Enforcing Secure and Robust Routing with Declarative Policies

Palanivel Kodeswaran *, Filip Perich †, Wenjia Li *, Anupam Joshi * and Tim Finin*
*Computer Science and Electrical Engineering
University of Maryland, Baltimore County, Baltimore, Maryland 21250
palanik1,wenji1,joshi,finin@cs.umbc.edu
†Shared Spectrum Company
Vienna, Virginia 22182
Email: fperich@sharedspectrum.com

***Abstract*—Internet routers must adhere to many polices governing the selection of paths that meet potentially complex constraints on length, security, symmetry and organizational preferences. Many routing problems are caused by their misconfiguration, usually due to a combination of human errors and the lack of a high-level formal language for specifying routing policies that can be used to generate router congurations. We describe an approach that obviates many problems by using a declarative language for specifying network-wide routing policies to automatically congure routers and also inform software agents that can diagnose and correct networking problems. Our policy language is grounded in ontologies dened in the Semantic Web language OWL, supporting machine understanding and interoperability. polices expressed in it can be automatically compiled into low-level router congurations and intelligent agents can reason with them to diagnose and correct routing problems. We have prototyped the approach and evaluated the results in both a simulator and on a small physical network. Our results show that the framework performs well on a number of use cases, including checking for policy coherence, selecting paths to enforce security constraints, preventing asymmetric routing patterns, applying organizational preferences, and diagnosing and correcting failures.**

I. Introduction

The rest of this text is from the icccn paper draft.

Recent studies [**?**] have shown that human error is a major cause for a number of BGP related routing failures. The absence of a high level language for modelling network-wide routing policies forces network operators to manually configure BGP routers with low level details. The resulting configurations do not always reflect the organization's routing policy in its entirety and also lack support for verification. Clearly, there is a growing need within both the operator and research communities for a high-level language to model and configure BGP routing policies. The goal of these high level languages is to allow operators focus on the policy decisions rather than on the low-level implementation details. For example, we would like to be able to automatically configure routers to implement the valley free property [**?**] of BGP routing by merely stating the agreement type between a pair of autonomous systems without having to manually construct the associated export and import filters.

In this paper we describe an ontology-based, declarative language for modelling and configuring BGP routing policies. Our language supports configuring export and import filters as well as expressing route preferences. It uses Semantic Web languages that have well defined syntax and logical semantics. Consequently, policies expressed in our language can be formally reasoned over, conflicts and vulnerabilities discovered and corrections made.

We build on this declarative framework and construct an argumentation protocol in which neighboring routers ahre information and argue with each other to diagnose the causes of problems and recovere from router misconfigurations. We differ from previous "black-box" approaches by emplying a white box framework in which we have visibility into ISP policies expressed in our language. Previous approaches, on the other hand, are ISP policy transparent and resort to tomographic approaches for locating the cause of the failure. Furthermore, in addition to locating the cause of failures, we also try to recover from the failure by router reconfigurations if so allowed by ISP policy. We fathom scenarios where ISP policies may prevent router reconfigurations due to privacy and securtiy concerns. In these cases, we alert the network operator providing them with the location and cause of the failure, thereby saving precious time in diagnosing network failures.

We see two main contributions in this work: defining an ontology-based, declarative language for expressing BGP routing policies and using an argumentation protocol for diagnosing and recovering from routing misconfigurations. The rest of the paper is organized as follows. In section II, we review related work. We describe our declarative language and argumentation protocol in sections III and IV, respectively. In section V, we present our system architecture and discuss issues related to the practical deployment of our approach in section VI. We present our evaluation in section VII and finally conclude in section VIII.

II. Related Work

In [**?**], the authors present an elaborate discussion of the causes and frequency of BGP misconfigurations. The authors found that, in general, Internet connectivity was highly resilient to misconfigurations with only 4% of the misconfigured announcements affecting connectivity. The authors attribute

the misconfigurations to a variety of causes such as human error, router bugs, route redistribution, lack of transactional semantics for configuration commands etc. The authors conclude by providing a list of recommendations that could go a long way in reducing the incidence of misconfigurations viz. better user interfaces, high level declarative languages and configuration checking, protocol extensions and consistency among the multiple databases that are used for configuring BGP routers.

