Available online at www.sciencedirect.com
SCIENCE DIRECT®
Science of Computer Programming 53 (2004) 125–141
Science of Computer Programming
www.elsevier.com/locate/scico

Using service grammar to diagnose BGP configuration errors

Xiaohu Qie[a,*], Sanjai Narain[b]

[a]*Google Inc., Mountain View, CA 94043, USA*
[b]*Telcordia Technologies, Piscataway, NJ 08854, USA*

Received 31 December 2003; accepted 31 December 2003

Available online 21 July 2004

Abstract

Often network components work correctly, yet end-to-end services don't. This happens if configuration parameters of components are set to incorrect values. Configuration is a fundamental operation for logically integrating components to set up end-to-end services.

Configuration errors arise frequently because transforming end-to-end service requirements into component configurations is inherently difficult. Such transformation is largely performed in a manual and localized fashion, resulting in high cost of network operations.

The *Service Grammar* technique has been developed to solve the configuration error diagnosis problem, and, more generally, to formalize the process of building complex systems via configuration.

At its core is a *Requirements Language* that contains global, high-level constraints upon configuration parameters. These are derived from identifying the notion of "correct configuration" associated with different protocols. These are composed to create system-wide requirements on architecture and policies. A *Diagnosis Engine* checks if constraints in the Requirements Language are true given definite component configurations, and is used recursively to check composite requirements.

This paper describes an application of Service Grammar to diagnosing BGP configuration errors. As BGP architecture and policies differ widely from one network to another, it is not possible using previous techniques to check if router configurations implement the *intended* requirements. Our tools enable administrators to specify system-wide, network-specific requirements and check if they are correctly implemented by component configurations.

* Corresponding address: Google Inc., 1600 Amphitheatre Parkway, 94043 Mountain View, CA, USA. Tel.: +1-650-623-5009.
E-mail address: qiexh@cs.princeton.edu (X. Qie).

0167-6423/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2003.12.007

1. Introduction

Traditional network management systems diagnose hard, localized errors such as fiber cuts or hardware/software component failures. It is quite possible, however, that network components work correctly yet end-to-end services don't. This happens if there are configuration errors, i.e. configuration parameters of components are set to incorrect values. Configuration is a fundamental operation for integrating components to implement end-to-end services. Configuration errors arise frequently because transforming end-to-end service requirements into configurations is inherently difficult: in realistic networks there are many components, configuration parameters, values, protocols and requirements. Yet, such transformation is largely performed manually. The resulting high cost of network operations as well as the potential for security breaches is well documented [7,8].

The *Service Grammar* [2,10–12] technique has been developed to solve the configuration error diagnosis problem, and more generally, to formalize the process of building complex systems via configuration. At its core is a *Requirements Language* that contains *global*, high-level abstractions that are set up in the process of setting up end-to-end services. A good heuristic for deriving this language is to ask the question "what does it mean for a group of agents executing a protocol to be correctly configured". This language is created for all of the protocols in a domain of interest. End-to-end service requirements can be naturally defined as logical conjunctions of requirements in the language at and across different protocol layers.

This is done by rules of the form A:- $B_1, \ldots, B_k, k \geq 0$, where each A and B_i is a requirement. A *Diagnosis Engine* checks if a language requirement is true given definite system configuration. By recursive use of this operation, complex algorithms for diagnosis can be developed. Service Grammar captures the intuition to regard a system not as a set of components but as a set of services that, in general, span multiple components.

The information flow of the diagnosis system is illustrated in Fig. 1. The diagnosis engine takes input from two sources: (1) service requirements expressed in the requirements language, and (2) vendor-neutral component configurations stored at a centralized database, e.g. an LDAP directory. Raw component configuration is parsed into vendor-independent data structures by vendor-specific adaptors. The diagnosis engine queries the component configuration database and verifies if the configurations are consistent with service requirements. If not, it notifies the administrator where the diagnosis process had failed. The administrator can then modify the configuration settings and rerun the diagnosis process.

Service Grammars have been built and used for adaptive Virtual Private Networks and mobile security [2,10–12]. This paper describes an application of Service Grammar to diagnosing configuration errors in Border Gateway Protocol (BGP) [14]. Previous solutions to diagnosing configuration errors have been network invariant [13] in that they contain a fixed set of constraints that must be satisfied by every BGP network. However, BGP requirements such as logical architecture and policies differ widely from one network to another. It is not possible in previous techniques to check if component configurations implement the *intended* requirements. Our tools enable administrators to specify network-specific requirements and check if they are correctly implemented by component configurations.

