A Hierarchical Model for BGP Routing Policies

Laurent Vanbever Bruno Quoitin Olivier Bonaventure

IP Networking Lab
Université catholique de Louvain
B-1348 Louvain-la-Neuve, Belgium

ABSTRACT

BGP routing policies are mainly used by network operators to enforce business relationships between Autonomous Systems (AS), and to prefer some routes over others. In this paper, we propose a hierarchical policy model to express each policy at the most appropriate level of abstraction. The model is structured around chains of filters that apply at a specific level of abstraction. To validate our approach, we implemented the model in a Java prototype and used it to reproduce several network-wide routing policies. In all studied networks, the model produced a high-level view of routing policies while preserving their semantics. Our model offers numerous advantages to network operators including, but not limited to, a better network documentation, an improved understanding of routing policies, redundancy suppression, and an easier way of managing their network.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

General Terms

Design, Management

Keywords

BGP, routing policies, network configuration

1. INTRODUCTION

The Border Gateway Protocol (BGP) is one of the key protocols in today's Internet. BGP is a path-vector protocol that allows to distribute reachability information between and inside Autonomous Systems (AS). A key feature of BGP is that it allows network operators to define routing policies. Classical routing policies include provider-customer (paid peering) and shared-cost (settlement-free peering) peerings [5]. During the last ten years, BGP implementations have been extended to support additional routing policies required by the network operators. As a side effect, this greatly increased the difficulty of building and validating BGP routing policies. Several studies have shown that BGP policies often contain errors [15, 17]. Some of these errors are minors, but others have caused significant operational problems.

To build BGP policies, network operators go through a complex configuration process. Generally speaking, *configuring* a network is very similar, if not identical, to *programming* a distributed infrastructure. However, the network is configured by using low-level proprietary languages or APIs provided by the network vendor. Despite the improvements made in such APIs, they still fail to offer *network-wide* abstractions to network operators. As a consequence, network configuration remains a tedious and error-prone process [17]. Validation tools have been proposed to cope with misconfigurations [9, 22], but they are not frequently used. In addition, the documentation of the network configuration is often limited to a few configuration templates and a history of changes edited by hand. The lack of detailed and up-to-date documentation is a problem in many networks.

In this paper, we consider a different but safer approach. Instead of assuming that the routers are configured manually, we develop on top of the pre-existing APIs a hierarchical framework which allows network operators to express network-wide BGP policies in a declarative way, just as a software programmer would do. There are many advantages in using a high-level program to build the configurations of all routers in a network [20]. First, the program contains the entire documentation of the network. Second, a program can easily ensure that the design guidelines are met in all parts of the network. Third, the program can automatically adapt to different router vendors or different versions of the router's operating systems. Fourth, validation tests can be automatically performed by the program.

This paper is organized as follows. Section 2 summarizes the main applications of BGP policies. Section 3 describes the hierarchical model we propose to express network-wide BGP routing policies. Section 4 describes the utilization of our model on the Abilene network. The paper ends with a discussion of the related work.

2. BGP POLICIES

Currently, BGP configurations are expressed as commands for router configuration languages. The BGP configuration of each router usually contains several parts. First, the iBGP and eBGP sessions attached to the router are listed. The definition of each session includes the IP address and AS number of the neighbor router as well as the filters to be applied on the session. Two types of filters can be defined. Import filters specify which routes should be accepted, possibly after having changed some of the BGP at-

		Level of abstraction				
		All	Group of AS	AS	Session	Prefix
SANITY-IN	deny bogons	✓				
	deny own prefixes	✓				
	deny default route	✓	✓			
	remove own communities	✓				
	deny too specific prefixes,limit number of prefix		✓			
	filter un-assigned/un-allocated			✓		
	limit as-path length		✓	✓	✓	
IMPORT	accept/reject MED	✓	✓	✓		
	allow blackhole-communities	✓	✓	✓		
	set initial local-pref		✓	✓		
	allow TE-communities	✓	✓			
	informational tagging			✓	✓	
TE-IN	change local-pref (relative)		✓	✓	✓	✓
TRANSIT	business relationships		✓			
	exceptions			✓	✓	✓
TE-OUT	as-path prepending (relative)		✓	✓	✓	✓
	export MED=IGP cost		✓	✓		
	apply TE-communities			✓	✓	✓
EXPORT	allow MED		✓	✓		
SANITY-OUT	deny bogons	✓				
	remove own communities	✓				

Table 1: BGP routing filters and their levels of abstraction.

tributes. Export filters specify which routes can be advertised, possibly after modifying some of their BGP attributes. Several filters can be attached to each BGP session.

