A Declarative Approach for Secure and Robust Routing

Palanivel Kodeswaran
University of Maryland,
Baltimore County
Baltimore MD 21250
palanik1@cs.umbc.edu

Anupam Joshi
University of Maryland,
Baltimore County
Baltimore MD 21250
joshi@cs.umbc.edu

Tim Finin
University of Maryland,
Baltimore County
Baltimore MD 21250
finin@cs.umbc.edu

Filip Perich
Shared Spectrum Company
Vienna, VA 22182
fperich@sharedspectrum.com

ABSTRACT

Many Internet failures are caused by misconfigurations of the BGP routers that manage routing of traffic between domains. The problems are usually due to a combination of human errors and the lack of a high-level language for specifying routing policies that can be used to generate router configurations. We describe an implemented approach that uses a declarative language for specifying network-wide routing policies to automatically configure routers and show how it can also be used by software agents to diagnose and correct some networking problems. The language is grounded in an ontology defined in OWL and polices expressed in it are automatically compiled into low-level router configurations. A distributed collection of software agents 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 reconfiguring routers and/or recommending actions to human operators. We have evaluated the framework in both a simulator and on a small physical network. Our results show that the framework performs well in identifying failure causes and automatically correcting them by reconfiguring routers when permitted by the policies.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Operations

General Terms

Management

Keywords

Policy, BGP Configuration, Argumentation

SafeConfig'10, October 4, 2010, Chicago, Illinois, USA.

1. INTRODUCTION

Recent studies [16] have shown that human error has been 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 at low level details. The configurations thereby generated do not always reflect the organization's routing policy in its entirety and also lack tool 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 [8] 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. The framework is based on a principled approach grounded in Semantic Web languages that possess strong logical underpinnings. Consequently, policies expressed in our language can be formally reasoned over and corrected in the case of conflicts. We build on top of our declarative framework and employ an argumentation protocol in which neighboring routers argue with each other in diagnosing and recovering from router misconfigurations.

We differ from previous black box approaches in that we pursue a white box approach in which we have visibility into ISP policies expressed in our language. Previous approaches [4], 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 envision scenarios where ISP policies may prevent router reconfigurations due to privacy and security 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.

This works makes two contributions: (i) an ontology-

based declarative language for expressing high-level policies BGP routing that can be compiled into router configuration commands and (2) the use of an argumentation protocol that allows a distributed system of agents to diagnose and recover from routing misconfigurations. In the remainder of the paper we review related work, describe our declarative language and argumentation protocol, present our prototype implementation's system architecture, discuss issues related to its practical deployment and describe our evaluation.

2. RELATED WORK

Mahajan et al. present an elaborate discussion of the causes and frequency of BGP misconfigurations in [16]. The authors found that Internet connectivity was generally highly resilient to misconfigurations with only 4% of the misconfigured announcements affecting connectivity. They attribute the misconfigurations to a variety of causes, such as human error, router bugs, route redistribution, and a lack of transactional semantics for configuration commands. The authors conclude by providing a list of recommendations that could go a long way in reducing the incidence of misconfigurations, including developing better user interfaces, the use of high level declarative languages and configuration checking, adding protocol extensions, and maintaining consistency among the multiple databases used for configuring BGP routers. Our work addresses the need for high level declarative languages by proposing an ontology based language for expressing routing policies that are automatically compiled into low level configuration details.

There has been a great deal of work related to diagnosing Internet routing failures from end hosts such as those in [22, 15]. Dhamdhere et al. propose a network tomography based algorithm for diagnosing and locating routing failures at the autonomous system (AS) level in [4]. The authors modified a well known network tomography algorithm for the multiple 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. Huang et al. [11] 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 five seconds. Our work can complement these approaches by diagnosing the cause of the failure as well as correcting misconfigurations where permitted by policy.