There have been recent works related to diagnosing Internet routing failures from end hosts such as those in [**?**], [**?**]. In [**?**], the authors propose a network tomography based algorithm for diagnosing and locating AS level routing failures. The authors modified a well known network tomography algorithm for the multi AS problem and use it to locate link as well as misconfiguration failures. The authors showed that additional control plane information such as routing data improved the accuracy of locating failures. The authors in [**?**] consider the practical issues related to using network tomography for fault diagnosis and find that tomography approaches perform well in detecting failures that last over 5 seconds.

Feamster et al. in [**?**] address the foundational problems associated with scalable policy based interdomain routing. The authors argue that while some of today's Internet routing failures may occur due to BGP specifics, there are intrinsic problems that need to be addressed such as interdomain policy disputes, lack of control and data plane security as well as issues arising from scalability such as prefix aggregation and AS abstractions. In [**?**], the authors propose a feedback based routing scheme to handle routing failures and attacks. The basic idea is to separate structural information such as topology from dynamic information such as latency, loss rate etc. which are learned by the routers themselves by probing. In their scheme, only the structural information is propogated among routers where as routing decisions are made by individual routers based on the collected dynamic information. Mahajan et al. in [**?**] propose Negotiation Based Routing (NBR). The basic idea is for a pair of ISPs to use opaque preference lists to negotiate and choose an optimal inter-AS link for each flow when there are multiple links between the ASes. They show that NBR based routing is beneficial to both ISPs compared to selfish routing. Furthermore, the scheme is flexible to enable each ISP to independently optimize for its own metric. The nature of the negotiations further ensures that ISPs do not benefit from cheating.

There has been a recent interest in the development of declarative languages for network management. In [**?**], the authors propose a generic declarative network management framework that can be used for configuring many pieces of network management such as ACLs, NATs, QoS etc. at a single place. Their policy engine applies policy decisions at the granularity of individual unidirectional flows. Their language also has in built support for conflict detection and policy prioritization for conflict resolution. Performance results from actual deployments showed that the policy based flow management framework typically performed well in these environments and promised good scalability. The authors in [**?**] propose, Nettle, a domain specific embedded language in Haskell for BGP configuration. They use the type safety checks of the language to ensure that common configuration errors are avoided. RPSL [**?**] is used by ISPs to express their routing policies that are stored in Internet Routing Registries. However, none of these languages have implicit support for reasoning.

III. Ontology Based Declarative Language

There are several advantages to developing an ontology based language for routing policies. The language is generic and can be used to express the policies of different organizations. The ontological approach naturally supports evolution and new policies can be modeled as they became relevant to the organization by updating the appropriate ontology. Furthermore, the logical basis of the language automatically supports reasoning and conflict resolution among policies. As part of this work, we have developed an ontology for representing routing policies. Our current ontology models concepts such as neighbors, autonomous system, network prefix etc. We also model the different types of policies viz. permissive, obligatory and prohibitive. These policies typically influence whether route updates are accepted, shared or denied. We also support policies that prefer routes from one AS over the other. Using concepts defined in our ontology, we can write policies that can be used to automatically create appropriate BGP Configurations. For example, we can express policies that specify which routes are accepted from neighbors based on the relationship with the neighbor. The relationship itself could be based on multiple factors such as economical, political etc. Our framework also supports prioritization of policies which becomes useful in the context of resolving conflicts among multiple policies. The policies expressed in our language are automatically translated into appropriate BGP level router configurations by our framework.

While our language can model typical BGP configuration issues, for the rest of the paper, we focus on aspects of the language that are used for configuring import and export filters. Also, we define a policy as a rule that specifies how to handle a route update. Since we limit our discussion to import/export policies, the set of allowable actions include accepting/rejecting a route update from a neighbor and sharing/denying a route advertisement to a neighbor. As mentioned above, our framework supports expressing policies that factor the relationship type between a pair of ASes. For example, the valley free policy [**?**] that requires a provider to announce its customer's prefixes to its upstream provider and peers can be expressed in our framework as begin

```
ShareUpdate(U,A,D) :-
  Origin(U,C),
  Customer(C,A),
  Provider/Peer(D,A)
```

The above rule is interpreted as RouteUpdate "U" is shared by AS "A" with its neighboring AS "D", if U contains a prefix that originates at AS "C", and "C" is "A"'s customer and "D" is either "A"'s peer or provider. This policy results in the following BGP configuration at A

```
router bgp ASN(A)
neighbor IpAddressOfRouter(C) filter-list 1 in
ip as-path access-list  1 permit ^ASN(C)$
```

where ASN(X) represents the Autonomous System Number of AS X. Similarly, we could have configured an export filter at AS C to share only C's prefixes. Furthermore, any route that is not explicitly shared is not advertised to a neighbor.

