Internet Draft
Expires December 3, 1999
draft-ietf-rps-appl-rpsl-06

David Meyer
Cisco Systems
Joachim Schmitz
America On-Line
Carol Orange
RIPE NCC
Mark Prior
Connect
Cengiz Alaettinoglu
USC/ISI
June 3, 1999

Using RPSL in Practice

Status of this Memo:

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

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 and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as 'work in progress.'

The list of current Internet-Drafts can be accessed at
`<http://www.ietf.org/ietf/1id-abstracts.txt>`

The list of Internet-Draft Shadow Directories can be accessed at
`<http://www.ietf.org/shadow.html>`.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Abstract

This document is a tutorial on using the Routing Policy Specification Language (RPSL) to describe routing policies in the Internet Routing Registry (IRR). We explain how to specify various routing policies and configurations using RPSL, how to register these policies in the IRR, and how to analyze them using the routing policy analysis tools, for example to generate vendor specific router configurations.

1 Introduction

This document is a tutorial on RPSL and is targeted towards an Internet/Network Service Provider (ISP/NSP) engineer who understands Internet routing, but is new to RPSL and to the IRR. Readers are referred to the RPSL reference document (RFC 2280) [1] for completeness. It is also good to have that document at hand while working through this tutorial.

The IRR is a repository of routing policies. Currently, the IRR repository is a set of five repositories maintained at the following sites: the CA*Net registry in Canada, the ANS, CW and RADB registries in the United States of America, and the RIPE registry in Europe. The five repositories are run independently. However, each site exchanges its data with the others regularly (at least once a day and as often as every ten minutes). CW, CA*Net and ANS are private registries which contain the routing policies of the networks and the customer networks of CW, CA*Net, and ANS respectively. RADB and RIPE are both public registries, and any ISP can publish their policies in these registries.

The registries all maintain up-to-date copies of one another's data. At any of the sites, the five registries can be inspected as a set. One should refrain from registering his/her data in more than one of the registries, as this practice leads almost invariably to inconsistencies in the data. The user trying to interpret the data is left in a confusing (at best) situation. CW, ANS and CA*Net customers are generally required to register their policies in their provider's registry. Others may register policies either at the RIPE or RADB registry, as preferred.

RPSL is based on RIPE-181 [2, 3], a language used to register routing policies and configurations in the IRR. Operational use of RIPE-181 has shown that it is sometimes difficult (or impossible) to express a routing policy which is used in practice. RPSL has been developed to address these shortcomings and to provide a language which can be further extended as the need arises. RPSL obsoletes RIPE-181.

RPSL constructs are expressed in one or more database "objects" which are registered in one of the registries described above. Each database object contains some routing policy information and some necessary administrative data. For example, an address prefix routed in the inter-domain mesh is specified in a `route` object, and the peering policies of an AS are specified in an `aut-num` object. The database objects are related to each other by reference. For example, a `route` object must refer to the `aut-num` object for the AS in which it is originated. Implicitly, these relationships define sets of objects, which can be used to specify policies effecting all members. For example, we can specify a policy for all routes of an ISP, by referring to the AS number in which the routes are registered to be originated.

When objects are registered in the IRR, they become 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. The syntax and semantics of the `route` object are described in Appendix A.3. Registered policies can also be compared with others for consistency and they can be used to diagnose operational routing problems in the Internet.

```
% whois -h whois.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.

The RAToolSet [6] is a suite of tools which can be used to analyze the routing registry data. It includes tools to configure routers (`RtConfig`), tools to analyze paths on the Internet (`prpath` and `prtraceroute`), and tools to compare, validate and register RPSL objects (`roe`, `aoe` and `prcheck`).

In the following section, we will describe how common routing policies can be expressed in RPSL. The objects themselves are described in Appendix A. Authoritative information on the IRR objects, however, should be sought in RFC-2280, and authoritative information on general database objects (person, role, and maintainers) and on querying and updating the registry databases, should be sought in RIPE-157 [4]. Section 3.2 describes the use of `RtConfig` to generate vendor specific router configurations.

2 Specifying Policy in RPSL

The key purpose of RPSL is to allow you to specify your routing configuration in the public Internet Routing Registry (IRR), so that you and others can check your policies and announcements for consistency. Moreover, in the process of setting policies and configuring routers, you take the policies and configurations of others into account.

In this section, we begin by showing how some simple peering policies can be expressed in RPSL. We will build on that to introduce various database objects that will be needed in order to register policies in the IRR, and to show how more complex policies can be expressed.

2.1 Common Peering Policies

The peering policies of an AS are registered in an aut-num object which looks something like that in Figure 2. We will focus on the semantics of the import and export attributes in which peering policies are expressed. We will also describe some of the other key attributes in the aut-num object, but the reader should refer to RFC-2280 or to RIPE-157 for the definitive descriptions.

