFZ do not pass *origin* validation.
- At least $\frac{1}{4}$, but likely more than a half of European *hops* in `AS_path` do not pass filter validation.
- The `AS_path` validation results exhibit great variance.
- These observations have come from reputable network based in Prague that generally care about technical correctness.

Globality

The *origin* validation results have global outreach, which means that almost identical[1] results would be computed form any BGP table dump in any AS in the world.

On the contrary, the *hop* and path validation depends on the upstream network and peers to great extent. Different results have to be expected with different BGP table dump. However, the fact that a large number of routing policy discrepancies along paths exist is unquestionable.

[1]Certain differences are expected because different autonomous system can have different DFZ views.

6.1.3 Accuracy hypotheses

The hypothesis concerning accuracy and extent of RPSL routing policies in RIPE NCC service region, which has been stated in the beginning of the thesis and analyzed in chapter 4, has not yet been concluded.

The evidence supporting the hypothesis presented in chapter 4 and in the appendix A is convincing enough when we consider the raw numbers. However, the threshold for the hypothesis has been set to 20% on the basis of analogies with different fields and simple reasoning about perception of usefulness.

Operational experience from the Internet suggests that 20% of potentially unreachable remote networks is completely unacceptable for virtually any Internet user and transitionally to any network operator. We can safely assume that the practical threshold is much lower, perhaps under 10% or maybe even under 5%. Finding enough evidence for our 20% threshold proves that the original *intuitively* stated hypothesis "*The utilization of RPSL in the current Internet is sub-optimal both in scale and accuracy of the information.*" holds.

6.2 RPSL processing

The previous chapter shows that complexity of the IRR data in RIPE DB exceeds any expectation. To read IRR data, we need to have a complete RPSL parser that is capable of interpreting the complex filters and resolving large recursive trees.

The RPSL specification is permissive and open to interpretation in many cases. Luckily, we can resort to **IRRToolSet** that serves as the reference implementation and which has RPSL syntax checking tool.

Another problem is understanding the semantics of certain RPSL constructions[2] that are poorly documented and their meaning has to be established by reverse-engineering the reference implementation.

6.2.1 Complexity of RPSL

Complexity of filter selectors

The most severe problem in RPSL data parsing is the impossibility of accelerating neither positive nor negative lookups in `aut-num` objects. It is caused by the selector expressions in `import` and `export` attributes. The selector expressions make it impossible to stop the resolution unless the positive match is found or the end of the object is reached.

This feature is troublesome in virtually any case, but namely in the following ones:

- For repeated matching the filters (which is the case of **bgpcrunch**) it imposes a requirement of processing and resolving the entire object to obtain the content of the object in a flat form. Only after that the filters can be evaluated. It might be possible to pre-process the `aut-num` objects and store the resulting flat data structure for future use. The obvious problem

[2]For instance filter repetitions with the overlapping selectors, loops in the `-set` objects and many other issues

is the potential size of such data-structure: The recursive `-set` objects and the repetition in the individual filters could increase the size of the flattened objects by two or three orders of magnitude. And it can change literally over night. In addition to that, the cache management would increase the overall software complexity and spoil *lazy* evaluation approach. The cache size is still only a potential problem, but it is serious one because a relatively small change in IRR data could render any software, using this technique, unusable.

- For routing policy translation, the entire `aut-num` object has to be translated anyway. The potential problem lies in connecting the two or more filters on different lines that have some overlapping selectors.

- Moreover, it is difficult to orientate in objects that use expressions in filter selectors. These objects are prone for errors during manual data entry and subsequent manual maintenance.

Complexity of filter expressions

Another problem that concerns a lot of objects in RIPE DB lies in complex filters that use multiple `refine` and `except` operators. It is extremely difficult to understand the meaning of these filters. Another problem for both human maintainers and software parsers is the combination of the selector expressions and the set operators.

This kind of constructions are mainly utilized to inherit certain parameters from the super-filter to the sub-filters that match parts of the selector expressions.

These constructions are supposed to decrease the filter size by limiting repetition of the statements. It would be better not to use overly complicated statements in IRR at the cost of greater repetition in many cases. The reason is that parsing and understanding the complex statements is too expensive in terms of CPU power or human effort, while data storage is relatively cheap and it would be easy to generate the repeated statements by simple scripts.

6.2.2 Missing orthogonality of the language

One of the most confusing feature in RPSL is the fact that there are often many different ways of expressing one thing. The examples are the above mentioned complex selectors that can be replaced by multiple `import` or `export` lines.

The filters in general offer more ways of expressing the same idea. The variability reach beyond ordinary algebra variations.

Moreover, the combinations of multiple different techniques and expression patterns in a routing policy of a single AS, or even in a single object, is often confusing for human interpretation. It further complicates maintenance of the routing policy in RPSL.

6.3 Reasons for low data quality in IRR

6.3.1 Technical freedom in the Internet

Reasons for specifying a routing policy

We have described technical and organizational roots of the Internet in the beginning of this thesis: The most important values that prevail in both aspects are decentralization of decision-making and freedom of act. Both principles are used in resource allocation policies, operation of BGP and use of IRRs.

At the moment, I am not aware of any document that would mandate using IRR system globally or set a requirement to specify a routing policy for the AS.

However, many transit providers require their customers to create `route` objects and filter customers' BGP announcements based on the registry contents.

Route servers in the internet exchange points often use `route` objects to filter the passing announcements. The RS operators occasionally use also certain information from proper `aut-num` objects for filtering and for automating configuration.

Consequences of not using IRRs

The consequences of avoiding IRR system are not emphasized enough by common networking courses and certifications. We can assume that not all members of the networking community are fully aware of the IRR system use cases, issues and dangers of ignoring it. Many transit providers do not even require the `route` object registration. They do not check the customer's routing policy and would not remind the customers that without these objects the announcements are likely to get filtered out in IXPs, which might cause suboptimal routing.

The rules that the transit providers and route servers apply, regarding the IRR data related to their customers and passing prefixes, are fragmented. Moreover, there is no unique and machine-readable way for describing the rules.

As a consequence, it is not immediately obvious to many people why they should register their announcements and specify the routing policy. The known consequences of not doing that are:

- Announcement without proper registration in IRR might get filtered out by the immediate transit provider.
- Announcement might get filtered out by another upstream provider along its path. The result might be a partial *invisibility*[3] of the prefix in question or suboptimal routing.
- Announcement might get filtered out by route servers in important internet exchange points. The immediate consequence is usually partial prefix invisibility or suboptimal routing.
- Announcements might get filtered out by any other party along its path. The consequences are the same as in the previous cases.

[3]Prefix invisibility means that the prefix in DFZ is not propagated to certain portion of the Internet.

In addition, the consequences might change over time, depending either on technical or policy changes in the neighboring networks.

A partial invisibility of the announcements or suboptimal routing is notoriously difficult to detect. Network operators that do not actively and repeatedly assess their prefix visibility usually learn about routing issues too late. In most cases, these operators find out that only after they hear from their customers who can not reach the network in question or suffer from degraded performance.

6.3.2 High demands on AS operators

Language complexity

Previous chapters described the RPSL standard and its relation to the described networks, technologies and involved parties. Chapter 4 and appendix A show that the quality of the IRR data in RIPE DB is low enough to raise concern about IRR data practical usefulness.

Finally, chapter 5 elaborated on the RPSL parsing techniques and provided arguments in favor of the hypothesis that the RPSL complexity might be into blame for the low data quality.

Learning resources

The cited RFC documents that serve as the RPSL standard contain a limited set of examples. Finding additional RPSL tutorials and more complex examples with proper explanations proves to be difficult.

Currently the best resources for learning RPSL are perhaps the training courses provided by RIPE NCC[4]. Two of the current (2016) courses cover relevant topics:

1. *BGP Operations and Security Training Course*,
2. *RIPE Database Training Course Outline*

In the past there was a specialized course focused directly on RPSL: *Routing Registry Training Course*. Unfortunately this course has been replaced by the *BGP Operations and Security Training Course* and the part covering RPSL has been reduced.

Another RIRs that operate IRR for their region usually offer comparable courses related to their specific IRR.

Debugging of RPSL

Testing and *debugging* routing policies in RPSL is an undocumented task. It seems that that many people use RPSL in *write-only* manner. The opinions presented in [62] and the overall error rate we measured support this assumption.

Basic debugging of the routing policy is possible with `prtraceroute` tool from **IRRToolSet**, but it requires access to as many remote autonomous systems as possible. Unfortunately, this particular tool has been removed without replacement in the latest development version of **IRRToolSet**.

[4] `https://www.ripe.net/support/training/courses`

Another remaining possibility is to use `RtConfig` from **IRRToolSet** to generate filters and manually analyze results.

The **bgpcrunch** software could be easily transformed into BGP debugging tool: It is possible to create a self-contained tool for remote routing policy checking. The functionality would be similar to BGP *looking-glass*[5], with the difference that it would acquire and display not only the BGP table contents, but also routing policy validation report.

This idea seems to be worth including into future-work short-term plans.

IRR record maintenance

Maintenance of IRR records is not much different from the first registration of the resources. The problem lies in the complexity of parsing and reading the existing policies. There are three methods of maintaining the IRR records:

1. Creating automation for re-generating all the objects related to the AS in question.
2. Keeping local documentation in a format different from RPSL and creating the objects manually from the documentation when something changes.
3. Updating the relevant objects directly in the database.

Obviously the methods are sorted from the most difficult one to set up to the easiest one and in the reverse order from the easiest to use to the most difficult. The difficulty of the last method is derived from the fact that often a lot of complex RPSL objects have to be read and understood before updating them.

Another problem with the latest method is that inaccuracies and errors in the objects make it more difficult to perform maintenance tasks.

6.3.3 RIPE DB update mechanisms

LIRs operating within RIPE NCC service region are obliged to enter and maintain resource registration data in the RIPE DB. The obligation comes from the service contract between RIPE NCC and the LIR. IRR data share the same database and it is recommended to register routing policy at the same time with the IRR data.

Using RIPE DB update mechanisms is not always an easy task. The reasons for that are:

- Authentication model is based on `mntner` objects. The objects might reference three different authentication mechanisms. The mechanisms are bound to specific communication channels.
- Different channels with slightly different behavior can be used.
- Only the *Webupdates* method provides help with creating the objects and execute certain semantic checks before submitting the object to the RIPE DB.

[5]Looking-glass is a web site, or in this specific meaning a web application, that provides access to certain basic BGP `show` commands on a specific BGP router. Network administrators use remote looking-glasses for debugging BGP routing issues outside of their network.

Authentication

The supported authentication mechanisms and current (June 2016) compatibility with the update channels are summarized in the table 6.1.

	SSO	PGP	MD5
Web interface	Yes	No	Yes
Email	No	Yes	Yes
API	No	No	Yes

Table 6.1: RIPE DB Access authentication

MD5 is the weakest method and achieving even basic security with MD5 method is complicated: The `mnter` object has to be populated with the MD5 hash of the password. An update message has to contain a line with the clear-text password. In case of Syncupdates, the message can be encrypted by HTTPS protocol in transit. However, complex security measures have to be done to protect the software that contains the plain-text passwords and to protect the HTTPS session from protocol downgrade attacks and man-in-the-middle attack.

SSO (Single Sign-On) is a new method that is also based on password authentication, but it supports supplemental TOTP (Time-Based One-Time Password Algorithm) authentication method according to RFC 6238 [86]. It is easy to use with the RIPE DB web interface, but it is currently unavailable with the rest of the access methods.

PGP is the most secure method for signing the requests, but the supported channels are limited to e-mail, which is difficult to use with scripting.

Data formats

The diverse channels require different formats:

- The *older* channels, namely *whois*, *Syncupdates* and e-mail require raw RPSL objects in textual representation.
- The *Webupdates* requires user interaction with the web form where either individual attributes can be modified or the complete object can be passed to the RIPE DB software through bulk data entry form.
- RIPE DB API uses XML or JSON form of the objects.

The problems of the old text format have been described in the previous chapters and sections. Newer API that uses XML and JSON format is, in my opinion, a step in the right direction. The obvious benefits of the standard data markup format and the simple REST API promise easier integration with the client software.

Unfortunately, the issues related to complexity of parsing the filters remain in the returned objects from the REST API. The reason is that the pre-parsing is only on the object level.

The following example shows the `import` attribute from the `aut-num` object **AS20535**. The same object has been used in the previous chapters in various examples and this particular filter has been discussed in detail:

```
import:  {  from AS-ANY accept NOT { 0.0.0.0/0 }
            AND NOT { 0.0.0.0/0^25-32 };
         } refine {
            from AS-ANY action pref=40; accept community(20535:60);
            from AS-ANY action pref=30; accept community(20535:70);
            from AS-ANY action pref=0; accept ANY;
         } refine {
            from AS13099 accept AS-AET and <AS-AET$>;
            from AS28910 accept AS-INTAL AND <AS-INTAL$>;
            from AS34639 accept AS-TOTEL and <AS-TOTEL$>;
            from AS39214 accept as-comintech and <as-comintech$>;
            from AS-INSAT accept PeerAS and <PeerAS$>;
         }
```

This particular filter is transformed to the following equivalent XML format:

```
<attribute name="import" value="{ from AS-ANY
accept NOT { 0.0.0.0/0 } AND NOT { 0.0.0.0/0^25-32 }; }
refine { from AS-ANY action pref=40;
accept community(20535:60); from AS-ANY action pref=30;
accept community(20535:70); from AS-ANY action pref=0;
accept ANY; }
refine { from AS13099 accept AS-AET and <AS-AET$>;
from AS28910 accept AS-INTAL AND <AS-INTAL$>;
from AS34639 accept AS-TOTEL and <AS-TOTEL$>;
from AS39214 accept as-comintech and <as-comintech$>;
from AS-INSAT accept PeerAS and <PeerAS$>; }"/>
```

Thus the REST API helps only with the simplest portion of the RPSL parsing issues. However, the most important problem lies in the fact that this modern API is only a RIPE DB proprietary extension, and standardization effort that could unify the access to the IRR databases on the RPSL object level is missing.

7. IRR system reform

The current state of IRR system is deemed unsatisfactory. This section discusses the possible amendments and changes in the system that might, in my opinion, increase the chance that the IRR system would cover Internet routes more accurately, would be easier to maintain and would be more of use for the network operators.

Possible high-level steps that might help the IRR system include:

- Creating IRR automation software both for producing routing policies and for parsing them,
- writing documentation, creating larger examples,
- amending or reworking the standards.

The most interesting part of this list is the last point: Reform of the standards.

To fulfill one of the thesis objectives we have to provide recommendation for future development of standards related to routing policies. I decided to use this opportunity for creating a high-level specification of the new IRR system that might serve as a starting point for development of a new standard to supplement and subsequently replace the existing RPSL standards.

7.1 Reform of IRR standards

7.1.1 Requirements for the IRR system

The requirements for any new Internet standard have to be collected from the community members, elaborated in detail and meticulously considered, taking into account all known benefits and disadvantages.

The following list contains the requirements we know at this point. It can be used as an input for future requirement collection phase of the possible IRR system reforming effort.

There are two principal groups of requirements:

1. Feature requirements
2. Design requirements

Feature requirements

The required features include:

- The IRR system should allow describing existence of an autonomous system and link the ASN to the responsible AS operator.
- It should provide facility for publishing contacts and legal information about the AS operators in machine-readable from.
- The AS representation in IRR should be able to keep information relevant to the routing, QoS and peering with the AS in question.

- The AS representation should be able to keep arbitrary human-readable information.
- The IRR system should have means for describing peerings. Level of detail can be variable, depending on the operator's need.
- The peering representation in IRR should be able to keep all information needed to set up the peering.
- There should be data unit for grouping the peerings, that would allow parameter inheritance.
- The peering representation should be able to contain the filters and BGP attribute manipulation rules for the peering.
- The peering should be able to hold any supplemental information in human-readable form.

Design requirements

Many design requirements are obvious and adhere to common design patterns that almost any modern data model follows. The brief list of the design features include:

- The IRR system should be publicly available and potentially free to use.
- There must not be a central authority or designated database by design.
- The contacts and personal information in the IRR should be protected from possible data harvesting and abuse.
- The IRR system should support cryptographical signing of the information. Though the signature might be an optional field in the information units.
- The IRR system should allow delegation of name space partitions to different providers and cross-connecting name space partitions with *hyperlinks*.
- There should exist analyzable metasyntax and detailed specification of the syntax.
- There should exist a detailed data model for the IRR information units with extensive descriptions of semantics.
- The language, data model and the procedures related to the IRR should be simple.

7.1.2 Differences from RPSL

There are several immediate differences between the current RPSL-based IRR system and the new suggested IRR sytem:

1. RPSL does not have any tool for interconnecting IRR databases and linking data in foreign name space.

2. RPSL databases are inherently centralized. De-centralization of the system adds considerable complexity with database mirroring and finding authoritative data.

3. RPSL derives data security from the database access control mechanisms.

4. Peering information are limited to the pre-defined values and the data model is rigidly standardized which complicate any effort to amend it.

The differences depends on the selected implementation details. The most important difference might be the simplicity of the model and reuse of the existing technologies that would keep the overall standard complexity low and would simplify implementation both in software and in policies.

7.2 New IRR outline

This section describes an implementation outline for the suggested new IRR system. The most important design decisions are described from *high-level* standpoint. Selected subset of them are elaborated into more detail.

The new IRR system outline took certain ideas from NETCONF protocol. Moreover, we suggest to re-use NETCONF data modeling language YANG and incorporate parts of certain data models that are already specified for use with NETCONF.

7.2.1 Data

Language and syntax

The worst problems with the RPSL are related with parsing and understanding the language. The issue concerns both the syntax of the format as well as the semantics and data model. The syntax of RPSL is too permissive and complex at the same time. It proved to be unnecessarily complicated even to parse the RPSL objects and validate their syntactical correctness into detail.

The new format should avoid these problems. One possible way that would rule out any possibility of these problems is not defining the language at all and provide only a data model.

Another way of avoiding the parsing and syntax problems and providing a practically usable standard is suggesting one or more existing and standardized data markup languages. The best current candidates are obviously XML [87], JSON [88] and YAML [89]. Selecting the finite list of suggested existing markup formats might help with the initial implementation because the standard would limit the implementation complexity. If it proves too limiting later, we can easily lift the limits.

This would be an ideal compromise: We can re-use existing software tools for generating, parsing and validating the selected format. We can also refer to the existing format specification, which would reduce the size of the IRR system specification length and therefore the overall complexity. In addition, there are tools for transforming data among the above mentioned data markup formats, and it is likely that any prospective standard data format will be compatible with them.

The disadvantages might include greater overhead both in size and in processing power needed for generating and parsing the data units. Further analysis of this issue is needed to compare the size and processing power requirements with the existing RPSL standard and the current software tools.

Data model

The most important questions are how to define the data model which components have to be standardized, and how to prepare the data model for both future standard extensions and proprietary extensions.

One tool that might give an answer to the question "How to define the data model?" is the specialized language for data modeling: YANG [90]. It has been created for modeling NETCONF messages. The advantages of YANG are:

- It is a format-agnostic language: The data exchange format might be XML, JSON or any comparable markup language.
- It is general data modeling language that allows to express complex structures.
- It contains domain-specific features for networking and the Internet.
- It is supported by many software projects in several programming languages.
- It is a modern and maintained standard that has a great base of commercially successful users. NETCONF is spreading among the networking hardware vendors and NETCONF management tools are being actively developed and deployed in many important networks.

Further analysis of YANG features and IRR system requirements is needed to verify the assumption that YANG can cover the IRR system domain.

Regardless of the data modeling language or method we can specify the components that should be part of the new IRR standards. The brief list of features that need their data model component and semantic specification include:

- Data units: The new counterpart of RPSL *object*.
- Namespace and identification of the data units.
- Data unit referencing, namespace partitioning and delegation.
- Data unit grouping and searching in the groups.
- Data unit signing and signature verification.
- Basic unit types: We propose to have only the counterpart of RPSL `aut-num` object and a few supplementary data units to for grouping and interconnecting the objects.
- Filter expressions: The proposal calls for radically simplified filtering expressions. The RPSL set operators should be dropped and the filter structure should be *flattened*. However, either references in the filtering expressions or recursive templates in the filters will be likely required to limit the filter length and repetitions.

- Peering information: On the contrary the peering descriptions should contain much more information than the RPSL counterparts. We propose to add fields for MTU, QoS, TTL security, dampening and many other details of BGP sessions.
- Human readable information: The machine-readable parts should be supplemented by human readable information for each element in the *active* part of the data unit. Additional *inactive* comments should be also allowed.

The extensibility of the language can be achieved by adding special multi-valued attribute to each data unit that could contain an identifier of the standard or proprietary implementation that can parse the data unit. Optionally, the complex data unit that requires specialized software might contain also an identifier of less specialized data unit that is compatible with subset of the data in the unit. The highest possible level of detail would be used, depending on abilities of the software that attempts to read the data unit.

Data units

RPSL uses different types of objects and it proves difficult to remember them. The new IRR system should be simpler to use and the ultimate goal in this aspect would be to create an intuitive and clear syntax that could be used without constant searching in specification. The usage pattern should resemble writing a document in a markup language and using validator along the way rather than struggling with unknown "*programming/description*" language without strong types and without the compiler and validator.

The suggested data units in the new IRR include:

- Delegation of a group or a range of Autonomous System Numbers
- Autonomous System representation

The most important feature of this simple data model should be the possibility of self-hosting the data units on arbitrary HTTP server and linking remote resources and referring to the parts of either local or remotely referenced data units.

The peerings, *peer-groups* and filters should be expressed within the Autonomous System data unit. Links among different parts of the data units play crucial role in expressing the information in compact and efficient way. The purpose for keeping the information concerning a particular AS in a single document is to support consistency of the information and reduce repeated queries to the IRR system. The IRR system should allow the files to be static, so the current IRR *whois* databases that perform the searches and process data could be replaced by simple web servers serving static documents.

However, the IRR system is hierarchic and the data units at the top level might contain a lot of records that would be irrelevant for vast majority of queries. Due to this fact we should leave space for server-side response optimization. Inspiration for this might be taken from DNS recursive resolution procedure: The clients should be able to ask for specific part of the requested object and the server could reply with either the full object or the requested part, based on the server abilities.

Filters

The filters in the new IRR system have to be radically simplified. Unlike the RPSL filters we suggest to:

- Use *unique* filters: The filters for a particular peering can be either inherited from the *peer-group* or explicitly specified. The peering can be part of a single *peer-group*. The explicit filter can be specified only once for each direction in the *peer-group* and in the peering context. The peering filter has precedence over the *peer-group* filter. Effectively only one filter can be active for a single peering in a particular direction and it is simple to decide which filter is active and download its contents form the IRR.
- Use *flattened* filters: The filter expressions have to be explicitly stated and no further links are allowed with only two exceptions:
    1. References to *flat* lists of prefixes either in own or remote data units.
    2. References to *flat* lists of regular expressions for matching `AS_path` BGP attributes
- The default filters have to be consistent with default BGP operation.
- The filters should be treated as a sequence of rules rather than arbitrary algebraic expressions. This conforms to the common BGP configuration elements.

The outlined filters are simpler than the RPSL counterparts but the overall size of the objects might grow. Further analysis is needed to address questions regarding size and expressive power. Nonetheless, the possible size growth can be alleviated by recommending AS operators to self-host the objects.

7.2.2 Database and lookup system

Hierarchy

The basic principles of the new IRR system are technical decentralization on one side and consensus-based unification on the other side. There should be no obstacle for anybody to run a parallel IRR system for testing and development purposes or due to the desire to have an alternate system for any technical or political reason.

The Internet coordination bodies should run the *consensus-based official* IRR servers that would be consistent with and linked to the RIR registry data.

The figure 7.1 shows the hierarchy of the proposed new IRR system. The important difference in the system is represented by the delegations and data self-hosting. The figure also shows that the data hierarchy is derived from the Internet coordination hierarchy. In addition, the system is more transparent than the current RPSL hierarchy that has been displayed in the figure 1.1.