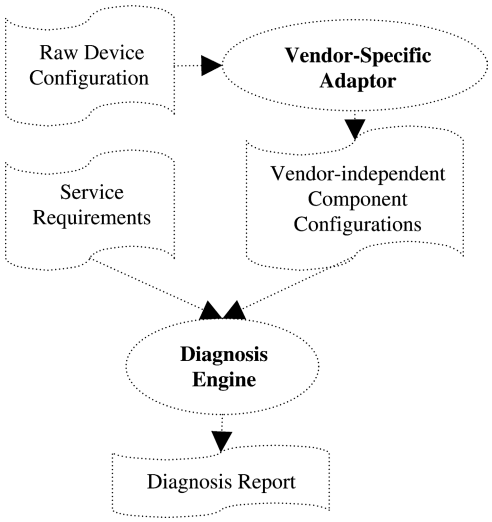

Fig. 1. Diagnosis system.

2. BGP background

The Internet consists of Autonomous Systems (ASes) operated by different institutions. Within an AS, routing is controlled by intra-domain protocols such as OSPF, IS-IS, and RIP. Border Gateway Protocol (BGP) is the Internet's inter-domain routing protocol. BGP has two distinct modes of operation: External BGP (EBGP) and Internal BGP (IBGP). Routers in different ASes use EBGP to exchange information on how to reach destinations throughout the Internet. Routers in the same AS use IBGP to exchange external reachability information. IBGP is essentially the same as EBGP except for the treatment of certain attributes and the forwarding manner of BGP messages [14].

BGP is a path-vector based protocol. A BGP route consists of a network prefix N and an *AS Path* of the form $\{AS_k, \ldots, AS_0\}$, which is the ordered list of ASes to traverse to reach N. The AS path is constructed by successively propagating reachability information: each AS prepending its own AS number to the path (one or more times) before sending it to neighbors. Fig. 2 illustrates how routing information about network *200.12.0.0/16* is propagated between ASes. For instance, AS160 knows its traffic will traverse AS172, AS180 and AS200 before reaching the destination.

BGP is capable of enforcing policies based on various preferences and constraints. BGP policies affect the route selection and export process, thereby controlling how traffic enters and leaves an AS. Each AS can define BGP policies according to its own criteria. BGP chooses the best route based on a number of metrics, such as the AS path length. In Fig. 2 AS172 chooses {*180, 200*} over {*190, 200, 200*} as the best route to N because its policy favors a shorter AS path.

Policy can be also applied to the route propagation process. An AS decides what to tell its neighbors. If an AS is unwilling to carry certain traffic for a neighbor, its policy will

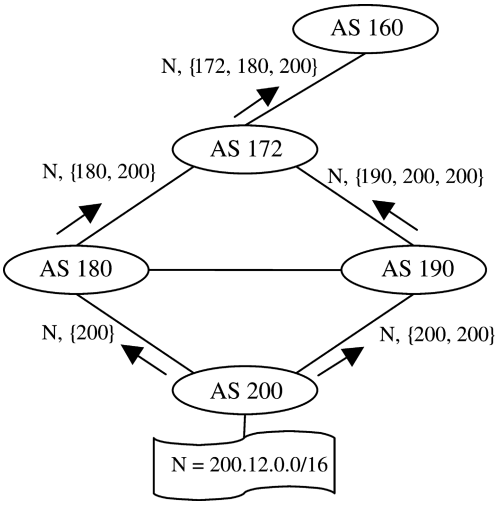

Fig. 2. BGP network example.

disallow routing advertisements about particular destinations being sent to the neighbor. For instance, AS180 and AS190 chose not to export routes to N to each other. As a result, the horizontal link between the two ASes will not be used to carry traffic to N. In a less restrictive case, AS200 tells AS190 about N, but prepends its AS number twice to make the path longer, indicating the route is considered a less attractive one. The policy eventually affects AS172's route selection process: it chose AS180 instead of AS190 as the next hop AS to reach N.

3. Challenges of setting up BGP

To set up BGP, network administrators configure individual routers in the AS using a configuration language. The following is a sample configuration in Cisco CLI format [3] for a router in AS160. The configuration involves originating routes, establishing peer relationship with neighbors, and applying policy filters. In this example, the router announces network *172.1.1.0/24* and peers with a remote BGP router in AS172. The policy filter allows only routes with an empty AS path (i.e. locally originated routers) to be advertised to AS172.

```
router bgp 160
  network 172.1.1.0 mask 255.255.255.0
  neighbor 172.16.24.1 remote-as 172
  neighbor 172.16.24.1 filter-list 1 out
!
ip as-path access-list 1 permit ^$
!
```

As BGP is a complex protocol, manually configuring individual routers is a time-consuming and error prone task. This is especially challenging in a large network with hundreds to thousands of routers. To maintain a consistent view of routing inside an AS, all BGP routers must be correctly configured to form a full-mesh or some well-structured internal hierarchy, such as route reflector clusters. At a lower-level, two BGP speakers must be able to talk to each other in order to exchange routing information. This seemingly obvious requirement has certain intricacies due to the fact that BGP relies on pre-existing connectivity provided by Interior Gateway Protocols (IGP) or static routes. For example, the remote peer address specified by a BGP router must match the outgoing interface of the IGP route used by the remote peer. Otherwise the connection will not be established, unless the remote peer explicitly specifies the matching interface. This type of implicit requirement can be easily overlooked by administrators, or violated due to change of the network.

