RIPE

Representation of IP Routing Policies in a Routing Registry (ripe-81++)

Tony Bates
Elise Gerich
Laurent Joncheray
Jean-Michel Jouanigot
Daniel Karrenberg
Marten Terpstra
Jessica Yu

Document-ID: ripe-181
Obsoletes: ripe-81

October, 1994

ABSTRACT

This document is an update to the original 'ripe-81'[1] proposal for representing and storing routing polices within the RIPE database. It incorporates several extensions proposed by Merit Inc.[2] and gives details of a generalised IP routing policy representation to be used by all Internet routing registries. It acts as both tutorial and provides details of database objects and attributes that use and make up a routing registry.

Table of Contents

1. Introduction

This document is a much revised version of the RIPE routing registry document known as ripe-81 [1]. Since its inception in February, 1993 and the establishment of the RIPE routing registry, several additions and clarifications have come to light which can be better presented in a single updated document rather than separate addenda.

Some of the text remains the same the as the original ripe-81 document keeping its tutorial style mixed with details of the RIPE database objects relating to routing policy representation. However this document does not repeat the background and historical remarks in ripe-81. For these please refer to the original document. It should be noted that whilst this document specifically references the RIPE database and the RIPE routing registry one can easily read "Regional routing registry" in place of RIPE as this representation is certainly general and flexible enough to be used outside of the RIPE community incorporating many ideas and features from other routing registries in this update.

As you can see this document has a new RIPE document identification number but can also be referred to as ripe-81++. Appendix F summarises the changes from ripe-81 plus the motivation for these changes.

We would like to acknowledge many people for help with this document. Specifically, Peter Lothberg who was a co-author of the original ripe-81 document for his many ideas as well as Gilles Farrache, Harvard Eidnes, Dale Johnson, Kannan Varadhan and Cengiz Alaettinoglu who all provided valuable input. We would also like to thank the RIPE routing working group for their review and comment. Finally, we like to thank Merit Inc. for many constructive comments and ideas and making the routing registry a worldwide Internet service. We would also like to acknowledge the funding provided by the PRIDE project run in conjunction with the RARE Technical Program, RIPE and the RIPE NCC without which this paper would not have been possible.

IT should be noted that all features described in this document will be usable in the RIPE database at a time specified in [ ]. Please refer to this document for more details.

2. Organisation of this Paper

This paper acts as both a basic tutorial for understanding routing policy and provides details of objects and attributes used within an Internet routing registry to store routing policies. Section 3 describes general issues about IP routing policies and their representation in routing registries. Experienced readers may wish to skip this section. Section 4 provides an overview of the RIPE database, its basic concepts, schema and objects which make up the database itself. It highlights the way in which the RIPE database splits routing information from allocation information. Sections 5, 6, 7 and 8 detail all the objects associated with routing policy representation. Section 9 gives a fairly extensive "walk through" of how these objects are used for expressing routing policy and the general principles behind their use. Section 10 provides a list of references used throughout this document. Appendix A, B, C and D document the formal syntax for the database objects and attributes. Appendix F details the main changes from ripe-81 and motivations for these changes. Appendix G tackles the issues of transition from ripe-81 to ripe-81++.

3. General Representation of Policy Information

Networks, Network Operators and Autonomous Systems

Throughout this document an effort is made to be consistent with terms so as not to confuse the reader.

When we talk about "networks" we mean physical networks which have a unique classless IP network number: Layer 3 entities. We do not mean organisations.

We call the organisations operating networks "network operators". For the sake of the examples we divide network operators into two categories: "service providers" and "customers". A "service provider" is a network operator who operates a network to provide Internet services to different organisations, its "customers". The distinction between service providers and customers is not clear cut. A national research networking organisation frequently acts as a service provider to Universities and other academic organisations, but in most cases it buys international connectivity from another service provider. A University networking department is a customer of the research networking organisation but in turn may regard University departments as its customers.

An Autonomous System (AS) is a group of IP networks having a single clearly defined routing policy which is run by one or more network operators. Inside ASes IP packets are routed using one or more Interior Routing Protocols (IGPs). In most cases interior routing decisions are based on metrics derived from technical parameters like topology, link speeds and load(1).

ASes exchange routing information with other ASes using Exterior Routing Protocols (EGPs). Exterior routing decisions are frequently based on policy based rules rather than purely on technical parameters. Tools are needed to configure complex policies and to communicate those policies between ASes while still ensuring proper operation of the Internet as a whole. Some EGPs like BGP-3 [8] and BGP-4 [9] provide tools to filter routing information according to policy rules and more. None of them provides a mechanism to publish or communicate the policies themselves. Yet this is critical for operational coordination and fault isolation among network operators and thus for the operation of the global Internet as a whole. This document describes a "Routing Registry" providing this functionality.