A. Creating a Routing Knowledge Base

The knowledge base contains facts, rules and any piece of information that is useful in deciding the routing policy of the organization such as the operating policy, traffic matrix, time of the day etc. Minimally, the knowledge base contains a high level representation of the network wide routing policies of the organization. The knowledge base is initially loaded with the following minimal set of base rules that specify the operating policy of the node, conditions for sharing/denying route updates as well as the condition for retracting policies.

```
(?i follow ?x ):-
  (?i trust ?y),
  (?y issues ?x)-(1)
```

Rule 1 states that a node follows a policy issued by a trusted entity. Nodes assert that they trust their parent organization. Additionally, nodes could be configured to trust external organizations for business purposes as well.

```
(?i shareRouteAdvt ?d) :-
  (?i follow ?x),
  (?x sharesRouteAdvt ?d)-(2)
```

Rule 2 asserts that a node shares a route advertisement as long as it follows a policy that permits sharing the advertisement through the "sharesRouteAdvt" predicate.

```
(?i denyRouteAdvt ?d) :-
  (?i follow ?x),
  (?x deniesRouteAdvt ?d)-(3)
```

Rule 3 is similar to Rule 2 for denying route advertisements to neighbors.

```
(?y retracts ?d) :-
  (?y issues ?a),
  (?y issues ?d),
  (?a replaces ?d)-(4)
```

Rule 4 specifies the condition for an entity to retract a policy. When an entity issues two policies, and one policy replaces the other, the entity essentially retracts the replaced policy. Retractions typically occur when a higher priority policy replaces a lower priority one as well as when a later version of a policy replaces the older one.

The knowledge base is futher updated with the creation of new policies. When a new policy is created, the rules representing the policy are asserted into the knowledge base along with the direct and inferred facts. For example, the valley free policy that denies advertising non-originating prefixes to a provider makes the following assertion into the knowledge base of AS C

```
``ValleyFreePolicy deniesRouteAdvt 12.1/16''
```

where 12.1/16 corresponds to a prefix not originating in AS "C". We can thus automatically create the knowledge base from the policies expressed in our language. The knowledge base thus created is used in the argumentation protocol described in the next section.

IV. Argumentation Framework for Detecting Routing Misconfigurations

In this section, we present our argumentation framework for detecting and recovering from routing failures. Typically, a routing failure is followed by frantic phone calls[**?**] among network operators trying to locate and diagnose the cause of the failure. Most routing failures however, are caused by BGP misconfigurations that are human generated. These misconfigurations arise due to the fact that humans configure routers at the lowest level details and do not necessarily represent the high level goals of the policy. Furthermore, since no usable semantics is associated with the low level configuration data, diagnosis of the failure is further impeded. It is now well recognized [**?**] that most of the BGP related misconfigurations can be avoided through the development of high level languages for router configuration. We argue that in addition to automatically configuring routers, a logic based declarative language that supports reasoning can be used to diagnose and recover from routing failures. The diagnosis problem can be modelled as a multiagent problem as follows. Each border router in an Autonomous System can be considered as an agent. Each agent must now co-ordinate with peer agents in other ASes to arrive at a conclusion about the cause and location of a failure. On the other hand, the declarative nature of the policies enables meaningful communication among the agents during argumentation. While some failures such as link failure are transient and are handled by the underlying routing protocol, other failures such as misconfigurations are long lasting and need operator intervention. It is this class of non-transient failures that we aim to diagnose and recover automatically through agent communication.

A. Design of the Argumentation Framework

We now describe our argumentation protocol for diagnosing misconfigured routers and automatically correcting them. Although our framework can handle generic situations, in this particular work, we focus on diagnosing and correcting export filters. Our argumentation protocol is principled along the Fatio [**?**] Argumentation Protocol. The protocol itself consists of only the following six utterances/messages.