Policy routing is an important functionality of BGP, but also provides numerous opportunities for configuration errors. In the face of this type of errors, BGP may continue to operate, but does not enforce the intended policy. Policy violation could lead to connectivity, security and economic problems. A well-known problem is address space hijacking, in which one AS accidentally announces networks "owned" by other ASes, forming a "blackhole" within the Internet. Other polices problems are commonly related to the commercial relationships an AS participates in. A multi-homed AS, for instance, shouldn't provide transit service to non-local traffic. The causes of errors are diverse, ranging from typographical errors to poor understanding of configuration semantics. An excellent empirical study of BGP policy configuration errors is presented in [9].

4. Service grammar for BGP

Correct BGP configuration means all routers in an AS achieve the joint goal of exchange routing information, maintaining a consistent view of routing and enforcing intended policies. However, there is quite a large conceptual gap between this global requirement and individual router configurations. Configuration errors arise because manual compilation of these high-level requirements into low-level "machine language" is difficult. If for some reason the global requirements are not satisfied there are no systematic tools to automatically diagnose configuration errors. Network administrators today manually perform these tasks.

Diagnosing why routers don't work together requires global reasoning about the logical structure of the network as well as dependencies between services. The BGP Service Grammar captures these global abstractions that are set up in the process of constructing routing services. By making these definitions explicit, network administrators can formally state high-level network-specific requirements and policies using these definitions.

A subset of the requirement language is shown in Table 1 followed by detailed explanations. The requirements fall into two categories: connectivity and policy.

Table 1
Service grammar for BGP

BGP Requirements Language	
Requirement	**Meaning**
`ibgp_session (RouterA, RouterB, LocalAS)`	Both RouterA and RouterB are BGP speakers of the local AS. A BGP session can be successfully established between them.
`ebgp_session (LocalRouter, LocalAS, RemoteRouter, RemoteAS)`	A BGP session can be successfully established between the local BGP speaker and the remote BGP speaker.
`reflector_client_session (ReflectorRouter, ClientRouter, LocalAS)`	In addition to ibgp_session requirements, the reflector is set up to forward routing updates from other IBGP peers to the client.
`cluster (Reflectors, Clients, LocalAS)`	Reflectors and clients form a cluster, i.e. reflectors are fully meshed and all clients are peered with all reflectors.
`as_full_mesh (Clusters, Non-clients, LocalAS)`	Clusters and non-clients form an AS, i.e. all reflectors and non-clients are fully-meshed. No client peers with a non-client.
`route_originate (Subnets, LocalAS)`	The LocalAS originates routes represented by subnets.
`link_to_provider (LocalRouter, RemoteRouter)`	The session represents a link to the local AS's provider. On this session, the local AS should accept everything, but only announce its own routes.
`link_to_customer (LocalRouter, RemoteRouter)`	The session represents a link to the local AS's customer. On this session, the local AS should announce everything, but only accept the customer's routes.
`link_to_peer (LocalRouter, RemoteRouter)`	The session represents a link to the local AS's peer. On this session, the local AS should only announce its customers' routes, and only accept the peer's customers' routes.
`provider_as (LocalAS, RemoteAS)`	RemoteAS is a provider of LocalAS.
`customer_as (LocalAS, RemoteAS)`	RemoteAS is a customer of LocalAS.
`peer_as (LocalAS, RemoteAS)`	RemoteAS is a peer of LocalAS.
`preferred_outgoing_link (LocalRouter, RemoteRouter, RemoteDestination)`	The session is the preferred outgoing link to reach a remote destination, expressed in either subsets or ASPath.
`preferred_incoming_link (LocalRouter, RemoteRouter, LocalDestination)`	The session is the preferred incoming link to reach a local destination, expressed in either subsets or ASPath.
`preferred_neighbor_entry (LocalRouter, RemoteRouter, LocalDestination)`	The session is the preferred entry from the neighbor AS to reach a local destination, expressed in either subsets or ASPath.

ibgp_session(*RouterA, RouterB, LocalAS*)

- *Meaning: Both RouterA and RouterB are BGP speakers of the local AS. A BGP session can be successfully established between them so they can exchange routing information.*
- *Diagnosis procedure:*
  1. *RouterA.as_num == LocalAS RouterB.as_num == LocalAS*
  2. ∃ *Ia, Ib, Na, Nb, s.t*
     - *Ia* ∈ *RouterA.interfaces*
     - *Ib* ∈ *RouterB.interfaces*
     - *Na* ∈ *RouterA.neighbors*
     - *Nb* ∈ *RouterB.neighbors*
     - *Ia.ip_addr == Nb.peer_address*
     - *Ib.ip_addr == Na.peer_address*
     - *Na.remote_as==LocalAS==Nb.remote_as*
     - *RouterA has an IGP route Ir_a to reach Ib*
     - *RouterB has an IGP route Ir_b to reach Ia*
     - *If Ia is a loopback interface then Na.update_source == Ia, else Ia is the outgoing interface of Ir a*

Fig. 3. IBGP session diagnosis procedure.