Feamster et al. in [6] address the foundational problems associated with scalable policy based inter-domain 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 inter-domain policy disputes, lack of control and data plane security as well as issues arising from scalability such as prefix aggregation and AS abstractions. Zhu et al. [23] 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 propagated among routers where as routing decisions are made by individual routers based on the collected dynamic information. Mahajan et al. propose Negotiation Based Routing (NBR) in [17]. 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 autonomous systems. 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. Our work is similar in spirit in that it involves negotiating with neighboring ASes.

There has been a recent interest in the development of declarative languages for network management. Hinrichs et al. [9] 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. Voellmy et al. propose Nettle, a domain specific embedded language in Haskell for BGP configuration in [21]. They use the type safety checks of the language to ensure that common configuration errors are avoided. RPSL [1] is used by ISPs to express their routing policies that are stored in Internet Routing Registries. Our work differs from these languages in that they do not implicitly support reasoning.

3. ONTOLOGY BASED LANGUAGE

In this section, we present our ontology based declarative language for expressing network wide routing policies. 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. It also naturally supports a hierarchical policy architecture in which operators can write policies at various granularities depending on their position in the organizational hierarchy.

At the top of the hierarchy, subsumption capabilities could be used to write generic policies at a high level, while more specific policies could be written at the lower levels of the hierarchy. These policies can then be merged to create a final policy that represents the routing policy of the organization at all levels of the hierarchy. 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 using the Semantic Web languages OWL [2] and RDF [14] for representing routing policies and associated concepts. Our ontology models concepts such as nodes, autonomous systems and routes as well as properties and relations like neighbors and network prefix. We also model the three different deontic types of policy rules or constraints – permissive, obligatory and prohibitive. These policies typically influence whether route updates are accepted, shared or denied. We support policies that prefer routes from one AS over the other.

Using the concepts defined in our ontology, we can write policies that can be used to automatically create appropri-

ate BGP configurations in a lower-level language accepted by current hardware systems. 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, including business relations, economic considerations and political constraints [13]. Our framework supports prioritization of policies which becomes useful in the context of resolving conflicts among multiple policies. The policies expressed in our language can be reasoned about t a high level as well automatically translated into appropriate BGP-level router configurations that can be used for enforcement.

While our language can model typical BGP configuration scenarios, the rest of the paper will focus on aspects of the language 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 four possible actions: accepting or rejecting a route update from a neighbor and sharing or denying a route advertisement to a neighbor. As mentioned above, our framework supports expressing policies that factor the relationship type between a pair of autonomous systems. For example, the *valley free policy* [8] requires a provider to announce its customer's prefixes to its upstream provider and peers. It can be expressed in our framework with the following rule, expressed as a Horn clause [10] using the familiar Prolog-like syntax.

```
shared_update(U, A, D) :-
  origin(U, C),
  customer(C, A),
  provider_or_peer(D, A).

provider_or_peer(AS1, AS2) :- provider(AS1, AS2).
provider_or_peer(AS1, AS2) :- peer(AS1, AS2).
```

The implemented system uses rules in more restricted subset of Horn clauses and with a slightly different syntax. These are based on the SWRL rule language [19] that works in conjunction with a knowledge base of OWL axioms and assertions.

The rule for the *shared_update* relation given above is interpreted as follows. A route update "U" is shared by autonomous system "A" with its neighboring autonomous system "D", if three conditions are simultaneously true for a single set of variable bindings. Tokens beginning with an uppercase character are variables. The conditions are that U contains a prefix that originates at autonomous system "C", that "C" is "A"'s customer and that "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.

3.1 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).
```

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).
```

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).
```

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).
```

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 further 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.

4. DETECTING ROUTING MISCONFIGURATIONS VIA ARGUMENTATION

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 [12] 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 [16] 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.

4.1 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 [18] 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.