```
aut-num:    AS2
as-name:    CAT-NET
descr:      Catatonic State University
...
import:     from AS1 accept ANY
import:     from AS3 accept <^AS3+$>
export:     to AS3 announce ANY
export:     to AS1 announce AS2 AS3
...
admin-c:    AO36-RIPE
tech-c:     CO19-RIPE
mnt-by:     OPS4-RIPE
changed:    orange@ripe.net
source:     RIPE
```

Figure 2: Autonomous System Object

Now consider Figure 3 (AS4 and AS5 in the figure will be discussed later). The peering policies expressed in the AS2 aut-num object in Figure 2 are typical for a small service provider providing connectivity for a customer AS3 and using AS1 for transit. That is, AS2 only accepts announcements from AS3 which:

- are originated in AS3; and
- have paths composed of only AS3's (`^` in `<^AS3+$>` means that AS3 is the first member of the path, `+` means that AS3 occurs one or more times in the path, and `$` means that no

other AS can be present in the path after AS3)[1].

To AS1, AS2 announces only those routes which originate in their AS or in their customer's AS.

```
AS1--------AS2--------AS3
            |          |
            |          |
           AS4--------AS5
```

Figure 3: Some Neighboring ASes.

In the example above, "`accept ANY`" in the `import` attribute indicates that AS2 will accept any announcements that AS1 sends, and "`announce ANY`" in the `export` attribute indicates that any route that AS2 has in its routing table will be passed on to AS3. Assuming that AS1 announces "ANY" to AS2, AS2 is taking full routing from AS1.

Note that with this peering arrangement, if AS1 adds or deletes route objects, there is no need to update any of the `aut-num` objects to continue the full routing policy. Added (or deleted) route objects will implicitly update AS1's and AS2's policies.

While the peering policy specified in Figure 2 for AS2 is common, in practice many peering agreements are more complex. Before we consider more examples, however, let's first consider the aut-num object itself. Note that it is just a set of attribute labels and values which can be submitted to one of the registry databases. This particular object is specified as being in (or headed for) the RIPE registry (see the last line in Figure 2). The source should be specified as one of ANS, CANET, CW, RADB, or RIPE depending on the registry in which the object is maintained. The `source` attribute must be specified in every database object.

It is also worth noting that this object is "maintained by" OPS4-RIPE (the value of the `mnt-by` attribute), which references a "mntner" object. Because the `aut-num` object may be used for router configuration and other operational purposes, the readers need to be able to count on the validity of its contents. It is therefore required that a mntner be specified in the `aut-num` and in most other database objects, which means you must create a `mntner` object before you can register your peering policies. For brief information on the "mntner" object and object writeability, see Appendix A of this document. For more extensive information on how to set up and use a mntner to protect your database objects, see Section 2.3 of RIPE-157.

2.2 ISP Customer - Transit Provider Policies

It is not uncommon for an ISP to acquire connectivity from a transit provider which announces all routes to it, which it in turn passes on to its customers to allow them to access hosts on the

[1] AS-PATH regular expressions are POSIX compliant regular expressions.

global Internet. Meanwhile, the ISP will generally announce the routes of its customers networks to the transit ISP, making them accessible on the global Internet. This is the service that is specified in Figure 2 for AS3.

Consider again Figure 3. Suppose now that AS2 wants to provide the same service to AS4. Clearly, it would be easy to modify the import and export lines in the `aut-num` object for AS2 (Figure 2) to those shown in Figure 4.

