Internet Draft
Expires January 29, 1998
draft-ietf-rps-appl-rpsl-01.txt

David Meyer
University of Oregon
Joachim Schmitz
DFN-NOC
Cengiz Alaettinoglu
USC/ISI
July 29, 1997

Application of Routing Policy Specification Language (RPSL) on the Internet

Status of this Memo

This document is an Internet Draft, and can be found as draft-ietf-rps-appl-rpsl-01.txt in any standard internet drafts repository. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material, or to cite them other than as a "working draft" or "work in progress."

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

1 Introduction

This document is a tutorial on using the Routing Policy Specification Language (RPSL) to specify routing policies. It covers specifying routing policies using RPSL, registering these policies in the Internet Routing Registry (IRR), and analysing them using the routing policy analysis tools, for example to generate vendor specific router configurations. This document is targeted towards an Internet/Network Service Provider (ISP/NSP) engineer who is new to RPSL and to IRR. Readers are referred to the RPSL reference document [3] for completeness. We recommend reading this document before reading the reference document. We hope that for many cases, this document will be sufficient.

IRR is a repository of routing policies. It currently consists of five sites: CA*Net registry in Canada, ANS, MCI and RADB registries in the United States of America, and RIPE registry in Europe. Each of these sites run independent of each other. However, each site exchanges its data with each other at some frequency (at least once a day or as often as every ten minutes). MCI, Ca*Net and ANS are private registries and contain routing policies of MCI, Ca*Net, ANS, and their customers respectively. RADB and RIPE are public registries, and any ISP can publish their policies in these registries. Since registries exchange their data regularly, you need to register your policies in only one of them. If you are an MCI, ANS or CA*Net customer, we recommend you register your policies with them. Otherwise, you may register your policies either at the RIPE or RADB registry, whichever is closer to you. We recommend against registering in multiple registries since it often eventually leads to inconsistent data between the registries.

Routing policies registered in IRR are specified using RPSL. RPSL is based on an earlier language known as RIPE-181 [4, 5]. Through operational use of RIPE-181 it has become apparent that certain policies cannot be specified and a need for an enhanced and more general language is needed. RPSL addresses RIPE-181's limitations. RPSL obsoletes RIPE-181 [4, 5].

RPSL is object oriented; that is, objects contain pieces of policy and administrative information. For example, each address prefix routed in the inter-domain mesh is specified in a `route` object and policies of each AS are specified in an `aut-num` object. Objects have relations between each other. For example, all `route` objects of an ISP refer to the `aut-num` object of the ISP. These relations form sets of objects, we can then use these sets to specify policies collectively to all their members. For example, we can specify policy against all routes of an ISP, by refering to the AS number of the ISP. In the following sections, we will describe each of these objects (rather object classes) in more detail and give numerous examples for you to create your own objects. In most cases, you should be able to cut and paste our examples to create your own objects.

Once you register your policies in IRR, they are available for others to query using a whois service. Figure 1 illustrates the use of the `whois` command to obtain the route object for `128.223.0.0/16`. The output of the `whois` command is the ASCII representation of the `route` object. We will describe the details of the `route` object in Section 2.3.

```
% whois -h radb.ra.net 128.223.0.0/16
  route:        128.223.0.0/16
  descr:        UONet
  descr:        University of Oregon
  descr:        Computing Center
  descr:        Eugene, OR 97403-1212
  descr:        USA
  origin:       AS3582
  mnt-by:       MAINT-AS3582
  changed:      meyer@ns.uoregon.edu 19960222
  source:       RADB
```

Figure 1: `whois` command and a route object.

That is not all, once you register your policies in IRR, they can be analyzed for consistency or used to diagnose Internet's operational routing problems. RAToolSet [1] is a suite of tools for analyzing this data. It contains tools (`RtConfig`) to configure routers, tools (`prpath` and `prtraceroute`) to analyse paths on the Internet, tools (`roe`, `aoe` and `prcheck`) to compare, validate and register RPSL objects, and others.

The remainder of this document is organized as follows: Section 2 introduces the fundamental RPSL objects. Section 3 discusses implementation of various common policies using RPSL. Finally, Section 4 describes the use of `RtConfig` to generate vendor specific router configurations.

2 RPSL Objects

This section introduces the fundamental RPSL objects required to implement many typical Internet routing policies. The basic elements are:

- maintainer objects (`mntner`)
- autonomous system number objects (`aut-num`)
- route objects (`route`)
- set objects (`as-set`, `route-set`)

and they are described in the following sections. These objects must be registered in the IRR, in only one of the existing registries. In general, registration is done by sending electronic mail to a registry robot. The email addresses for existing registries are listed in Figure 2. The contents of the mail consists of the objects you want to have registered, separated by empty lines, and often some kind of authorization (see below). The registry robot automatically processes your mail, entering new objects into the database, deleting old ones, or activating changes. Moreover, it may send notifications and replies with an error or success report about its actions. The first object which has to be registered, normally is the `mntner`. In general, to have it properly authenticated, a maintainer object is added manually by registry staff. Afterwards, all other actions should be done through the registry robot. Each registry provides documentation on how to use it. If problems arise your registry staff is willing to assist you.