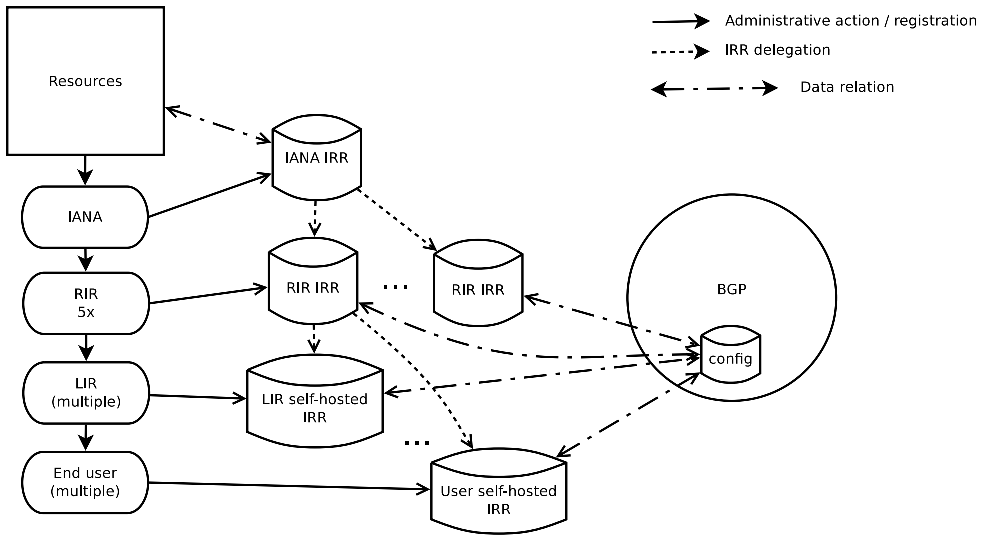

Figure 7.1: The proposed New IRR hierarchy

Query protocol

In the first stage, the suggested query protocols are HTTP or HTTPS. The possible REST interface for obtaining the relevant parts of the data units can be also specified in the standard or it can be left to discretion of the implementation authors.

We strongly suggest to re-use existing protocols, such as HTTP or HTTPS and existing standards and programming patterns like REST. This could potentially reduce size and complexity of the IRR system specification and make it considerably easier to implement the standard.

Update protocol

We suggest to specify an update protocol based on HTTPS, using HTTP authentication methods and using REST interface to create, update and delete objects in the public databases, especially in future IRR databases operated by RIRs.

Using HTTPS for sending the updates is easy and the required HTTP methods are implemented in most HTTPS libraries that are available in most languages. However, using HTTPS for updating files is unnecessary for self-hosted IRR site that may publish only a few IRR data units.

The update protocol should therefore be optional for the IRR servers and the simple static files without any explicit IRR update mechanism could be used on self-hosted IRR sites.

7.2.3 New IRR operation

The outlined IRR system offers not only the technical change. We also suggest changing operational principle and dependent standards. The modifications should affect following aspects of the Internet:

1. Resource registration process: We believe that the resource registration process should be more closely linked with the IRR.

2. The routing policies in certain autonomous systems, especially in autonomous systems that represent IXPs, should explicitly state that the AS validates routing policies for IRR and the failed prefixes will get filtered out.

3. The implementation should be easy for all parties. The standard should be clear and the data model should be formally defined in machine-readable way. The reference implementation should provide an Open Source, complete and easy to use IRR library for integration with arbitrary network management software.

4. The IRR system should provide framework for on-line validation of the IRR data. This system should help with debugging the IRR data and the BGP configuration as well.

5. The IRR system should allow autoconfiguration in both directions: It should be possible and reasonably simple to transform IRR data to a router configuration or, in the opposite direction, transform router configuration to a IRR data unit fragment.

Resource registration

The question of linking resource allocation to the resource registration and subsequent IRR descriptions and routing requirements is clearly a responsibility of the RIR policy developers. In case of RIPE it is the responsibility of the RIPE community and the designated working groups within RIPE.

We believe that certain portion of the operators would prefer to have simple written rules that would make IRR system obligatory. On the contrary, the Internet community is known for being sensitive to any new directives or rules. The new technology would have to prove that it is useful for the community in the first place. Once it is deployed and widely accepted, then the community might possibly consider making the IRR system mandatory.

With regard to the known community attitude we do not recommend linking the new IRR system to the resource registration from the beginning. We rather suggest preparing the data model for possible subsequent integration.

Validation

We believe that the validation and *debugging* tools are important for any software system. The new IRR system should have built-in validation and analysis tools from the outset.

The experience with **bgpcrunch** software and reactions from the community members show that the missing validation and analysis tools in the RPSL based IRR system probably caused a lot of current issues. Lack of IRR validation tools delayed deployment. Moreover, lack of global and recent analysis results also complicated discussion about development of the standards. It also prevented many parties from using autoconfiguration because the results were unpredictable. Finally, it leaves many AS operators in doubt when it comes to correctness of their RPSL routing policies.

Autoconfiguration

We suggest putting less emphasis on autoconfiguration in the new IRR system in comparison with RPSL. On the contrary, we believe in integration with the current and prospective network management tools and with NETCONF protocol.

The purpose of the IRR based software should range from creating small custom scripts that network operators often use to perform repetitive tasks up to the large network management and orchestration tools. However, the IRR system should mainly provide the data model and library to access it. The autoconfiguration functionality should be separated from the standards and from the library in order to keep generality of the standard and the software.

An important new feature is also the reverse autoconfiguration: The IRR system should have support for generating IRR data units from router configurations or arbitrary internal representation. This feature might help populating the IRR system with accurate data and make it easy to deploy the new IRR system.

8. Impact of the results and future work

8.1 Presentations of the results

I have presented the analysis results on three major events:

1. RIPE 71, DB-WG (November 2015, Bucharest)
2. NIX WG (November 2015, Prague)
3. CEE Peering Days (March 2016, Budapest).

Moreover I have presented the thesis results on the following networking events:

- CZ.NIC Labs seminar (January 2016, Prague).
- Peering workshop, MIXP & University of Montenegro (June 2016, Podgorica).

8.1.1 RIPE 71

Event details

A RIPE Meeting is a five-day event where Internet Service Providers, network operators and other interested parties from all over the world gather. The presentation topics cover range from research into new networking technologies, traffic analysis and routing techniques to resource assignment policing and IRR operation.

The thesis results have been presented on Database Working Group (DB-WG). The recording of the presentation and the slides can be downloaded from the conference web site for further reference: `https://ripe71.ripe.net/programme/meeting-plan/db-wg/`.

Ideas from Q&A

Questions in the subsequent Q&A session inspired several new ways of possible future work: Ruediger Volk from Deutsche Telecom asked for deeper analysis of `route` objects and noted that people often misinterpret net mask range of the `route` objects. He also asked for possible correlation of DFZ data, IRR data and S-BGP/RPKI validation results.

8.1.2 NIX WG

Event details

NIX WG (Neutral Internet Exchange Working Group) is a one-day event for network operators in the Czech Republic and in Central Europe.

Ideas from Q&A

Numerous questions in the Q&A session dealt with two major topics:

1. Requests for the fine-grained regional statistics and focus on the Czech Republic and Central Europe.
2. Questions regarding route server operation and the impact of the IRR data on peering relations conveyed by the route servers.

The apparent and deep concern about the route servers in relation to IRRs and routing policies is the reason why this thesis repeatedly discuss many details of route server operation.

8.1.3 CEE Peering Days

Event details

The Central and Eastern European Peering Days is a two-day technical event for network operators in Central and Eastern European countries. The topics are more technical and operation-oriented than the topics on RIPE meeting.

Ideas from Q&A

The most important question during the Q&A session came from Martin Levy, the Network Strategist at CloudFlare and an important figure of Internet history and protocol standardization. He called for creating an initiative for mending IRR data that would try to reach the operators, confront them with the error reports that concern their network and suggest possible steps for correction.

This idea goes far beyond my ambition, however it is worth noting that any future IRR system reform should allow this task to be carried out eventually.

8.2 Future work

8.2.1 Future presentations and education effort

Before any RPSL reform effort can be initiated, the need for the reform has to be established within the Internet community. To help spread awareness of the analysis I have conducted, its results and arguments supporting future RPSL reform I will continue presenting the results on relevant networking events and possibly publish the results.

8.2.2 Creating a RPSL looking-glass

The previous chapter described RPSL debugging issues and the lack of modern RPSL debugging software.

Consequently, the idea of turning **bgpcrunch** into BGP debugging tool has been outlined. The resulting RPSL *looking-glass* software might help with updating the RPSL data and finding currently unknown errors.

The most important requirements are:

- Self-contained package: The Python standard means of distribution can be used and the web interface could be provided by Flask framework.
- Easy installation: The only unresolved problem is connecting the BGP routers. The best option would be using an OpenSource BGP implementation directly on the same server. However, that would most likely increase configuration complexity. Another possibility is to use either NETCONF or *pexpect*-based console scripts to interact directly with the routers.
- Low memory and data foot-print: This requirement dictates that online IRR access have to be utilized for data acquisition. Nonetheless, this access method should fit the needs well in this case.

OpenSource software that fulfills these requirements could perhaps achieve wider deployment in many different autonomous systems. After that it could fill the current gap in RPSL tools.

8.2.3 NLNet Labs RDL effort

NLNet Labs has created a draft document [66] and there is an ongoing effort to finish and standardize the new language. A workshop on RDL is going to be held in the summer 2016. I am going to attend the workshop and present my notes on the existing syntax outline. The most important note I am going to present is a concern about low machine-readability of the new format.

8.2.4 Internet community role

Potentially involved parties

The research into RPSL and its deficiencies has been a process that I was able to do generally on my own. Discussions about the available tools, procedures and results on mailing lists and on relevant events brought a few novel ideas. It still provided only a limited help in comparison with the required community involvement in a possible RPSL reform.

Unfortunately, this topic diverges form expertise of most networking professionals and researchers. Therefore the community that appreciate the work in the early phase is fairly small. The academia seems to take Internet standards as granted and prefer research into completely new ideas and technologies, rather then amending the old ones.

Networking industry representatives, and networking equipment vendors especially, are busy with different challenges related to IPv6 transition, cloud services and SDN.

On the contrary, the IRR operation and related issues are the subject of RIPE Database Working Group charter. Therefore the Database Working Group seems to be the best place to discuss the future of IRR and possible RPSL reform at the preparation stage.

Specification of a RPSL successor

The ultimate ambition in this field is to specify a succeeding standard to RPSL and achieve better coverage, data accuracy and general usefulness of the standard.

The first step in this effort is to prove the need for the change and attract people to involvement in the effort. This thesis provided enough evidence and thorough analysis of the current state and brought many arguments in favor of the change.

The subsequent steps should be:

1. Discussing the existing standard and the existing proposals within Internet community,
2. collecting requirements for the new IRR system and reviewing them in cooperation with the Internet community,
3. devising arguments for the change and proving that the existing standards or the existing proposals does not meet the requirements,
4. creating a draft of the new standard and discussing the standard on the Internet community events,
5. creating a reference software according to the proposed standard,
6. then proceeding to the standardization phase: Bringing the proposed standard to IETF and to proper RIR working groups.

Turning ideas into Internet standard

IETF is most likely the proper place for the possible standardization effort.

The Standard Development Process (SDP) in IETF is described in the RFC 2026 (BCP 9) [70].

Creation of a new Internet Standard comprises of many stages, and the process is time consuming and exceptionally difficult. Estimates of the expected complexity and length of the process can be given only in terms of lower bound. The educated guess of the minimum required effort varies between one and two man-years.

Fortunately, the Internet standards are developed with the help of the community. Involvement of the experienced and capable community members helps to alleviate the amount of work, that would be otherwise prohibitive.

Conclusion

The objectives, set in the thesis abstract, have been fulfilled in the following aspects:

- I have thoroughly studied the RPSL standard and many other resources that are related to the topic of this thesis.
- I have examined the data in RIPE DB and studied the policies in its IRR part. On that basis, I created a software that analyzed RIPE DB and BGP data and computed many quantitative parameters that concern Internet routing, IRRs and their mutual relations.
- The analysis software has been published under OSS license and parts of it have already been re-used by renowned Internet researchers.
- The results obtained from the analysis provide evidence supporting both previously stated hypotheses:
    1. The first hypothesis "The utilization of RPSL in the current Internet is sub-optimal both in scale and accuracy of the information. The situation is not improving and there is no perspective of change in this trend." has been discussed and concluded in favor of the hypothesis because of the poor match between IRR and BGP. The matching errors affect both `origin` validation and `AS_path` hop matches for IPv4 and IPv6 as well.
    2. The second hypothesis "High added workload compared to relatively low benefit obtained from extensive utilization of RPSL prevents the AS operators from wider deployment." has been elaborated on and many supporting arguments for the hypothesis have been collected, though the hypothesis is too broad to be decided on exact scientific bases.
- I have striven for describing relations of the data analysis results with the known problems of the standard.
- The thesis gave me the opportunity to investigate a problem that stands aside from the main stream of networking research, even though the routing management is exceptionally important for the entire Internet operation. I used the opportunity to outline a completely new IRR system that avoids the described problems and known shortcomings of the current system. Specification and implementation of the new IRR system is one of the future work goals set in the last chapter.

Accurate and up-to-date routing policies represent a fundamental mechanism that divide well operated and maintained network from chaos and disorder. Now (in June 2016) the most threatening Internet issue - the IPv4 address space exhaustion has its solution and nobody seriously challenge the need for transition to IPv6 anymore. It is perhaps the right time to help consolidate the Internet and make it better, safer and more robust by mending the IRR system.

Bibliography

[1] Hlaváček, T., *Routing policies* Bachelor's thesis, 2011, Prague

[2] Hubbard, K., Kosters, M., Conrad, D., Karrenberg, D., Postel, J. *Internet Registry IP Allocation Guidelines*, RFC 2050, November 1996 `http://www.ietf.org/rfc/rfc2050.txt`

[3] Daigle, L. *WHOIS Protocol Specification*, RFC 3912, September 2004 `http://www.ietf.org/rfc/rfc3912.txt`

[4] *Number Resources* Website of Internet Assigned Numbers Authority `http://www.iana.org/numbers`

[5] Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D., Terpstra, M. *Routing Policy Specification Language (RPSL)*, RFC 2622, June 1999 `http://www.ietf.org/rfc/rfc2622.txt`

[6] Blunk, L., Damas, J., Parent, F., Robachevsky, A. *Routing Policy Specification Language next generation (RPSLng)*, RFC 4012, March 2005 `http://www.ietf.org/rfc/rfc4012.txt`

[7] *List of Routing Registries* by Merit Network, Inc. `http://www.irr.net/docs/list.html`

[8] Rekhter, Y., Ed., Li, T., Ed., Hares, S., Ed. *"A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, January 2006 `http://www.ietf.org/rfc/rfc4271.txt`

[9] Jouanigot, J., Bonito, A., Dupont, F., Fassbender, S., Hillbo, A. Hommes. F., Klein, L., Porten, W., Stikvoort, D., Terpstra, M., Volk, R. *Policy based routing within RIPE*, ripe-60, May 1992 `ftp://ftp.ripe.net/ripe/docs/ripe-060.txt`

[10] Bates, T., Jouanigot, J., Karrenberg, D., Lothberg, P., Terpstra, M. *Representation of IP Routing Policies in the RIPE Database*, ripe-81, February 1993 `ftp://ftp.ripe.net/ripe/docs/ripe-081.txt`

[11] Bates, T., Gerich, E., Joncheray, L., Jouanigot, J., Karrenberg, D. Terpstra, M., Yu, J. *Representation of IP Routing Policies in a Routing Registry*, ripe-181, October 1994 `ftp://ftp.ripe.net/ripe/docs/ripe-181.txt`

[12] Meyer, D., Schmitz, J., Orange, C., Prior, M., Alaettinoglu, C. *Using RPSL in Practice*, RFC 2650, August 1999 `http://www.ietf.org/rfc/rfc2650.txt`

[13] Huston, G. *Analyzing the Internet's BGP Routing Table*, July 2003 `http://impossible.rand.apnic.net/papers/ipj/2001-v4-n1-bgp/bgp.pdf`

[14] Huston, G., Armitage, G. *Projecting Future IPv4 Router Requirements from Trends in Dynamic BGP Behaviour* Proc. ATNAC, Australia, December 2006
`http://caia.swin.edu.au/pubs/ATNAC06/Huston_1m.pdf`

[15] Houston, G. *BGP Routing Table Analysis Reports*
`http://bgp.potaroo.net/`

[16] Tangmunarunkit, H., Govindan, R., Shenker, S., Estrin, D. *T*he impact of routing policy on Internet paths Proc. INFOCOM 2001, Pages 736-742 vol.2 ISBN 0-7803-7016-3

[17] Feldmann, A., Maennel, O., Mao, Z. M., Berger, A., Maggs, B. *Locating internet routing instabilities* Proc. SIGCOMM 2004, Pages 205-218 ISBN 1-58113-862-8

[18] Zheng, H., Lua, E. K., Pias, M., Griffin, T. G. *Internet Routing Policies and Round-Trip-Times* Proc. PAM 2005, Pages 236-250 ISBN 978-3-540-31966-5

[19] Labovitz, C., Ahuja, A. *The Impact of Internet Policy and Topology on Delayed Routing Convergence* Proc. INFOCOM 2001, Pages 537 - 546 vol.1 ISBN 0-7803-7016-3

[20] Wang, F., Gao, L. *On Inferring and Characterizing Internet Routing Policies* Proc. IMC 2003, SIGCOMM, Pages 15-26 ISBN 1-58113-773-7

[21] Feamster, N., Balakrishnan, H. *Detecting BGP configuration faults with static analysis* Proc. NSDI 2005, Volume 2, May 2005, Pages 43-56

[22] Kruegel, C., Mutz, D., Robertson, W., Valeur, F. *Topology-Based Detection of Anomalous BGP Messages*

[23] Govindan, R., Alaettinoglu, C., Eddy, G. Kessens, D., Kumar, S., Lee, W. *An architecture for stable, analyzable Internet routing* IEEE Network Magazine, Volume 13 Issue 1, Jan-Feb 1999, Pages 29-35 ISSN 0890-8044

[24] Griffin, T., Shepherd, B., Wilfong, G. *Policy Disputes in Path-Vector Protocols* Proc. International Conference on Network Protocols, Oct-Nov 2003, Pages 21-30 ISSN 1092-1648

[25] Feamster, N., Winick, J., Rexford, J. *A model of BGP routing for network engineering* Proc. ACM SIGMETRICS, Volume 32 Issue 1, June 2004, Pages 331-342 ISBN 1-58113-873-3

[26] Feamster, N., Jung, J., Balakrishnan, H. *An empirical study of "Bogon" route advertisements* Proc. ACM SIGCOMM Computer Communications Review, Volume 35 Issue 1, January 2005, Pages 63-70

[27] Caesar, M., Rexford, J. *BGP routing policies in ISP networks* IEEE Network Magazine, Volume 19 Issue 6, November 2005, Pages 5-11 ISSN 0890-8044

[28] , Griffin, T., Jaggard, A., Ramachandran, V. *Design Principles of Policy Languages for Path Vector Protocols* Proc. SIGCOMM 2003, Pages 61-72 ISBN 1-58113-735-4

[29] Stone, G., N., Lundy, B., Xie, G. G. *Network policy languages: a survey and a new approach* IEEE Network Magazine, Volume 15 Issue 1, Jan-Feb 2001, Pages 10-21 ISSN 0890-8044

[30] Lepinski, M. Kent, S. *An Infrastructure to Support Secure Internet Routing*, RFC 6480, February 2012
`http://www.ietf.org/rfc/rfc6480.txt`

[31] Bush, R. Austein, R. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*, RFC 6810, January 2013
`http://www.ietf.org/rfc/rfc6810.txt`

[32] Mohapatra, P., Scudder, J., Ward, D., Bush, R., Austein, R. *BGP Prefix Origin Validation*, RFC 6811, January 2013
`http://www.ietf.org/rfc/rfc6811.txt`

[33] *Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing* Goodell, G., Aiello, W., Griffin, T., Ioannidis, J., McDaniel, P., Rubin, A. Internet Society, NDSS Symposium 2003

[34] Kent, S., Lynn, C., Mikkelson, J., Seo, K. *Secure Border Gateway Protocol (S-BGP) — Real World Performance and Deployment Issues* Internet Society, NDSS Symposium 2000

[35] Bush, R., Austein, R., Patel, K., Gredler, H., Waehlisch, M. *Resource Public Key Infrastructure (RPKI) Router Implementation Report*, RFC 7128, February 2014
`http://www.ietf.org/rfc/rfc7128.txt`

[36] Wahlisch, M., Schmidt, R., Schmidt, T., C., Maennel, O., Uhlig, S., Tyson, G. *RiPKI: The Tragic Story of RPKI Deployment in the Web Ecosystem* `arXiv:1408.0391 [cs.NI]`, November 2015

[37] *YouTube Hijacking: A RIPE NCC RIS case study* RIPE NCC Publications, web site, March 2008
`https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study`

[38] Litke, P., Stewart, J. *BGP Hijacking for Cryptocurrency Profit* Dell SecureWorks CTU RESEARCH, web site, August 2014
`https://www.secureworks.com/research/bgp-hijacking-for-cryptocurrency-profit`

[39] Cowie, J. *The New Threat: Targeted Internet Traffic Misdirection* Dyn Research report, web site, November 2013
`http://research.dyn.com/2013/11/mitm-internet-hijacking/`

[40] Yun, J., Song, J. *Policy-Based AS Path Verification with Enhanced Comparison Algorithm to Prevent 1-Hop AS Path Hijacking in Real Time* International Journal of Multimedia and Ubiquitous Engineering Volume 11 No.1, January 2016, Pages 11-22 ISSN 1942-2636

[41] Sloman, M., Lupu, E. *Policy Specification for Programmable Networks* Active Networks, Springer Berlin Heidelberg, 1999, Pages 73-84 ISBN 978-3-540-48507-0

[42] Voellmy, A., Kim, H., Feamster, N. *Procera: a language for high-level reactive network control* Proc. HotSDN '12, Pages 43-48 ISBN: 978-1-4503-1477-0

[43] Schlamp, J., Wahlisch, M., Schmidt, T., Carle, G., Biersack, E. *CAIR: Using Formal Languages to Study Routing, Leaking, and Interception in BGP* `arXiv:1605.00618 [cs.NI]`, May 2016

[44] Foster, N., Freedman, M., Guha, A., Harrison, R., Katta, N., Monsanto, C., Reich, J., Reitblatt, M., Rexford, J., Schlesinger, C., Story, A., Walker, D. *Languages for software-defined networks* IEEE Communications Magazine, Volume 51 Issue 2, February 2013, Pages 128-134

[45] Reich, J., Monsanto, C., Foster, N., Rexford, J., Walker, D. *Modular SDN Programming with Pyretic* USENIX ;login Magazine, Volume 38 No. 5, Pages 128-134, 2013

[46] Doriguzzi-Corin, R., Salvadori, E., Aranda Gutierrez A., Stritzke, C., Leckey, A., Phemius, K., Rojas, E., Guerrero, C. *NetIDE: removing vendor lock-in in SDN* NetSoft 2015, London (UK) `http://www.netide.eu/sites/www.netide.eu/files/publications/netsoft2015_demo.pdf`

[47] Gutierrez, P., Karl, H., Rojas, E., Leckey, A. *On Network Application Representation and Controller Independence in SDN* Proc. EuCNC 2015, June-July 2015

[48] Jasinska, E., Hilliard, N., Raszuk, R., Bakker, N. *Internet Exchange Route Server* Internet-Draft – work in progress 05, IETF, June 2014 `https://tools.ietf.org/html/draft-ietf-idr-ix-bgp-route-server-10`

[49] *IRRtoolset* project Trac home `http://irrtoolset.isc.org/`

[50] Estrin, D., Postel, J., Rekhter, Y. *Routing Arbiter Architecture* `ftp://ftp.isi.edu/pub/hpcc-papers/ra/ra-arch.ps`

[51] *PERL RPSL::Parser* module in CPAN `http://search.cpan.org/~lmc/RPSL-Parser-0.04000/lib/RPSL/Parser.pm`

[52] *BGF project page - SourceForge*
`http://sourceforge.net/projects/bgflib/`

[53] *NOC project webpage*
`https://kb.nocproject.org/display/DOC/Home`

[54] Konstantaras, S. *ENGRIT (Extensible Next Generation Routing Information Toolkit)* Routing WG at RIPE 72, May 2016
`https://ripe72.ripe.net/presentations/143-RIPE72_ENGRIT_SK.pdf`