4.1. Connectivity requirements

The language provides two basic primitives—`ibgp_session` and `ebgp_session`—for describing BGP neighbor relationships. We outline the diagnosis procedure for `ibgp_session` in Fig. 3.

Regarding establishing BGP neighbor relationship, the types of configuration errors that can arise include:

- Incorrect AS number or neighbor address at two session end points, peer values are not mirror images of each other (usually typos).
- The neighbor's address is not reachable via IGP. This can happen when a loop-back interface is used but that interface does not participate in any IGP.
- A router tries to connect to a reachable interface of a remote neighbor, but the neighbor uses a different outgoing interface in the reverse IGP route. This happens when the neighbor has multiple reachable interfaces.

Any of the above errors can lead to connectivity problems preventing the BGP session from being established. This example demonstrates that even a very basic BGP requirement implies a number of assumptions and global relationships that the administrator must keep in mind and configure correctly on every router. There are many places for errors. The diagnosis engine systematically validates these assumptions and global relationships, catching all potential errors and providing useful information for the debugging process.

cluster(Reflectors, Clients, LocalAS)

- *Meaning: Reflectors and clients form a cluster, i.e. reflectors are fully meshed and all clients are peered with all reflectors.*
- *Diagnosis procedure:*
  1. $\forall A \in Reflectors, \forall B \in Clients,$ *reflector_client_session(A, B, LocalAS) is TRUE*
  2. $\forall X, \forall Y (X \neq Y) \in Reflectors$ *ibgp_session(X, Y, LocalAS) is TRUE*
  3. All reflectors have the same *cluster_id*
  4. $\forall C \in Clients$
     $\forall N \in C.neighbors$
     if $(N.remote\_as == LocalAS)$ && $(N \notin Reflectors \cup Clients)$
     return FALSE;
     / Clients shouldn't have IBGP sessions to non-clients */*

Fig. 4. Cluster diagnosis procedure.

Notice these basic primitives are already higher-level than raw router configurations. They can be used to compose other higher-level requirements that specify an AS′ logical structure, such as `reflector_client_session` and `cluster`.

Fig. 4 illustrates how to validate if a group of routers form a cluster. The algorithm verifies three global properties: all reflectors are fully meshed, all clients can receive updates from all reflectors, and every client only has BGP sessions with routers in the same cluster. IBGP session test is embedded as part of the procedure.

4.2. Policy requirements

Routing policies are configured via policy filters. A filter consists of a match criteria and a set of actions. Nearly all attributes of a routing update can be used to specify the match criteria, with AS path and network prefix being the most common ones. When a routing update satisfies the conditions set in the match criteria, associated actions (permit, deny, or modify) are invoked to control the propagation of the update. A filter can be applied to route origination, import and export process. It serves as the low-level building block for composing arbitrary routing policies. Our BGP Service Grammar supports the direct use of low-level filters to specify policy requirements. What we highlight in this section is the grammar that describes global AS-level properties, rather than that of an individual filter. These abstractions give network administrators a set of templates for defining common

link_to_provider(*LocalRouter, RemoteRouter*)

- *Meaning: The session between LocalRouter L and RemoteRouter R is a link to the provider of LocalAS.*
- *Diagnosis procedure:*

```
Let P = Get_provider_AS(L.as_num)
    ∪ Get_peer_AS(L.as_num)
Let N ∈ L.neighbors corresponding to R
∀p ∈ P
  Construct a route update r, let r.
as_path="_p_"
  if (Routemap_Eval(N.map_out, r) ≠
DENY)
      return FALSE;
/* LocalAS shouldn't leaks routes
learned from
      providers back to providers */
```

Fig. 5. Link to provider diagnosis procedure.

AS routing policies at high-level. Low-level filters can then be used for further refinement. We believe such a design would largely reduce the need for administrators to go into the low-level configuration details of each filter.

Typical commercial relationships between two neighboring ASes can be characterized as *customer, provider, or peer*, as defined in [4]. To test if a remote AS is a customer (provider, or peer) of the local AS, we need to verify that the relationship holds on all sessions between the two ASes. Fig. 5 outlines the diagnosis procedure for `link_to_provider`. When exporting routers to a provider, an AS exports its own and its customer routers, but usually does not export routes learned from providers or peers. A properly configured export filter on this session should block those routes. For each provider and peer AS, the diagnosis procedure constructs an AS path containing the AS number, and feeds it to the export filter. Any of these paths passing the filter is a violation of the policy. In that case, the diagnosis procedure fails. This procedure uses several utilities functions. `Get_Provider_AS` returns the set of ASes that are marked as a provider of the local AS. `Routemap_Eval` mimic the processing of a policy filter on a route update.