ANS	auto-dbm@ans.net
CANET	auto-dbm@canet.net
MCI	auto-rr@mci.net
RADB	auto-dbm@radb.ra.net
RIPE	auto-dbm@ripe.net

Figure 2: Registry email addresses to register policy objects in IRR.

2.1 The Maintainer Object

The maintainer object is used to introduce some kind of authorization for registrations. It lists various contact persons and describes security mechanisms that will be applied when updating objects in the IRR. Registering a `mntner` object is the first step in creating policies for an AS. An example is shown in Figure 3. The maintainer is called `MAINT-AS3701`. The contact person here is the same for administrative (`admin-c`) and technical (`tech-c`) issues and is referenced by the NIC-handle `DMM65`. NIC-handles are unique identifiers for persons in registries. Refer to registry documentation for further details on person objects and usage of NIC-handles.

The example shows two authentication mechanisms: `CRYPT-PW` and `MAIL-FROM`. `CRYPT-PW` takes as its argument a password that is encrypted with Unix crypt(3) routine. When sending updates, the maintainer adds the field `password: <cleartext password>` to the beginning of any requests that are to be authenticated. `MAIL-FROM` takes an argument that is a regular expression which covers a set of mail addresses. Only users with any of these mail addresses are authorized to work with objects secured by the corresponding maintainer[1].

The security mechanisms of the `mntner` object will only be applied on those objects referencing a specific `mntner` object. The reference is done by adding the attribute `mnt-by` to an object using the name of the `mntner` object as its value. In Figure 3, the maintainer `MAINT-AS3701` is maintained by itself.

```
mntner:     MAINT-AS3701
descr:      Network for Research and Engineering in Oregon
remark:     Internal Backbone
admin-c:    DMM65
tech-c:     DMM65
upd-to:     noc@nero.net
auth:       CRYPT-PW  949WK1mirBy6c
auth:       MAIL-FROM .*@nero.net
notify:     noc@nero.net
mnt-by:     MAINT-AS3701
changed:    meyer@antc.uoregon.edu 970318
source:     RADB
```

Figure 3: Maintainer Object

2.2 The Autonomous System Object

The autonomous system object describes the import and export policies of an AS. Each organization registers an autonomous system object (`aut-num`) in the IRR for its AS. Figure 4 shows the `aut-num` for AS3582 (UONET).

The autonomous system object lists contacts (`admin-c`, `tech-c`) and is maintained by (`mnt-by`) `MAINT-AS3701` which is the maintainer displayed in Figure 3.

The most important attributes of the `aut-num` object are `import` and `export`. The `import` clause of an `aut-num` specifies import policies, while the `export` clause specifies export policies. The corresponding clauses allow a very detailed description of the routing policy of the AS specified. The details are given in section 3.

With these clauses, an `aut-num` object shows its relationship to other autonomous systems by describing its peerings. In addition, it also defines a routing entity comprising a group of IP networks which are handled according to the rules defined in the `aut-num` object. Therefore, it is closely linked to `route` objects.

In this example, AS3582 imports all routes from AS3701 by using the keyword `ANY`. AS3582 imports only internal routes from AS4222, AS5650, and AS1798. The import policy for for AS2914 is slightly more complex. Since AS2914 provides transit to various other ASes, AS3582 accepts routes with ASPATHs that begin with AS2194 followed by members of AS-WNA, which is an as set (see section 2.4.1 below) describing those customers that transit AS2914.

[1] Clearly, neither of these mechanisms is sufficient to provide strong authentication or authorization. Other public key (e.g., PGP) authentication mechanisms are available from some of the IRRs.

Since AS3582 is a multi-homed stub AS (i.e., it does not provide transit), its export policy consists simply of "`announce AS3582`" clauses; that is, announce internal routes of `AS3582`. These routes are those in `route` objects where the `origin` attribute is `AS3582`.

```
aut-num:     AS3582
as-name:     UONET
descr:       University of Oregon, Eugene OR
import:      from AS3701 accept ANY
import:      from AS4222 accept <^AS4222$>
import:      from AS5650 accept <^AS5650$>
import:      from AS2914 accept <^AS2914+ (AS-WNA)*$>
import:      from AS1798 accept <^AS1798$>
export:      to AS3701 announce AS3582
export:      to AS4222 announce AS3582
export:      to AS5650 announce AS3582
export:      to AS2914 announce AS3582
export:      to AS1798 announce AS3582
admin-c:     DMM65
tech-c:      DMM65
notify:      nethelp@ns.uoregon.edu
mnt-by:      MAINT-AS3582
changed:     meyer@antc.uoregon.edu 970316
source:      RADB
```

Figure 4: Autonomous System Object

The `aut-num` object forms the basis of a scalable and maintainable router configuration system. For example, if AS3582 originates a new route, it need only create a route object for that route with origin AS3582. AS3582 can now build configuration using this route object without changing its `aut-num` object.