[55] *Repository of PolicyParser, mantaBGP project* GitHub repository
`https://github.com/stkonst/PolicyParser`

[56] Walker, D. and others *proposal for a review of the RIPE Database data model* Thread in RIPE DB-WG Mailing list
`https://www.ripe.net/ripe/mail/archives/db-wg/2016-May/005239.html`

[57] Schmitz, J., Gunduz, E., Kerr, S., Robachevsky, A., Damas, J. *Routing Registry Consistency Check*, ripe-201, 2001
`https://www.ripe.net/publications/docs/ripe-201`

[58] *RRCC* RIPE NCC Archived Projects, web site, December 2010
`https://www.ripe.net/analyse/archived-projects/rrcc/rrcc`

[59] Siganos, G., Faloutsos, M. *Analyzing BGP Policies: Methodology and Tool* Proc. INFOCOM 2004, Pages 1640 - 1651, vol.3 ISBN: 0-7803-8355-9

[60] Snijders, J., Hilliard, N. *The 'via' keyword in RPSL Policy Specifications* Internet-Draft, IEEE, Network Working Group, June 2013
`https://tools.ietf.org/html/draft-snijders-rpsl-via-00`

[61] Vohra, Q., Chen E. *BGP Support for Four-Octet Autonomous System (AS) Number Space*, RFC 6793, December 2012
`http://www.ietf.org/rfc/rfc6793.txt`

[62] Hilliard N. *Whither RPSL?* Routing WG at RIPE 61, November 2010
`http://ripe61.ripe.net/presentations/231-228-inex-ripe-rome-routingwg-whitherrpsl-2010-11-17.pdf`

[63] Farinacci, D. Meyer, D. Lewis, D. *The Locator/ID Separation Protocol (LISP)*, RFC 6830, January 2013
`http://www.ietf.org/rfc/rfc6830.txt`

[64] Jen, D., Meisel, M., Yan, H., Massey, D., Wang, L., Zhang, B., Zhang, L. *Towards A New Internet Routing Architecture: Arguments for Separating Edges from Transit Core* Proc. HotNets-VII, October 2008
`http://conferences.sigcomm.org/hotnets/2008/papers/18.pdf`

[65] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., van der Merwe, J. *The case for separating routing from routers* Proc. FDNA SIGCOMM, 2004, Pages 5-12

[66] Bilse, P. et al. *Routing Documentation Language* Internet Draft, April 2014
`https://tools.ietf.org/html/draft-bilse-rdl-00`

[67] Enns, R., Ed. *NETCONF Configuration Protocol*, RFC 4741, December 2006
`http://www.ietf.org/rfc/rfc6241.txt`

[68] *IRRToolSet Use @ Deutsche Telekom* ToolSet BOF at RIPE 64
`https://ripe64.ripe.net/presentations/210-ToolSet-BOF.pdf`

[69] *Route Registry: who uses them?* Thread in NANOG Mailing List
`http://seclists.org/nanog/2000/Oct/271`

[70] Bradner, S. *The Internet Standards Process – Revision 3*, BCP 9, RFC 2026, October 1996
`http://www.ietf.org/rfc/rfc2026.txt`

[71] Resnick, P. *On Consensus and Humming in the IETF*, RFC 7282, June 2014
`http://www.ietf.org/rfc/rfc7282.txt`

[72] Louridas, P., Spinellis, D., Vlachos, V. *Power laws in software* ACM Transactions on Software Engineering and Methodology, Volume 18 Issue 1, September 2008

[73] Newman, M. *Power laws, Pareto distributions and Zipf's law* `arXiv:cond-mat/0412004 [cond-mat.stat-mech]`, May 2006

[74] *polling for ripe dbase software changes* Thread in RIPE DB-WG Mailing list
`https://www.ripe.net/ripe/mail/archives/db-wg/1996-March/000496.html`

[75] *RIPE Database Query Reference Manual* RIPE Database Documents, RIPE NCC web site
`https://www.ripe.net/manage-ips-and-asns/db/support/documentation/ripe-database-query-reference-manual`

[76] Newton, A., Ellacott, B., Kong, N. *HTTP Usage in the Registration Data Access Protocol (RDAP)*, RFC 7480, March 2015
`http://www.ietf.org/rfc/rfc7480.txt`

[77] Hollenbeck, S., Kong, N. *Security Services for the Registration Data Access Protocol (RDAP)*, RFC 7481, March 2015
`http://www.ietf.org/rfc/rfc7481.txt`

[78] Newton, A., Hollenbeck, S., *Registration Data Access Protocol (RDAP) Query Format*, RFC 7482, March 2015
`http://www.ietf.org/rfc/rfc7482.txt`

[79] Newton, A., Hollenbeck, S. *JSON Responses for the Registration Data Access Protocol (RDAP)*, RFC 7483, March 2015
`http://www.ietf.org/rfc/rfc7483.txt`

[80] Blanchet, M. *Finding the Authoritative Registration Data (RDAP) Service*, RFC 7484, March 2015
`http://www.ietf.org/rfc/rfc7484.txt`

[81] Vohra, Q., Chen, E. *BGP Support for Four-octet AS Number Space* RFC 4893, May 2007
`http://www.ietf.org/rfc/rfc4893.txt`

[82] Rekhter, Y., Sangli, S., Tappan, D. *Four-octet AS Specific BGP Extended Community* IETF draft, May 2009
`https://tools.ietf.org/html/draft-ietf-l3vpn-as4octet-ext-community-02`

[83] *WHOIS REST API documentation* GitHub Wiki page
`https://github.com/RIPE-NCC/whois/wiki/WHOIS-REST-API`

[84] *IPv6 World Launch website*
`http://www.worldipv6launch.org/`

[85] *bgpcrunch* GitHub repository
`https://github.com/tmshlvck/bgpcrunch`

[86] M'Raihi, D., Machani, S., Pei, M., Rydell, J. *TOTP: Time-Based One-Time Password Algorithm*, RFC 6238, May 2011
`http://www.ietf.org/rfc/rfc6238.txt`

[87] Bray, T., Paoli, J., Sperberg-McQueen C., Maler, E., Yergeau, F. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* W3C Recommendation, November 2008
`https://www.w3.org/TR/2008/REC-xml-20081126/`

[88] Bray, T. *The JavaScript Object Notation (JSON) Data Interchange Format*, RFC 7159, March 2014
`http://www.ietf.org/rfc/rfc7159.txt`

[89] Ben-Kiki, O., Evans, C., Net, I. *YAML Ain't Markup Language (YAML^{TM}) Version 1.2* YAML specification, October 2009
`http://www.yaml.org/spec/1.2/spec.html`

[90] Bjorklund, M. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, RFC 6020, October 2010
`http://www.ietf.org/rfc/rfc6020.txt`

List of Tables

List of Figures

List of Abbreviations

AFI	Address Family Identifier
AFRINIC	African Network Information Center
AMS-IX	Amsterdam Internet Exchange
API	Application Programming Interface
APNIC	Asia Pacific Network Information Centre
ARIN	American Registry for Internet Numbers
AS	Autonomous System
ASBR	Autonomous System Border Router
ASCII	American Standard Code for Information Interchange
ASN	Autonomous System Number
BCP	Best Current Practices
BGF	Border Gateway Filter
BGP	Border Gateway Protocol
BoF	Birds of a Feather
CEE	Central and Eastern Europe
CPAN	Comprehensive Perl Archive Network
CPE	Customer-Premises Equipment
CPU	Central Processing Unit
CSV	Comma Separated Values
DB	Datbase
DB-WG	Database Working Group in RIPE
De-CIX	Deutsche Commercial Internet Exchange
DFZ	Default Free Zone
DNS	Domain Name System
EBNF	Extended Backus–Naur Form
EGP	Exterior Gateway Protocol
ENGRIT	Extendible Next Generation Routing Information Toolkit

FIB	Forwarding Information Base
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IOS	Internetwork Operating System, (Cisco proprietary OS)
IP	Internet Protocol
IRC	Internet Relay Chat
IRIS	Internet Registry Information Service
IRR	Internet Routing Registry
ISOC	Internet Society
ISP	Internet Service Provider
IT	Information Technology
IX	Internet Exchange
IXP	Internet Exchange Point
JSON	JavaScript Object Notation
LACNIC	Latin America and Caribbean Network Information Centre
LINX	London Internet Exchange
LIR	Local Internet Registry
LISP	Locator Identifier Separation Protocol
MED	Multi-Exit Discriminator
MIXP	Montenegro Internet Exchange Point
MPLS	Multiprotocol Label Switching
MPLS-TE	Multiprotocol Label Switching Traffic Engineering
MTU	Maximum Transmission Unit

NANOG	North American Network Operations Group
NAT-PT	Network Address Translation - Protocol Translation
NETCONF	Network Configuration
NetIDE	Network Interactive Development Environment
NFS	Network File System
NIX	Neutral Internet Exchange, *IXP based in Prague, Czech Republic*
NLRI	Network Layer Reachability Information
NOC	Network Operations Center
OSS	Open Source Software
PA	Provider Aggregatable *IP address space*
PERL	Practical Extraction and Report Language
PGP	Pretty Good Privacy
PI	Provider Independent *IP address space*
RDAP	Registration Data Access Protocol
RDL	Routing Documentation Language
REST	Representational State Transfer
RFC	Request For Comment
RIB	Routing Information Base
RIPE	Réseaux Internet Protocol Européens
RIPE NCC	Réseaux Internet Protocol Européens Network Coordination Centre
RIR	Regional Internet Registry
RIS	Routing Information Service, *RIPE NCC service*
RPC	Remote Procedure Call
RPF	Reverse Path Forwarding
RPKI	Resource Public Key Infrastructure
RPSL	Routing Policy Specification Language
RS	Route Server
RTBH	Remotely Triggered Black Hole
RTT	Round-Trip Time

S-BGP	Secure BGP
SDN	Software Defined Network
SDP	Standard Development Process
SIX	Slovak Internet eXchange
SSO	Single Sign-On, *RIPE NCC resources authentication method*
STD	Internet Standard
TCP	Transmission Control Protocol
TOTP	Time-Based One-Time Password Algorithm
TTL	Time To Live
VRF	Virtual Routing and Forwarding
WG	Working Group
XML	Extensible Markup Language
YAML	Yet Another Markup Language
YANG	Yet Another Next Generation, *data modeling language*

Attachments

A. RPSL and BGP data analysis

The appendix presents selected results of the data analysis, referenced in previous chapters. The results came from a software called **bgpcrunch**, which has been created during the preparation phase of this thesis. The functional description of the software in this chapter is going to be brief. The next appendix contains detailed API documentation and programmer's manual. The source code of the **bgpcrunch** software is published under OSS license on GitHub [85] (`https://github.com/tmshlvck/bgpcrunch`) for further reference.

Please note that this appendix contains only a small portion of the selected results. Complete results in both text and graphical form are published on the web site `http://aule.elfove.cz/~brill/bgpcrunch`. The reason for not including the full results in appendices or in electronic form is the size of the entire result set that reaches several hundreds of gigabytes.

A.1 Available data

A.1.1 Outline of the experiment

Extent and subject

The idea of conducting an Internet routing system analysis from two different points of view arose in fall 2011. The primary concern was to measure the accuracy of routing policy data in RIPE DB and observe IP routing trends that could be seen in turbulent time of IPv4 to IPv6 transition. In 2011, the IPv6 was generally deployed in backbone networks, but it was still generally disabled by content providers. People were concerned about turning IPv6 on and, at the same time, they were anxious about the future of the Internet because the IPv4 address pool exhaustion was imminent at that time. Despite the fact that IPv6 exists from late 1990's, the community did not have consistent and broad experience with end-to-end IPv6 services even after more than 15 years of small-scale dual-stacking. This situation was interesting not only from operational point of view, but also as an opportunity to capture the dynamics of IPv6 deployment and observe practices, concerning the new protocol in the Internet routing system.

The first part of the work that preceded this IRR measurement initiative was based on collecting data and storing them until the software for its analysis is finished. The plan was to seek for additional support later, namely obtaining more processing power to analyze the collected data, if it proves to be necessary.

Data collection period

The data collection started on November 11, 2011. There has been a brief discussion with the thesis supervisor, the representatives of my former employer who offered resources for the experiment, and a few people who showed interest in this topic about what data to collect and what to do with them afterwards.

With the known limits of available resources, I have created an outline for the data analysis, which is also the abstract for this thesis.

The collection process required subsequent adjustments: This is the reason why we have the first data that are usable for the subsequent analysis since March 3, 2012. Data collected in the period from November 11, 2011 to March 3, 2012 are incomplete and do not provide enough evidence for comprehensive analysis, though some information could be extracted. Later, the analysis process proved to be time-consuming and requiring a lot of resources in general. This fact discouraged me from putting an extra effort to analyze partial and possibly inconsistent data from the beginning of the experiment.

A.1.2 Data types

The collected data on a daily basis consists of two basic parts:

1. Snapshot of BGP table in text form,
2. Snapshot of RIPE DB contents.

BGP table

The BGP snapshot is a captured output generated by Cisco commands `show bgp ipv4 unicast` and `show bgp ipv6 unicast`. An example of a short output capture fragment follows:

```
    Network       Next Hop       Metric LocPrf Weight Path
*>i 1.0.0.0/24    217.31.48.125       0    128      0 6939 15169 i
*                 195.39.49.133            127      0 5588 15169 i
*>i 1.0.4.0/24    217.31.48.125       1    256      0 6939 4826
                                                        38803 56203 i
*                 91.210.16.201       1    240      0 6939 4826
                                                        38803 56203 i
*                 195.39.49.133            127      0 5588 6939
                                                        4826 38803
                                                        56203 i
*>i 1.0.5.0/24    217.31.48.125       1    256      0 6939 4826
                                                        38803 56203 i
*                 91.210.16.201       1    240      0 6939 4826
                                                        38803 56203 i
*                 195.39.49.133            127      0 5588 6939
                                                   4826 38803 56203 i
*>i 1.0.6.0/24    217.31.48.125       1    256      0 6939 4826
                                             38803 56203 56203 56203 i
*                 91.210.16.201       1    240      0 6939 4826
                                             38803 56203 56203 56203 i
*                 195.39.49.133            127      0 5588 6939
                                        4826 38803 56203 56203 56203 i
*>i 1.0.64.0/18 217.31.48.125         1    256      0 6939 4725
                                          4725 7670 7670 7670 18144 i
*                 91.210.16.201       1    240      0 6939 4725
                                          4725 7670 7670 7670 18144 i
*                 195.39.49.133            127      0 5588 6939
```

```
                                             4725 4725 7670 7670 7670 18144 i
```

The first two lines, that follow the header, describe the two possible paths towards prefix **1.0.0.0/24**. The most interesting part of it for routing policy analysis is the `Path` column. This example contains two distinct paths over two different upstreams **AS6939** and **AS5588**. We can also observe that the originator of the prefix is **AS15169**, BGP `origin` attribute is **i** (*internal*). The path over **AS6939** is being used because it has greater `LocPrf` (*local preference*) and therefore it has been selected by the BGP best path selection algorithm as the *best path*. This fact is displayed by the ">" character in the first column.

The complete output capture contains information about all prefixes in the Internet that are reachable from our observation point at the capture generation time.

However, some prefixes might be less straight-forward to read, namely the prefixes created by route summarization, redistribution from EGP or prefixes manipulated by some non-standard mechanism.

These prefixes are less frequent and the non-standard results should not occur in DFZ, even though some appear anyway.

Moreover, for the first three months the router, which has been used for obtaining the data, did not support 32-bit ASNs and therefore path data have been mangled by the backward compatibility mechanism, described in RFC 6793 [61]. However, since data from this period are not used in the subsequent analysis, it does not affect our results.

The size of the IPv4 table dumps varies from 783 555 lines and 67 250 680 bytes in the beginning (2012), up to 1 220 336 lines and 105 552 786 bytes in June 2015. For IPv6 the data size is considerably lower: It consisted of only 30 491 lines in November 2011 and in June 2015 it contained 181 681 lines.

Please note that the BGP table size is larger than the RIB[1] table size because the BGP table contains multiple paths for many prefixes. The reason for holding redundant information is to be able to decide on the best path and to switch to the second best path immediately when the best path fails. Generally speaking, the BGP table in our case contains at least two possible paths for most prefixes. Nonetheless, for some prefixes we might have as much as 10 competing BGP paths.

RIPE DB

In order to conduct analysis of the routing policies, I have decided to collect RIPE DB snapshots and store them in compressed form. The RIPE DB content dumps are available on FTP server of RIPE NCC. FTP makes it easy to download and pack daily snapshots.

The files from FTP are anonymized in order to protect personal data in certain RIPE DB object types, but the anonymization does not interfere with our goals because IRR and routing related information are preserved. The RIPE DB format has already been described in previous chapters: It is based on text files that contain RPSL *objects* grouped by the object type.

[1]Routing Information Base - the effective routing table on the router.

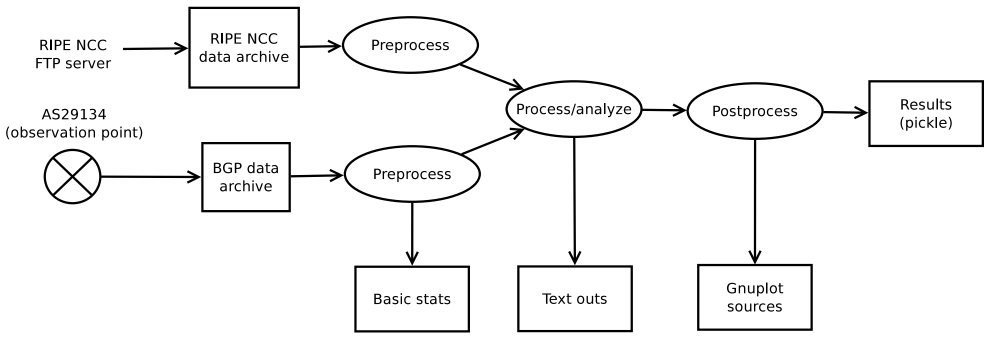

Figure A.1: BGP crunch architecture

A.1.3 Data size

Overall data size of the captured data prior to the first analysis (June 2015) in compressed form was 198 GB. Major part of the data was the RIPE DB dump archive. Results and byproducts of the analysis took 776 GB, while the actual results without byproducts take approximately 26 GB.

A.2 Data processing

A.2.1 Software requirements

Programming language

The presented outline calls for batch processing of great amount of data. The basic decision I made in the beginning was to use scripting language. The language should interact well with many different data analysis tools, in should be fast and should support distributed processing.

The language of choice for this project is Python 2.7, even though it does not excel in all the mentioned requirements. It proved to be reasonable choice and I was able to create and debug all components needed for the data analysis in a period of several months.

Parallel execution

I needed to scale out the CPU intensive parts of the analysis to more computers to get the results in reasonable time - in order of days for partial results and weeks for the complete analysis. Luckily, the analysis is easy to partition into independent tasks: Analysis of snapshots from a particular day does not depend on other days. The daily results can be merged and summarized in relatively short time.

This feature of the data set allowed to automate scheduling to N different servers with M CPU cores to achieve almost linear scaling.

A.2.2 Software architecture

The figure A.1 shows the high-level architecture of the analysis software **bgpcrunch**.

Processing of data can be divided into three steps:

1. Pre-processing: Scripts take the raw text data, generates internal data structures and several by-products for preliminary statistics and debugging purposes.

2. Processing: The actual matching of routing policies and BGP path vectors. The most CPU intensive and time-consuming part of the process.

3. Post-processing: Merging data, creation of the summary outputs and formatting outputs for subsequent visualization.

Pre-processing

This step begins with unpacking the archived data. Then we take the raw text data, parse them and create a few text outputs and very basic *Gnuplot* charts as well as Python lookup data structures that are stored into *pickle* files. The saved files are actually much larger than the input files, mostly because input files are compressed. Since the raw data are text files with a lot of repetitions, the compression ratio is high for these files. In fact, decompressing the raw input was a surprisingly time-consuming operation and it generated huge I/O load as well. Therefore I was able to decompress the input files on the main server, where all the data physically resided, in only 6 threads and it took a few days to complete.

Processing

The resulting *pickle* files with the pre-processed data structures are passed to the main and the most CPU intensive part of the analysis, which is called "processing". At this point we take the path vectors from the pre-processed BGP dumps and match the prefixes and corresponding paths to the `route` or `route6` objects in the pre-created lookup structures and record results to another *pickle* file.

Then we analyze each hop in the AS paths of each path vector in the BGP table and we try to validate the hops in AS path according to filters in the corresponding `aut-num` objects. The objects are in the pre-created lookup structures loaded from the *pickle* files. The difficult part is to resolve recursive selectors and then the recursive filters that might refer to many different objects that are in another pre-processed lookup structures. The depth of the recursion is bound by the recursive tree depth in `-set` objects superposed with depth of inherent recursive nature of the filter expressions.

The matching process has to lookup the proper `aut-num` object, loop through selectors and evaluate the filters that have the matching selectors. This task represents a lot of CPU-bound work but it also burdens memory and I/O as well.

This procedure is exceptionally time consuming. The analysis of the data, covering one day, takes almost a day on a single contemporary (May 2015) state-of-the-art CPU core. Therefore, distribution of the workload is needed. To achieve that, the data has been shared among many servers via NFS. The distribution of the work has been semi-automatic in the aspect of distributing work among the servers, because the workload assignment required special consideration with respect to the particular server power and spare capacity I was allowed to utilize. The work assignment has been fully-automatic at the servers in the aspect of distributing work among the available CPU cores and throttling the bulk processing to avoid blocking the primary application that ran on the server.

Phase	Running time	No. of CPU cores
pre-processing	127 hours	6
processing	247 hours	56
post-processing	27 hours	1

Table A.1: Analysis running time

Post-processing

The results of the "processing" phase still consist only of a few text outputs and several *pickle* files that contain the validation results in internal data representation. The last portion of the **bgpcrunch** software does the post-processing to take the analysis results in the *pickle* files and create main text outputs, count totals and generate sources of charts, that can be later processed by Gnuplot. Unfortunately this part has to run in a single thread because it needs to process the data sequentially and generate the output in the precise order. Of course it could be parallelized to a certain degree, but the running time is only several hours even on a single CPU core, so there is no real need to justify an extra effort needed for speeding this step up.

Postprocessing tools

The collection of tools used to post-process results from my Python software is limited to **R** and **Gnuplot**. Both tools helped with visualization and simple reasoning on basic features of the results and generated data.

A.2.3 Running time

The table A.1 shows the running time of the data analysis that covers 1 119 days of BGP and RIPE DB snapshots: From March 22, 2012 to June 21, 2015.

A.2.4 Data and product size

Another interesting aspect of the analysis is the size of the data and byproducts generated in the process. The table A.2 contains an overview of data types and their approximate total size.

A.2.5 Syntax errors

During the data pre-processing, I also discovered a bug in RIPE DB software that permitted a syntactically invalid object into RIPE DB. It was an unfortunate coincidence that the offending object **aut-num: AS2852** belongs to CESNET z.s.p.o. - Czech academical and research network. The problem was the following line:

```
mp-import: afi ipv6.unicastfrom AS39790 action pref=150; accept AS39790
```

The `ipv6.unicastfrom` fragment consists of two keywords `ipv6.unicast` and `from`, that should be separated by one or more white space characters. The bug

Data	Compressed	Uncompressed
IANA IP space map	N/A	44 kB
BGP snapshots	4 GB	91 GB
RIPE DB snapshots	194 GB	5.1 TB
pre-processing results	N/A	522 GB
processing results	N/A	306 GB
post-processing daily results	N/A	132 GB
post-processing summary results	N/A	9 MB
post-processing timeline results	N/A	2 GB
Total	198 GB	6.2 TB

Table A.2: Size of data and products

has been reported and acknowledged by RIPE NCC staff and a correction is expected in a future release.

Apart from this obvious syntax error I encountered two other types of more complex syntax errors in selectors that, I believe, break the standard to a certain degree. I created a work-around for the parsing errors and noted them for later discussion in mailing lists.