```
import:     from AS1 accept ANY
import:     from AS3 accept <^AS3+$>
import:     from AS4 accept <^AS4+$>
export:     to AS3 announce ANY
export:     to AS4 announce ANY
export:     to AS1 announce AS2 AS3 AS4
```

Figure 4: Policy for AS3 and AS4 in the AS2 `as-num` object

These changes are trivial to make of course, but clearly as the number of AS2 customers grows, it becomes more difficult to keep track of, and to prevent errors. Note also that if AS1 is selective about only accepting routes from the customers of AS2 from AS2, the aut-num object for AS1 would have to be adjusted to accommodate AS2's new customer.

By using the RPSL "as-set" object, we can simplify this significantly. In Figure 5, we describe the customers of AS2. Having this set to work with, we can now rewrite the policies in Figure 2 as shown in Figure 6.

```
as-set:     AS2:AS-CUSTOMERS
members:    AS3 AS4
...
changed:    orange@ripe.net
source:     RIPE
```

Figure 5: The `as-set` object

Note that if the `aut-num` object for AS1 contains the line:

```
import:     from AS2 accept <^AS2+ AS2:AS-CUSTOMERS*$>
```

then no changes will need to be made to the `aut-num` objects for AS1 or AS2 as the AS2 customer base grows. The AS numbers for new customers can simply be added to the `as-set`

```
import:     from AS1 accept ANY
import:     from AS2:AS-CUSTOMERS accept <^AS2:AS-CUSTOMERS+$>
export:     to AS2:AS-CUSTOMERS announce ANY
export:     to AS1 announce AS2 AS2:AS-CUSTOMERS
```

Figure 6: Policy in the AS2 `aut-num` object for all AS2 customers

AS2:AS-CUSTOMERS, and everything will work as for the existing customers. Clearly in terms of readability, scalability and maintainability, this is a far better mechanism when compared to adding policy for the customer AS's to the `aut-num` objects directly. The policy in this particular example states that AS1 will accept route announcements from AS2 in which the first element of the path is AS2, followed by more occurrences of AS2, and then 0 or more occurrences of any AS2 customer (e.g. any member of the `as-set` AS2:AS-CUSTOMERS).

Alternatively, one may wish to limit the routes one accepts from a peer, especially if the peer is a customer. This is recommended for several reasons, such as preventing the improper use of unassigned address space, and of course malicious use of another organization's address space.

Such filtering can be expressed in various ways in RPSL. Suppose the address space 7.7.0.0/16 has been allocated to the ISP managing AS3 for assignment to its customers. AS3 may want to announce part or all of this block on the global Internet. Suppose AS2 wants to be certain that it only accepts announcements from AS3 for address space that has been properly allocated to AS3. AS2 might then modify the AS3 import line in Figure 2 to read:

```
import:      from AS3 accept { 7.7.0.0/16^16-19 }
```

which states that route announcements for this address block will be accepted from AS3 if they are of length upto /19. This of course will have to be modified if and when AS3 gets more address space. Moreover, it is again clear that for an ISP with a growing or changing customer base, this mechanism will not scale well.

```
route-set:  AS2:RS-ROUTES:AS3
members:    7.7.0.0/16^16-19
...
changed:    orange@ripe.net
source:     RIPE
```

Figure 7: The `route-set` object

Luckily RPSL supports the notion of a "route-set" which, as shown in Figure 7, can be used to specify the set of routes that will be accepted from a given customer. Given this set (and a similar one for AS4), the manager of AS2 can now filter on the address space that will be accepted from their customers by modifying the import lines in the AS2 `aut-num` object as shown in Figure 8.