Similarly, if for example, AS3701 originates a new route, it need only create a route object for that route with origin AS3701. Both AS3701 and AS3582 can now build configuration using this route object without modifying its `aut-num` object.

2.3 The Route Object

In contrast to `aut-num` objects which describe propagation of routing information for an autonomous system as a whole, `route` objects define single routes from an AS. An example is given in Figure 5.

This `route` object is maintained by `MAINT-AS3582` and references `AS3582` by the `origin` attribute. By this reference it is grouped together with other routes of the same origin AS, becoming member of the routing entity denoted by `AS3582`. The routing policies can then be defined in the `aut-num` objects for this group of routes.

Consequently, the `route` objects give the routes from this AS which are distributed to peer ASes according to the rules of the routing policy. Therefore, for any route in the routing tables of the backbone routers a `route` object must exist in one of the registries in IRR. `route` objects must be registered in the IRR only for the routes seen outside your AS. Normally, this set of *external* routes is different from the routes

```
route:      128.223.0.0/16
descr:      UONet
descr:      University of Oregon
descr:      Computing Center
descr:      Eugene, OR 97403-1212
descr:      USA
origin:     AS3582
mnt-by:     MAINT-AS3582
changed:    meyer@ns.uoregon.edu 960222
source:     RADB
```

Figure 5: Example of a `route` object

internally visible within your AS. One of the major reasons is that external peers need no information at all about your internal routing specifics. Therefore, external routes are in general aggregated combinations of internal routes, having shorter IP prefixes where applicable according to the CIDR rules. Please see the CIDR FAQ [8] for a tutorial introduction to CIDR. It is strongly recommended that you aggregate your routes as much as possible, thereby minimizing the number of routes you inject into the global routing table and at the same time reducing the corresponding number of `route` objects in the IRR.

While you may easily query single `route` objects using the `whois` program, and submit objects via mail to the registry robots, this becomes kind of awkward for larger sets. The RAToolSet [1] offers several tools to make handling of `route` objects easier. If you want to read policy data from the IRR and process it by other programs, you might be interested in using `peval` which is a low level policy evaluation tool. As an example, the command

```
peval -h radb.ra.net -expand_all AS3582
```

will give you all `route` objects from AS3582 registered with RADB.

A much more sophisticated tool from the RAToolSet to handle `route` objects interactively is the route object editor `roe` (please see Figure 6). It has a graphical user interface to view and manipulate `route` objects registered at any IRR. New `route` objects may be generated from templates and submitted to the registries. Moreover, the `route` objects from the databases may be compared to real life routes. Therefore, `roe` is highly recommended as an interface to the IRR for `route` objects. Further information on `peval` and `roe` is available together with the RAToolSet [1].

2.4 Set Objects

With routing policies it is often necessary to reference groups of autonomous systems or routes which have identical properties regarding a specific policy. To make working with such groups easier RPSL allows to combine them in set objects. There are two basic types of predefined set objects, `as-set`, and `route-set`. The RPSL set objects are described below.