A.3 Long-term changes and trends

A.3.1 Global changes

The period covered by the available data captures important events with an impact on the entire Internet. The time frame covers the culmination of IPv4 scarcity problem and subsequential run-out in all but one RIRs. It also captures major IPv6 event: World IPv6 Launch [84] on June 6, 2012. Unfortunately, this event is not directly visible on the BGP propagation plots because the IPv6 BGP announcements of the participating parties had been in DFZ for a long time before the World IPv6 Launch day. The *flip of the switch* on the World IPv6 Launch day happened inside DNS.

A.3.2 IPv4

IPv4 prefix count

Despite of IPv4 run-out, the IPv4 BGP table has been growing steadily during the whole period, as the figure A.2 shows.

The possible answer to the question "*How it is possible that IPv4 table is growing despite the fact that the IPv4 source has been depleted in major part of the world?*", is that there are three factors that allow ISPs to add new prefixes to the DFZ:

1. The first possibility is to announce old and previously unannounced prefixes that some of the ISPs had obtained in the past and have not used them yet. The problem is that, except the prefixes allocated in pre-RIR era, it would be a breach of RIR's policies. Still, it is possible.

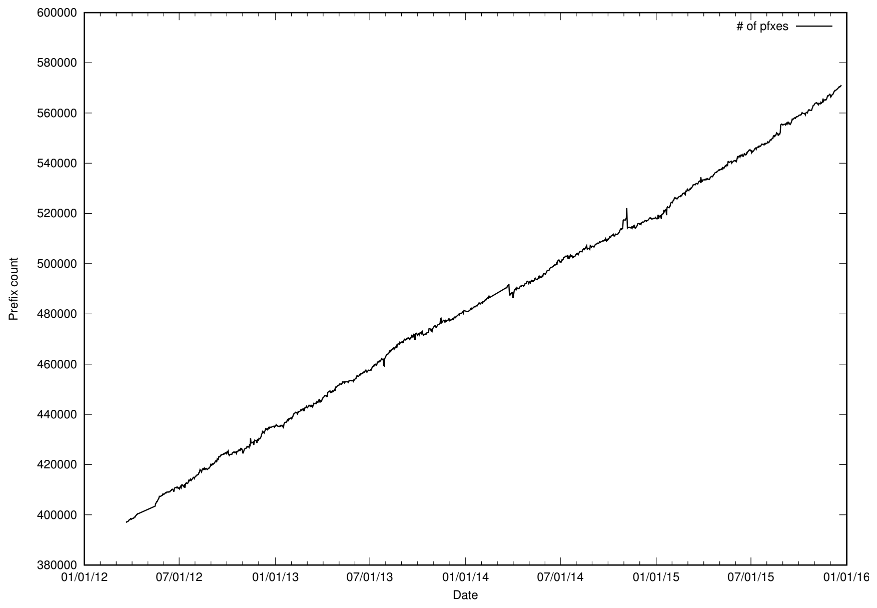

Figure A.2: IPv4 prefixes in BGP

2. The second possibility is obtaining small prefixes from RIRs' last /8. IPv4 run-out is not yet total (May 2015): RIRs decided a few years ago that they needed to change IPv4 distribution policy, when they hit a certain level of remaining addresses in the pool. According to this policy, most RIRs halted IPv4 distribution according to "*justifiable need*" when they reached their last */8* IPv4 prefix. New distribution policy for the last */8* in RIPE allows each LIR to obtain a single uniformly-sized */22* prefix from the last */8* under similar conditions regarding the justification of the need as for prefixes allocated before this policy. The intention is to support future newcomers with modest amount of IPv4 addresses to allow basic connection to IPv4 Internet and deployment of IPv6 transition mechanisms like NAT-PT.

3. The third possibility is deaggregation of existing prefixes in the DFZ. This is actually the most problematic consequence of IPv4 depletion for most ISPs, because prefix deaggregation on a large scale can cause substantial increase of IPv4 routes count in DFZ in a short time. Since BGP routers are usually hardware-assisted routing platforms, there is a certain hardware limitation on maximum routes in FIB. Reaching the limitation implies that the router can not operate in the Internet correctly anymore and has to be replaced or carefully configured to drop certain routes, which has a huge potential to cause further problems.

Average prefix length

The possible deaggregation and injection of a large number of */22* routes at the same time should be visible. In fact, the time series of average prefix length displayed in the figure A.3 shows rather steady growth throughout the whole period. The expected trend is visible in longer time frame in BGP analysis reports created by Geoff Houston from APNIC that are published on website [15].

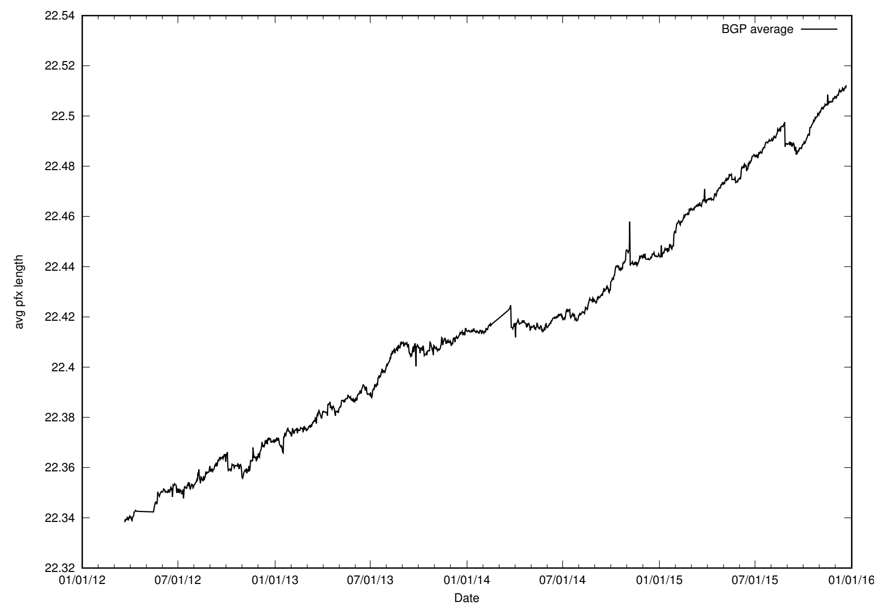

Figure A.3: IPv4 average prefix length in BGP

IPv4 /22 prefixes

Another interesting point is the possible increase of */22* prefixes in routing table due to policy for handling the last */8*. The problem is that the RIPE NCC started to allocate IPv4 address from the last */8* on September 14, 2012 which means that my data set is not old enough to capture the change in the trend. The figure A.4 shows rather steady growth, and once again I have to refer to BGP analysis reports by Geoff Houston [15].

IPv4 lower prefix lengths

But it fact the IPv4 depletion consequences are visible in plots of IPv4 prefix counts for lower prefix lengths, which translates to greater portion of IPv4 address space. For instance, share of one of the most common IPv4 prefixes - */20* is still increasing but it decelerates, as the figure A.5 shows.

In addition, even shorter prefixes started to drop. The most visible case is */16* in the figure A.6.

A.3.3 IPv6

Prefix count

Even though IPv6 growth is an interesting point on its own, the IPv6 growth in my data does not show anything exceptional or surprising. The growth in the captured period is steady and the shape in the figure A.7 seems to be linear in time.

Prefix size distribution

The growth of IPv6 prefix count is concentrated mostly in the default allocation units, which are */32* for PA allocations and anything between */32* and */48* for PI

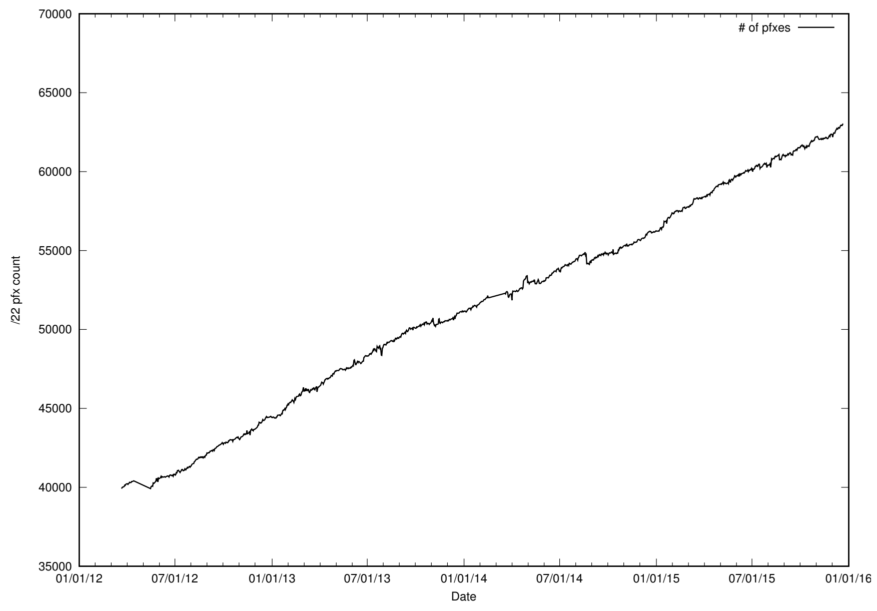

Figure A.4: Number of IPv4 /22 prefixes in BGP

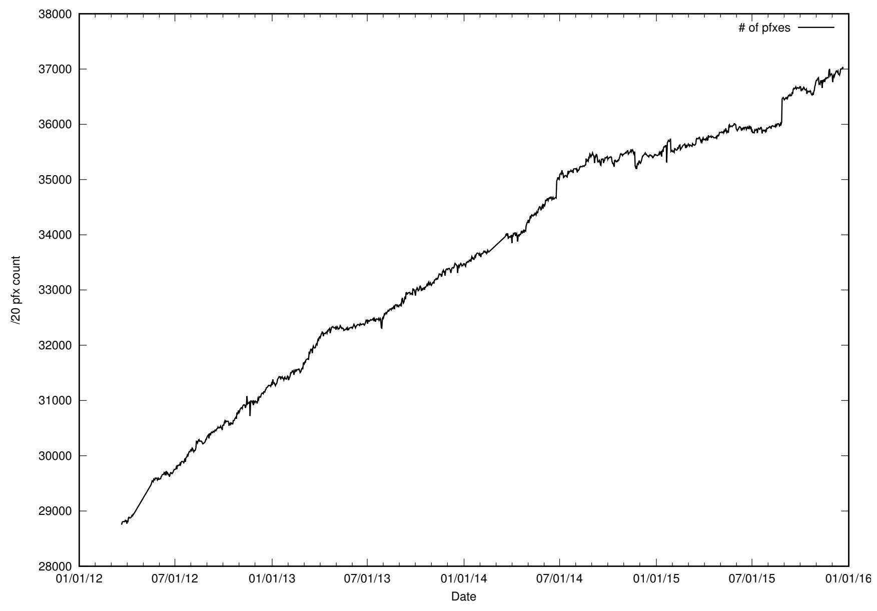

Figure A.5: Number of IPv4 /20 prefixes in BGP

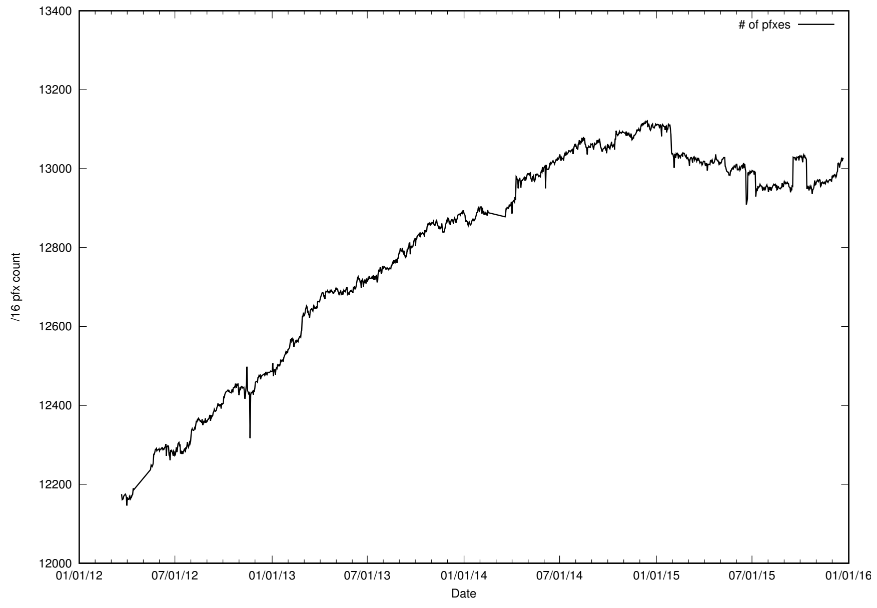

Figure A.6: Number of IPv4 /16 prefixes in BGP

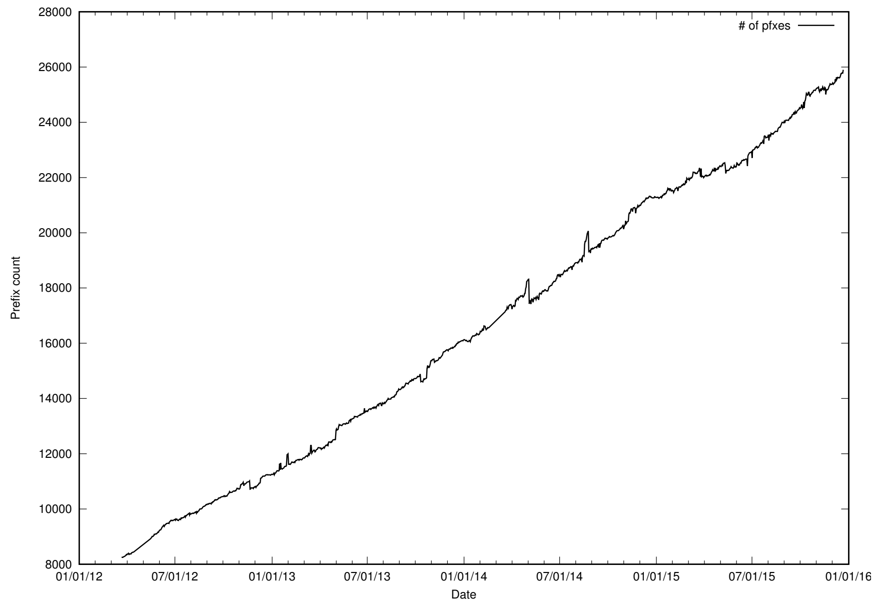

Figure A.7: IPv6 prefixes in BGP

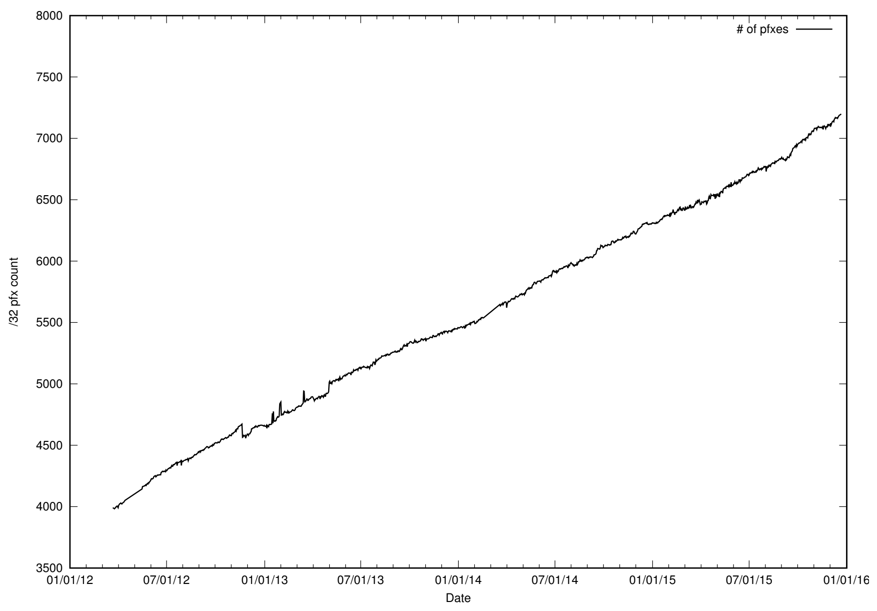

Figure A.8: Number of IPv6 /32 prefixes in BGP

allocations in most RIR service regions.

The figures for */32* - A.8 and for */48* - A.9 shows the growth throughout the captured period. It is also interesting that since IPv6 allocation policies allow allocation of even shorter prefixes than */32*, there are prefixes as short as */20*, which are visible in the figure A.10.

Average prefix length

The average prefix length in IPv6 was also growing for most of the captured period and the numbers over 40 are surprisingly high. Luckily it seems that it is slowing down and there is some hope that it might be turning for good to even decreasing trend in the future as the figure A.11 shows.

A.3.4 Changes near the observation point

The covered period captures global changes in scale of entire RIR service regions and in scale of the entire Internet, but in addition, the collected data are also affected by local changes that have happened in the proximity of the observation point.

Observation point

The observation point is an autonomous system border router in **AS29134**. The primary purpose of this router is to provide connectivity for an important Czech ISP that is focused on server hosting and related networking business.

The advantage of this observation point is that we have first-hand data from the real router, which is involved in day-to-day operation of backbone network. The ISP in question operates the autonomous system fully on its own. At the same time, the entire ISP network is fairly simple because there are only two BGP routers and a few IGP routers connected to them. Moreover, the network

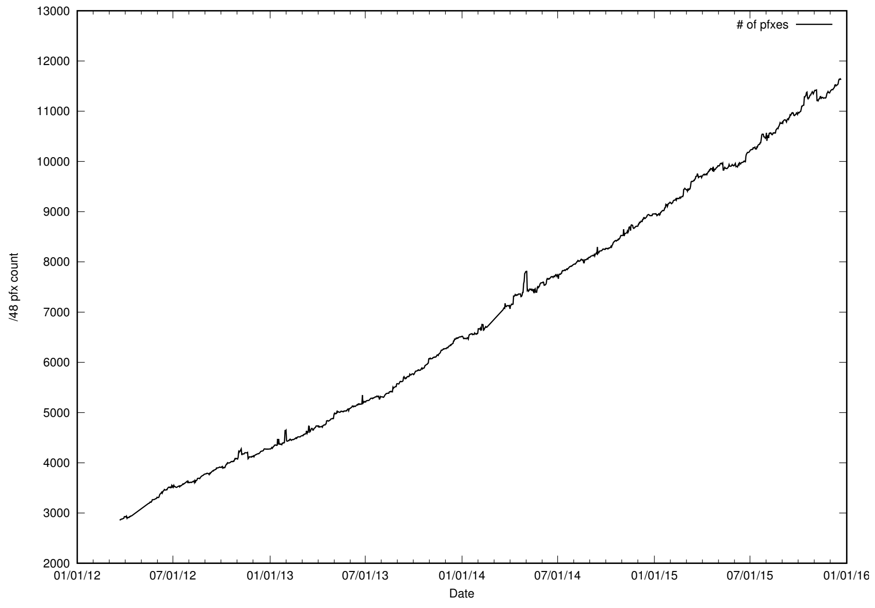

Figure A.9: Number of IPv6 /48 prefixes in BGP

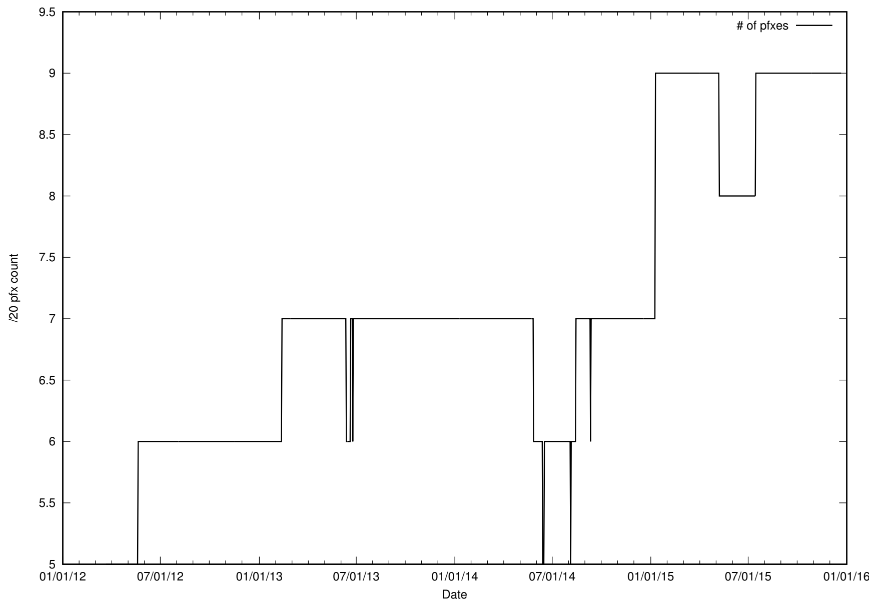

Figure A.10: Number of IPv6 /20 prefixes in BGP

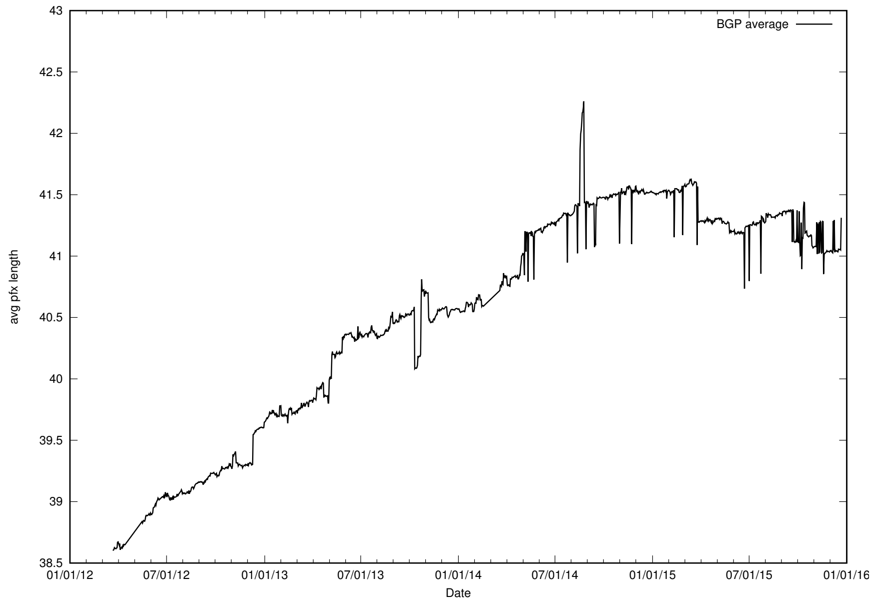

Figure A.11: IPv6 average prefix length in BGP

architecture follows design patterns and network engineering best practices to a great extent.

Upstreams of the observation point

On the contrary, the disadvantage of having this particular observation point, at this place and at this particular time frame is related to the fact that the European and especially Czech ISP business underwent transformation and restructuralization during the observation period.

Analysis of the European ISP business trends is clearly beyond the scope of this thesis. The consequences for the observation point **AS29134** have been mainly several changes of upstream providers and experiments with low-cost upstream and *remote peering*[2] services, provided by certain networks that had started to operate in the Czech Republic not long ago.

IPv4 path length

The consequences were rapid changes in DFZ view and quality of visible paths. The figure A.12 shows average daily path length for all prefixes in DFZ. To interpret the figure please note that longer paths do not need to be worse than shorter ones. Nevertheless, fluctuations and instability is always harmful in networking.

The reason for the changes is that the observation point changed the main upstream networks several times:

1. September - November 2012: Upstream changed from AS174 to AS6939.
2. June 2013: Remote peerings to AMS-IX (Amsterdam) and De-CIX (Frankfurt) added.

[2]Remote peering is a partial upstream service. Usually a network that have points of presence in one or more important locations connects to the IXPs at these locations and it provides connectivity to the IXPs from remote PoPs via MPLS.