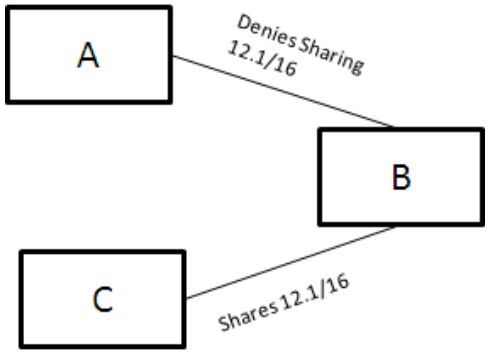

Figure 1: Router topology with misconfigured export filter at Router B

We now present a simple example illustrating our argumentation protocol. 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")

When the Assert message is received, 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.

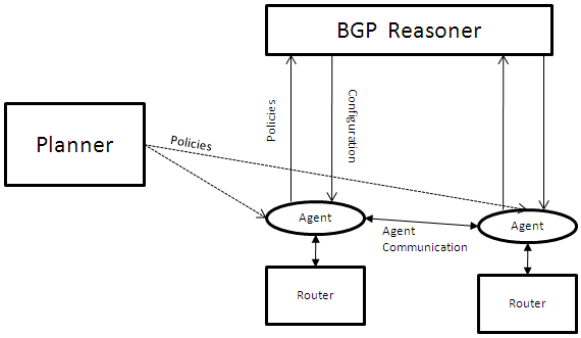

Figure 2: Our logical architecture assumes a set of independent routers each with an agent process that performs various services for it. The planner generates high-level policies and the BGP reasoner can compile them into appropriate low-level configuration commands.

- 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 [5] 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 arguing 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.

5. SYSTEM ARCHITECTURE

The system architecture used in our prototype implementation is shown in Figure 2. It consists of a set of routers, each with an associated agent, a reasoner and a planner.

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

Figure 3: The planner in our prototype implementation allows us to easily experiment with new policies and networking configuration for experimentation.

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. Figure 3 shows a snapshot of the planner used in our prototype implementation.

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.

Our policies has a hierarchical structure that reflects the network organization. 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

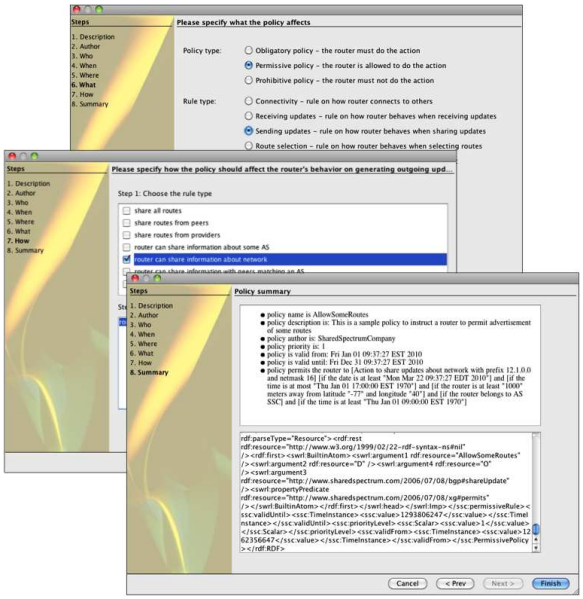

Figure 4: A simple user interface using the familiar "wizard" approach allows an engineer to inspect, maintain or author a policy without having to understand the underlying representation languages.

that network wide policies have higher priority compared to local policies during conflict resolution.

6. PRACTICAL ISSUES IN DEPLOYMENT

In this section we discuss issues related to the practical deployment of our approach, including issues of privacy, operator involvement and policy authoring.

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 [4] that periodically probes destinations to check for route availability and initiates argumentation when no route is available.