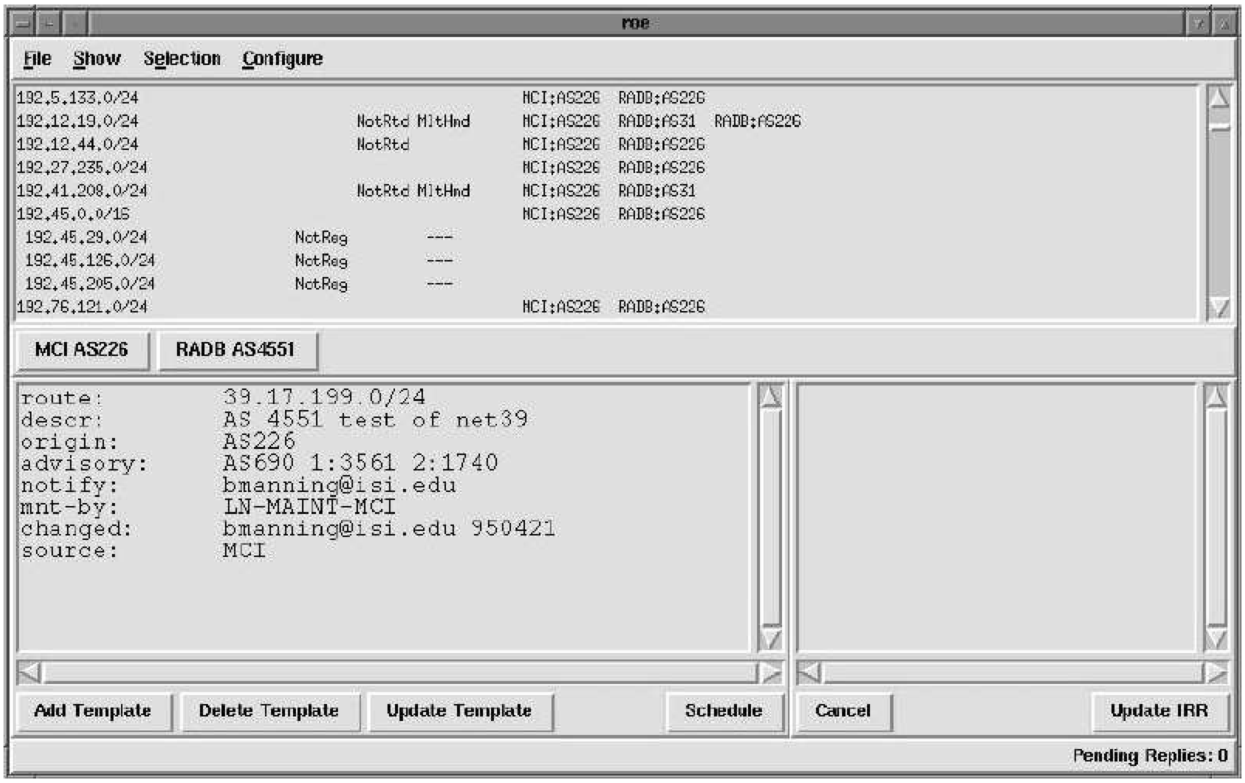

Figure 6: route object editor (`roe`).

2.4.1 AS-SET Object

Autonomous system set objects (`as-set`) are used to group autonomous system objects into named sets. An `as-set` has an RPSL name that starts with “AS-”. In the example in Figure 7, an `as-set` called `AS-NERO-PARTNERS` and containing AS3701, AS4201, AS3582, AS4222, AS1798 is defined. The `as-set` is the RPSL replacement for the RIPE-181 `as-macro`. It has been extended to include ASes in the set indirectly by referencing as set names in the `aut-num` objects.

AS-SETs are particularly useful when specifying policies for groups such as customers, providers, or for transit. You are encouraged to register sets for these groups because it is most likely that you will treat them alike, i.e. you will have a very similar routing policy for all your customers which have an autonomous system of their own. You may as well discover that this is also true for the providers you are peering with, and it is most convenient to have the ASes combined in one `as-set` for which you offer transit. For example, if a transit provider specifies its import policy using its customer’s `as-set` (i.e., its `import` clause for the customer contains the customer’s `as-set`), then that customer can modify the set of ASes that its transit provider accepts from it. Again, this can be accomplished without requiring the customer or the transit provider to modify its `aut-num` object.

The ASes of the set are simply compiled in a comma delimited list following the `members` attribute of the `as-set`. This list may also contain other AS-SETs.

```
as-set:    AS3582:AS-PARTNERS
members:   AS3701, AS4201, AS3582, AS4222, AS1798
```

Figure 7: `as-set` Object

2.4.2 ROUTE-SET Object

A `route-set` is a way to name a group of routes. The syntax is similar to the `as-set`. A `route-set` has an RPSL name that starts with "RS-". The `members` attribute lists the members of the set. The value of a `members` attribute is a list of address prefixes, or route-set names. The members of the `route-set` are the address prefixes or the names of other route sets specified.

Figure 8 presents some example `route-set` objects. The set `rs-uo` contains two address prefixes, namely `128.223.0.0/16` and `198.32.162.0/24`. The set `rs-bar` contains the members of the set `rs-uo` and the address prefix `128.7.0.0/16`.

```
route-set: rs-uo
members: 128.223.0.0/16, 198.32.162.0/24

route-set: rs-bar
members: 128.7.0.0/16, rs-uo

route-set: AS3582:RS-MARTIANS
remarks: routes not accepted from any peer
members: 0.0.0.0/0,             # default route
         0.0.0.0/0^32,          # host routes
         224.0.0.0/3^4-32,      # multicast routes
         127.0.0.0/8^9-32, . . .
```

Figure 8: `route-set` Objects

3 Specifying Policies using RPSL

In this section we show how the various RPSL objects can be used to specify typical Internet policies.

3.1 Provider-Customer Policies

In typical customer-provider relationships, the customer imports all the routes that the provider has and exports its routes to the provider. The provider's policies are symmetrical in the sense that it exports all routes that it has to the customer, and it imports only the customers routes. Figure 9 illustrates one way of expressing these policies using RPSL where AS3701 is the provider and AS3582 is the customer. Refer to Figure 4 for AS3582's `aut-num`.

```
aut-num:     AS3701
as-name:     NERO-NET
descr:       Network for Engineering and Research in Oregon
...
import:      from AS3582 accept <^AS3582$>
export:      to AS3582 announce ANY
...
admin-c:     DMM65
tech-c:      DMM65
notify:      noc@nero.net
mnt-by:      MAINT-AS3701
changed:     meyer@antc.uoregon.edu 970316
source:      RADB
```

Figure 9: Provider-Customer Policies in RPSL

In this example, "`announce ANY`" means export any route that AS3701 has in its routing table, and "`accept <^AS3582$>`" means accept only AS3582's routes (i.e., that have AS-PATHs of length one, where the AS in the path is AS3582)[2]. Note that AS3582 is taking full routing from AS3701; this can be seen in that AS3701 is announcing "ANY", and AS3582 is accepting "ANY". The important point in this example is that if AS3582 adds or deletes route objects, there is no need to update the `aut-num` objects. The added (or deleted) objects will implicitly update AS3582's and AS3701's policies, and thus affect their router configuration files.