Ask. Used by a router to query a neighbor. The query itself could be application dependant such as prefix reachability, accepting/dropping traffic for a particular destination etc. In the use case presented in this paper, the query is whether the neighbor has a route to a particular destination prefix.

Confirm. On receiving a Ask message, the recipient consults its internal data store which might be either a knowledge base or routing table to evaluate the query in the Ask message. If the query evaluates to be true, the recipient replies with a Confirm message.

Deny. Similar to the confirm message, if the query in the Ask message evaluates to be false, the recipient replies with a Deny message.

Challenge. On receiving a Confirm/Deny message from a neighbor, the recipient can challenge the response with a Challenge Message.

Justify. On receiving a challenge from a neighbor, the recipient responds with a justification of why it Confirmed/Denied the query in the Ask message. Most generally, the justification is a set of policy statements that the node currently believes in and against which the Ask query was evaluated.

Assert. On receiving a Justify message, the recipient issues an Assert message if it believes the neighbor's justification is not valid. This could arise for a number of reasons such as the neighbor following an old policy, the neighbor following a lower priority policy and so on. On receiving an Assert, the neighbor evaluates the assertion. If the assertion is valid, generally a reconfiguration takes place at the neighbor. The reconfiguration could be in the form of either updating export/import filters or updating a policy with a newer version. When the reconfiguration phase completes, the neighbor responds with a Confirm message which ends the argumentation round.

Fig. 1. Router topology with misconfigured export filter at Router B

We now present an example of using the above described argumentation protocol for reconfiguring export filters. We consider the topology shown in figure 1 . We make the following assumptions in the example.

- "Policy1" denies advertising routes to 12.1/16
- "Policy2" allows advertising routes to 12.1/16
- "Policy2" has higher Priority compared to "Policy1" and replaces the latter.

At the beginning of the example, B follows "Policy1", and hence does not share reachability information for 12.1/16 to Router A which follows "Policy2" . When router A finds that it has no route to a destination in 12.1/16, A sets up an argumentation with B as follows

A→ B: Ask(B hasRoute to 12.1/16)

On receiving Ask, B queries its route table to see if it has a route to 12.1/16. Since we assume C has no misconfigured export filters, B has a route for 12.1/16 and replies with a Confirm

B→A:Confirm(B hasRoute to 12.1/16)
A→B:Ask(B deniesRouteAdvertisementFor 12./16)
B→A:Confirm(B deniesRouteAdvertisementFor 12.1/16)
since B follows "Policy1".
A→B:Challenge(B deniesRouteAdvertisementFor 12.1/16)
B→A: Justify(B deniesRouteAdvertisementFor 12.1/16
since B follows "Policy1", "Policy1" deniesRouteAdvertisementFor 12.1/16)
A→B: Assert ("Policy2" hasHigherPriorityThan "Policy1", "Policy2"
replaces "Policy1","Policy2" allowsRouteAdvertismentFor "12.1/16")

On receiving the Assert message, B evaluates the Policy statements in the message which results in B following "Policy2". Since the current operating policy changes, B reconfigures its filters in line with the new policy, "Policy2", thereby sharing the reachability information for 12.1/16 with A.

B→A: Confirm(ok)

Several points are noteworthy from this argumentation example. First is the need for high level declarative policies that can be reasoned over. Each step in the argumentation protocol requires a query to be evaluated against the knowledge base of a router. Also, for the Justification step, the reasoner needs to generate the proof tree[**?**] for query evaluation. Secondly, the argumentation may not always converge or may take more than a single round. The example we have described is a simple use case in which argumentation does converge in a single round. However there could be situations in which the argumentation may not converge in a single round and may need multiple rounds and consequently multiple reconfigurations at both routers.

For example, consider the scenario where in a node is argueing with its third hop neighbor. Further, the third hop neighbor has an export filter that denies a route advertisement where as the second hop neighbor has an import filter rejecting route updates for the target prefix. In the first round of argumentation, the export filter of the third hop neighbor is reconfigured to share the route advertisement. However, in order to obtain a route to the target prefix, the import filter at the second hop neighbor needs to be reconfigured as well, calling for a second round of argumentation. On the other hand, there exist scenarios where in the argumentation may not converge at all. These situations may arise for example due to policy conflicts. A node may believe it has a higher priority policy than its neighbor while the neighbor may not agree with that. These failures, which can only be resolved with operator involvement, are flagged off to the operator with a log of the argumentation protocol executed so far.