When multiple routes to a remote destination (network or AS) exist, one link is usually designated as the primary route and others serve as backup. Such a policy can be expressed with `preferred_outgoing_link`. Its diagnosis procedure (Fig. 6) examines the import filter on all EBGP sessions. For each session, the procedure calculates the **local-preference** that a route update to the remote destination would get if it arrives on this session. To pass the test, the import filter on the preferred session must be the one that generates the highest **local-preference**.

preferred_outgoing_link(*LocalRouter, RemoteRouter, RemoteDestination*)

- *Meaning: The session between LocalRouter L and RemoteRouter R is the preferred link for outgoing traffic to RemoteDestination D.*
- *Diagnosis procedure:*

```
Let S = Get_EBGP_Sessions(L.as_num)
Construct a route update r, let r.NLRI = D
Let N ∈ L.neighbors corresponding to R
Let H = Routemap_Eval(N.map_in,
r).local_pref
∀s ∈ S, s≠{L, R}
 if (Routemap_Eval(s.local.map_in,
r).local_pref > H)
    return FALSE;
 /* Another session is more preferable to
  this one */
```

Fig. 6. Preferred outgoing link diagnosis procedure.

Both `preferred_incoming_link` and `preferred_neighbor_entry` are used to control incoming traffic by designating a primary route to a local destination. The difference is that the latter only concerns two neighboring ASes. The diagnosis procedures are similar. Both procedures examine export filters, except that the former looks for the filter that generates the shortest AS path, while the latter looks for the one that generates the lowest multi-exit-discriminator (**med**).

5. Sample network study

We have designed an experimental BGP network consisting of 9 CISCO routers in 5 ASes, shown in Fig. 7. The goal is to demonstrate different BGP architecture, peering relationship and routing policies.

Under this setup:

- AS172 represents a large service provider with 4 routers, 3 of which are BGP speakers. The 3 BGP speakers form a cluster with **PR3** being the reflector. Therefore IBGP peering between **CR3** and **CR4** is not required. Inside the AS OSPF is running as IGP.
- AS160 represents a small customer ISP. It connects to the Internet solely via AS172, and thus it is a stub.
- AS180 and AS190 represent two intermediate level service providers. They subscribe service from AS172 and provide connectivity for AS 200. They also enter a bilateral peering agreement.
- AS200 represents a multi-homed customer ISP with 2 BGP speakers. It has multiple links to AS180 and AS190.