As mentioned above, if the customer is itself a provider, i.e. it has its own customers, the set of routes passed to the provider includes its customers' routes as illustrated in Figure 10. In this example, "`accept AS3582:AS-PARTNERS`" means that for each AS X in the set defined by `AS3582:AS-PARTNERS` accept AS X's routes.

```
aut-num:     AS4600
as-name:     NERO-TRANSIT
descr:       Network for Engineering and Research in Oregon
...
import:      from AS3701 accept <^A3701+ (AS3582:AS-PARTNERS)*$>
export:      to AS3701 announce ANY
...
admin-c:     DMM65
tech-c:      DMM65
notify:      noc@nero.net
mnt-by:      MAINT-AS4600
changed:     meyer@antc.uoregon.edu 970316
source:      RADB
```

Figure 10: Provider-Customer Policies in RPSL

In this case shown in Figure 10, if AS3701 gets a new customer, then it can update the definition of the `AS3582:AS-PARTNERS` set to include the new AS. The policies specified in the `aut-num` objects for AS4600 and AS3701 do not change.

[2] AS-PATH regular expressions are POSIX compliant regular expressions, see section 3.3.

3.2 Inter-Provider Policies

In this case, the policies of both providers are to export *only* their customer routes to the other provider, and to import *only* the customer routes of the other provider. This is commonly referred to as *peerage*. Figure 11 illustrates how this is expressed using RPSL where both AS3701 and AS2914 are providers. In this example, AS3701 advertises only the AS paths described by the AS-SET AS3582:AS-PARTNERS (i.e., customer routes). Likewise, we filter routes that come from AS2914, accepting only those defined by the filter "<^AS2914+ (AS-WNA)*\$>", i.e., those routes whose AS-PATH attribute ends with an AS in the set defined by the AS-WNA AS-SET.

```
aut-num:     AS3701
as-name:     NERO-NET
descr:       Network for Engineering and Research in Oregon
...
import:      from AS2914 accept <^AS2914+ (AS-WNA)*$>
export:      to AS2914 announce AS3582:AS-PARTNERS
...
admin-c:     DMM65
tech-c:      DMM65
notify:      noc@nero.net
mnt-by:      MAINT-AS3701
changed:     meyer@antc.uoregon.edu 970316
source:      RADB
```

Figure 11: Inter-Provider Policies in RPSL

3.3 AS Path Based Policies

Our routing policy examples have used AS path expressions in them to describe the routes accepted from or announced to peering partners. these expressions do not only cover the ASes themselves but also the interdomain topology between them. This is achieved by a mechanism known as *regular expressions*. Those familiar with the UNIX world or with programming languages like C or perl will most likely already understood the details of the AS-PATHS displayed in the examples. RPSL uses POSIX compliant regular expressions over AS numbers. In this section, we briefly describe their use.

Regular expressions follow certain rules. Several characters have special meanings, e.g. "^" denotes the beginning of a string, "$" its end. Then it becomes obvious that the AS-PATH <^AS3582$> accepted from AS3582 in figure 10 has length one and consists of AS3582 only.

Besides these positional indicators there are also operators, e.g. the unary postfix operators "+" or "*" as seen in figure 11 which ASes are accepted by AS2914: <^AS2914+ (AS-WNA)*$>. These operators work on the directly preceding regular expressions, i.e. + only affects AS2914, and * only works on (AS-WNA). Operator "+" means one or more occurrences, operator "*" means zero or more occurrences. Now it becomes obvious that the AS-PATHS accepted start with at least one AS2914 but may as well be stuffed allowing several occurrences of AS2914. Then the AS-PATH continues with no or any number of any ASes out of the as-set AS-WNA. No other ASes may then follow.

Apparently, quite complicated AS-PATHS can be expressed in a very handy and short way. For a complete list of operators and rules for regular expressions applicable to AS-PATHS in RPSL, please refer to the RPSL

```
+--------------------+          +--------------------+
|            7.7.7.1 |-----+    +-----| 7.7.7.2            |
|                    |     |    |     |                    |
| AS1                |    ========    |                AS2 |
|                    |    IX    |     |                    |
|                    |          +-----| 7.7.7.3            |
+--------------------+          +--------------------+
```

Figure 12: Including interfaces in peerings definitions

document [3].