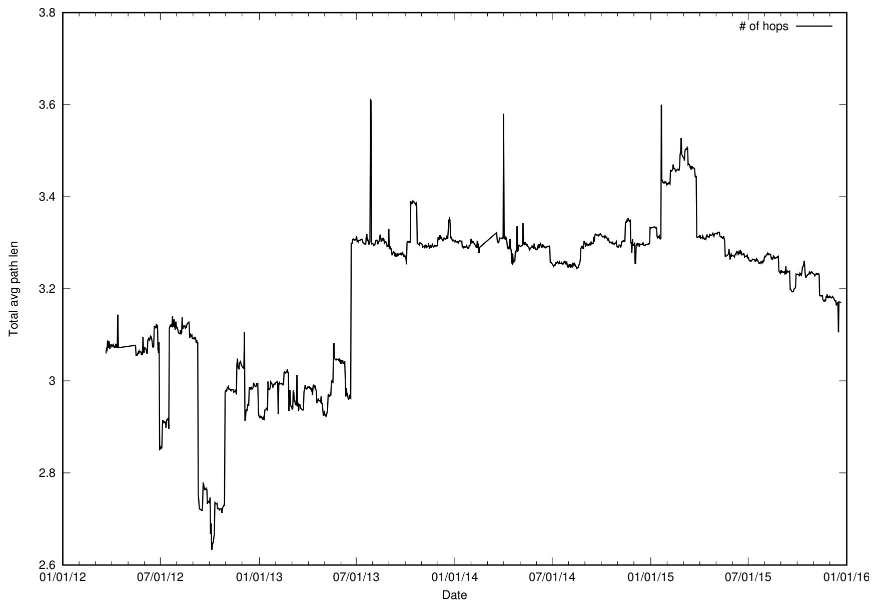

Figure A.12: IPv4 BGP average path length

3. May 2014 - Remote peerings to SIX (Bratislava) and LINX (London) added.
4. January 2015 - Remote peerings migrated to AS5580. New upstream AS42000.
5. March 2015 - Remote peerings shutdown. New upstream AS2819.
6. July 2015 - Change of upstream priorities. Removal of AS42000 upstream.

The first upstream hop has a great potential to change the statistics because average path length to all IPv4 prefixes in DFZ from the observation point is only approximately 3.8.

IPv6 path length

Generally, any fluctuations and rapid changes in BGP translates to lower quality of the connection.

Quality of the Internet connection in general means high throughput, lack of blind sports[3] and consistency. The rapid changes in path length usually contribute to breaching consistency.

Unfortunately, the plot of IPv6 average path length A.13 shows even more fluctuations. But this results can be explained by the lower number of prefixes in IPv6 DFZ and lower stability of the new address family.

IPv4 path length relative to prefix length

Unlike in the previously discussed peaks in the total average charts that have been correlated with known changes in the **AS29134** connectivity, we can observe general trends in plots of path length relative to prefix length and time. The figure A.14 shows the path length plot in 3D. The two independent variables are prefix length and time.

[3]Blind spots are unreachable or badly reachable remote networks.

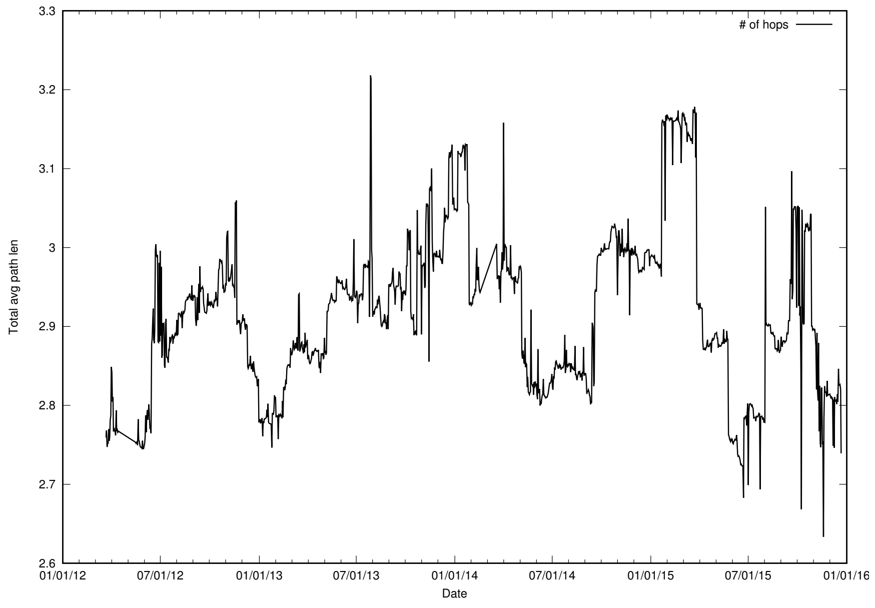

Figure A.13: IPv6 BGP average path length

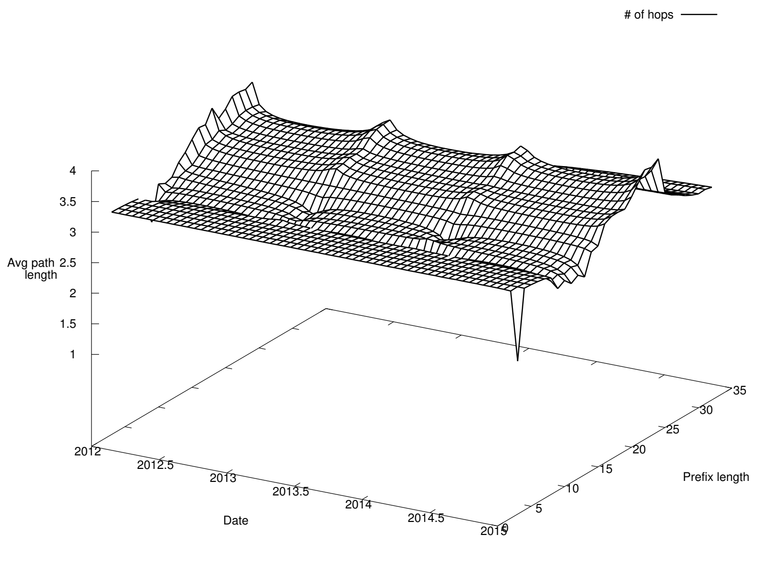

Figure A.14: IPv4 BGP path length per prefix length

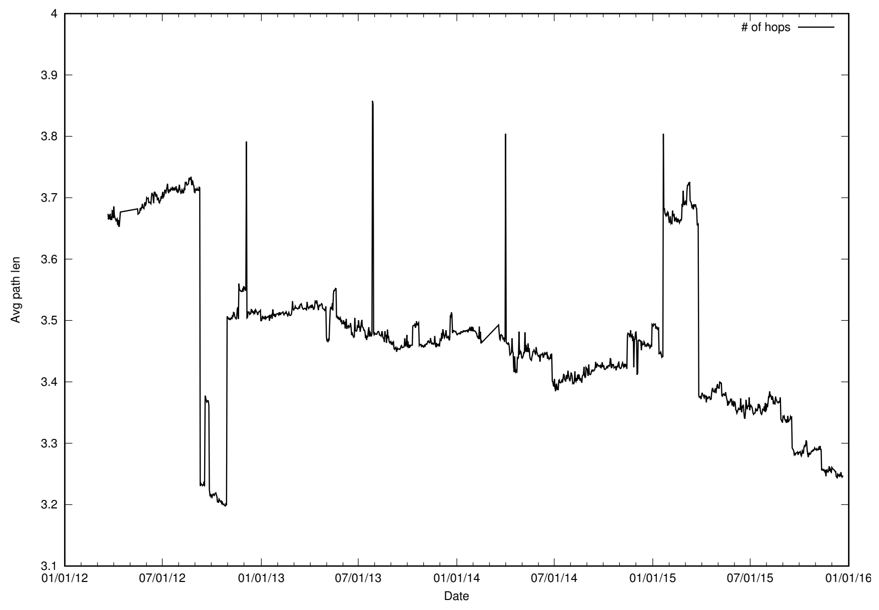

Figure A.15: IPv4 BGP path length time series for /16

This figure shows two groups of peaks from left to right in the chart and the peaks invert each other in the path length axis. The peaks happens to be at */16* and */24* marks. The */16* group shows decrease of the average path length. The */24* group seems to be increasing.

It is worth examining that particular prefix lengths in more detail. The figures A.15 and A.16 shows 2D plots of the groups in question. The first one is the decreasing */16* group and the second one is the increasing */24* group. These charts show the bad effect of deaggregation: Paths are getting longer for smaller and less specific prefixes are disappearing and deaggregating. The side effect of the dissolution of less specific prefixes is decreasing length of the remaining paths since deaggregation is the case of remote networks that are relatively far from our observation point.

The 3D plot reveals the rough profile of the IPv4 path length plotted relative to prefix length, but it is imprecise and too difficult to analyze.

Another possibility to analyze relation between prefix length and path length is to plot them on a daily basis. The figure A.17 shows the prefix length to path length profile for June 21, 2015. It is in fact only Y - Z projection of the proper X point in the 3D plot.

The IPv4 path length profile have actually changed substantially several times because of upstream changes or because the new connections have been created either locally or remotely. The remote changes have to happen near enough to the observation point to affect the local view of the Internet.

For instance, we can see the completely different profile in the chart for the first day of the observations - March 22, 2012 in the figure A.18.

These charts reveal the local quality of the IPv4 connection to a certain level, but the interpretation of the captured data and reasoning about them clearly depends on operational context and network needs.

The figures are presented in this chapter because it is important to get a high-level view on the DFZ data, scale and speed of changes in the captured period

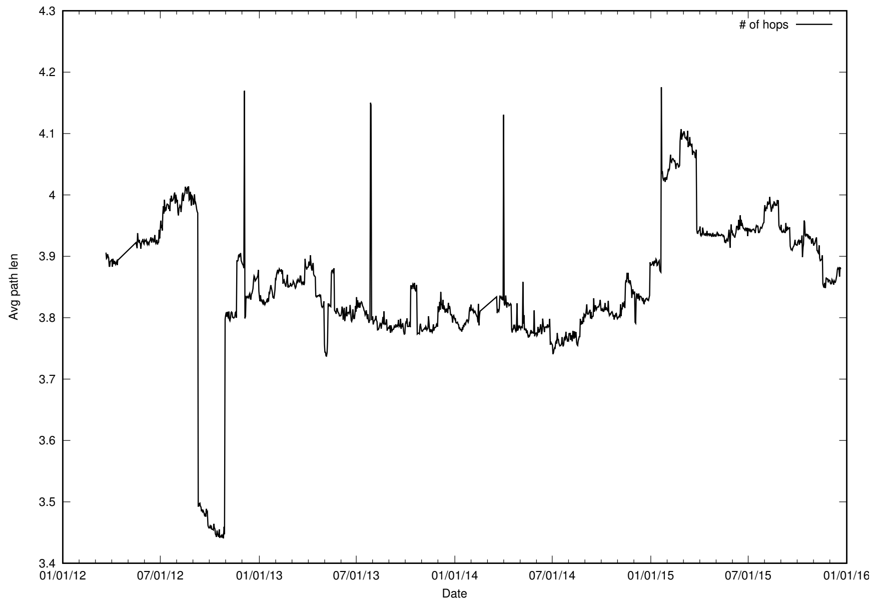

Figure A.16: IPv4 BGP path length time series for /24

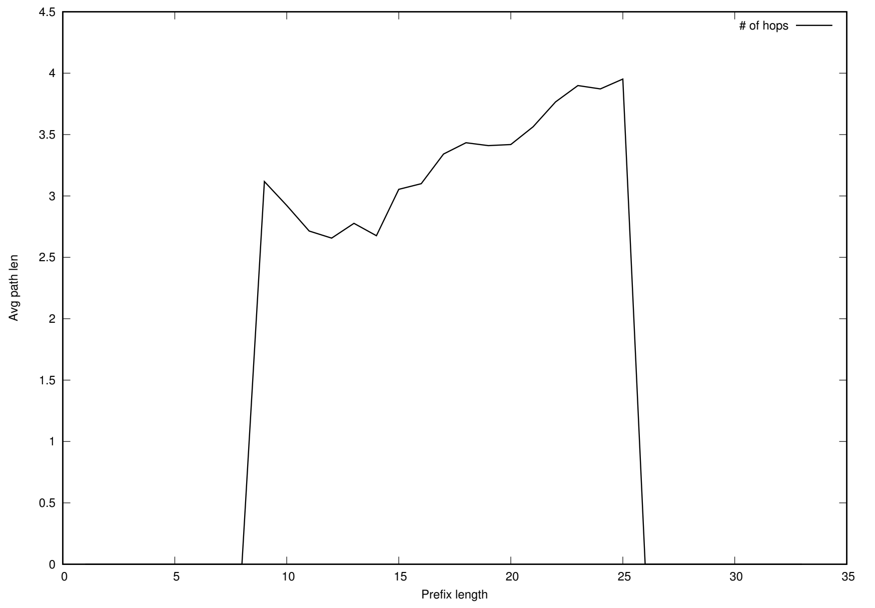

Figure A.17: 2015-06-21 - IPv4 BGP path length per prefix length

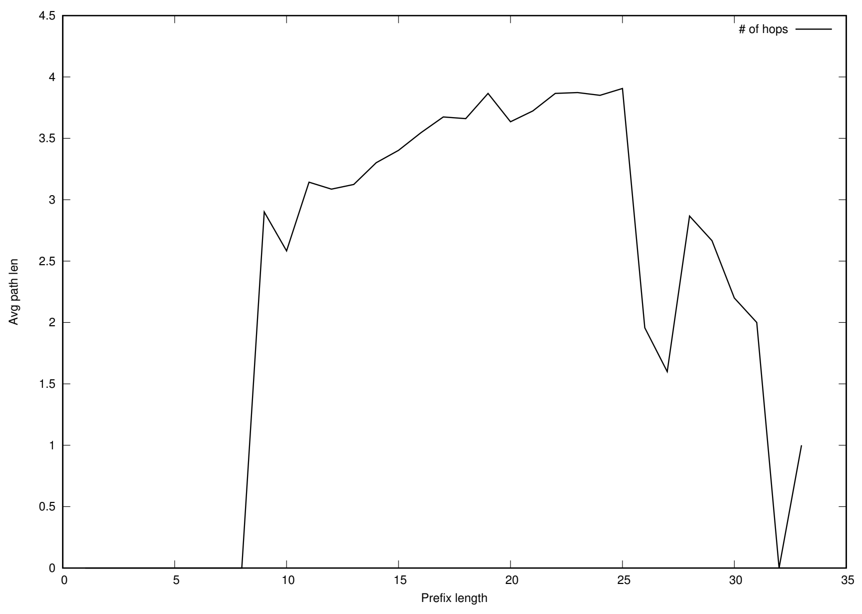

Figure A.18: 2012-03-22 - IPv4 BGP path length per prefix length

before we can continue to discuss the main results.

IPv6 path length relative to prefix length

Quality of IPv6 connectivity is correlated to a certain degree with IPv4 service provider because the **AS29134** applies a policy that dictates to operate always in fully dual-stacked mode.

The requirement of dual-stacked networking has been applied in all contractual relationships with upstream providers or any other service providers. However, IPv4 is still more important than IPv6 both from technical and business point of view. This natural prioritization affects the quality of IPv6 paths in a negative way almost everywhere in the Internet.

The figure A.19 shows the basic BGP path lengths plotted relative to time and prefix length in the same manner as in 3D plot for IPv4.

The major changes perpetuated by the upstream connection migrations to various service providers over time are visible at almost same points as in case of IPv4.

IPv6 path length time series of the most common prefix lengths are in the following figures: The figure A.20 shows the path length for */28* prefixes. The */28* prefixes are shorter than the preferred default in most RIR regions. However, it is easy to justify the need for such a large IPv6 prefix, especially for large ISPs that have plans for deployment of *6rd*[4].

In IPv6 the preferred default allocation size for PA IPv6 address space is usually */32*. The figure A.21 shows path length time series for */32* prefixes.

Even though IPv6 is expected to provide better opportunity for aggregating IP addresses, it seems that a lot of networks announce the longest practically

[4]6rd is acronym for IPv6 Rapid Deployment transition mechanism, which needs a large amount of IPv6 addresses for direct mapping of IPv4 address space to network part of IPv6 address. It usually uses */28* prefix combined with 32 bits of IPv4 address as a prefix delegation to the CPE. It allows the CPE to assign 16 networks with standard */64* prefix for each one.

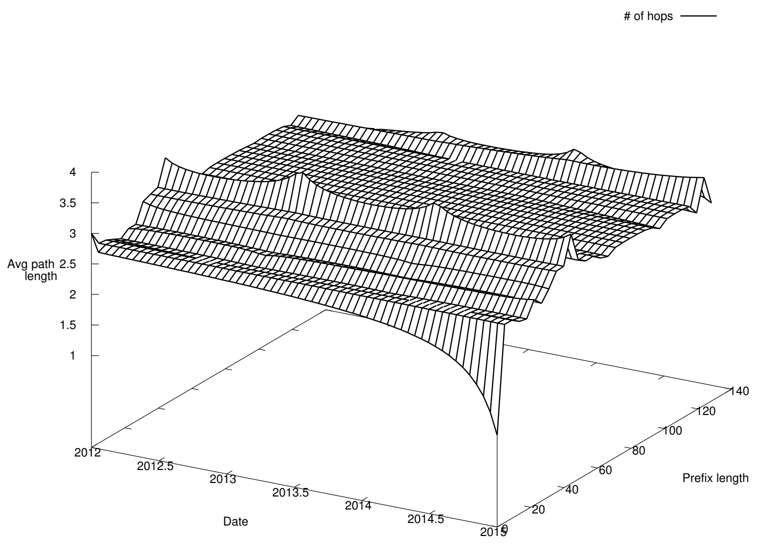

Figure A.19: IPv6 BGP path length per prefix length

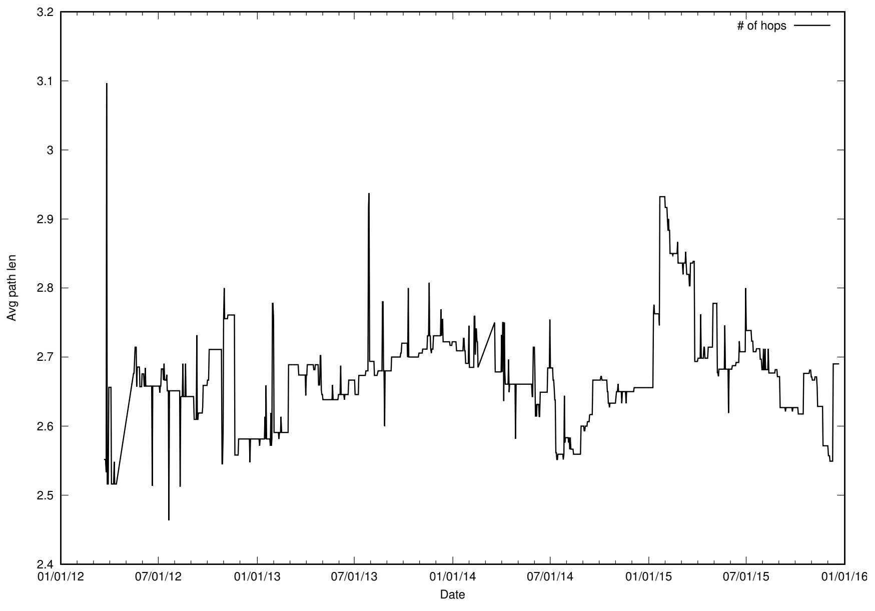

Figure A.20: IPv6 BGP path length time series for /28

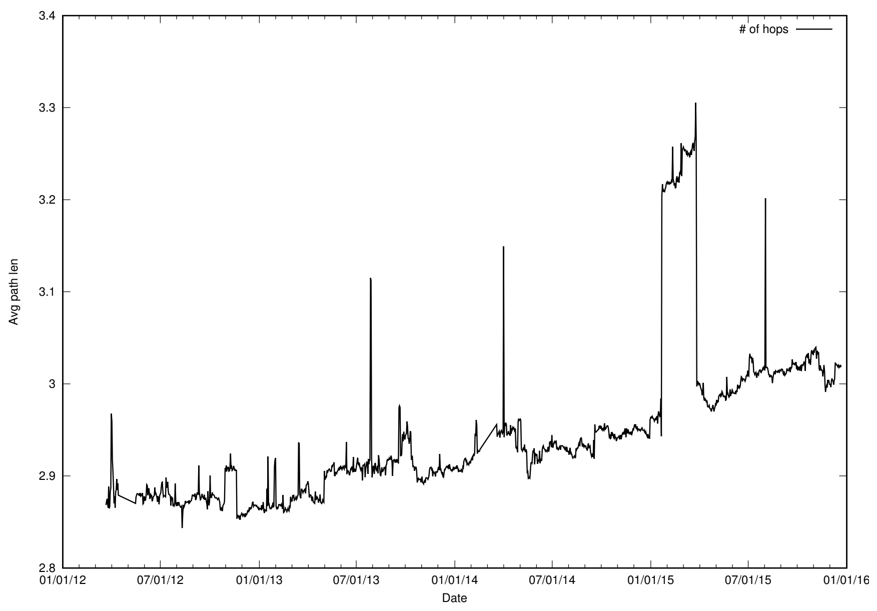

Figure A.21: IPv6 BGP path length time series for /32

usable[5] prefixes, which is */48*. The figure A.22 shows the path length time series for these prefixes.

We know from the figure A.9, which shows the number of visible */48* prefixes in DFZ in time, that this group forms a considerable portion of IPv6 DFZ and it is growing quickly.

It is actually not surprising because in IPv4 the */24*, the longest and therefore smallest prefixes that are practically usable, form the most ample group of prefixes in DFZ. In fact, more than half of all IPv4 prefixes are the */24*s. And it seems that this unfortunate situation of large scale utilization of the longest possible prefixes transitions to IPv6 as well.

The limit of practical usability is a matter of the best practice, an unwritten rule and a common belief in the Internet community, that there are and should be filters on certain boundaries. Effects of these limits are interesting: Because the **AS29134** does not impose any limits on incoming prefixes, there are occasionally some shorter prefixes visible in corresponding time series. Please refer to the **IPv4 BGP timeline of prefix count per prefix length** or **IPv6 BGP timeline of prefix count per prefix length** figures on the website `http://aule.elfove.cz/~brill/bgpcrunch` for more information about short prefixes.

A.3.5 RIR service region differences

Even though this thesis is focused on RIPE NCC service region that covers Europe, the Middle East and parts of Central Asia, the Internet is a global network and we get all the routes blended in the BGP table.

[5]The practical usability does not have any specification or general rule. The current defacto standard of filtering more specific prefixes than */24* in IPv4 and */48* in IPv6 has been established on strictly informal and unwritten basis. Any AS operator is free to set his own rules and impose his own limits on prefix length.

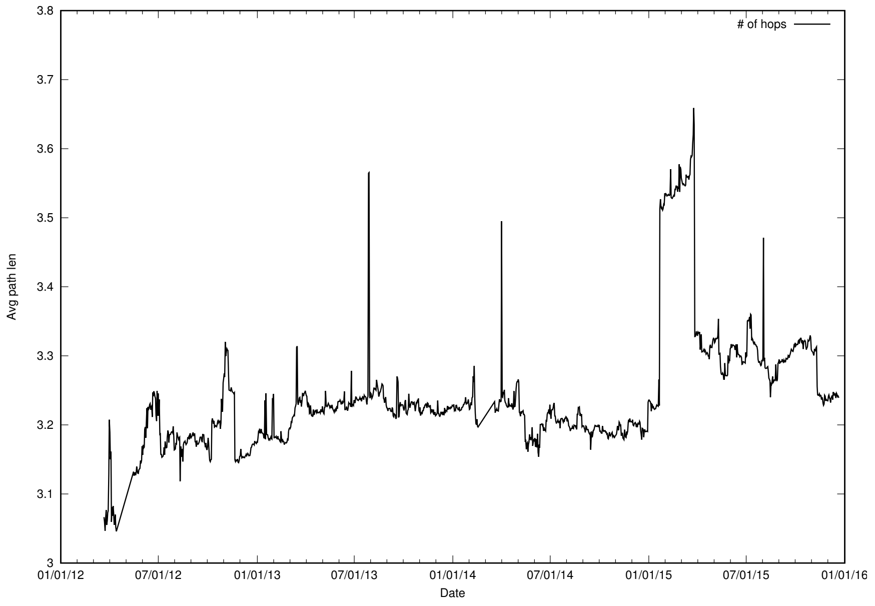

Figure A.22: IPv6 BGP path length time series for /48

RIR	No. of prefixes	Average prefix length
LACNIC	65045	22.33
APNIC	133151	22.33
ARIN	189472	22.33
RIPE NCC	136158	22.33
AFRINIC	12273	22.33

Table A.3: IPv4 RIR regions size - 2015-06-21

Region prefix share

The figure A.23 shows the share of the routes in DFZ of each RIR region in the IPv4 Internet. The chart is a time series that captures the observation period. The figure A.24 shows the same numbers for IPv6 AFI.