```
import:      from AS1 accept ANY
import:      from AS3 accept AS2:RS-ROUTES:AS3
import:      from AS4 accept AS2:RS-ROUTES:AS4
export:      to AS2:AS-CUSTOMERS announce ANY
export:      to AS1 announce AS2 AS2:AS-CUSTOMERS
```

Figure 8: Policy in the AS2 `aut-num` object for address based filtering on AS2 customers

Note that this is now only slightly more complex than the example in Figure 6. Furthermore, nothing need be changed in the AS2 `aut-num` object due to address space changes for a customer, and this filtering can be supported without any changes to the AS1 `aut-num` object. The additional complexity is due to the two route set names being different, otherwise we could have combined the two import statements into one. Please note that the set names are constructed hierarchically. The first AS number denotes whose sets these are, and the last AS number parameterize these sets for each peer. RPSL allows the peer's AS number to be replaced by the keyword `PeerAS`. Hence,

```
import:      from AS3 accept AS2:RS-ROUTES:PeerAS
import:      from AS4 accept AS2:RS-ROUTES:PeerAS
```

has the same meaning as the corresponding import statements in Figure 6. This lets us combine the two import statements into one as shown in Figure 9.

```
import:      from AS1 accept ANY
import:      from AS2:AS-CUSTOMERS accept AS2:RS-ROUTES:PeerAS
export:      to AS2:AS-CUSTOMERS announce ANY
export:      to AS1 announce AS2 AS2:AS-CUSTOMERS
```

Figure 9: Policy in the AS2 `aut-num` object using PeerAS

2.3 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, where peerings can be specified between physical routers

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

Figure 10: Including interfaces in peerings definitions

using IP addresses in the `import` and `export` attributes. Figure 10 shows a simple example in which AS1 and AS2 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+$>
...
```

Because AS1 has only one connection to the exchange point in this example, this specification does not differ from that in which no boundary router is specified. However, AS1 might want to choose to accept only those announcements from AS2 which come from the router with IP address 7.7.7.2 and disregard those announcements from router 7.7.7.3. AS1 can specify this routing policy as follows:

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

By selecting certain pairs of routers in a peering specification, others can be denied. If no routers are included in a policy clause then it is assumed that the policy applies to all peerings among the ASes involved.

2.4 Describing Simple Backup Connections

The specification of peerings among ASes is not limited to one router for each AS. In figure 10 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+$>
...
```

`pref` is opposite to `local-pref` in that the smaller values are preferred over larger values. 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 [7]. 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_cost; announce <^AS2+$>
```

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

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

RFC 1998 [9] describes the use of the BGP `community` attribute to provide support for load balancing and backup connections of multi-homed autonomous systems. In this section, we use stepwise refinement of an example to illustrate how those policies might be specified using RPSL.

The basic premise of RFC 1998 is to use the BGP `community` attribute to allow a customer to configure the BGP "`LOCAL_PREF`" on a provider's routers. This will allow the customer to influence the provider's route selection, normally by lowering the BGP "`LOCAL_PREF`" to indicate backup arrangements.

In this example, we illustrate how `AS1` (the provider) might specify their policy so that a customer (AS4) connected to two of AS1's direct customers (AS2 and AS3) might signal to AS1 which connection is to be preferred.

AS1's base policy is to only accept routes from customers that are originated by the customer, or by the customer's customers. This leads to a policy such as:

```
aut-num:    AS1
import:     from AS2
            accept (AS2 OR AS4) AND <^AS2+ AS4*$>
import:     from AS3
            accept (AS3 OR AS4) AND <^AS3+ AS4*$>
import:     from AS5
            accept AS5 AND <^AS5+$>
```

Note that AS4 is a customer of AS2 and AS3, and AS5 does not have its own customers.