3.4 Including Interfaces in Peering Definitions

In the above examples peerings were only given among ASes. However, the peerings may be described in much more detail by RPSL. Actually, peerings are introduced among physical routers using real IP addresses. These can be specified in the `import` and `export` attributes. Figure 12 shows a simple example of two ASes AS1 and AS2 which are connected to an exchange point IX. While AS1 has only one connection to the exchange point via a router interface with IP address 7.7.7.1, AS2 has two different connections with IP address 7.7.7.2 and 7.7.7.3. The first AS may then define its routing policy in more detail by specifying its boundary router.

```
aut-num:   AS1
import:    from AS2 at 7.7.7.1 accept <^AS2$>
...
```

This is very simple and does not make much of a difference compared to a policy where no boundary router is specified because AS1 in this example has only one connection to the exchange point. However, AS1 might want to choose to accept only announcements from AS2 which come from the router with IP address 7.7.7.2 and disregard router 7.7.7.3. AS1 then defines the following routing policy towards AS2:

```
aut-num:   AS1
import:    from AS2 7.7.7.2 at 7.7.7.1 accept <^AS2$>
...
```

So both routers involved in a peering may be specified and by selecting certain pairs of routers other connections can be denied. If no routers are included in a policy clause then it is assumed that the policy is true for all peerings among the ASes involved.

3.5 Describing Simple Backup Connections

The specification of peerings among ASes is not limited to one router for each AS. In figure 12 one of the two connections of AS2 to the exchange point IX might be used as backup in case the other connection fails.

Let us assume that AS1 wants to use the connection to router 7.7.7.2 of AS2 during regular operations, and router 7.7.7.3 as backup. In a router configuration this may be done by setting a local preference. The equivalent in RPSL is a corresponding *action* definition in the peering description. The action definitions are inserted directly before the `accept` keyword.

```
aut-num:   AS1
import:    from AS2 7.7.7.2 at 7.7.7.1 action pref=10;
           from AS2 7.7.7.3 at 7.7.7.1 action pref=20;
           accept <^AS2$>
...
```

Actions may also be defined without specifying IP addresses of routers. If no routers are included in the policy clause then it is assumed that the actions are carried out for all peerings among the ASes involved.

In the previous example AS1 controls where it sends its traffic and which connection is used as backup. However, AS2 may also define a backup connection in an `export` clause:

```
aut-num:   AS2
export:    to AS1 7.7.7.1 at 7.7.7.2 action med=10;
           to AS1 7.7.7.1 at 7.7.7.3 action med=20;
           announce <^AS2$>
```

The definition given here for AS2 is the symmetric counterpart to the routing policy of AS1. The selection of routing information is done by setting the *multi exit discriminator* metric `med`. Actually, `med` metrics will not be used in practice like this; they are more suitable for load balancing including backups. For more details on `med` metrics refer to the BGP-4 RFC [9]. To use the `med` to achieve load balancing, one often sets it to the “IGP metric”. This is specified in RPSL as:

```
aut-num:   AS2
export:    to AS1 action med=igp;
           announce <^AS2$>
```

Hence, both routers will set the `med` to the IGP metric at that router, causing some routes to be prefferred at one of the routers and other routes at the other router.

3.6 Multi-Home Routing Policies using the `community` Attribute

RFC 1998 [7] describes the use of the BGP `community` attribute [6] to handle the load balancing and backup connection of multi-homed autonomous systems. In this section, we simply illustrates how those policies are specified in RPSL.

`AS3561` bases its route selection preference on the community attribute. Other ASes may indirectly affect `AS3561`'s route selection by including the appropriate communities in their route announcements. In our example, `AS1` is connected to `AS2` and `AS3` which are both connected to `AS3561`. `AS1` wants `AS3561` to prefer the `AS2` connection over the `AS3` connection.

AS3561 prefers the routes with no community attribute followed by the routes with community {3561,90}, followed by the routes with community {3561,80}, and followed by the routes with community {3561,70}. AS1 achieves its policy by adding the community {3561,90} to the routes it exports to AS2, and the community {3561,80} to the routes it exports to AS1. Hence, AS3561 will assign a lower preference value to the routes it receives from AS2 (note that, in RPSL smaller integers represents higher preferences). This is illustrated in Figure 13.

```
as-set: AS3561:AS-PEERS
members: AS2, AS3, . . .

aut-num: AS3561
import: from AS3561:AS-PEERS
        action pref = 30;
        accept community.contains({3561,70})
import: from AS3561:AS-PEERS
        action pref = 20;
        accept community.contains({3561,80})
import: from AS3561:AS-PEERS
        action pref = 10;
        accept community.contains({3561,90})
import: from AS3561:AS-PEERS
        action pref = 0;
        accept ANY

aut-num: AS1
export: to AS2 action community.={3561,90};
        to AS3 action community.={3561,80};
        announce AS1
```

Figure 13: Policy example using the community rp-attribute.

3.7 The aut-num Object Editor

All the examples shown in the previous sections may well be edited by hand. They may be extracted one by one from the IRR using the whois program, edited, and then handed back to the registry robots. However, again the RAToolSet [1] provides a very nice tool which makes working with aut-num objects much easier: the aut-num object editor aoe (please see Figure 14).