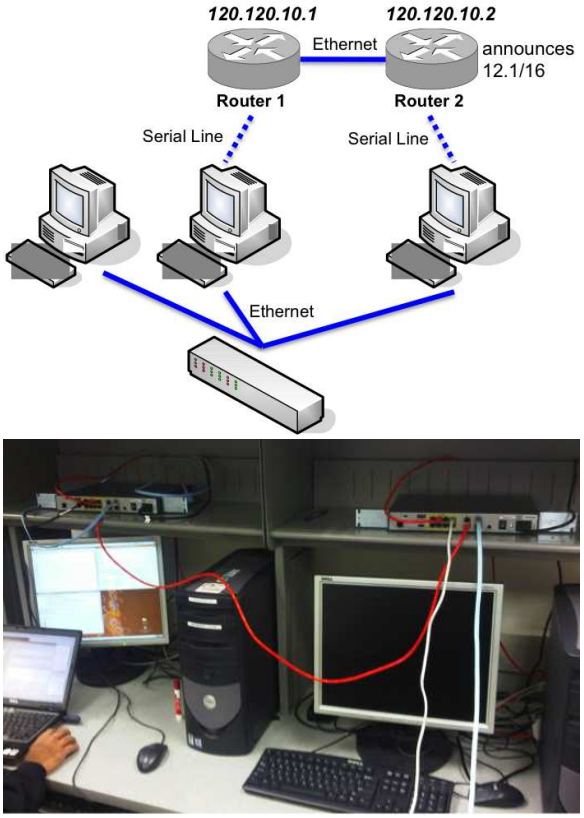

Figure 5: We tested the system on a physical network consisting of two Cisco routers. Router 2 denies sharing a route for prefix 12.1/16 to Router 1.

Our policies are represented as rules grounded in concepts and predicates specified in an ontology. These representations will be unfamiliar and opaque to most network engineers and operators who nonetheless may need to author, modify or maintain a set of policies. We have implemented a prototype "policy wizard" system that walks a user through the process of examining, maintaining or authoring policy rules. Figure 4 shows examples of from this user interface for policy inspection and authoring.

7. EVALUATION

We evaluate our approach using both a testbed implementation and simulations. We have verified our approach on a small testbed in our lab and perform simulation studies using CBGP [20] which can be used to model arbitrarily large topologies.

7.1 Testbed Implementation

We have implemented our approach on a two router testbed

- 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 belief (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 belief (i,denyRouteAdvt,12.1.0.0/16).
- Worker[id=120.120.10.2]: challenge: 120.120.10.1 challenges null because it 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 it believes [(POLICY1,deniesRouteAdvt,12.1.0.0/16), (i,follow,POLICY1)]
- 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 belief [(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 it believes (i,denyRouteAdvt,12.1.0.0/16) because it believes null instead, which implies it is false

Figure 6: The agents for two routers engage in an argumentation dialogue in attempt to come to a common model of their shared environment.

of Cisco 1811 Series [3] routers as shown in Figure 5. 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 [7] 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 shown in Figure 6.

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.

7.2 Simulation

We now illustrate a simple example evaluating our approach in the presence of link failures using CBGP [20]. We use the same agent framework as above with the only change being

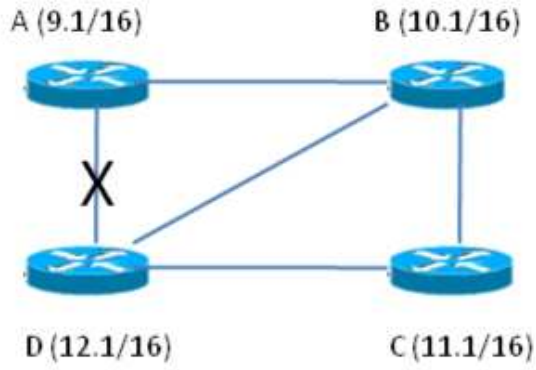

Figure 7: Simulated topology in which a link failure initiates argumentation.

that the Jess reasoner now generates CBGP configurations from our policies. We use the topology as shown in Figure 7. 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.

8. CONCLUSION AND FUTURE WORK

We have presented the case for using argumentation and declarative policies for diagnosing and recovering from Internet routing failures. We have developed an ontology based declarative language for expressing network wide routing policies that are then compiled into appropriate BGP configurations. We leverage the declarative nature of the policies by developing an argumentation protocol in which routing agents reason about their high level policies and negotiate with neighboring routers to recover from routing misconfigurations.