Now suppose we want to add some policy to describe that if a customer tags a route with community 1:1 then AS1 will act on this to reduce the BGP "`LOCAL_PREF`" by 10.

```
aut-num: AS1
import:  from AS2
         action pref=10;
         accept (AS2 OR AS4) AND <^AS2+ AS4*$> AND community.contains(1:1)
import:  from AS2
         action pref=0;
         accept (AS2 OR AS4) AND <^AS2+ AS4*$>
import:  from AS3
         action pref=10;
         accept (AS3 OR AS4) AND <^AS3+ AS4*$> AND community.contains(1:1)
import:  from AS3
         action pref=0;
         accept (AS3 OR AS4) AND <^AS3+ AS4*$>
import:  from AS5
         action pref=10;
         accept AS5 AND <^AS5+$> AND community.contains(1:1)
import:  from AS5
         action pref=0;
         accept AS5 AND <^AS5+$>
```

We can see here that basically we are adding identical statements for each peering to the policy. This is the ideal candidate for RPSL's `refine` statement. This will make the policy more concise and avoid some of the potential for errors as more peering statements are added in the future:

```
aut-num:     AS1
import: {
             from AS-ANY
                  action pref=10;
                  accept community.contains(1:1);
             from AS-ANY
                  action pref=0;
                  accept ANY;
        } refine {
             from AS2 accept (AS2 OR AS4) AND <^AS2+ AS4*$>;
             from AS3 accept (AS3 OR AS4) AND <^AS3+ AS4*$>;
             from AS5 accept AS5 AND <^AS5+$>;
        }
```

Now, we can clearly see that any route that has been accepted from a customer that contains the community 1:1 will have it's local preference value reduced by 10.

The refinement has cleaned up some of the policy but we still have a large number of individual policies representing the same basic provider policy "from the customer, accept customer routes". These can be simplified by using AS sets.

First, we will collect together all of AS1's customers into a single AS set, AS1:AS-CUSTOMERS. We use a hierarchical set name that start with AS1 to avoid possible set name clashes in IRR with other ASes:

```
as-set:      AS1:AS-CUSTOMERS
members:     AS2, AS3, AS5
```

We also define one set for each customer which lists the AS numbers of any of their customers.

```
as-set:      AS1:AS-CUSTOMERS:AS2
members:     AS4

as-set:      AS1:AS-CUSTOMERS:AS3
members:     AS4

as-set:      AS1:AS-CUSTOMERS:AS5
members:     # AS5 has no customers yet, so keep blank for now
```

We can now use the keyword PeerAS with these AS sets to simplify the policy further:

```
aut-num:     AS1
import: {
             from AS-ANY
                  action pref=10;
                  accept community.contains(1:1);
             from AS-ANY
                  action pref=0;
                  accept ANY;
        } refine {
             from AS1:AS-CUSTOMERS
                  accept (PeerAS OR AS1:AS-CUSTOMER:PeerAS)
                         AND <^PeerAS+ AS1:AS-CUSTOMER:PeerAS*$>
        }
```

The use of PeerAS with AS1:AS-CUSTOMERS is basically equivalent to looping over the members of AS1:AS-CUSTOMERS, expanding the policy by replacing PeerAS with a member from the set AS1:AS-CUSTOMERS.

To illustrate how this policy might be utilised by AS4 , we present the following policy fragment:

```
aut-num: AS4
export: to AS2
        action community.append(1:1);
        announce AS1
export: to AS3
        announce AS1
```

Here, AS4 is signalling AS1 to prefer the routes from AS3.

3 Tools

In this section, we briefly introduce a number of tools which can be used to inspect data in the database, to determine optimal routing policies, and enter new data.

3.1 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 [6] provides a very nice tool which makes working with `aut-num` objects much easier: the `aut-num` object editor `aoe` (please see Figure 11).