BGP routing filters are configured on each router by using low-level proprietary languages. The primary goal of these configuration languages is to pass the router all the parameters needed for its proper operation. These configuration languages are well suited to set up most of one router's configuration knobs. However, they are not appropriate for the expression of more complex statements such as BGP routing policies. As a consequence, network operators are often forced to express their routing filters on a per-session or per-prefix basis.

A BGP routing filter is typically composed of a sequence of rules. Each rule is a couple of one predicate and a series of actions. The predicate defines which criteria a route must satisfy in order for the actions to be applied on the route. Actions can be as simple as accepting or denying the route. They can also be used to change the attributes of the route.

The majority of BGP routing filters can be roughly classified in three main categories: enforcing **transit policies** according to the business relationships with neighbor AS, performing **traffic engineering** and **sanitizing the routes**. By studying reference books such as [7] as well as a large set of router configuration files obtained from several networks, we found that there are appropriate levels of abstraction for different categories of routing filters. We distinguish 5 main levels: the **prefix** level, the **session** level, the **AS** level, the **AS-group** level, and the **network** level. In the remaining of this section, we review the different categories of BGP routing filters and we indicate their appropriate levels of abstraction. We summarize our classification in Table 1.

2.1 Transit policies

Autonomous Systems need to control how routes are propagated between their neighbors according to their **business relationships**. There are two main business relationships [10]: *customer-provider* and *peer-to-peer*. The policy for advertising routes among customers, peers and providers is usually as follows. Customer routes are distributed to all neighbors. Peer and provider routes are propagated to customers only.

The **transit policies** are not limited to the above business relationships and more specific policies exist [18]. For example, it is common for one ISP to buy its connectivity from one provider and to have a peering link with another AS used as backup only [11]. Another example is the provision of partial transit [18].

Such transit policies can easily be abstracted by a square transit matrix where rows and columns are indexed by groups of AS that share the same business relationships. In the case of the typical customer-provider and peer-to-peer relationships, the following matrix fully describes the transit policies:

		to		
		customers	peers	providers
from	customers	✓	✓	✓
	peers	✓		
	providers	✓		

In real router configuration files, it is frequent to find filters that override or alter the behavior defined by the above high-level transit policies. These filters are often targeted at specific couples of neighbors or sessions and can even be applied on a prefix-basis. We call these filters **exceptions** to transit policies. To express such exceptions, it is not possible to rely on the above transit matrix as an higher level of granularity than groups of AS is needed.

2.2 Traffic Engineering

BGP provides the network operator with an arsenal of mechanisms to influence how BGP ranks and selects routes towards a specific destination. For example, **changing the local-preference** attribute can be used to force BGP to prefer one route over another in the local network. It is common to **set the initial local preference** of routes according to the business relationship with the announcer [5]. Such attribute modifications are often done **relatively** to a base attribute value. For example, a change of the local-preference will be done by adding or subtracting a delta to the previous local-preference value. Another example is using **AS-Path prepending**.

2.3 Sanitization

Every self-respecting BGP network operator configures his network with BGP routing filters that makes sure routes towards invalid prefixes or **bogons** are denied [8].

In addition to filtering invalid prefixes, it is recommended to filter **too specific prefixes** in order to limit the burden on BGP [3]. Such filters are usually applied on sessions with customers and peers. Careful operators can also deny their **own prefixes** from upstreams or peers. The **default route** (`0.0.0.0/0`) is usually denied from customers or peers who have no reason to announce it. A default route is sometimes accepted from upstream providers by networks that do not provide transit.