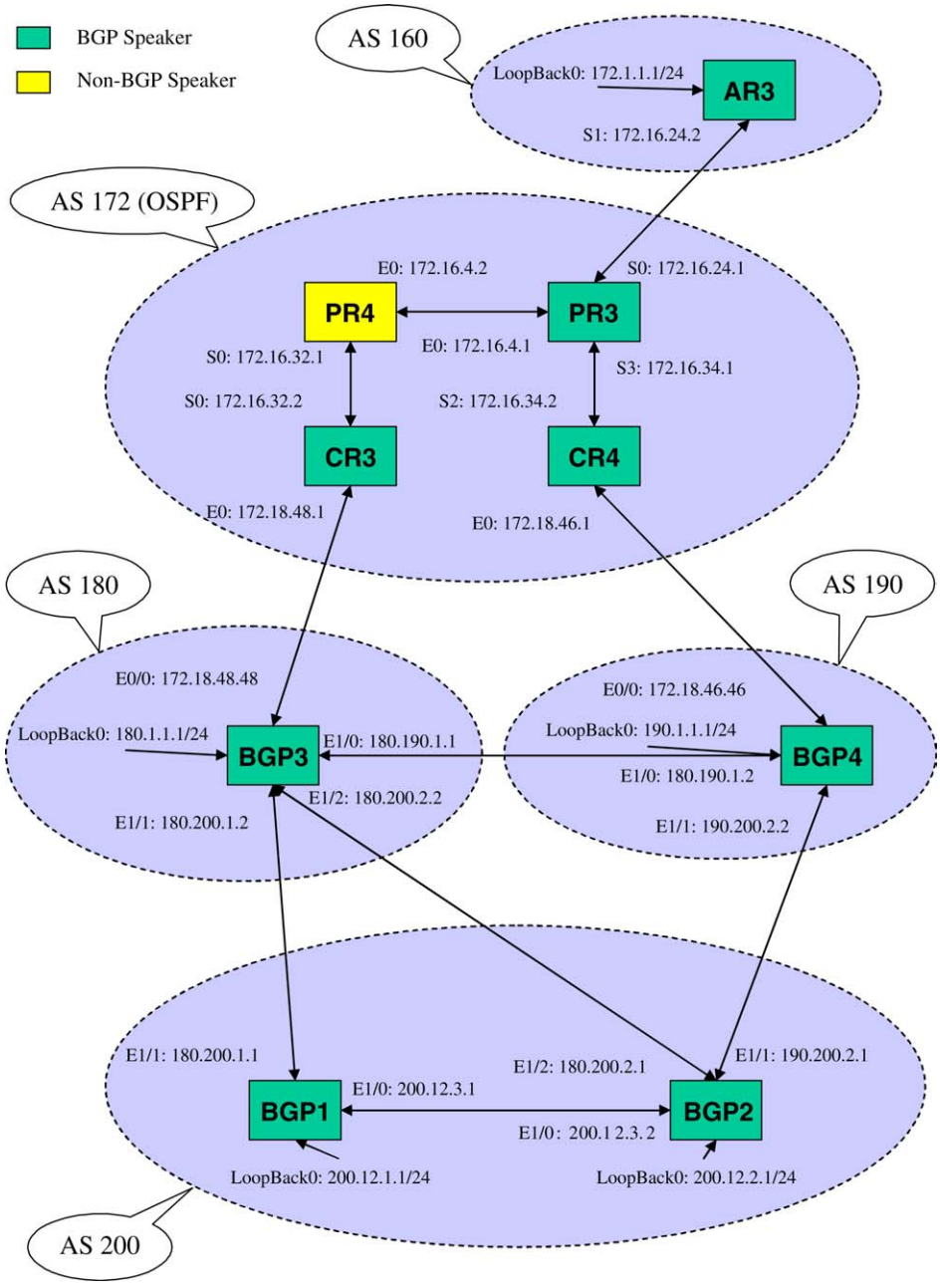

Fig. 7. Experimental setup.

Suppose AS200's network administrator wants to enforce the following policies:

1. AS200 announces two networks: *200.12.1.0/24* and *200.12.2.0/24*.
2. AS200 is a multi-homed AS. AS180 and AS190 are its providers.
3. AS190 is the preferred AS for outgoing traffic to AS172.
4. **BGP1** is the preferred Border Router for outgoing traffic to AS180.
5. **BGP1** is the preferred ingress Border Router for traffic to network *200.12.1.0/24* from AS180.
6. **BGP2** is the preferred ingress Border Router for traffic to network *200.12.2.0/24* from AS180.
7. AS180 is the preferred AS for all incoming traffic.

To realize these policies, network administrators first need to analyze the underlying requirements that support them. Typically, Policy 1 requires the two networks be originated by the two BGP speakers. Policy 2 requires outbound filters on every EBGP session that only allows locally originated routes to be advertised. Policies 3 and 4 require inbound filters to set up the **local-preference** attribute correctly. More precisely, routes to AS172 learned from AS190 should be given a higher **local-preference**, as should routes to AS180 learned via **BGP1**.

Policies 5 and 6 require manipulation of the **med** path attribute. Both **BGP1** and **BGP2** advertise network *200.12.1.0/24* and *200.12.2.0/24* to **BGP3**. **BGP1** should give a more favorable **med** value to network *200.12.1.0/24* than to *200.12.2.0/24*, and **BGP2** should do the opposite. **BGP3** then is able to decide which router is the best to reach these networks based on the metric. The **med** value should be set by outbound filters of **BGP1** and **BGP2**.

For Policy 7, a common practice is for AS200 to prepend its own AS number to all updates sending to AS190. This would discourage incoming traffic from going through AS190 because everything else being equal BGP will select the route with the shortest AS Path.

Based on these requirements, administrators then choose the appropriate configuration commands and parameter values for each router to satisfy them. Figs. 8 and 9 give a snapshot of the working configuration of **BGP1** and **BGP2**.

The configuration errors that can arise include (in fact, we inadvertently made most of them in setting up our network):

1. Forget to configure the static route to the loopback interface of **BGP1** and **BGP2**. Neighbor address becomes unreachable. BGP session could not be established.
2. Forget to specify the **update-source** in the **neighbor** command. TCP connection is rejected and BGP session could not be established.
3. Forget the AS path access list. AS200 becomes a transit AS of AS180 and AS190.
4. Incorrect route maps and filters due to misunderstanding of the syntax of regular expression and meaning of path attributes. For example, a lower **med** value is considered better, which is in contrast to a higher **local-preference** favored in the route selection process.
5. For Policies 5 and 6, **BGP1** and **BGP2** should give, respectively, more and less favorable values to **med**. It is entirely possible that both give equally favorable values.

```
!
hostname BGP1
!
router bgp 200
 no synchronization
 network 200.12.1.0
 neighbor 180.200.1.2 remote-as 180
 neighbor 180.200.1.2 route-map SETLOCALIN in
 neighbor 180.200.1.2 route-map SETMEDOUT out
 neighbor 180.200.1.2 filter-list 1 out
 neighbor 200.12.2.1  remote-as 200
 neighbor 200.12.2.1  update-source Loopback0
!
ip as-path access-list 1 permit ^$
ip as-path access-list 2 permit 180$
!
access-list 1 permit 200.12.1.0 0.0.0.255
route-map SETLOCALIN permit 10
 match as-path 2
 set local-preference 400
!
route-map SETLOCALIN permit 20
 set local-preference 100
!
route-map SETMEDOUT permit 10
 match ip address 1
 set metric 10
!
route-map SETMEDOUT permit 20
 set metric 20
!
ip route 200.12.2.0 255.255.255.0 200.12.3.2
!
```

Fig. 8. BGPI configuration.