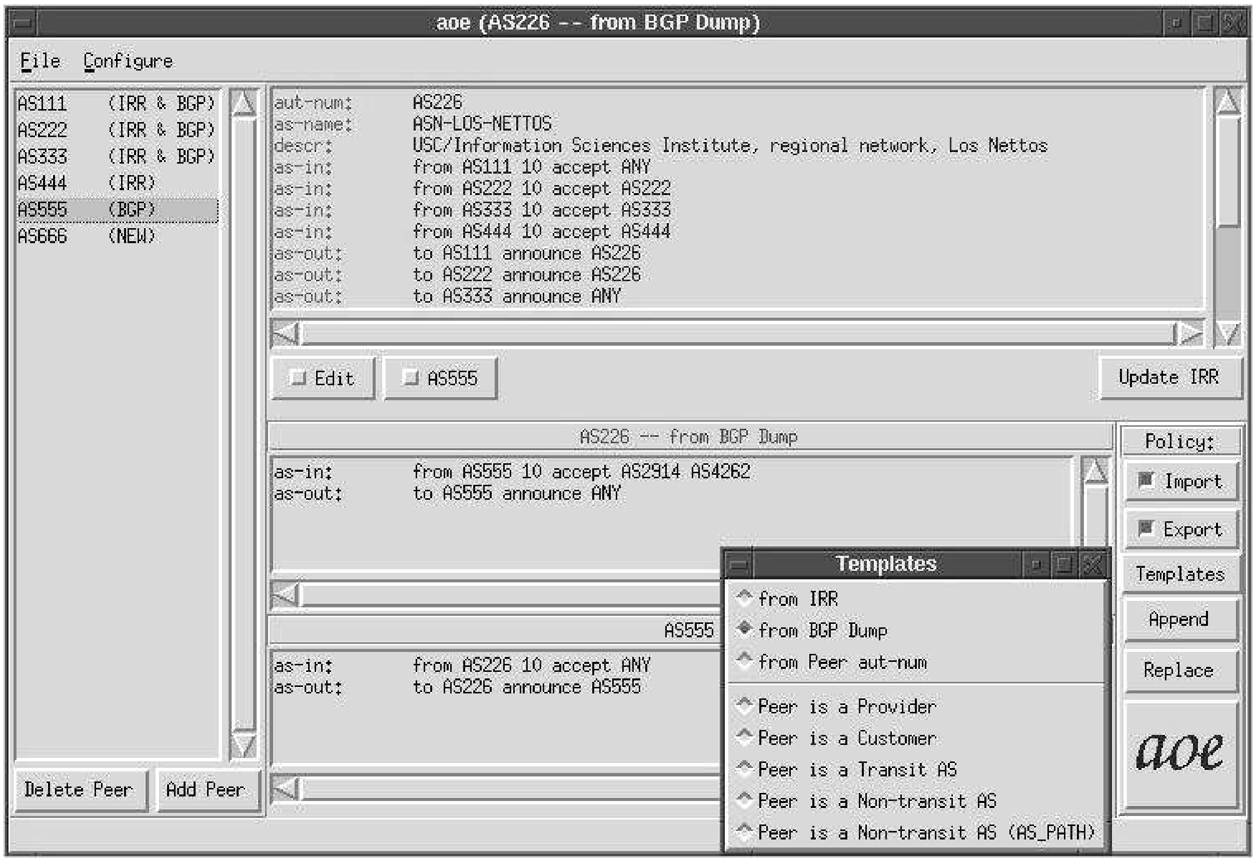

Figure 11: Autonomous System object editor (`aoe`).

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 [6].

3.2 Router Configuration Using `RtConfig`

`RtConfig` is a tool developed by the Routing Arbiter project [8] to generate vendor specific router configurations from the policy data held in the various IRRs. `RtConfig` currently supports Cisco, gated and RSd configuration formats. It has been publicly available since late 1994, and is currently being used by many sites for router configuration. The next section describes a methodology for generating vendor specific router configurations using RtConfig.[2]