As part of the routes sanitization, filters are often deployed to prevent a neighbor from attaching your **own internal communities**, i.e. communities that belong to the local AS and can be used internally to implement transit policies or for informational purpose. The communities attribute being transitive, it is also considered a good practice to filter local community values from outgoing routes unless needed.

Finally, we found that some aspects of the BGP session configuration can be classified as part of the route sanitization task. Among them, **limiting the number of prefixes** received from a neighbor.

2.4 Other filters

An ISP might provide its neighbors with a mean to influence the processing of their routes within the ISP network. A typical example is to allow customers to prepend specific **TE-communities** to their routes [19]. These communities, when present, cause the assignment of predefined local-preferences to the routes. Another similar example is a community whose meaning is to request the ISP to **blackhole** a customer route's prefix, to prevent the customer's network to be overwhelmed with traffic generated by a DDoS attack [23]. Another common utilization of the communities is to tag routes for **informational or debugging** purposes.

3. HIERARCHICAL MODEL

As explained in Section 2, BGP routing filters are used for different purposes and have an appropriate level of expression. For this reason, we propose a high-level model of BGP routing policies where routing filters can be expressed at the most appropriate level. The model is then used to automatically derive routing filters expressed in vendor-dependent router configuration languages.

In our model, the network-wide BGP routing policies are expressed by using two chains of high-level routing filters. The *import policy chain* (resp. *export policy chain*) completely specifies the routing filters that must be applied when routes are received by (resp. sent to) an eBGP session. Each node in a policy chain is a sequence of **rules**, each rule containing a **predicate** and a **routing filter template**, i.e. a block of routing filter statements. Predicates specify the conditions one eBGP session must meet to allow the corresponding template to be included in the session's filter. Such predicates can test the attributes of a BGP session, e.g. the router-id, the remote ASN, the group of neighbors it belongs to, etc. The filter template contains routing filter statements such as setting the local-preference or denying a route.

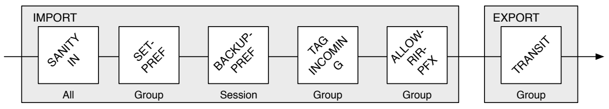

Figure 1: High-level model of BELNET routing policies.

The *import policy chain* (resp. *export policy chain*) is used to generate the import (resp. export) filters for eBGP session s. The algorithm's operation is similar for both chains, therefore we only describe how the import chain is processed. The algorithm starts with the first node and traverses the whole chain. For each node, it traverses the rules in sequence. For each rule, it checks if the predicate evaluates to true for s. If so, the filter template specified in the rule becomes part of s's routing filters.

3.1 Example

To illustrate our high-level model, we show an example inspired of BELNET, the Belgian national research and education network. We show in Figure 1 the *import* and *export policy chains* that globally describe BELNET's routing filters. The *import policy chain* is composed of five nodes, while the *export policy chain* is composed of a single node named TRANSIT. The different nodes are expressed using three different levels of abstraction: *All* when all sessions are concerned, *Group* when groups of sessions are used and *Session* when individual sessions are concerned.

Each node in the two policy chains is described in Table 2. The SANITY-IN node for example declares that on import, all sessions (the predicate is always true) must include a filter template that rejects all routes whose destination prefix belongs to a list of bogons. The ALLOW-RIR-PFX node declares that each session must include a filter that denies prefixes not allocated/assigned by the RIR to this AS. Finally, the TRANSIT node declares that the export filter of customer sessions must announce the default route while the export filter of peer and provider sessions must only announce routes tagged with the "customer" community or routes originated internally.

3.2 Implementation

Our prototype is implemented in Java and is freely available [1]. It is composed of three main components: a *data repository*, a *predicate matching engine*, and a *generation engine*.

The *data repository* contains the description of all BGP sessions as well as the import and export policy chains. In our prototype, the nodes in the policy chains are Java classes to which we attach routing filter templates written in vendor-specific configuration language. At the moment, our prototype generates configurations expressed in the JunOS language, but other configuration languages (*e.g.*, Cisco IOS, Alcatel TimOS, etc.) could be easily added. An example of a particular policy template is shown in Listing 1. This template expresses in JunOS a routing filter that subtract some value from one BGP route's local-preference attribute.