Routing Policies

The exchange of routing information between ASes is subject to routing policies. Consider the case of two ASes, X and Y exchanging routing information:

```
NET1 ......  ASX  <--->  ASY  ....... NET2
```

ASX knows how to reach a network called NET1. It does not matter whether NET1 is belonging to ASX or some other AS which exchanges routing information with ASX either directly or indirectly; we just assume that ASX knows how to direct packets towards NET1. Likewise ASY knows how to reach NET2.

---

(1) The entity we refer to as an AS is frequently and more generally called a routing domain with the AS just being an implementation vehicle. We have decided to use the term AS exclusively because it relates more directly with the database objects and routing tools. By using only one term we hope to reduce the number of concepts and to avoid confusion. The academically inclined reader may forgive us.

In order for traffic from NET2 to NET1 to flow between ASX and ASY, ASX has to announce NET1 to ASY using an external routing protocol. This states that ASX is willing to accept traffic directed to NET1 from ASY. Policy thus comes into play first in the decision of ASX to announce NET1 to ASY.

In addition ASY has to accept this routing information and use it. It is ASY's privilege to either use or disregard the information that ASX is willing to accept traffic for NET1. ASY might decide not to use this information if it does not want to send traffic to NET1 at all or if it considers another route more appropriate to reach NET1.

So in order for traffic in the direction of NET1 to flow between ASX and ASY, ASX must announce it to ASY and ASY must accept it from ASX:

```
  resulting packet flow towards NET1
<<===================================
                  |
                  |
   announce NET1  |  accept NET1
  --------------> + ------------->
                  |
      AS X        |    AS Y
                  |
   <------------- + <--------------
     accept NET2  |  announce NET2
                  |
                  |
 resulting packet flow towards NET2
 ===================================>>
```

Ideally, and seldom practically, the announcement and acceptance policies of ASX and ASY are identical.

In order for traffic towards NET2 to flow, announcement and acceptance of NET2 must be in place the other way round. For almost all applications connectivity in just one direction is not useful at all.

Usually policies are not configured for each network separately but for groups of networks. In practise these groups are almost always defined by the networks forming one or more ASes.

Routing Policy limitations

It is important to realise that with current destination based forwarding technology routing policies must eventually be expressed in these terms. It is relatively easy to formulate reasonable policies in very general terms which CANNOT be expressed in terms of announcing and accepting networks. With current technology such policies are almost always impossible to implement.

The generic example of a reasonable but un-implementable routing is a split of already joined packet streams based on something other than destination address. Once traffic for the same destination network passes the same router, or the same AS at our level of abstraction, it will take exactly the same route to the destination(2).

In a concrete example AS Z might be connected to the outside world by two links. AS Z wishes to reserve these links for different kinds of traffic, let's call them black and white traffic. For this purpose the management of AS Z keeps two lists of ASes, the black and the white list. Together these lists comprise all ASes in the world reachable from AS Z.

```
        "W"
       <--->
...                AS Z .... NET 3
       <--->
        "B"
```

It is quite possible to implement the policy for traffic *originating* in AS Z: AS Z will only accept announcements for networks in white ASes on the white link and will only accept announcements for networks in black ASes on the black link. This causes traffic from networks within AS Z *towards* white ASes to use the white link and likewise traffic for black ASes to use the black link.

Note that this way of implementing things makes it necessary to decide on the colour of each new AS which appears before traffic can be sent to it from AS Z. A way around this would be to accept only white announcements via the white link and to accept all but white announcements on the black link. That way traffic from new ASes would automatically be sent down the black link and AS Z management would only need to keep the list of white ASes rather than two lists.

Now for the unimplementable part of the policy. This concerns traffic towards AS Z. Consider the following topology:

```
B AS ---)                      "W"
W AS ---)                      --->
B AS ---)>>  AS A  ---> ...             AS Z .... NET 3
B AS ---)                      --->
W AS ---)                      "B"
```

As seen from AS Z there are both black and white ASes "behind" AS A. Since ASes can make routing decisions based on destination only, AS A and all ASes between AS A and the two links connecting AS Z can only make the same decision for traffic directed at a network in AS Z, say NET 3. This means that traffic from both black and white ASes towards NET 3 will follow the same route once it passes through AS A. This will either be the black or the white route depending on the routing policies of AS A and all ASes between it and AS Z.

The important thing to note is that unless routing and forwarding decisions can be made based on both source and destination addresses, policies like the "black and white" example cannot be implemented in general because "once joined means joined forever".