6. For Policy 7, AS200 should prepend its own AS number in all updates to AS190, and this rule should be enforced both at **BGP1** and **BGP2**. If it is forgotten at one router, Policy 7 will not be implemented.

This example shows that manual compilation of high-level requirements into low-level configuration is a rather demanding and error-prone process. Even for a small network, the resulting configuration is already complex. It is not so obvious how each individual command relates to intended policies. A large network has many more routers to manage and much more complicated policies in place. The service requirement also changes more frequently. It will be even harder for the administrator to keep the mental map of how each policy is effected on each individual router, and make sure adding or modifying devices and services does not violate existing requirements.

Using Service Grammar, the global requirements of AS200 can be described as shown in Table 2.

```
!
hostname BGP2
!
router bgp 200
 no synchronization
 network 200.12.2.0
 neighbor 180.200.2.2 remote-as 180
 neighbor 180.200.2.2 route-map SETMEDOUT out
 neighbor 180.200.2.2 filter-list 1 out
 neighbor 190.200.2.2 remote-as 190
 neighbor 190.200.2.2 route-map SETLOCALIN in
 neighbor 190.200.2.2 route-map SETASPATH out
 neighbor 190.200.2.2 filter-list 1 out
 neighbor 200.12.1.1 remote-as 200
 neighbor 200.12.1.1 update-source Loopback0
!
ip as-path access-list 1 permit ^$
ip as-path access-list 2 permit 172$
!
access-list 1 permit 200.12.2.0 0.0.0.255
route-map SETLOCALIN permit 10
 match as-path 2
 set local-preference 300
!
route-map SETLOCALIN permit 20
 set local-preference 100
!
route-map SETMEDOUT permit 10
 match ip address 1
 set metric 10
!
route-map SETMEDOUT permit 20
 set metric 30
!
route-map SETASPATH permit 10
 set as-path prepend 200 200
!
ip route 200.12.1.0 255.255.255.0 200.12.3.1
!
```

Fig. 9. BGP2 configuration.

The description is concise and hides most low-level details. More importantly, it highlights the constraints spanning multiple routers that have to be enforced. Keeping track of such global constraints in low-level configuration language would be much harder. The diagnosis engine can effectively identify the configuration errors listed above. For instance, the first two errors will cause `ibgp_session_test` to fail. A missing AS path access list can be detected by `link_to_provider`. Similarly, misconfigured route-maps are caught by policy grammar rules.

Table 2
AS200 service grammar

- ***bgpAS200*** *:-*
  AS200basicConnectivity, AS200policies.
- ***AS200basicConnectivity*** *:-*
  ibgp_session(*BGP1, BGP2, AS200*),
  ebgp_session(*BGP1, AS200, BGP3, AS180*),
  ebgp_session(*BGP2, AS200, BGP3, AS180*),
  ebgp_session(*BGP2, AS200, BGP4, AS190*),
  as_full_mesh({}, {*BGP1, BGP2*}, *AS200*).
- ***AS200policies*** *:-*
  policy1, policy2, policy3, policy4, policy5, policy6, policy7.
- ***policy1*** *:-*
  route_originate({*200.12.1.0/24, 200.12.2.0/24*}, *AS200*).
- ***policy2*** *:-*
  provider_AS(*AS200,AS180*),
  link_to_provider(*BGP1, BGP3*),
  link_to_provider(*BGP2, BGP3*),
  provider_AS(*AS200,AS190*),
  link_to_provider([*BGP2, BGP4*).
- ***policy3*** *:-*
  preferred_outgoing_link(*BGP2, BGP4, AS172*).
- ***policy4*** *:-*
  preferred_outgoing_link(*BGP1, BGP3, AS180*).
- ***policy5*** *:-*
  preferred_neighbor_entry(*BGP1, BGP3, 200.12.1.0/24*).
- ***policy6*** *:-*
  preferred_neighbor_entry(*BGP2, BGP3, 200.12.2.0/24*).
- ***policy7*** *:-*
  preferred_incoming_link(*BGP1, BGP3, ALL*),
  preferred_incoming_link(*BGP2, BGP3, ALL*).

6. Related work

Our system provides a language to express the logical structure of an AS and its BGP policies. It can automatically check expressions in this language against router configurations and thereby provide a useful diagnosis service, which has not been available to date. The main difference between this approach and previous diagnosis systems is that we can describe the BGP architecture of an AS in a high level language and check that it has been correctly configured. If someone changes a router configuration, he can just run the diagnosis again to ensure that the logical structure and policies have not been violated. In Netsys [13], there is no way to describe the administrator's *intention*, i.e. *network-specific* policies and structure. It just runs a collection of network-invariant tests.

Routing Policy Specification Language (RPSL) [1] allows a network operator to specify routing policies in a high-level language. The goal of RPSL is to contain a view of the global routing policy in a single cooperatively maintained distributed database so that the integrity of the Internet's routing can be checked. RPSL shares some common views with our approach. We feel Service Grammar is better suited for diagnosing configuration errors

because of its expressive power. In addition to policies, Service Grammar can also describe AS structure, non-BGP properties, which provide important diagnostic information but are not available with RPSL.