The *predicate matching engine* of our prototype relies on regular expressions matching. In the *data repository*, we encoded each BGP session as a concatenation of the textual representation of its attributes. For example, the backup BGP session with AS 24 defined on router LOSA and belonging to peer-group NREN is represented as "LOSA:NREN:24:FEDNET:207.231.246.3:backup". We can then use regular expressions to express simple predicates. For example the regular expression `.*backup.*` is our prototypical way to express the predicate $s.type = backup$.

The *generation engine* iterates over all BGP sessions. For each

	Predicate	Filter template
SANITY-IN	$true$	$(r.pfx \in BOGONS) \implies reject$
SET-PREF	$s.group = CUST$	$r.lp = 5000$
	$s.group = PEER$	$r.lp = 4000$
	$s.group = PROV$	$r.lp = 3000$
BACKUP-PREF	$s.type = backup$	$r.lp = r.lp - 500$
TAG-INCOMING	$s.group = CUST$	$r.comm \uplus \{CUST\}$
	$s.group = PEER$	$r.comm \uplus \{PEER\}$
	$s.group = PROV$	$r.comm \uplus \{PROV\}$
ALLOW-RIR-PFX	$s.group = CUST$	$(r.pfx \notin RIR\_PFX(s.asn)) \implies reject$
TRANSIT	$s.group = CUST$	$announce\ default\ route$
	$s.group = PEER \vee$ $s.group = PROV$	$((r.comm \ni CUST) \vee (r.pfx \in INTERNAL)) \implies accept$

Table 2: High-level network-wide description of BELNET's BGP routing policies.

```
policy-statement BACKUP-PREF {
  term down-pref {
    then {
      local-preference subtract $value$;
      accept;
    }
  }
}
```

Listing 1: Example of a JunOS backup template.

BGP session, it applies the import and export policy chains to generate the JunOS import and export filters assigned to the session. For each node of one policy chain, the *predicate matching engine* is used to identify whether one filter template applies. When a match is found, the corresponding template is associated to the session. Finally, the generation engine outputs, for each BGP session, all the associated templates. The result is thus a complete configuration expressed in the templates' configuration language.

4. CASE STUDY

In this section, we show how our hierarchical model can be applied to produce network-wide BGP policies and help operators deploy new BGP-based services. As an illustration, we reproduce a subset of Abilene BGP routing policies [14]. Abilene is the Internet2 backbone network interconnecting points of presence spreaded all-over the United States. It is composed of nine BGP routers in a fully-meshed iBGP configuration. Abilene is composed of more than 125 IPv4 eBGP sessions.

In addition to interconnecting universities, Abilene offers advanced services such as commercial peering, an international transit network, an US federal peer network, etc. Abilene's BGP policies are thus customized according to the type of neighbor. For example, routes learned from commercial peers will not be redistributed to international or federal peers. As a consequence, Abilene routing policies are more complex than the policies found in typical ISP networks [5].

Due to space limitations, we focus ourselves on modeling the export policies of Abilene which were found to be the most interesting. Indeed, most Abilene import policies are similar to the policies described in Section 3.

4.1 Analyzing Abilene policies

To understand the high-level objectives expressed by BGP routing policies, we built a *configuration analyzer tool*. This tool is able to identify patterns of policies shared through the network configuration and outputs semantically equivalent abstractions. The tool helped us to identify the abstraction level of each policy.

Listing 2 shows a typical output for two BGP sessions belonging to the same peer-group. Two trends are apparent. First, both sessions share two groups of policy statements: [**SANITY-OUT**, **REMOVE-COMMS-OUT**, **ORIGINATE4**] and [**FEDNET-OUT**]. We called these two groups of shared policies: *predecessor* and *successor*. Typically, *predecessor* policies are applied before every other policies while *successor* policies are applied after every others polices. Second, lower-level policies are encompassed inside these two groups (*e.g.*, *REDCLARA-TO-USGS*). This configuration pattern was often encountered in policies configurations (64 out of 125 policies configuration). In our hierarchical model, *predecessor* and *successor* policies are represented by high-level filters (*e.g.*, group-level) because they could be abstracted among sharing sessions. Between these high-level filters, lower-level filters (*e.g.*, AS-level) are added to represent more-specific policies.