The aut-num object editor has a graphical user interface to view and manipulate aut-num objects registered at any IRR. New aut-num objects may be generated using templates and submitted to the registries. Moreover, the routing policy from the databases may be compared to real life peerings. Therefore, aoe is highly recommended as an interface to the IRR for aut-num objects. Further information on aoe is available together with the RAToolSet [1].

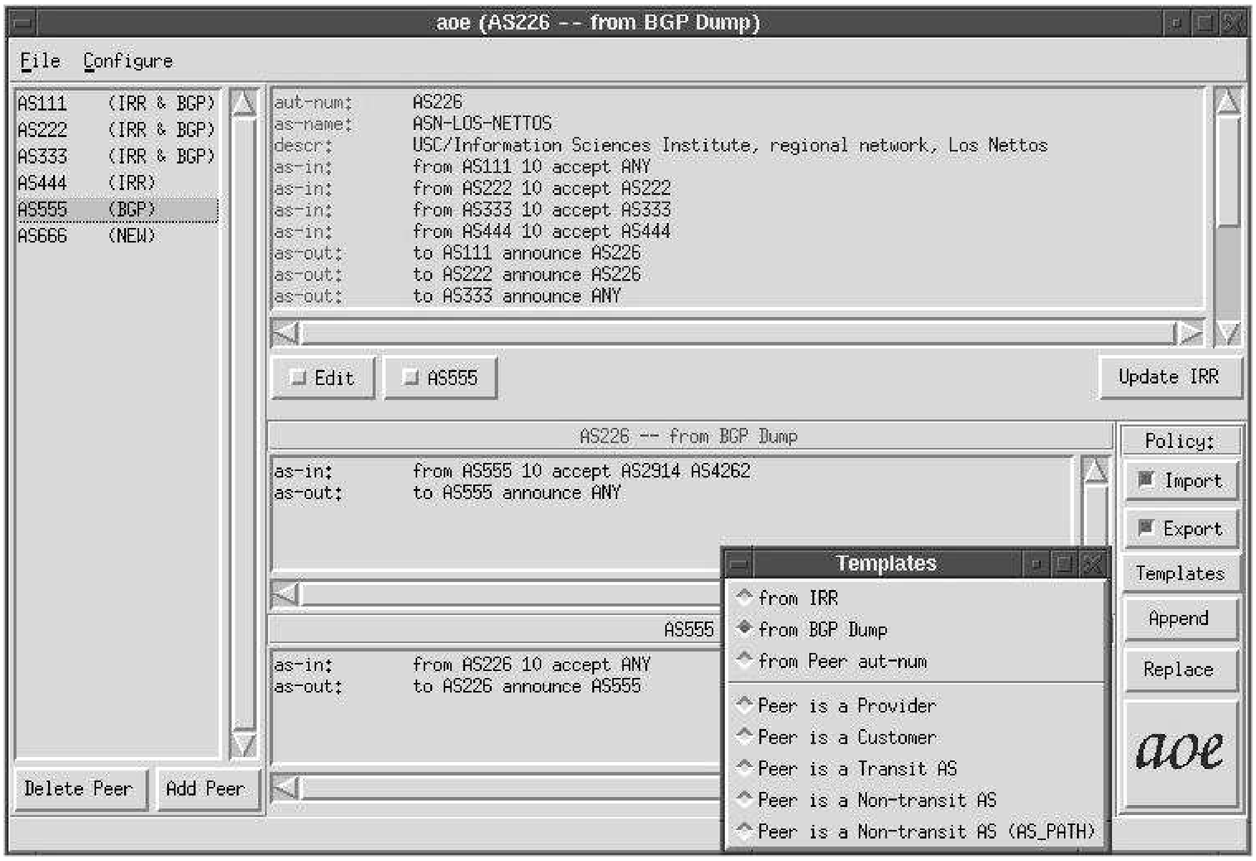

Figure 14: Autonomous System object editor (`aoe`).

4 Router Configuration Using `RtConfig`

`RtConfig` is a tool developed by the Routing Arbiter project [2] to generate vendor specific router configurations from the policy data held in the various IRRs. `RtConfig` currently supports Cisco, gated and RSd configuration formats. `RtConfig` written in C++, C, lex, and yacc. It has been publicly available since late 1994, and is currently being used by several sites for router configuration. A few of the sites currently using `RtConfig` include the Routing Arbiter Project (USA), ANS (USA), CA*Net (Canada), IMNet (Japan), VERIO (USA), Oregon-IX (USA), IAfrica (South Africa), Connect (Australia). The next section describes a methodology for generating vendor specific router configurations using `RtConfig`.[3]

4.1 Using `RtConfig`

The general paradigm for using `RtConfig` involves registering policy in an IRR, building a RtConfig source file, then running running `RtConfig` against the source file and the IRR database to create vendor specific router configuration for the specified policy. The source file will contain vendor specific commands as well

[3] Discussion of `RtConfig` internals is beyond the scope of this document.

as `RtConfig` commands. To make a source file, pick up one of your router configuration files and replace the vendor specific policy configuration commands with the `RtConfig` commands.