The most interesting points are that RIPE NCC service region clearly surpassed ARIN region in IPv6 deployment, and it is also the fastest growing and consistently increasing region in IPv6 AFI.

On the contrary, the IPv4 time series shows a huge gap among ARIN region and all others.

IPv4

The table A.3 shows the share of IPv4 DFZ by the RIR regions and the average prefix length in each region on June 21, 2015. The average prefix length indicates the level of deaggregation within the region.

The most alarming figure of this section is the RIR average path length time series for IPv4 in the figure A.25. Although we know that service region of AFRINIC comes through troubled times and there are understandable reasons for deaggregations and and changes in subnet announcements, this result exceeds any fears and the worst expectations. The chart proves that the Internet is

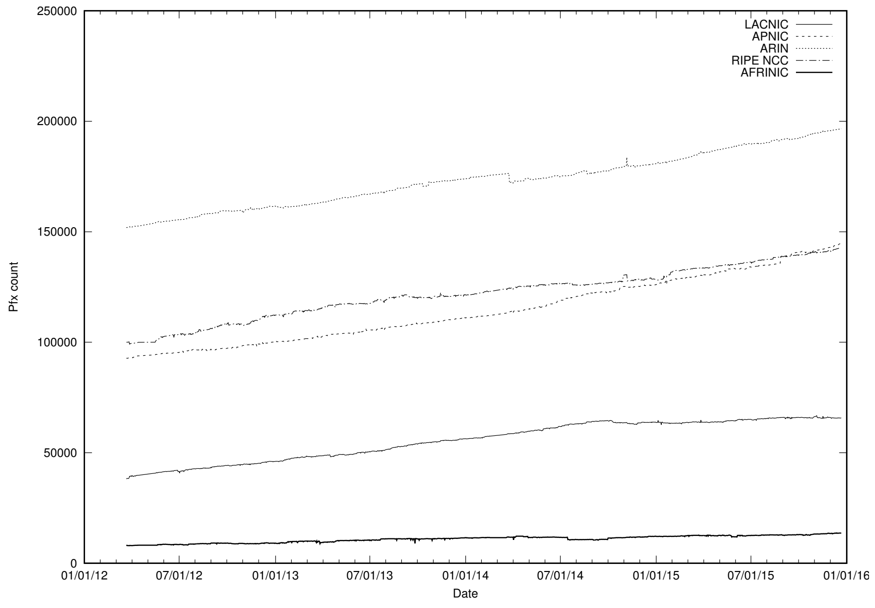

Figure A.23: IPv4 RIR DFZ share

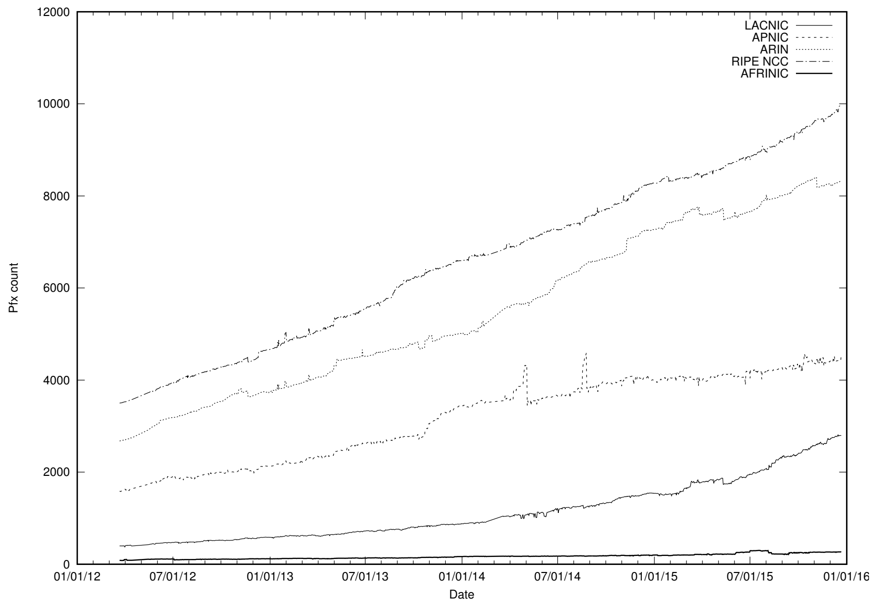

Figure A.24: IPv6 RIR DFZ share

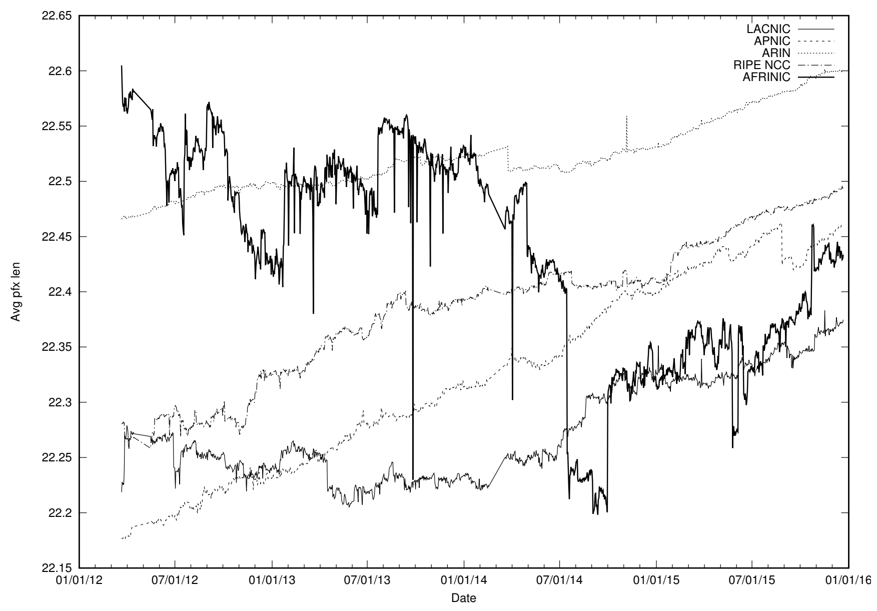

Figure A.25: IPv4 RIR average prefix length

RIR	No. of prefixes	Average prefix length
LACNIC	1890	38.19
APNIC	4060	43.61
ARIN	7629	43.25
RIPE NCC	8822	38.99
AFRINIC	258	40.05

Table A.4: IPv6 RIR regions size - 2015-06-21

extremely dynamic even in its technical base and it seems that people are trying to attain more performance, revenue and any advantage from their resources by constant changes. The changes seem to be unfortunately too huge and violent.

IPv6

The table A.4 shows the prefix count and average prefix length for IPv6 on June 21, 2015. The absolute numbers are naturally much lower than in the previous case.

The level of deaggregation in IPv6 DFZ is a huge disappointment because in IPv6 the common allocation for a LIR is */32*, but the numbers are reaching close to */48*, which is the suggested minimum assignment for the end network. The end networks are supposed to get aggregated at the ISP level, but it is obviously not happening.

The average prefix length chart for IPv6 is in the figure A.26. It shows more consistent but still alarming numbers. In case of IPv6, the most concerning point is mainly the scale of changes within short periods of time.

APNIC region exhibits prefix length amplitude over 2 in the last year and it oscillates within weeks or even days.

In IPv6 it is highly unlikely for one single ISP to affect the whole region

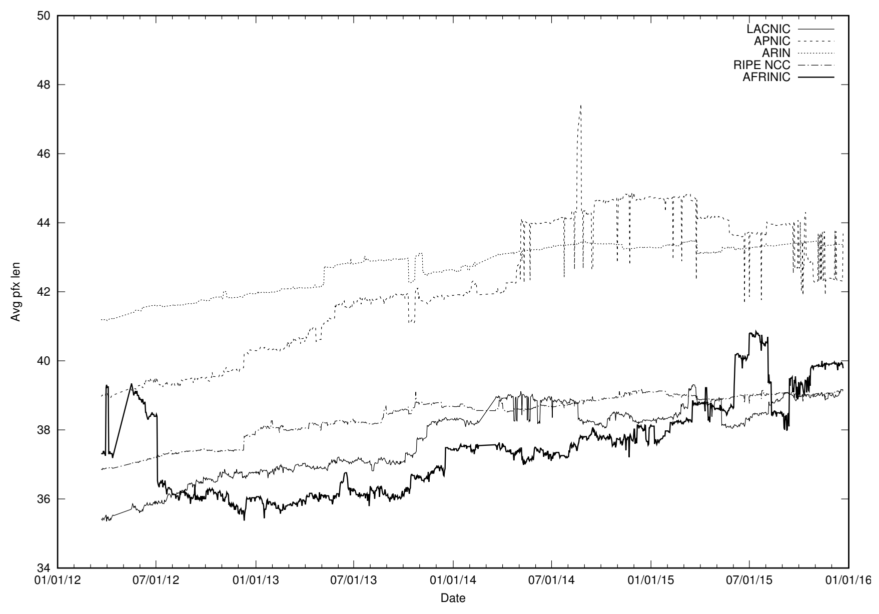

Figure A.26: IPv6 RIR average prefix length

because even the major ISPs are supposed to have a single PA allocation. They might deaggregate, however it is hard to imagine one single ISP to announce one short prefix on one day and thousands of longer prefixes the next day. More research into this aspect might yield interesting results in the future, but it is not required for this analysis to further elaborate on different RIR region.

This section intentionally presents only a few selected charts to show long-term trends and the scale of the parameters. There are over 1800 charts that contain relevant results on the above mentioned **bgpcrunch** results web site. The web site contains charts with even more data points and the full text outputs with detailed categorization of IP resources.

A.4 Origin validation

A.4.1 Method

Objects

The *origin* of BGP routes is moderately easy to validate. The idea of this measurement has been formally described in chapter 4. The question is whether we can find `route` or `route6` objects for each path vector in DFZ BGP table. If yes we try to match the `origin` attribute in one of the objects with the first AS in `AS_path` BGP attribute of the path vector. If there is a match, the path vector is valid. If not, the vector is either invalid or undecidable, depending on the region to which the prefix belongs.

Apart from the object type in IRR, the checking process is the same for both IPv4 and IPv6.

Prefix	Announcing AS	`route` object	Result
RIPE	RIPE	missing	fail (missing `route` obj)
RIPE	RIPE	prefix match, no `origin` match	fail (AS not match)
RIPE	RIPE	prefix match, `origin` match	OK
RIPE	no AS (local prefix)	N/A	N/A (origin missing)
RIPE	anything, but the path is summarized	N/A	N/A (no-search aggregate)
RIPE	non-RIPE	missing	fail (missing `route` obj)
RIPE	non-RIPE	prefix match, no `origin` match	fail (AS not match)
RIPE	non-RIPE	prefix match, `origin` match	OK
non-RIPE/legacy	anything	anything	N/A (non-RIPE)

Table A.5: Origin validation states

Object multiplicity

It is possible to find more `route` or `route6` objects for one prefix in one routing database and the meaning is logical **OR**. Therefore, the path vector is valid when it matches at least one `route` or `route6` object, according to the proper AFI.

IRR resolution and undecidable path vectors

Since the experiment is limited to European prefixes and RIPE DB data, we do not have to resolve the fundamental problem of multiple Internet routing registries: Deciding what data are authoritative for each path vector in the Internet.

In case of `route` or `route6` objects, the problem seems to be reasonably simple because a single match is sufficient for marking the path vector as valid. The problem is that more routing registries might contain outdated and even conflicting data, which would be an issue. In our case with only one database the possible situations are described in the table A.5. The table lists regions of origin for prefixes and the first (right-most) autonomous systems in `AS_path` and the possible outcomes of the checks.

Path vector multiplicity

The outlined resolution procedure has a hidden limitation that concerns conflicting path vectors: A prefix could be present in BGP table of the observation point multiple times. It effectively means that there are more competing path vectors with the same prefix component and different remaining components.

It is perfectly normal and desirable to have alternate paths to one destination. In this case, BGP selects one of these path vectors as the best, installs the corresponding route into FIB and further propagates the best path vector.

It is even possible to have one prefix with different origins, though it is highly unusual and each case raises suspicion since there is practically no valid reason to do this intentionally. On the contrary, it might be a symptom of ongoing prefix hijacking attack.

We have many different paths to each prefix in DFZ and we want to save CPU time spent on matching them with the `route` or `route6` objects. In this situation it is reasonable to assume that all paths have the same origin and examine only the best paths.

It means we have to examine approximately only 500.000 path vectors for each day instead of over two millions. An obvious downside is that we miss possible prefix hijacking incidents. To remove this issue we have searched through data specifically for this incidents and we found none. In fact, it is not likely to capture this kind of attacks because they are rare and it would have to happen near (BGP-wise) the observation point. Furthermore, we would need to have diverse connectivity to the victim AS and to the attacker at the same time.

Decision rules

Decision rules for the origin validation procedure derive not only from standards and RFC documents, but also from the current best practices and IP allocation policies of RIRs. These rules are subject for interpretation and some of the rules are unwritten and customary. The main issue is whether we can ignore path vectors that are parts of IP space allocated through RIPE NCC, but the announcing AS is not a part of RIPE NCC blocks. According to current RIPE NCC allocation policies it seems correct to count failed path vectors that fall within RIPE IP space into errors regardless of the announcing AS. The reason is that policies forbid using the majority of resources allocated by RIPE NCC outside of the RIPE NCC region. Therefore the autonomous systems of different provenience should operate in RIPE NCC service region and should submit to the same rules. Moreover, it is possible and encouraged to register foreign AS into RIPE DB and use it as an `origin` in `route` or `route6` objects withing RIPE DB.

A.4.2 Results

Absolute numbers

The figure A.27 shows the BGP origin validation time series for IPv4. The IPv6 counterpart is in the figure A.28. Both charts contain absolute numbers of the result states according to the table A.5.

The complete origin validation results in text form are too long to be presented here or attached to the thesis in any other form. The results are accessible on the above mentioned web site in the sections called **Daily IPv4 BGP route matching report (text)** and **Daily IPv6 BGP route matching report (text)**.

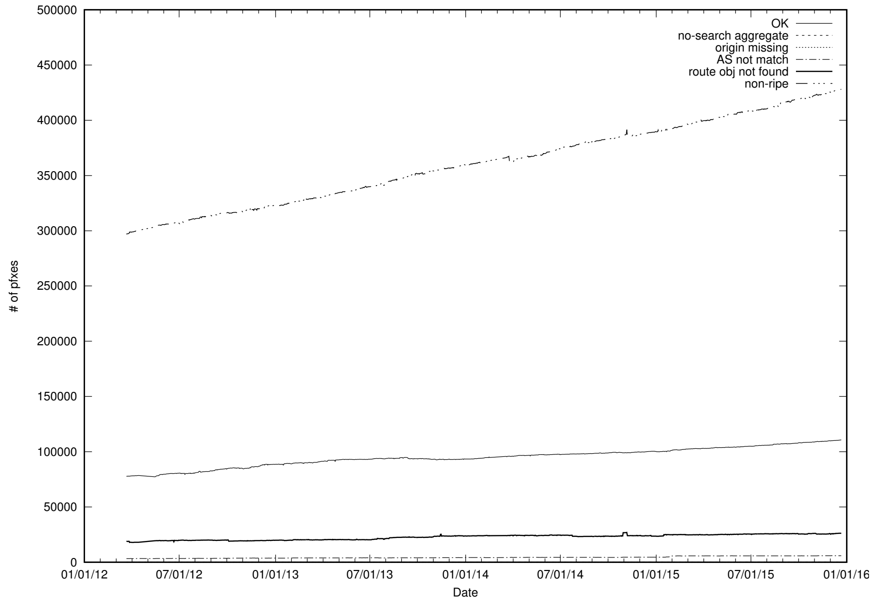

Figure A.27: IPv4 BGP origin validation time series

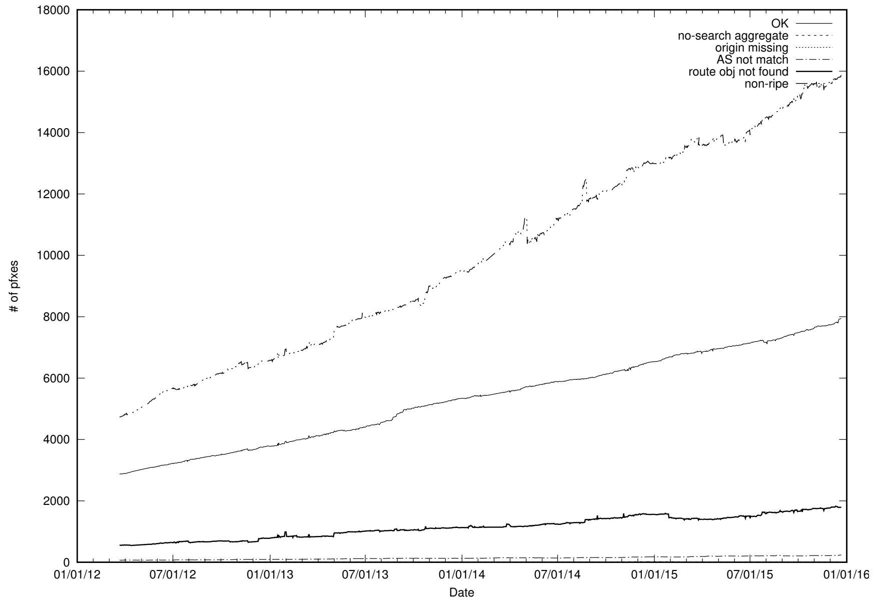

Figure A.28: IPv6 BGP origin validation time series

Status	No. of path vectors
OK	104778
no-search aggregate	164
origin missing	3
AS not match	5778
route obj not found	25555
non-ripe	407049

Table A.6: IPv4 Origin validation states

Status	No. of path vectors
OK	7113
no-search aggregate	14
origin missing	2
AS not match	204
route obj not found	1498
non-ripe	13837

Table A.7: IPv6 Origin validation states

The most interesting output is the section **Route violations timeline**. And the same results are attached on the disc that comes with the printed version of this thesis. This collection of results comprises of one file for each prefix that failed the BGP origin validation test at least once in the whole captured period. Each file contains all changes that happened either in BGP or in IRR, that concern the prefix in question.

The table A.6 shows IPv4 and the table A.7 shows results for IPv6 from June 21, 2015.

The following two charts show high-level summary of the results. The figure A.29 contains IPv4 totals and the figure A.30 shows the same numbers for IPv6.

Relative success ratio

It is useful to plot relative success rate in order to provide evidence for the hypothesis in chapter 4. The figure A.31 shows time series of invalid to valid ratio (error ratio) for IPv4 and the figure A.32 shows the same ratio for IPv6.

Conclusion

The origin validation results proved that almost one fourth of the prefixes in the RIPE NCC service region have invalid origin with respect to the IRR at any given time. It happens despite the fact that the RIPE community encourages LIRs, ISPs and network operators within the RIPE NCC service region to use `route` and `route6` objects to describe the most elementary BGP routing configuration.

There is an obvious question: Is it worse not to have any matching `route` or `route6` object for the prefix at all (which is the *route obj not found* status in validation) or to have some object with the wrong `origin`?

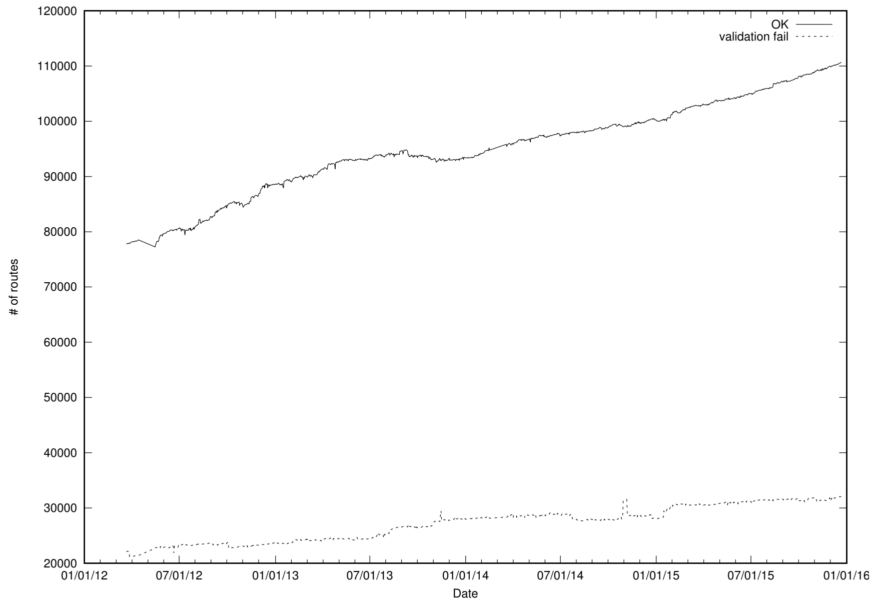

Figure A.29: IPv4 route origin validation results

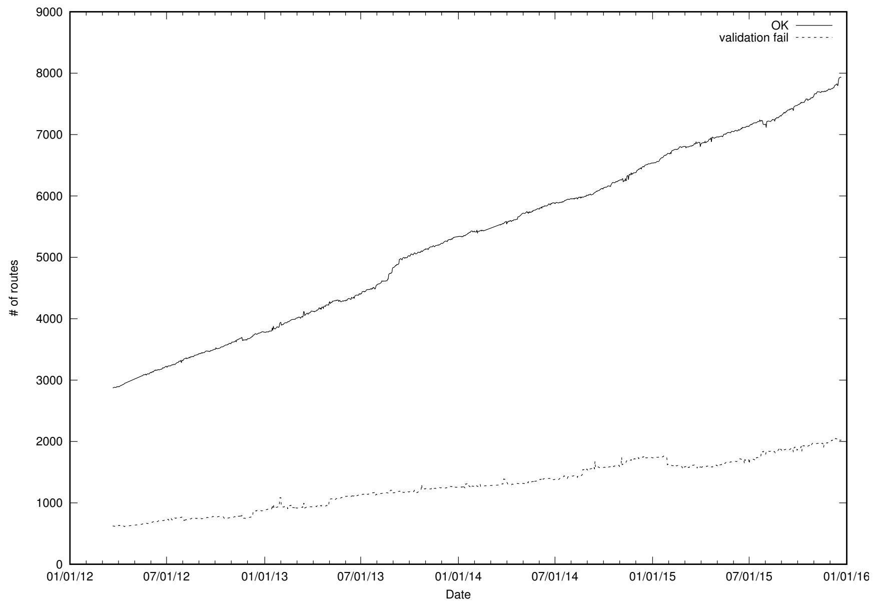

Figure A.30: IPv6 route origin validation results

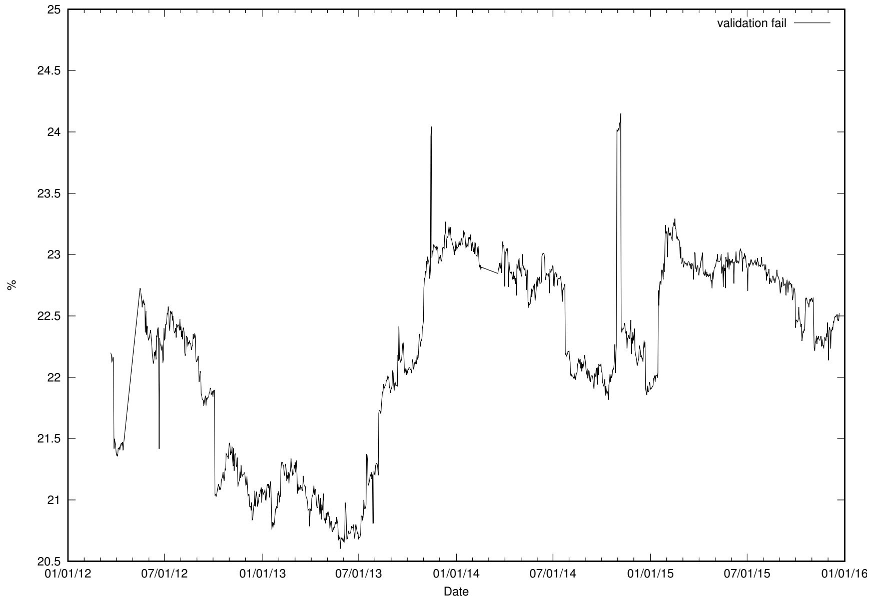

Figure A.31: IPv4 route origin validation percentage

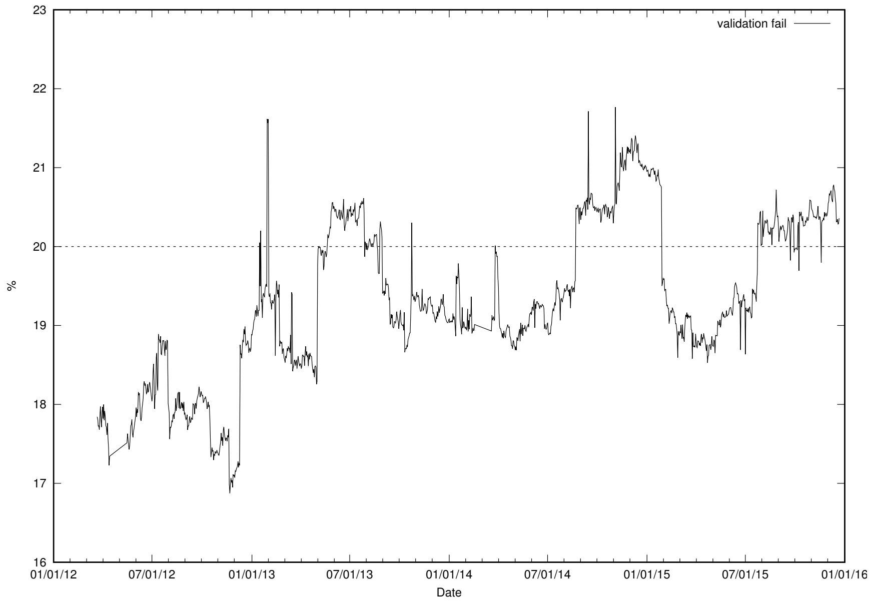

Figure A.32: IPv6 route origin validation percentage

The level of incorrectness depends on the reason why the discrepancy among BGP path vector and the available `route` and `route6` objects in IRR exists:

- It might be transitional state. The prefix might be in process of migration from one AS to another. In this case the correct procedure is to create a new `route` or `route6` object with the new origin and then migrate the prefix in BGP. Only after that the old `route` or `route6` objects can be removed.