In addition, our *configuration analyzer tool* could also be used to detect abstraction errors. For instance, when all but one session of a given peer-group share the same *predecessor* and *successor* groups, it is very likely that the remaining one is an error. Based on that observation, we detected several problems which were later confirmed as misconfigurations by Abilene's network operators.

```
...
<198.32.153.121 - AS#22284 (USGS)>
 EXPORT : [SANITY-OUT REMOVE-COMMS-OUT ORIGINATE4
           REDCLARA-TO-USGS FEDNET-OUT]
<198.32.11.81 - AS#24 (NREN)>
 EXPORT : [SANITY-OUT REMOVE-COMMS-OUT ORIGINATE4
           GEANT-TO-NREN CLARA-TO-NREN FEDNET-OUT]
...
```

Listing 2: Example output from the *configuration analyzer tool*.

4.2 Modeling Abilene's export policies

Based on our *configuration analyzer tool* and on manual analysis, we have been able to group policies by their level of abstraction and represent them as a chain of filters. Figure 2 depicts the export chain describing Abilene's export routing policies. The text at the

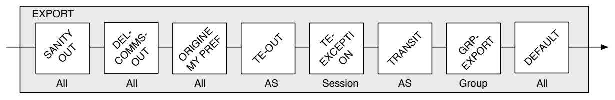

Figure 2: Excerpt of the filters chain representing Abilene's export policies.

bottom of each filter represents its abstraction level. The chain is composed of 8 nodes. Table 3 describes the most significant nodes.

The first three nodes —SANITY-OUT, DEL-COMMS-OUT, and ORIGINE MY PREF— apply on all sessions. They respectively sanitize announced routes, prevent local communities to leak out of the network, and announce Abilene's prefix to neighbors. The TE-OUT node declares traffic engineering policies. It is used to prepend Abilene's AS number to routes announced on the sessions with AMPATH, GEANT and ESNET. In a filter template, the notation $r.aspath[0]$ identifies the AS which originated route r. The TE-EXCEPTION node represents traffic engineering exceptions. For instance, we found that AS-path prepending is done on routes originated by AS AMPATH and redistributed to AS ESNET, but only on the Atlanta router (cf. predicate $s.rid = ATLA \wedge s.asn = ESNET$). The TRANSIT node represents a transit matrix. It declares which routes will be redistributed to each ASes. For instance, routes coming from AS ESNET will be redistributed on all the sessions Abilene has with ASes AMPATH and GEANT (see Table 3, Equations **3.1** and **3.2** respectively). The GRP-EXPORT node is a group-based filter that rejects routes based on the routes' communities. Finally, the DEFAULT node accepts all routes that weren't previously rejected.

4.3 Modeling complex routing policies

In this section, we give an example of scenario where our framework can help modeling complex routing policies. For instance, let's say a large ISP wants to offer a new service to its customers: national transit. Typically, the implementation process implies adding a new BGP import policy on *each* national customer sessions. This policy will *tag* every incoming routes with a given community (*e.g.*, NAT). Then, on the same sessions, BGP export policies have to be set to redistribute routes tagged with the aforementioned community. In this scenario, we suppose the default action for a policy is to reject a route.

Within our framework, this work could have been done much more easily. The only requirement is that each session must be flagged with a type to identify whether it is facing a national peer or not. Only two filters need to be added to the chain. The first filter is placed in the *import* chain. It matches sessions based on the predicate: $s.type = NATIONAL$. The filter's template simply sets the community accordingly: $r.comm \uplus \{NAT\}$. The second filter is an export one. It shares the same predicate and is associated with a template which accepts NAT tagged routes: $r.comm \ni NAT \implies accept$. Based on those two filters, our tool will iterate on each BGP session and apply routing policies accordingly.

Even if the result will be the same in both scenarios, our modeling offers numerous advantages. Among others, it gives a clear visualization of network policies. Routing policies are thus much more easier to understand, and to reason about. Moreover, it allows network operators to modify their policies easily and safely by reducing the required modifications to the minimum necessary. In this case, only two filters need to be added.