3.3 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 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 12. In this example, commands such as "@RtConfig import AS3582 198.32.162.1 AS3701 198.32.162.2" 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 [8] 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 AS3701 198.32.162.2
@RtConfig set cisco_map_name = "AS3701-IMPORT"
@RtConfig import AS3582 198.32.162.1 AS3701 198.32.162.2
!
!       WNA/VERIO
neighbor 198.32.162.6 remote-as 2914
@RtConfig set cisco_map_name = "AS2914-EXPORT"
@RtConfig export AS3582 198.32.162.1 AS2914 198.32.162.6
@RtConfig set cisco_map_name = "AS2914-IMPORT"
@RtConfig import AS3582 198.32.162.1 AS2914 198.32.162.6
...
```

Figure 12: RtConfig Template File

Once a source file is created, the file is processed by RtConfig (the default IRR is the RADB, and

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

the default vendor is Cisco; however, command line options can be used to override these values). The result of running `RtConfig` on the source file in Figure 12 is shown in Figure 21 in Appendix B.

A RPSL Database Objects

In this appendix, we introduce the RPSL objects required to implement many typical Internet routing policies. RFC-2280 and RIPE-157 provide the authoritative description for these objects and for the RPSL syntax, but this appendix will often be sufficient in practice.

The frequently needed objects 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. To make your routing policies and configuration public, these objects should be registered in exactly one of the IRR registries.

In general, you can register your information by sending the appropriate objects to an email address for the registry you use. The email should consist of the objects you want to have registered or modified, separated by empty lines, and preceded by some kind of authentication details (see below). The registry robot processes your mail and enters new objects into the database, deletes old ones (upon request), or makes the requested modifications.

You will receive a response indicating the status of your submission. As the emails are handled automatically, the response is generally very fast. However, it should be remembered that a significant number of updates are also sometimes submitted to the database (by other robots), so the response time cannot be guaranteed. The email addresses for submitting objects to the existing registries are listed in Figure 13.

ANS	auto-dbm@ans.net
CANET	auto-dbm@canet.net
CW	auto-rr@cw.net
RADB	auto-dbm@ra.net
RIPE	auto-dbm@ripe.net

Figure 13: Email addresses to register policy objects in IRR.

Because it is required that a maintainer be specified in many of the database objects, a `mntner` is usually the first to be created. To have it properly authenticated, a mntner object is added manually by registry staff. Thereafter, all database submissions, deletions and modifications should be done through the registry robot.

Each of the registries can provide additional information and support for users. For the public registries this support is available from the email addresses listed in Figure 14.

RADB	db-admin@ra.net
RIPE	ripe-dbm@ripe.net

Figure 14: Support email addresses.

If you are using one of the private registries, your service provider should be able to address your questions.

A.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 15. 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[3].

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 15, the maintainer `MAINT-AS3701` is maintained by itself.

[3] 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.

```
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 15: Maintainer Object

A.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 16 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 15.

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 2.

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 A.4.1 below) describing those customers that transit AS2914.

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 16: 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.

A.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 17.

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

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

Figure 17: Example of a `route` object

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 *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 [5] 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 [6] 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 whois.ra.net 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 18). 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 [6].