- Another possibility is that the corresponding `route` or `route6` objects are completely missing through the entire history of the prefix. This is not a rare situation at all. These prefixes are present even in case they get filtered out when the origins are validated in IXPs, especially when the prefixes traverse route servers. It effectively means that the failed prefixes are more likely to take longer and more expensive paths and are generally limited in number of alternate paths in the entire Internet.

- It is also possible that some failed prefixes are actually hijacked. It means that the prefix is announced to DFZ by other autonomous system than the entitled to do it. The most common type of hijacking concerns either unused *legacy* prefixes that have not been reclaimed, or generally unused prefixes. In the Internet any autonomous system can start to announce the prefix into DFZ even though it is unrelated to the prefix in question. Responsibility for spreading that announcement to the neighboring networks and subsequently to the Internet is shared between the originating AS and its upstreams that accepted and forwarded the announcement.

- It is also possible to hijack an utilized prefix. This type of hijacking almost certainly causes network outage to the legitimate user of the IP space. These hijacking attacks, either malicious or unintentional, should be prevented by filtering on the upstream input and by peering filters.

Despite the current practice that AS operators use many different side channels (e-mail, written contracts, etc.) to distribute information about the announcements, the primary source for generating the BGP filters should be IRR.

The `route` and `route6` objects contain the most important information that can prevent BGP hijacking attacks if it is used correctly.

Nonetheless automatic filtering seems not to work on a global scale despite the relative simplicity of the concept. By observing over 30 000 prefixes without valid `route` or `route6` objects in RIPE NCC service region we can conclude that input and peering filters are neither correctly generated from IRR nor used at all.

Since a huge number of prefixes work without the proper origin records, their originators feel no need to care about their routing policy, and more specifically about validity of their announcements with respect to `route` and `route6` objects.

The time series shows that the number of failures is steady and actually slightly declining in IPv4. In IPv6 the situation is a bit worse: Time series shows that the number of failures is increasing, but fortunately the increase is slower than the growth of the correct prefixes. According to the table A.7, the total number of errors in IPv6 is not too high. Moreover, in IPv6 keeping of the records should be straight-forward and simple, because even the biggest networks usually need no more than a single IPv6 prefix.

A.5 Path validation

A.5.1 Method

Path validation is the most complex and novel part of this work. Unlike the basic data analysis and origin matching that has been done before by other researchers or by IXP operators, I am not aware of any other recent attempt to validate paths in the entire RIR region.

Procedure outline

The basic idea is to take each path vector in DFZ and traverse the entire path from the originator up to the observation point. At the start of the path we have to check the origin. The question is what to do with a path that fails the origin validation test in the beginning? To resolve this for sake of the measurement simplicity, we can assume that the origin validation error is only another failure mode in the path verification.

For each AS, apart from the first and the last one in the `AS_path`, we can lookup and verify conformance of each prefix that traverses the AS with its routing policy. The basic source of the routing policy specification for the AS is the corresponding `aut-num` object.

The object should have `import` and `default` filters that contain a selector for the neighboring AS that have announced the path vector in question. And the AS should allow the path vector to its routing table only if the import filter matches.

The opposite direction works in the same manner: There should be `export` filters in the `aut-num` object of the AS. Each export filter contains a selector and a filter that allows the corresponding path vectors to be announced to the neighbors listed by the selector.

Filter evaluation

The selector might be directly an autonomous system number of the neighbor, which is fortunately the most common case. Otherwise the selector can contain selector expressions that can contain ASNs, lists of ASNs, references to `-set` objects and operators. The expression might require recursive resolution.

In order to evaluate the path vector imported into the AS we have to search through all selectors and subsequently all the corresponding filters in the `import` lines of its `aut-num` object.

We can stop searching only when we find a selector and corresponding filter that accepts the path vector, or when we reach the end of the `aut-num` object. The exhaustive search is necessary because finding a selector with a filter that rejects the prefix does not mean that there is not another selector that matches the neighboring AS as well. The corresponding filter for the later selector could accept the path vector in question. Therefore, we need to potentially examine each `import` and `default` line in the `aut-num` object and attempt to match filters of the lines that match in the selectors.

The same procedure can be applied to `export` filters.

Asymmetrical filter evaluation

Evaluation of the path vectors with respect to IRR filters is asymmetric in a certain aspect: We can only find validation errors in path vectors that are visible at the observation point in BGP and violate routing policy at some point in the `AS_path`. It means that routing policy effectively does not allow propagation of the particular path vector, but the vector propagates anyway.

On the contrary, we can not observe the opposite situation: Prefix propagation might be suppressed by BGP configuration even though the prefix should be allowed to propagate according to the routing policy.

In theory, we could build a graph of all relations among autonomous systems in the Internet according to routing policies. Then we could compute the best path matrix in this enormous graph and then compare real path for each prefix in BGP with the theoretically devised best paths for them.

There are obvious technical problems with that *thought* experiment and it is beyond the scope of this thesis.

Limitation on the best paths

Another drawback is that, given the validation computational complexity and finite resources and time, we have managed to validate only the best paths in the BGP table of the observation point.

It would be easy to switch on evaluation of the alternate paths in **bgpcrunch** software. Increase of the available paths would be three to five times the count of the best paths, depending on the upstream configuration that was active to the measurement date.

Error types

There are many possible failure modes in this path validation experiment. The basic outline has been formally specified in chapter 4. The table A.8 shows possible results from validation of one hop (one transition from one AS to another in the `AS_path`). Both the `import` and `export` filters are taken into consideration.

There is a special case for transit autonomous systems that do not operate within RIPE NCC service region and in spite of that decided to create `aut-num` object in RIPE DB for registration and authorization.

These objects for foreign entities might have also been generated automatically. It means that the objects have been left with virtually no useful information. Strictly speaking, traversing such an object is a routing policy failure but we conjecture that it is less severe type of a failure. Therefore, we define a special failure type that is subsequently counted to missing `aut-num` object in RIPE DB rather than to the counter for *missing filter* error.

Checking the hop with corresponding `aut-num` object consists of:

1. Considering the proper `import` and `default` lines one by one, if some exist, to find the previous AS in the `AS_path` and verify the corresponding filters until the first match is found or the end of the object is reached.

2. Taking proper `export` lines (if some exist) for the next AS in the `AS_path` and verifying corresponding filters until the first match is found or the end of the object is reached.

AS	`aut-num`	filter	result
RIPE	present	match	OK
RIPE	present	not-match	fltr fail
RIPE	missing	N/A	fltr not found
non-RIPE	present	match	OK
non-RIPE	present	not-match	fltr fail
non-RIPE	dummy	not-match	unknown
non-RIPE	missing	N/A	unknown

Table A.8: Path validation states

`import` or `default`	`export`	result
match	match	OK
no-match	match	import fltr fail
match	no-match	export fltr fail
no-match	no-match	import fltr fail
N/A, originating AS	match	OK
N/A, originating AS	no-match	export fltr fail
match	N/A, last AS	OK
no-match	N/A, last AS	import fltr fail

Table A.9: Hop validation states

There are obviously special cases: The first and the last autonomous systems in the `AS_path`. In these cases we can check only the available portion of the filter: The `export` for the right-most AS and `import` for the left-most one.

The possible results of the entire *hop validation* process are listed in table A.9. Please note that the result for the case when both `import` and `export` filters fail is actually "*import filter failure*". This result is selected because the prefix should not have been imported in the first place. Obviously, the export filter is also violated in this case. Nonetheless, due to software internal limitation we can only have one exit status from the validation procedure.

The reason for categorizing various failure modes is that we have many of them at different points of `AS_paths`. Therefore, we have to distinguish errors in order to learn about structure, dynamics and trends in failures relative to time. Thus it is not enough to mark any path that contains at least one error as "invalid". On the contrary, we count the errors by their type and by their position in the `AS_path`, relative to the observation point, and record many metrics that help to describe the errors.

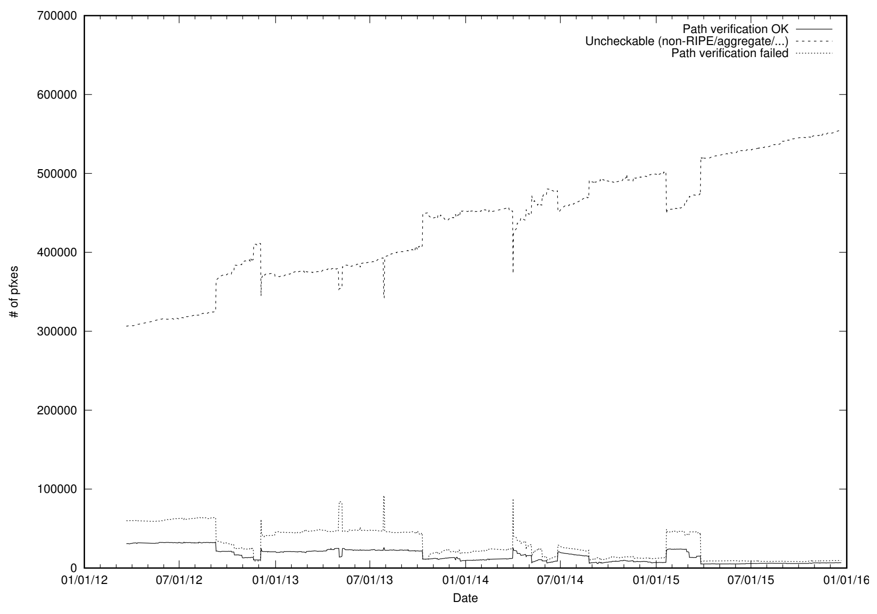

Figure A.33: IPv4 BGP path validation time series

A.5.2 Results

IPv4 overview

The figure A.33 shows the summary time series of path validation results for IPv4 through the entire captured period. From the chart we can see that judging the result based on one single filter failure in the `AS_path` is not sufficient. The fluctuations are heavily dependent on network paths that the prefixes take from the originator to the observation point at the time of the BGP table snapshot creation.

It changes rapidly over time and we can see that the changes correlate with the major fluctuations of the average BGP path length presented in figure A.12, that depend on local changes in the BGP setup, which have been already discussed.

It is obvious that the summary time series is too coarse and too dependent on the observation point location and network connectivity, that we can not judge on trends or global tendencies based on this fact only.

The first possible approach to obtain more information is to decompose the validation results based on detailed verification state. Unfortunately, that is still not enough to see how many errors are in the routing policies when we compare them to the actual BGP state.

The figure A.34 shows error count time series for each validation failure type that occurred during the entire validation process. It means that each traversed AS in `AS_path` that failed to validate counts once and results for all path vectors have been summed up.

To interpret this validation error time series chart we have to compare it to the number of IPv4 prefixes in DFZ from the figure A.2 and to the average path length that has been shown in the figure A.3.

It is also promising approach to compute combined time series from the previously mentioned parameters. The figure A.35 shows the time series of average path length and compare it with the average number of errors per path. It shows

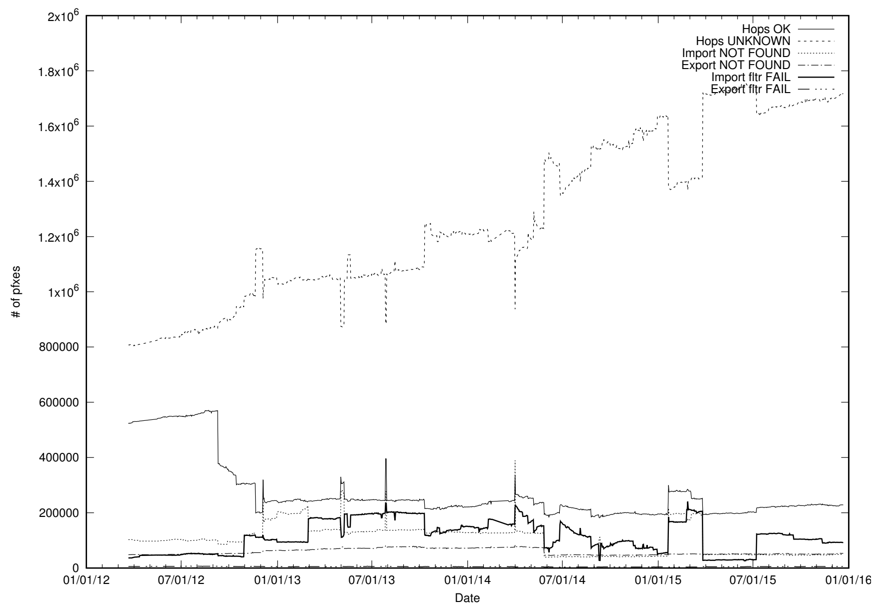

Figure A.34: IPv4 BGP path validation details

that in average case each path has 3.5 to 4 hops in `AS_path`. From that 4 hops 0.5 to 1 hop is invalid.

The detailed results for each prefix in DFZ and for each day within the captured period are published in text form on web site mentioned above in section **Daily IPv4 BGP path matching report (text)**. The summary for June 21, 2015 follows:

```
Total prefixes: 543327
Path verification OK: 5234
Uncheckable (non-RIPE/aggregate/...): 529193
Path verification failed: 8900

Total hops observed: 2083177
Total hops valid: 199971
Total hops unknown (non-RIPE/aggregate/filter-unknown): 1748076
Total hops unknown due to filter syntax error (included in unknown): 90
Total import filter not-found: 49342
Total import filter invalid: 29628
Total export filter not-found: 50933
Total export filter invalid: 5227
```

This following text output shows decomposed points of the chart A.35 for June 21, 2015:

```
Avg path length: 3.83
Avg unknown per path: 3.22
Avg unknown per path: 0.25
```

IPv4 hop validation

The number of the decidable paths in the end of the previous section is low. Both the correctly verified and the failed paths are a minority compared to undecidable

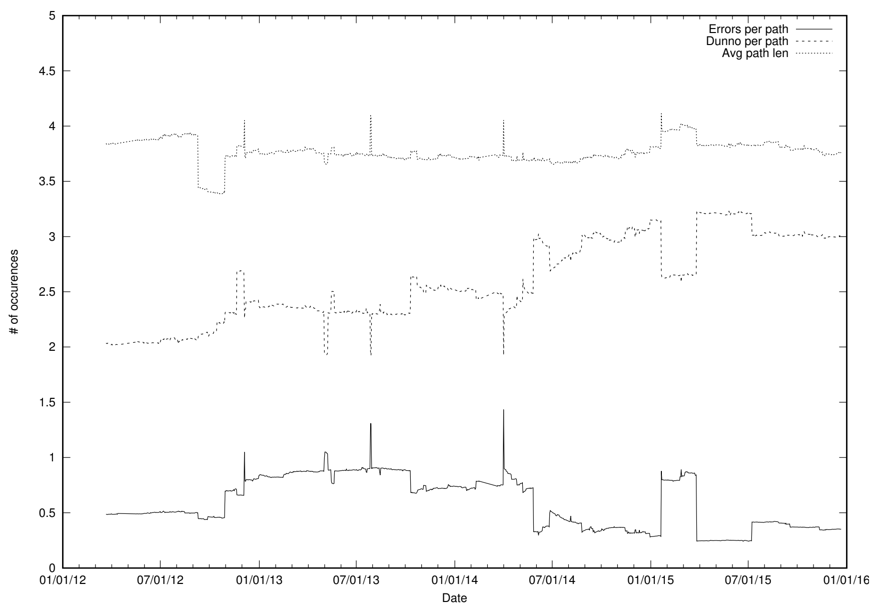

Figure A.35: IPv4 BGP validation errors per path

ones.

The reason is that the **AS29134** used upstream connection to the **AS6939** to the date of capture. And this AS is an international carrier that is based in the USA. Furthermore, the AS operates within ARIN service region and therefore the `aut-num` object for the main upstream provider of the observation point went out of our scope.

The paths that can be completely verified to the mentioned date consist only of the ASes within Europe, which can be reached form **AS29134** via peerings.

It is still possible to create a meaningful analysis of the data despite the fact that we do not have information regarding the first hop in most paths. We can focus on individual *hops* and thus ignore only the first undecidable *hop* in paths that traverse the upstream.

The numbers of observed prefixes and sizes of verification status groups relative to the distance from the observation point are displayed by the following output for June 21, 2015:

```
Hop 0 : 543324 pfx traversed, 8841 ok, 6554 errors, 527929 unknown
Hop 1 : 540934 pfx traversed, 60375 ok, 74560 errors, 405999 unknown
Hop 2 : 450513 pfx traversed, 53877 ok, 33922 errors, 362714 unknown
Hop 3 : 259788 pfx traversed, 30320 ok, 13476 errors, 215992 unknown
Hop 4 : 116076 pfx traversed, 17952 ok, 3781 errors, 94343 unknown
Hop 5 : 59055 pfx traversed, 9718 ok, 858 errors, 48479 unknown
Hop 6 : 36046 pfx traversed, 6255 ok, 422 errors, 29369 unknown
Hop 7 : 23248 pfx traversed, 3776 ok, 316 errors, 19156 unknown
Hop 8 : 16240 pfx traversed, 2821 ok, 218 errors, 13201 unknown
Hop 9 : 10421 pfx traversed, 1892 ok, 161 errors, 8368 unknown
Hop 10 : 6765 pfx traversed, 1333 ok, 206 errors, 5226 unknown
Hop 11 : 5281 pfx traversed, 942 ok, 181 errors, 4158 unknown
Hop 12 : 3971 pfx traversed, 516 ok, 191 errors, 3264 unknown
Hop 13 : 3225 pfx traversed, 405 ok, 200 errors, 2620 unknown
Hop 14 : 2422 pfx traversed, 294 ok, 57 errors, 2071 unknown
Hop 15 : 1945 pfx traversed, 117 ok, 27 errors, 1801 unknown
Hop 16 : 735 pfx traversed, 73 ok, 0 errors, 662 unknown
Hop 17 : 688 pfx traversed, 60 ok, 0 errors, 628 unknown
Hop 18 : 596 pfx traversed, 49 ok, 0 errors, 547 unknown
Hop 19 : 227 pfx traversed, 25 ok, 0 errors, 202 unknown
Hop 20 : 186 pfx traversed, 24 ok, 0 errors, 162 unknown
Hop 21 : 134 pfx traversed, 23 ok, 0 errors, 111 unknown
Hop 22 : 116 pfx traversed, 22 ok, 0 errors, 94 unknown
Hop 23 : 108 pfx traversed, 20 ok, 0 errors, 88 unknown
Hop 24 : 101 pfx traversed, 20 ok, 0 errors, 81 unknown
Hop 25 : 100 pfx traversed, 19 ok, 0 errors, 81 unknown
Hop 26 : 97 pfx traversed, 19 ok, 0 errors, 78 unknown
Hop 27 : 96 pfx traversed, 18 ok, 0 errors, 78 unknown
Hop 28 : 96 pfx traversed, 18 ok, 0 errors, 78 unknown
Hop 29 : 77 pfx traversed, 18 ok, 0 errors, 59 unknown
Hop 30 : 49 pfx traversed, 18 ok, 0 errors, 31 unknown
Hop 31 : 49 pfx traversed, 18 ok, 0 errors, 31 unknown
Hop 32 : 49 pfx traversed, 18 ok, 0 errors, 31 unknown
Hop 33 : 48 pfx traversed, 18 ok, 0 errors, 30 unknown
Hop 34 : 25 pfx traversed, 3 ok, 0 errors, 22 unknown
Hop 35 : 25 pfx traversed, 3 ok, 0 errors, 22 unknown
Hop 36 : 25 pfx traversed, 3 ok, 0 errors, 22 unknown
Hop 37 : 25 pfx traversed, 3 ok, 0 errors, 22 unknown
Hop 38 : 25 pfx traversed, 3 ok, 0 errors, 22 unknown
Hop 39 : 25 pfx traversed, 3 ok, 0 errors, 22 unknown
Hop 40 : 24 pfx traversed, 3 ok, 0 errors, 21 unknown
Hop 41 : 24 pfx traversed, 3 ok, 0 errors, 21 unknown
Hop 42 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 43 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 44 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 45 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 46 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 47 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 48 : 23 pfx traversed, 3 ok, 0 errors, 20 unknown
Hop 49 : 3 pfx traversed, 3 ok, 0 errors, 0 unknown
Hop 50 : 3 pfx traversed, 3 ok, 0 errors, 0 unknown
Hop 51 : 3 pfx traversed, 3 ok, 0 errors, 0 unknown
Hop 52 : 3 pfx traversed, 3 ok, 0 errors, 0 unknown
```

The first hop is for the vast majority of prefixes unfortunately the upstream AS, which actually breaks the validation of the entire path. However, the rest of the paths shows more reasonable numbers of validated as well as failed prefixes and less `unknown` results. The most important results are the first four *hops*. The second and the third *hops* exhibit less `unknown` results because for the majority of European path vectors the path went out of scope at the first *hop*, and it returns to the autonomous systems operated within RIPE NCC service region in a subsequent *hop*, and therefore these hops are decidable.