V. System Architecture

Fig. 2. System Architecture

In this section, we describe our system architecture as shown in Figure 2.

Planner: The planner is the front end used by the network administrator to create network wide routing policies. Typically, this is a graphical user interface with support to view network status such as topology, traffic load on links etc. Furthermore, the planner is used to create policies which are then distributed to router agents in the network.

Agent Framework: Each agent is responsible for controlling the underlying physical router. Policies created by the network operator are sent to the agents which are responsible for configuring the routers in line with the policies. Typically, the agents have an onboard reasoner or invoke a centralised reasoner to transform the policies into appropriate configurations. The agents also support a query and control interface to the underlying router which is used during the

argumentation phase. Agents communicate with each other during argumentation using an out of band communication channel such as an overlay network.

Reasoner: The function of the reasoner is to translate high level policies into appropriate low level configurations. Generally, each agent has its own reasoner, although a centralized reasoner for all agents is also possible. The configuration generated by the reasoner is sent back to the calling agent which then applies it to the underlying router.

Router: The physical router which performs packet processing and is controlled by the router agent.

A. Policy Architecture

We pursue a hierarchical policy structure in our framework. The planner generates network wide policies which are then distributed to the individual agents. Each agent can have own its own local policies as well. Additionally, an agent can make local assertions about its neighbors including configuration data like IP addresses, AS Number etc. The local assertions could also include the type of relationship with the neighbor. The network wide policies are then merged with local policies to form a unified policy which is used in generating the configurations. The ontological basis of our framework enables merging of policies as well as detecting conflicts. In our framework, we assume that network wide policies have higher priority compared to local policies during conflict resolution.

VI. Discussion

In this section we discuss issues related to the practical deployment of our approach.

ISP routing policies are generally considered to be private information. On the other hand, during network failures, operators across ISPs often collaborate revealing network configuration data with the goal of locating and rectifying the cause of failure. Our argumentation protocol does not necessitate complete information disclosure for its operation. Network operators can still retain control over the pieces of information that are exchanged during argumentation. For example, a network operator could require that a certain policy be private and never be revealed to neighbors. Under such scenarios, if the private policy happens to be the cause of failure, the agent responds with a "Cannot Reveal" message in the justification step. In this case, the neighbor could either attempt to recover with partial information or alert the human operator. Similar constraints can be placed on local reconfigurations as well. The operator could specify that, while it is acceptable to share policies for diagnosis, only certain reconfigurations may be performed locally. In these cases, every reconfiguration act that is initiated through argumentation is first locally checked through a list of network operator approved configuration acts before execution. When a reconfiguration is not permitted, the network operator is flagged with a log of the argumentation execution and the reconfiguration to be performed. Another issue to consider is which agent will be responsible for initiating the argumentation and under what conditions. We propose periodically checking for route availability to a set of prefixes determined by the operator. When there is no local route available to a listed prefix, the agent responsible for the router/AS initiates the argumentation with its neighbors. Alternatively, we could use a monitoring infrastructure similar to [**?**] that periodically probes destinations to check for route availability and initiates argumentation when no route is available.

VII. Evaluation

A. Testbed Implementation

Fig. 3. Testbed set up in which Router 2 denies sharing a route for prefix 12.1/16 to Router 1

We have implemented our approach on a two router testbed of Cisco 1811 Series [**?**] routers as shown in Figure 3. Our agent infrastructure is written in Java and uses a directory based service for agent communication. During set up, each agent is bound to a router and communication is established with the router through telnet. We use the open source Jess [**?**] rule engine for generating BGP configurations from policies specified in our language.