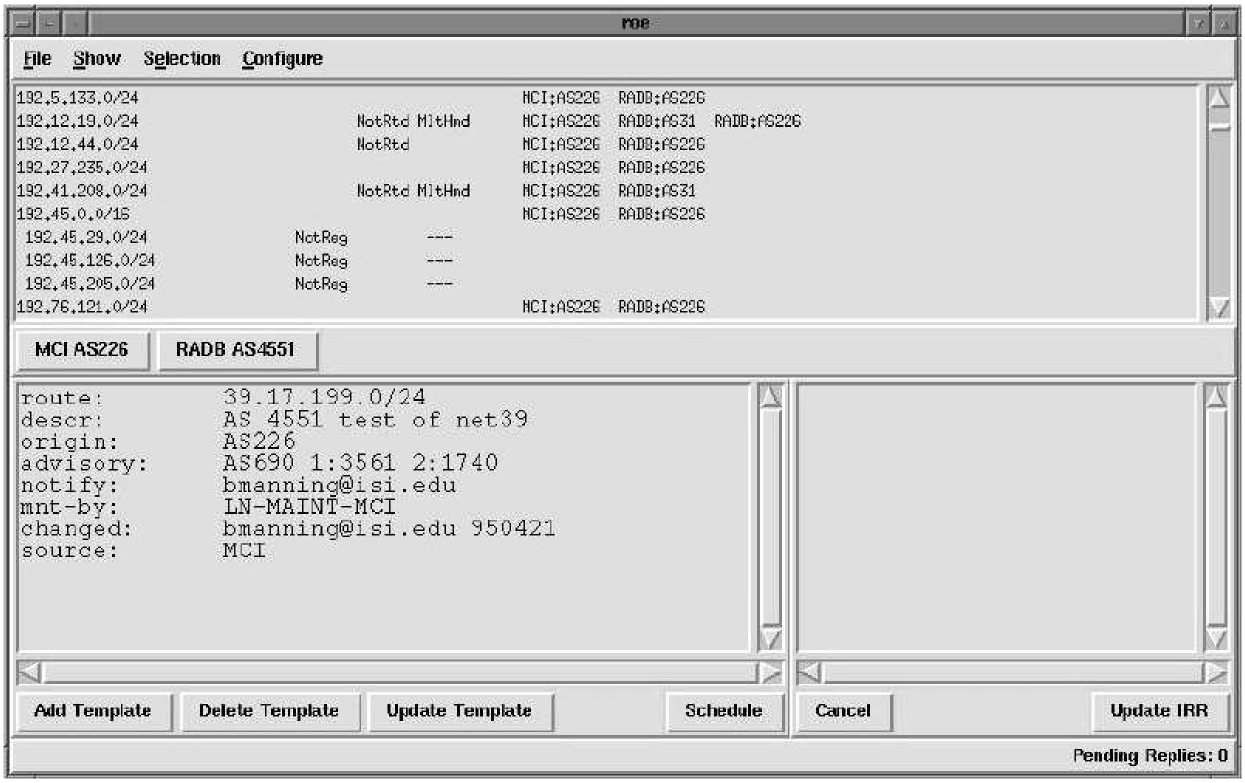

Figure 18: route object editor (`roe`).

A.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.

A.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 19, 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.

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

Figure 19: `as-set` 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-SET names.

A.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 20 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`. The set `rs-martians` illustrate the use of range operators. `0.0.0.0/0^32` are the length 32 more specifics of `0.0.0.0/0`, i.e. the host routes; `224.0.0.0/3^+` are the more specifics of `224.0.0.0/3`, i.e. the routes falling into the multicast address space. For more complete list of range operators please refer to RFC-2280.

B Output of RtConfig: An Example

In Figure 21, you see the result of running `RtConfig` on the source file in Figure 12.

```
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: 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^+,         # multicast routes
         127.0.0.0/8^9-32, . . .
```

Figure 20: `route-set` Objects

```
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
```

Figure 21: Output of `RtConfig` (cont)

C Security Consideration

This document is a tutorial to RPSL, it does not define protocols or standards that need to be secured.

```
!
!       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
```

Figure 22: Output of `RtConfig`

References

[1] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizer: Routing Policy Specification Language (RPSL), RFC 2280.

[2] T. Bates, J-M. Jouanigot, D. Karrenberg, P. Lothberg, and M. Terpstra. Representation of IP Routing Policies in the RIPE database, Technical Report ripe-81, RIPE, RIPE NCC, Amsterdam, Netherlands, February 1993.

[3] T. Bates, E. Gerich, J. Joncharay, 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.

[4] A. M. R. Magee. RIPE NCC Database Documentation. Technical Report RIPE-157, RIPE NCC, Amsterdam, Netherlands, May 1997.

[5] Hank Nussbacher. The CIDR FAQ. Tel Aviv University and IBM Israel. http://www.ibm.net.il/~hank/cidr.html

[6] The RAToolSet. http://www.ra.net/ra/RAToolSet/

[7] Y. Rekhter adn T. Li. A Border Gateway Protocol 4 (BGP-4). Request for Comment RFC 1654. Network Information Center, July 1994.

[8] RtConfig as part of the RAToolSet. http://www.ra.net/ra/RAToolSet/RtConfig.html

[9] E. Chen, T. Bates. An Application of the BGP Community Attribute in Multi-Home Routing. RFC 1998.

D Authors' Addresses

David Meyer
Cisco Systems
email: dmm@cisco.com

Joachim Schmitz
America On-Line
email: SchmitzJo@aol.com

Carol Orange
RIPE NCC
email: orange@ripe.net

Mark Prior
connect.com.au pty ltd
email: mrp@connect.com.au

Cengiz Alaettinoglu
USC Information Sciences Institute
email: cengiz@isi.edu