Our system focuses on diagnosis because we believe in the short run diagnosis is more important than service provision—it gives administrators the desirable level of control and predictability, and can be used immediately. In the long run, it is natural to extend our system for service provision. Given a comprehensive service specification, it can be compiled into vendor-neutral component configurations. Vendor-specific adaptors can then be applied to generate low-level router configuration commands. Reference [11] demonstrates a system that implements Service Grammar rules in Prolog. Because of the relational nature of Prolog, service specification simultaneously serves provisioning purposes. Another system that performs a set of specialized provisioning tasks is described in [5].

Reference [6] studies IBGP routing anomalies and proposes sufficient conditions that guarantee correctness. These conditions can be incorporated into our Service Grammar as policy templates. Reference [9] presents an empirical study of BGP misconfigurations. Several classes of errors, such as reliance on upstream filtering, forgotten filter, incorrect summary, bad route map, etc., are caused by simple high-level policies that are not obvious for operators to express at the CLI level. Our system would reduce these types of errors given the high-level, system-wide requirements.

7. Summary

Our Service Grammar system enables a new way of configuration error diagnosis in a distributed environment. Network administrators can express the global requirements in a high-level language. The diagnosis engine then systematically verifies if the expressions in this language are satisfied by device configurations in a top-down fashion. Our system highlights global reasoning—i.e. why a group of components fail to jointly compose the intended service—rather than why a single component fails.

We demonstrate how to use such a system to diagnose BGP configuration errors in an AS. Previous solutions to diagnosing configuration errors have been network invariant in that they contain a fixed set of constraints that must be satisfied by every BGP network. However, BGP requirements such as logical architecture and policies differ widely from one network to another. It is not possible in previous techniques to check if component configurations implement the network-specific requirements. Our language consists of a small set of abstractions that can be composed to describe most BGP features.

One limitation of this approach is that Service Grammars and diagnosis procedures must be developed for each protocol of interest. We have prototyped Service Grammars for RIP, OSPF, BGP, BGP/MPLS, PIM, GRE, IPSEC, DiffServ and the Spread group communication protocol [15]. The difference between grammars tends to be significant because they are very protocol-specific. Based on our experience, the amount of effort required to develop the Service Grammar for a protocol is not terribly large. The notion of correct configuration is already implicit in the definition of protocols since their intended use is a part of the definition. The job of the Service Grammar designer is essentially to make this “configuration logic” explicit by analyzing these definitions.

The challenge for end users is that they need to go through another learning curve, and may still write incorrect specifications in this language. However, we believe the chances of errors should be lowered in this high-level language as opposed to low-level configuration commands.

References

[1] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, M. Terpstra, Routing Policy Specification language (RPSL), Request for Comments 2622, June, 1999.
[2] M. Barton, D. Atkins, S. Narain, D. Ritcherson, K. Tepe, Integration of IP mobility and security for secure wireless communications, in: Proceedings of IEEE International Communications Conference, New York, NY, 2002.
[3] BGP Commands, http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/np1_r/1rprt1/1rbgp.htm.
[4] L. Gao, J. Rexford, Stable internet routing without global coordination, in: Proceedings of ACM SIGMETRICS, June, 2000.
[5] J. Gottlieb, A. Greenberg, J. Rexford, J. Wang, Automated provisioning of BGP customers, IEEE Network, no. 6, November, 2003.
[6] T.G. Griffin, G. Wilfong, On the correctness of IBGP configuration, in: Proceedings of ACM SIGCOMM, August, 2002.
[7] P. Horn, Autonomic Computing: IBM's Perspective on the State of Information Technology, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.
[8] B. Lampson, Computer security in the real world, in: Proceedings of Annual Computer Security Applications Conference, 2000.
[9] R. Mahajan, D. Wetherall, T. Anderson, Understanding BGP misconfigurations, in: Proceedings of ACM SIGCOMM, August, 2002.
[10] S. Narain, A. Shareef, M. Rangadurai, Diagnosing configuration errors in virtual private networks, in: Proceedings of IEEE International Communications Conference, Helsinki, Finland, 2001.
[11] S. Narain, T. Cheng, B. Coan, V. Kau, K. Parmeswaran, W. Stephens, Building autonomic systems via configuration, in: Proceedings of AMS Autonomic Computing Workshop, Seattle, WA, June, 2003.
[12] S. Narain, R. Vaidyanathan, S. Moyer, W. Stephens, K. Parmeswaran, A. Shareef, Middleware for building adaptive systems via configuration, in: Proceedings of ACM SIGPLAN Workshop on Optimizing Middleware, Salt Lake City, UT, June, 2001.
[13] Netsys, http://www.cisco.com/warp/public/cc/pd/nemnsw/nesvmn/index.shtml.
[14] Y. Rekhter, T. Li, A Border Gateway Protocol 4 (BGP-4), Request for Comments 1771, March, 1995.
[15] The Spread Toolkit, http://www.spread.org/.