5. RELATED WORK

The Routing Policy Specification Language (RPSL) [2] was initially proposed as a language to both document and verify that there are no conflicts among routing policies from different ASes [13]. Some ASes, notably in Europe rely on RPSL to document their peering relationships and some use the RPSL information to automatically generate part of their filters. However, it does not seem that RPSL is still used to verify the coherence of the routing policies. This is mainly because some operators do not provide their policies in RPSL format. Furthermore, RPSL does not allow operators to easily express complex policies. For example, most operators explain their usage of BGP communities using free text comments instead of pure RPSL syntax. RPSL does not allow to easily express complex routing policies.

Several network operators have developed tools to automate the configuration of parts of their routing policies. Among others, Gottlieb et al. described in [12] tools to build the configurations of the eBGP sessions with customers in a large ISP network based on provisioning databases. No information is provided in [12] about the support of complex routing policies. Bohm et al. propose in [4] an XML-based configuration language that allows to express the network-wide routing policies in an ISP network. This XML configuration is then converted in RPSL format and the RPSL tools are used to generate the corresponding router configurations. Our hierarchical model provides a clearer way of expressing and documenting complex routing policies. By using string templates, we also generate the actual router configuration commands.

Levanti et al. proposed in [16] the NetPolis software that starts from the low-level router configuration files to detect and document the abstract routing policies used in a network. We used a method similar to the one proposed in [16] to reverse engineer the Abilene routing policies as explained in Section 4. NetPolis could thus serve as a first step to allow an operator to start using our approach.

6. FUTURE WORK

They are three main directions for future work. First, we need to extend the scope of our study to more networks. Actually, chains of filters were found to be the easiest construction allowing us to represent every polices we encountered. Though, highly complex BGP policies (*e.g.*, the ones using branching instructions or subroutines) might not be supported by this model. In that case, we could easily extend our model to use directed acyclic graphs instead of chains.

The second direction is to study how this framework could be used as a building block to more advanced services for network operators. In particular, we plan to study how such tool could help operators doing inter-domain traffic engineering. For instance, it could be used to generate BGP policies that divert traffic from a given link while minimizing data loss (*e.g.* by updating the BGP routing policies or active BGP sessions thanks to the Route Refresh capabilities [6]).

A third direction would be to build a network-wide management control-plane instead of basing ourselves on the low-level APIs provided by network vendors. This could address one of the biggest problem of ISPs today: deploy *rapidly* and *safely* new services [21].

7. CONCLUSION

In this paper, we have first analyzed the routing policies that are used in today's network. We have then proposed a hierarchical model to express complex BGP routing policies. Our model expresses the routing policies as chains. Each node in a chain is a sequence of rules that contain a predicate and a routing filter tem-

	Predicate	Filter template
TE-OUT	$s.asn = AMPATH$ $s.asn = GEANT$	$r.aspath[0] = ESNET \implies [11537] + r.aspath$
	$s.asn = ESNET$	$r.aspath[0] = GEANT \implies [11537] + r.aspath$
TE-EXCEPTION	$s.rid = ATLA \wedge s.asn = ESNET$	$r.aspath[0] = AMPATH \implies [11537] + r.aspath$
	$s.rid = CHIC \wedge s.asn = USGS$	$r.aspath[0] = REDCLARA \implies [11537] + r.aspath$
TRANSIT	$s.asn = AMPATH$	$r.aspath[0] = ESNET \implies accept$ **(3.1)**
	$s.asn = CLARA$	$r.aspath[0] = NREN \implies accept$
	$s.asn = CUDI$	$r.aspath[0] = NISN \implies accept$
	$s.asn = ESNET$	$(r.aspath[0] = AMPATH$ $\vee r.aspath[0] = NREN) \implies accept$
	$s.asn = GEANT$	$(r.aspath[0] = ESNET \vee r.aspath[0] = NREN$ **(3.2)** $\vee r.aspath[0] = USGS) \implies accept$
	$s.asn = NISN$	$r.aspath[0] = CUDI \implies accept$
	$s.asn = NREN$	$r.aspath[0] = GEANT \implies accept$
	$s.asn = REDCLARA$	$r.aspath[0] = USGS \implies accept$
GRP-EXPORT	$s.group = FEDNET$	$(r.comm \cap \{FEDNET, ITN, NITN\}) \neq \emptyset \implies reject$
	$s.group = ITN$	$(r.comm \cap \{FEDNET, NITN\}) \neq \emptyset \implies reject$
DEFAULT	$true$	$accept$