The presented text outputs can be visualized to make the point clear. The

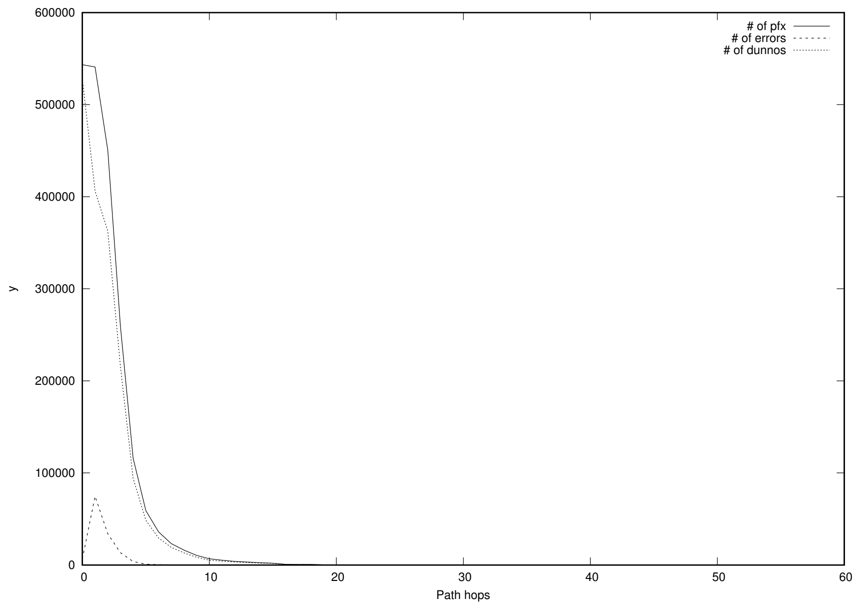

Figure A.36: IPv4 BGP filter matching along the paths

	2012-03-22	2014-03-20	2015-06-21
Valid	524723	243988	199971
Unknown	807522	1227030	1748076
`import` not-found	102792	126817	49342
`import` invalid	36181	156308	29628
`export` not-found	47749	75866	50933
`export` invalid	5543	5195	5227
Total hops	1524510	1835204	2083177

Table A.10: IPv4 Path AS transition validation results

figure A.36 shows the number of errors in relation to the number of prefixes and the number of undecidable hops along the path lengths on June 21, 2015.

The chart shows that most of the path validation failures happens within first five *hops* and peak seems to be in the second hop in our case.

We can analyze the structure of prefixes relative to path validation failures further and see whether there is any observable pattern. The figure A.37 shows the number of valid *hops*, failed *hops* and undecidable *hops* for all possible prefix lengths on June 21, 2015.

The table A.10 summarizes hop failure observations presented in this chapter. It shows the three different points in time when the observation point utilized different upstream providers. At the beginning, the upstream was leading European ISP. In the middle of the period, the routing table consisted mostly of an European low cost and the USA-based low cost providers. In the end, there is only one low-cost provider based in the USA, while other providers are kept in backup mode.

It is clearly visible from the table and from the time series that the validation errors are a persistent problem. But it actually matters the most what upstream provider is used because some have more documented paths and some do not

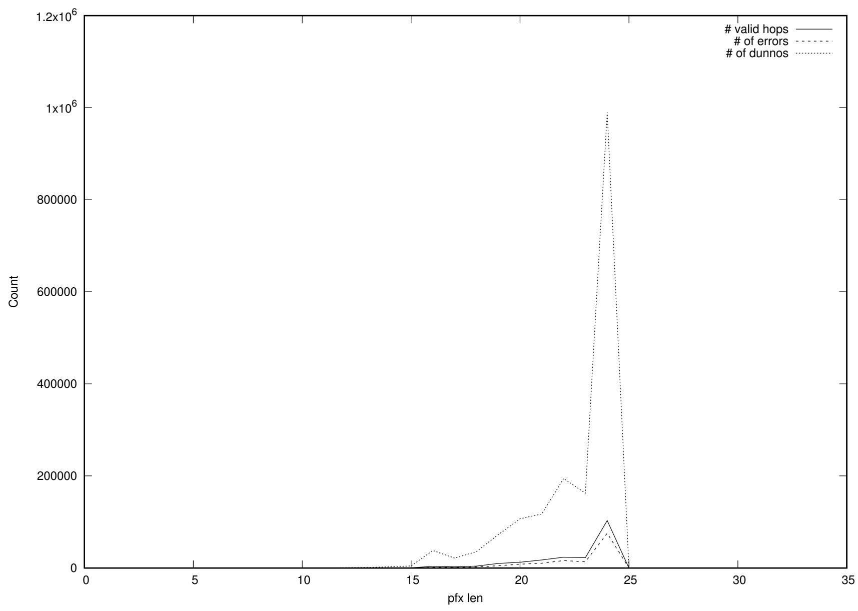

Figure A.37: IPv4 BGP filter matching results per prefix length

	2012-03-22	2014-03-20	2015-06-21
Valid	31063	11699	5234
Filter fail	59756	22799	8900
Unknown	306015	455926	529193
Total prefixes	396834	490424	543327

Table A.11: IPv4 Path validation results

actually bother with record keeping, which dramatically affects our results.

The table A.11 shows the summary of the entire path validation results in the three selected times.

Unfortunately, the presented numbers prevent us from concluding on global trends. The observation clearly shows that over the time the changes have made the validation status worse in the **AS29134** that hosted the observation point, even though it is a decent and respectable Czech ISP. It also shows that despite the high number of undecidable paths and hops, we have seen that IPv4 DFZ contains a high number of prefixes that cause errors of various kinds. This state is unfortunately consistent and long-term issue.

This section discussed the results briefly and only to the extent needed to support hypotheses and conclusions presented in main part of the thesis. Web site mentioned above contains the rest of the data and results that provide far more evidence and information about the Internet routing system.

IPv6 overview

In the following section on IPv6, there will be the same type of data, tables and visualizations from the same point of view and at the same points in time as for the IPv4. It will allow easy comparison of IPv4 and IPv6 routing in the Internet.

The figure A.38 shows the path validation summary time series for IPv6. It

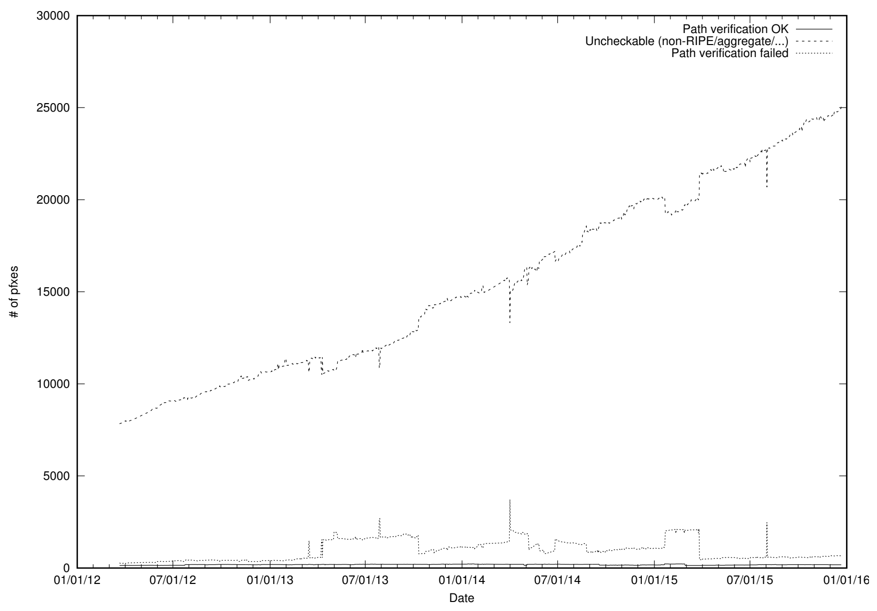

Figure A.38: IPv6 BGP path validation time series

covers the entire captured period and the same observation as for IPv4 holds: The chart is too coarse to judge any trends or global tendencies from it. We can also refer to the figure A.13 to see correlation with the IPv6 average path length time series.

Validation results, in the more detailed figure A.39, show how many errors of each type occurred during the entire validation process. It works in the same manner as in the IPv4 case. To interpret this validation error time series chart, we have to compare it to the number of IPv6 prefixes in DFZ from the figure A.7 and to the average path length that has been shown in the figure A.11.

The figure A.40 shows the time series of average path length and it allows comparison with the time series of average number of errors per path and average number of correctly validated hops per path.

Again, the numbers for IPv6 are worse than the numbers for IPv4. The reason might be that many people still consider IPv6 to be in “beta-testing” phase and do not care much about correctness of their routing policies.

IPv6 hop validation

The detailed results for each prefix in DFZ and for each day within the captured period are in text form on above mentioned web site in section **Daily IPv6 BGP path matching report (text)**. The summary report for June 21, 2015 follows:

```
Total prefixes: 22668
Path verification OK: 159
Uncheckable (non-RIPE/aggregate/...): 21984
Path verification failed: 525

Total hops observed: 72978
Total hops valid: 3444
Total hops unknown (non-RIPE/aggregate/filter-unknowm): 58561
```

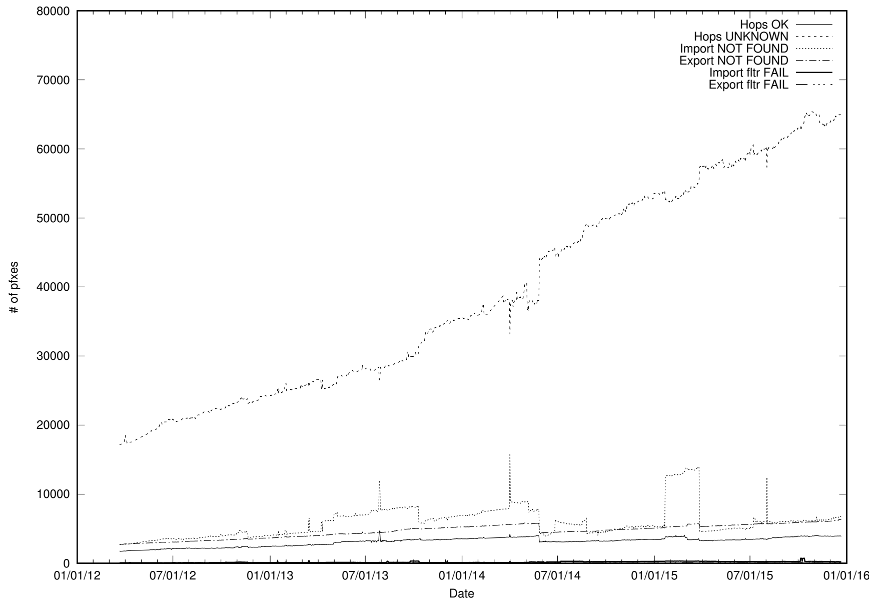

Figure A.39: IPv6 BGP path validation details

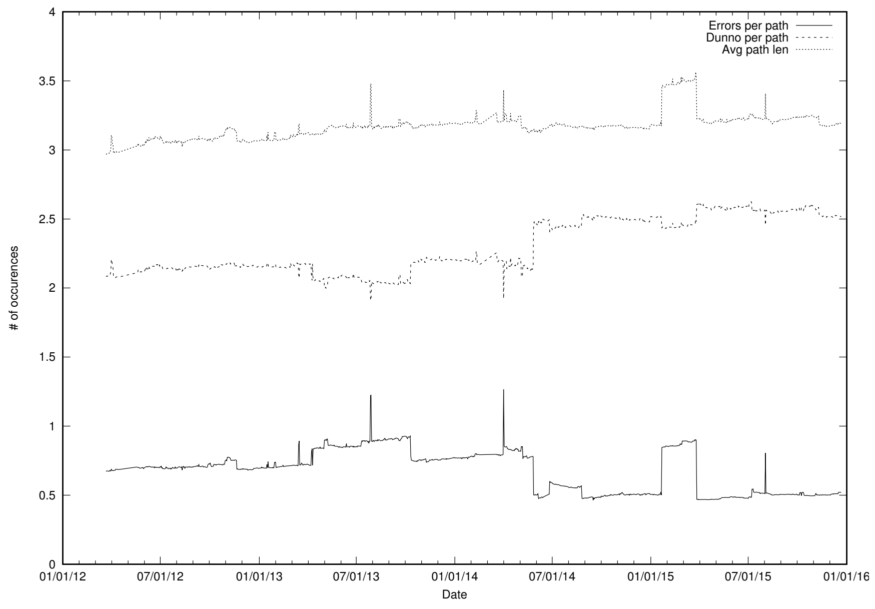

Figure A.40: IPv6 BGP validation errors per path

```
Total hops unknown due to filter syntax error (included in unknown): 5
Total import filter not-found: 5053
Total import filter invalid: 233
Total export filter not-found: 5605
Total export filter invalid: 82
```

Please note the low number of the decidable paths. The discussion and reasoning about these numbers is exactly the same as in IPv4 case. The problem lies in the upstream provider based in the USA and operating under ARIN rules.

Interesting information can be found in the following outputs from the same day. The numbers of observed prefixes and verification status relative to the length of the path from the observation point to the originator are displayed in the following output:

```
Hop 0 : 22666 pfx traversed, 276 ok, 1057 errors, 21333 unknown
Hop 1 : 22436 pfx traversed, 1185 ok, 5518 errors, 15733 unknown
Hop 2 : 16165 pfx traversed, 854 ok, 3353 errors, 11958 unknown
Hop 3 : 6958 pfx traversed, 616 ok, 837 errors, 5505 unknown
Hop 4 : 2306 pfx traversed, 214 ok, 146 errors, 1946 unknown
Hop 5 : 1131 pfx traversed, 137 ok, 32 errors, 962 unknown
Hop 6 : 433 pfx traversed, 55 ok, 16 errors, 362 unknown
Hop 7 : 292 pfx traversed, 34 ok, 4 errors, 254 unknown
Hop 8 : 193 pfx traversed, 26 ok, 3 errors, 164 unknown
Hop 9 : 145 pfx traversed, 20 ok, 2 errors, 123 unknown
Hop 10 : 75 pfx traversed, 12 ok, 2 errors, 61 unknown
Hop 11 : 45 pfx traversed, 6 ok, 0 errors, 39 unknown
Hop 12 : 37 pfx traversed, 2 ok, 0 errors, 35 unknown
Hop 13 : 35 pfx traversed, 1 ok, 0 errors, 34 unknown
Hop 14 : 26 pfx traversed, 1 ok, 0 errors, 25 unknown
Hop 15 : 8 pfx traversed, 1 ok, 0 errors, 7 unknown
Hop 16 : 7 pfx traversed, 1 ok, 0 errors, 6 unknown
Hop 17 : 7 pfx traversed, 1 ok, 0 errors, 6 unknown
Hop 18 : 7 pfx traversed, 1 ok, 0 errors, 6 unknown
Hop 19 : 3 pfx traversed, 1 ok, 0 errors, 2 unknown
Hop 20 : 1 pfx traversed, 0 ok, 1 errors, 0 unknown
Hop 21 : 1 pfx traversed, 0 ok, 1 errors, 0 unknown
Hop 22 : 1 pfx traversed, 0 ok, 1 errors, 0 unknown
```

This output exhibits much less favorable numbers than the IPv4 case. Though it might be a temporary issue, because we are still in the middle of IPv6 transition period. We can only hope that people are going to fix the filters and document their connection to IPv6 DFZ in the future.

The last portion of the summary text output is actually the right most point of the chart A.40:

```
Avg path length: 3.22
Avg unknown per path: 2.58
Avg errors per path: 0.48
```

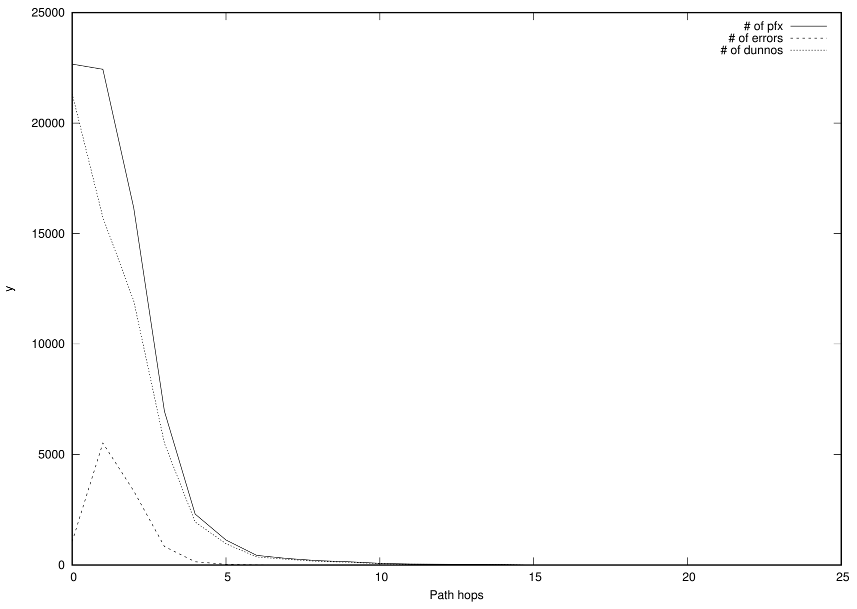

Figure A.41: IPv6 BGP filter matching along the paths

	2012-03-22	2014-03-20	2015-06-21
Valid	1770	3728	3444
Unknown	17184	38684	58566
`import` not-found	2687	7849	5053
`import` invalid	70	108	233
`export` not-found	2748	5568	5605
`export` invalid	38	111	82
Total hops	24492	56042	72978

Table A.12: IPv6 Path AS transition validation results

Visualization of the same type as in IPv4 case follows. The figure A.41 shows the number of errors in relation to number of prefixes and number of undecidable hops along the path lengths on June 21, 2015.

The figure A.42 shows the number of valid hops, failed hops and undecidable hops for all possible prefix lengths on June 21, 2015.

The table A.12 summarizes hop observations in this chapter. It shows the three different points in the same as in IPv4 case in previous section.

The table A.13 shows the summary of the entire path validation results in the same selected time points.

Conclusion

The presented numbers show the IPv6 Internet, still in its infancy, from three different points of view and in three different points in time to greater detail. The different points of view are based on three different upstream mixes that were used by the observation point.

The high number of severe failures in the results is alarming. It might still be the result of the low popularity of IPv6 among people and by perceivedness

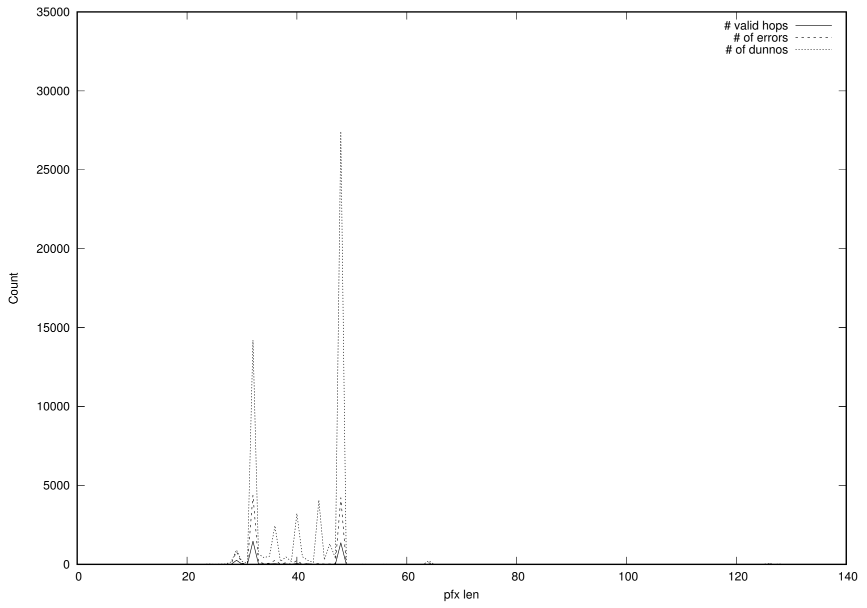

Figure A.42: IPv6 BGP filter matching results per prefix length

	2012-03-22	2014-03-20	2015-06-21
Valid	139	203	159
Filter fail	282	1365	525
Unknown	7825	15573	21984
Total prefixes	8246	17141	22668

Table A.13: IPv6 Path validation results

of IPv6 as a "*beta-testing*" technology.

A.6 Further results

At the end of this chapter I have to emphasize that we discussed only a small portion of the results. We have focused on the subset that is sufficient to prove that the routing policies in RIPE DB are inaccurate and incomplete to the point where their overall value becomes questionable.

The complete results, published on the web site `http://aule.elfove.cz/~brill/bgpcrunch`, contain over 7000 charts and several gigabytes of text outputs.

The discussion of the results in this chapter should, in my opinion, suffice to help orientate in the results. For further analysis of the BGP and IRR data, we can only refer to the raw results and to the **bgpcrunch** software at this point.

B. Contents of attached disc

B.1 Overview

The attached disc contains:

- Electronic version of the thesis,
- snapshot of the **bgpcrunch** software,
- programmer's manual and API documentation of the **bgpcrunch** software,
- selected subset of the primary results.

B.2 CD manifest

The listing of the CD contents follows:

- `README` - CD manifest
- `thesis.pdf` - The full (electronic) version of the thesis
- `bgpcrunch/` - The snapshot of the software for conducting the data anlysis; Please see chapters 4, 5 and appendix A of the thesis for details. The current (possibly newer) version could be found on GitHub: `https://github.com/tmshlvck/bgpcrunch`
- `bgpcrunch-doc.pdf` - Programmer's manual and API documentation of the bgpcrunch software.
- `results/`
  - `daily/`
    - `YYYY-MM-DD/` (date of the analyzed day)
      - `marge-pathlen4.png` - Daily IPv4 BGP average path length by prefix length
      - `marge-pathlen4.txt` - Daily IPv4 BGP path length report by prefix length
      - `marge-pathlen6.png` - Daily IPv6 BGP average path length by prefix length
      - `marge-pathlen6.txt` - Daily IPv6 BGP path length report by prefix length
      - `rirstats4-marge.txt` - Daily IPv4 BGP RIR share report
      - `rirstats6-marge.txt` - Daily IPv6 BGP RIR share report
      - `bgp2paths.png` - IPv4 BGP full paths matched against RIPE DB
      - `bgp2paths6.png` - IPv6 BGP full paths matched against RIPE DB

        - `bgppathbypfxlen4.png` - IPv4 BGP paths matched against RIPE DB by prefix length
        - `bgppathbypfxlen6.png` - IPv4 BGP paths matched against RIPE DB by prefix length
- `global/`
    - `route_violations_timeline/` - Route violations timeline dir (IPv4)
        - `aaa.bbb.ccc.ddd-nn` - Analysis of the violating IP prefix + netmask
    - `route6_violations_timeline/` - Route violations timeline dir (IPv6)
        - `aaaa:...:zzzz-nn` - Analysis of the violating IP prefix + netmask
    - `pfxcount4-N.png` - IPv4 BGP time series of prefix count of prefix length (N)
    - `pfxcount6-N.png` - IPv6 BGP time series of prefix count of prefix length (N)
    - `pfxcount4-avgpfxlen.png` - Average prefix length time series (IPv4)
    - `pfxcount6-avgpfxlen.png` - Average prefix length time series (IPv6)
    - `pfxcount4-sum.png` - IPv4 prefixes in BGP
    - `pfxcount6-sum.png` - IPv6 prefixes in BGP
    - `rirpfxcount4-marge.png` - IPv4 BGP prefix count per RIR
    - `rirpfxcount6-marge.png` - IPv6 BGP prefix count per RIR
    - `rirpfxlen4-marge.png` - IPv4 BGP average prefix length per RIR
    - `rirpfxlen6-marge.png` - IPv6 BGP average prefix length per RIR
    - `pathlen4-N.png` - IPv4 BGP time series of path length of prefix length (N)
    - `pathlen6-N.png` - IPv6 BGP time series of path length of prefix length (N)
    - `pathlen4-avg.png` - IPv4 BGP average path length
    - `pathlen6-avg.png` - IPv6 BGP average path length
    - `pathlen4-3d.png` - IPv4 BGP path length per prefix length 3D time series
    - `pathlen6-3d.png` - IPv6 BGP path length per prefix length 3D time series
    - `bgp2routes4.png` - IPv4 BGP origin verification results
    - `bgp2routes6.png` - IPv6 BGP origin verification results
    - `bgp2paths4.png` - Path verification results (IPv4)
    - `bgp2paths6.png` - Path verification results (IPv6)
    - `bgp2paths-detail4.png` - Path verification details (IPv4)
    - `bgp2paths-detail6.png` - Path verification details (IPv6)

* `bgp2paths-stats4.png` - Path verification errors per path (IPv4)
* `bgp2paths-stats6.png` - Path verification errors per path (IPv6)