Commands beginning with `@RtConfig` instruct `RtConfig` to perform special operations. An example source file is shown in Figure 15. In this example, commands such as "`@RtConfig import AS3582 198.32.162.1/32 AS3701 198.32.162.2/32`" instruct `RtConfig` to generate vendor specific import policies where the router `198.32.162.1` in AS3582 is importing routes from router `198.32.162.2` in AS3701. The other `@RtConfig` commands instruct the `RtConfig` to use certain names and numbers in the output that it generates (please refer to `RtConfig` manual [2] for additional information).

```
router    bgp 3582
network   128.223.0.0
!
!       Start with access-list 100
!
@RtConfig set cisco_access_list_no = 100
!
!       NERO
!
neighbor 198.32.162.2 remote-as 3701
@RtConfig set cisco_map_name = "AS3701-EXPORT"
@RtConfig export AS3582 198.32.162.1/32 AS3701 198.32.162.2/32
@RtConfig set cisco_map_name = "AS3701-IMPORT"
@RtConfig import AS3582 198.32.162.1/32 AS3701 198.32.162.2/32
!
!       WNA/VERIO
!
neighbor 198.32.162.6 remote-as 2914
@RtConfig set cisco_map_name = "AS2914-EXPORT"
@RtConfig export AS3582 198.32.162.1/32 AS2914 198.32.162.6/32
@RtConfig set cisco_map_name = "AS2914-IMPORT"
@RtConfig import AS3582 198.32.162.1/32 AS2914 198.32.162.6/32
...
```

Figure 15: `RtConfig` Template File

Once a source file is created, the file is processed by `RtConfig` (the default IRR is the RADB, and the default vendor is Cisco; however, `RtConfig` or command line options can be used to override these values). The result of running `RtConfig` on the source file in Figure 15 is shown in Figure 16.

References

[1] C. Alaettinoğlu. *RAToolSet Routing Policy Analysis Tool Set.* Marina del Rey, CA 90292, March 1996. Available from `http://www.isi.edu/ra/RAToolSet`.

[2] C. Alaettinoğlu. *RtConfig Manual.* Marina del Rey, CA 90292, March 1996. Available from `http://www.isi.edu/ra/RAToolSet`.

[3] C. Alaettinoğlu, T. Bates, E. Gerich, D. Karrenberg, M. Terpstra, and C. Villamizer. Routing Policy Specification Language (RPSL). Internet Draft draft-ietf-rps-rpsl-00.txt, Network Information Center, March 1996. Revised, June 1996.

[4] T. Bates, E. Gerich, L. Joncheray, J-M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu. Representation of IP Routing Policies in a Routing Registry. Technical Report ripe-181, RIPE, RIPE NCC, Amsterdam, Netherlands, October 1994.

[5] T. Bates, E. Gerich, L. Joncheray, J-M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu. Representation of IP Routing Policies in a Routing Registry. Technical Report RFC-1786, Network Information Center, March 1995.

[6] R. Chandra, P. Traina, and T. Li. BGP Communities Attribute. Request for Comment RFC-1997, Network Information Center, August 1996.

[7] E. Chen and T. Bates. An Application of the BGP Community Attribute in Multi-Home Routing. Request for Comment RFC-1998, Network Information Center, August 1996.

[8] H. Nussbacher. The CIDR FAQ. Available from `http://www.ibm.net.il/ hank/cidr.html`.

[9] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). Request for Comment RFC-1771, Network Information Center, March 1995.

```
router    bgp 3582
network   128.223.0.0
!
!       NERO
!
neighbor 198.32.162.2 remote-as 3701

no access-list 100
access-list 100 permit ip 128.223.0.0   0.0.0.0   255.255.0.0   0.0.0.0
access-list 100 deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
!
!
no route-map AS3701-EXPORT
route-map AS3701-EXPORT permit 1
 match ip address 100
!
router bgp 3582
neighbor 198.32.162.2 route-map AS3701-EXPORT out
!
!
no route-map AS3701-IMPORT
route-map AS3701-IMPORT permit 1
 set local-preference 1000
!
router bgp 3582
neighbor 198.32.162.2 route-map AS3701-IMPORT in
!
!       WNA/VERIO
!
neighbor 198.32.162.6 remote-as 2914
!
no route-map AS2914-EXPORT
route-map AS2914-EXPORT permit 1
 match ip address 100
!
router bgp 3582
neighbor 198.32.162.6 route-map AS2914-EXPORT out
no ip as-path access-list  100
ip as-path access-list 100 permit ^_2914(((_[0-9]+))*_            \
      (13|22|97|132|175|668|1914|2905|2914|3361|3381|3791|3937|    \
       4178|4354|4571|4674|4683|5091|5303|5798|5855|5856|5881|6083 \
       |6188|6971|7790|7951|8028))?$
!
no route-map AS2914-IMPORT
route-map AS2914-IMPORT permit 1
 match as-path 100
 set local-preference 998
!
router bgp 3582
neighbor 198.32.162.6 route-map AS2914-IMPORT in
!
! other cisco commands
```

Figure 16: Output of `RtConfig`