Table 3: Excerpt of the chain of filters modeling Abilene's export routing policies.

plate. Different nodes in the chain are applied at different levels of abstraction. We developed a prototype in Java and used it to completely reproduce the Abilene routing policies in JunOS. We also showed how network operators could use our framework to build their own routing policies.

8. ACKNOWLEDGMENTS

Laurent Vanbever is supported by a grant from FRIA (Fonds pour la Formation à la recherche dans l'Industrie et dans l'Agriculture, Belgium). Bruno Quoitin is funded by the SPINNET project. The authors would like to thank Bernard Lambeau and Virginie Van den Schrieck for their suggestions and comments on this work. We would also like to thank Dirk Haex from BELNET and Volodymyr Yakovenko from Google.

9. REFERENCES

[1] A hierarchical model for BGP routing policies. `http://inl.info.ucl.ac.be/softwares/hm-bgp`.

[2] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizar. Routing Policy Specification Language (RPSL). RFC 2280, Jan. 1998.

[3] S. Bellovin, R. Bush, T. Griffin, and J. Rexford. Slowing routing table growth by filtering based on address allocation policies. unpublished, June 2001.

[4] H. Bohm, A. Feldmann, O. Maennel, C. Reiser, and R. Volk. Network-wide inter-domain routing policies: Design and realization. NANOG 34, May 2005.

[5] M. Caesar and J. Rexford. BGP routing policies in ISP networks. *IEEE Network*, 19(6):5–11, 2005.

[6] E. Chen. Route Refresh Capability for BGP-4. RFC 2918 (Proposed Standard), Sept. 2000.

[7] G. Davies. *Designing and Developing Scalable IP Networks*. Wiley, September 2004.

[8] D. Deitrich. Bogons and bogon filtering. NANOG 33, Feb 2005.

[9] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *2nd Symp. on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[10] L. Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE Global Internet*, November 2000.

[11] L. Gao, T. Griffin, and J. Rexford. Inherently Safe Backup Routing with BGP. In *IEEE INFOCOM*, 2001.

[12] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang. Automated Provisioning of BGP Customers. *IEEE Network*, 17, 2003.

[13] R. Govindan, C. Alaettinoglu, K. Varadhan, and D. Estrin. An architecture for stable, analyzable internet routing. *IEEE Network Magazine*, 13(1):29–35, 1999.

[14] Internet2 Observatory. `http://www.internet2.edu/observatory/`.

[15] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb. Minerals: Using Data Mining to Detect Router Misconfigurations. In *MineNet '06*, pages 293–298, 2006.

[16] K. Levanti, H. Kim, and T. Wong. Netpolis : Modeling of inter-domain routing policies. In *IEEE GLOBECOM*, 2008.

[17] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *SIGCOMM '02*, pages 3–16, 2002.

[18] W. Muhlbauer, S. Uhlig, B. Fu, M. Meulle, and O. Maennel. In Search for an Appropriate Granularity to Model Routing Policies. In *Proceedings of ACM SIGCOMM*, 2007.

[19] B. Quoitin and O. Bonaventure. A survey of the utilization of the BGP community attribute, February 2002. Work in progress, draft-quoitin-bgp-comm-survey-00.txt.

[20] M. Shields. Automatic Configuration Generation and Auditing of Network. NANOG44, October 2008.

[21] J. van der Merwe and C. Kalmanek. Network Programmability is the answer! *PRESTO*, 2007.

[22] L. Vanbever, G. Pardoen, and O. Bonaventure. Towards validated network configurations with ncguard. In *Proc. of Internet Network Management Workshop 2008*, Orlando, USA, October 2008.

[23] R. White, D. McPherson, and S. Sangli. *Practical BGP*. Addison Wesley, 2004.