As part of experiment set up, we configure the interfaces and set up static routes on each of the routers for reachability. We also load the routers with the following policies. Router 2 is loaded with "Policy1" which denies sharing a route for the prefix 12.1/16 with Router 1. On the other hand, Router 1 is loaded with "Policy 2" which allows sharing advertisements for the prefix 12.1/16 and has higher priority than Policy 1. Essentially "Policy 2" replaces "Policy 1". When Router 2 announces 12.1/16, we query the routing table of Router 1 and find that it has no route to 12.1/16. We then inject a "RouteUnavailableEvent" for prefix 12.1/16 into the Agent for Router 1 which initiates an argumentation with Router 2's agent, a log of which is as follows

> Worker[id=120.120.10.1]: has no route to prefix: 12.1.0.0/16
>
> Worker[id=120.120.10.2]: question: 120.120.10.1 asks if I believe "("i","hasRoute","12.1.0.0/16)"?
>
> Worker[id=120.120.10.1]: confirmation: 120.120.10.2 confirms that s/he believes "("i","hasRoute","12.1.0.0/16)".
>
> Worker[id=120.120.10.2]: question: 120.120.10.1 asks if I believe "("i","denyRouteAdvt","12.1.0.0/16)"?
>
> Worker[id=120.120.10.1]: confirmation: 120.120.10.2 confirms that s/he believes "("i","denyRouteAdvt","12.1.0.0/16)".
>
> Worker[id=120.120.10.2]: challenge: 120.120.10.1 challenges "null" because s/he has a problem with "("i","denyRouteAdvt","12.1.0.0/16)"
>
> Worker[id=120.120.10.1]: justification: 120.120.10.2 justifies "("i","denyRouteAdvt","12.1.0.0/16)" because s/be believes "[("POLICY1","deniesRouteAdvt",
>
> "12.1.0.0/16), ("i","follow","POLICY1)]", which implies it.

Worker[id=120.120.10.2]: assertion: 120.120.10.1 asserts "[("POLICY2","replaces","POLICY1), ("POLICY2","sharesRouteAdvt","12.1.0.0/16)]"

Worker[id=120.120.10.1]: confirmation: 120.120.10.2 confirms that s/he believes "[("POLICY2","replaces","POLICY1), ("POLICY2","sharesRouteAdvt","12.1.0.0/16)]"

Worker[id=120.120.10.2]: question: 120.120.10.1 asks if I believe "("i","denyRouteAdvt","12.1.0.0/16)"?

Worker[id=120.120.10.1]: denial: 120.120.10.2 denies s/he believes "("i","denyRouteAdvt","12.1.0.0/16)" because s/he believes "null" instead, which implies that it cannot be true.

In conclusion, the argumentation results in a reconfiguration at Router 2 removing the export filter. This is verified by querying the routing table of Router 1 which now has a route for 12.1/16.

B. Simulation

Fig. 4. Simulated topology in which a link failure initiates argumentation.

We have also evaluated our approach in the presence of link failures using CBGP[**?**]. We use the same agent framework as above with the only change being that the Jess reasoner now generates CBGP configurations from our policies. We use the topology as shown in figure 4. The prefix in parenthesis represent the network prefix originated by the respective routers. Further, router A is configured with Policy 2 while router B is configured with Policy 1 which have the same interpretation as in the previous case. D announces the network 12.1/16, which is now reachable by all other nodes. We simulate a link failure between A and D, thereby causing A-B-D and A-B-C-D to be the only available physical paths for A to reach the prefix 12.1/16. However, B's export filter prevents announcing the route for 12.1/16 to A resulting in A having not route to 12.1/16. A now initiates an argumentation with B, eventually resulting in B removing the export filter.

VIII. Conclusion and Future Work

Internet routers must be configured to adhere to many polices governing the selection of paths that meet potentially complex constraints on length, security, symmetry and organizational preferences. Many routing problems are caused by their misconfiguration, usually due to a combination of human errors and the lack of a high-level formal language for specifying routing policies that can be used to generate router congurations. We have described an approach that obviates many problems by using a declarative language for specifying network-wide routing policies to automatically congure routers and also inform software agents that can diagnose and correct networking problems.

Our policy language is grounded in ontologies dened in the Semantic Web language OWL, supporting machine understanding and interoperability. Polices expressed in it can be automatically compiled into low-level router congurations for execution and enforcement. A distributed collection of software agents can use the high-level policies and a custom argumentation protocol to share and reason over information about routing failures, diagnose probable causes, and correct them by reconguring routers and/or recommending actions to human operators.

We have prototyped the approach and evaluated the results in both a simulator and on a small physical network. Our results show that the framework performs well on a number of use cases, including checking for policy coherence, selecting paths to enforce security constraints, preventing asymmetric routing patterns, applying organizational preferences, and diagnosing failures and automatically correcting them by reconguration.

Acknowledgement

This work was supported by a research contract from DARPA ....

References