We have verified our approach in a small physical network and using simulations. In future work, we plan on extending our framework to argue with non-immediate neighbors as well. We would also like to explore the case of simultaneously arguing with multiple neighbors and choosing the optimal reconfiguration based on an acceptable definition of optimality. While we do not fathom that our approach would be deployed on the Internet immediately due to economic and privacy considerations, we position our work as an initial step towards white box based approaches for automated diagnosis and recovery of Internet routing failures.

9. ACKNOWLEDGMENTS

We thank Wenjia Li for developing the front end of the planner. This work was supported in part by DARPA under contract W31P4Q-06-C-0395 and the Air Force Office of Scientific Research under MURI award FA9550-08-1-0265.

10. REFERENCES

[1] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing policy specification language (RPSL), 1999.

[2] S. Bechhofer, F. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L.Stein. OWL Web Ontology Language Reference W3C Recommendation. Technical report, W3C, February 2004.

[3] Cisco. Cisco Router Guide. Cisco Systems, Inc Website, 2007.

[4] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proceedings of the 3rd ACM International Conference on emerging Networking EXperiments and Technologies*, pages 1–12. ACM, 2007.

[5] A. Eriksson and A.-L. Johansson. Neat explanation of proof trees. In *Proceedings of the 9th international joint conference on artificial intelligence*, pages 379–381. Morgan Kaufmann Publishers Inc., 1985.

[6] N. Feamster, H. Balakrishnan, and J. Rexford. Some foundational problems in Interdomain routing. In *In HotNets, 2004. (Cited on*, pages 41–46, 2004.

[7] E. Friedman-Hill. *JESS in Action*. Manning, 2003.

[8] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking (TON)*, 9(6), 2001.

[9] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking - WREN '09*, page 1, New York, New York, USA, 2009. ACM Press.

[10] A. Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951.

[11] Y. Huang, N. Feamster, and R. Teixeira. Practical Issues with Using Network Tomography for Fault Diagnosis.

[12] P. Hunter. Pakistan YouTube block exposes fundamental Internet security weakness:: Concern that Pakistani action affected YouTube access elsewhere in world. *Computer Fraud & Security*, 2008(4):10–11, 2008.

[13] P. Kodeswaran, S. B. Kodeswaran, A. Joshi, and F. Perich. Utilizing semantic policies for managing BGP route dissemination. In *IEEE INFOCOM 2008 - IEEE Conference on Computer Communications Workshops*, pages 1–4. IEEE, 2008.

[14] O. Lassila and R. Swick. Resource description framework (RDF) model and syntax. Technical report, W3C, February 1999.

[15] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level internet path diagnosis. *Operating systems review*, 37(5):106–119, 2003.

[16] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–16. ACM, 2002.

[17] R. Mahajan, D. Wetherall, and T. Anderson. Negotiation-based routing between neighboring ISPs. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation - Volume 2*, 2005.

[18] P. McBurney and S. Parsons. Locutions for Argumentation in Agent Interaction Protocols. In *International Conference on Autonomous Agents*, 2004.

[19] M. OConnor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, and M. Musen. Supporting rule system interoperability on the semantic web with SWRL. In *Proceedings of the 4th International Semantic Web Conference*, pages 974–986. Springer, 2005.

[20] B. Quoitin and S. Uhlig. Modeling the Routing of an Autonomous System with C-BGP, 2005.

[21] A. Voellmy and P. Hudak. Nettle: A language for configuring routing networks. In *DSL '09: Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages*, pages 211–235, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] Y. Zhang, Z. M. Mao, and M. Zhang. Effective diagnosis of routing disruptions from end systems. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[23] D. Zhu, M. Gritter, and D. R. Cheriton. Feedback based routing. *SIGCOMM Computing Communication Revue*, 33(1):71–76, 2003.