Access Policies

Access policies contrary to routing policies are not necessarily defined in terms of ASes. The very simplest type of access policy is to block packets from a specific network S from being forwarded to another network D. A common example is when some inappropriate use of resources

(2) Disregarding special cases like "type of service" routing, load sharing and routing instabilities.

on network D has been made from network S and the problem has not been resolved yet. Other examples of access policies might be resources only accessible to networks belonging to a particular disciplinary group or community of interest. While most of these policies are better implemented at the host or application level, network level access policies do exist and are a source of connectivity problems which are sometimes hard to diagnose. Therefore they should also be documented in the routing registry according to similar requirements as outlined above.

Routing vs. Allocation information

The RIPE database contains both routing registry and address space allocation registry information. In the past the database schema combined this information. Because RIPE was tasked with running both an allocation and routing registry it seemed natural to initially combine these functions. However, experience has shown that a clear separation of routing information from allocation is desirable. Often the maintainer of the routing information is not the same as the maintainer of the allocation information. Moreover, in other parts of the world there are different registries for each kind of information.

Whilst the actual routing policy objects will be introduced in the next section it is worthy of note that a transition from the current objects will be required. Appendix G details the basic steps of such a transition.

This split in information represents a significant change in the representational model of the RIPE database. Appendix F expands on the reasons for this a little more.

Tools

The network operators will need a series of tools for policy routing. Some tools are already available to perform some of the tasks. Most notably, the PRIDE tools [3] from the PRIDE project started in September 1993 as well as others produced by Merit Inc [4] and CERN [5].

These tools will enable them to use the routing policy stored in the RIPE routing registry to perform such tasks as check actual routing against policies defined, ensure consistency of policies set by different operators, and simulate the effects of policy changes.

Work continues on producing more useful tools to service the Internet community.

4. The Routing Registry and the RIPE Database

One of the activities of RIPE is to maintain a database of European IP networks, DNS domains and their contact persons along with various other kinds of network management information. The database content is public and can be queried using the whois protocol as well as retrieved as a whole. This supports NICs/NOCs all over Europe and beyond to perform their respective tasks.

The RIPE database combines both allocation registry and routing registry functions. The RIPE allocation registry contains data about address space allocated to specific enterprises and/or delegated to local registries as well as data about the domain name space. The allocation registry is described in separate documents [6,7] and outside the scope of this document.

Database Objects

Each object in the database describes a single entity in the real world. This basic principle means that information about that entity should only be represented in the corresponding database object and not be repeated in other objects. The whois service can automatically display referenced objects where appropriate.

The types of objects stored in the RIPE database are summarised in the table below:

R	Object	Describes	References
B	person	contact persons	
A	inetnum	IP address space	person
A	domain	DNS domain	person
R	aut-num	autonomous system	person (aut-num,community)
R	as-macro	a group of autonomous systems	person, aut-num
R	community	community	person
R	route	a route being announced	aut-num, community
R	clns	CLNS address space and routing	person

The first column indicates whether the object is part of the allocation registry (A), the routing registry (R) or both (B). The last column indicates the types of objects referenced by the particular type of object. It can be seen that almost all objects reference contact persons.

Objects are described by attributes value pairs, one per line. Objects are separated by empty lines. An attribute that consists of multiple lines should have the attribute name repeated on consecutive lines. The information stored about network 192.87.45.0 consists of three objects, one inetnum object and two person objects and looks like this:

```
inetnum:   192.87.45.0
netname:   RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Amsterdam, Netherlands
country:   NL
admin-c:   Daniel Karrenberg
tech-c:    Marten Terpstra
rev-srv:   ns.ripe.net
rev-srv:   ns.eu.net
notify:    ops@ripe.net
changed:   tony@ripe.net 940110
source:    RIPE

person:    Daniel Karrenberg
address:   RIPE Network Coordination Centre (NCC)
address:   Kruislaan 409
address:   NL-1098 SJ Amsterdam
address:   Netherlands
phone:     +31 20 592 5065
fax-no:    +31 20 592 5090
e-mail:    dfk@ripe.net
nic-hdl:   DK58
changed:   ripe-dbm@ripe.net 920826
source:    RIPE

person:    Marten Terpstra
address:   RIPE Network Coordination Centre (NCC)
address:   PRIDE Project
address:   Kruislaan 409
address:   NL-1098 SJ Amsterdam
address:   Netherlands
phone:     +31 20 592 5064
fax-no:    +31 20 592 5090
e-mail:    Marten.Terpstra@ripe.net
nic-hdl:   MT2
notify:    marten@ripe.net
changed:   marten@ripe.net 931230
source:    RIPE
```

Objects are stored and retrieved in this tag/value format. The RIPE NCC does not provide differently formatted reports because any desired format can easily be produced from this generic one.

Routing Registry Objects

The main objects comprising the routing registry are "aut-num" and "route", describing an autonomous system and a route respectively. It should be noted that routes not described in the routing registry should never be routed in the Internet itself.

The autonomous system (aut-num) object provides contact information for the AS and describes the routing policy of that AS. The routing policy is described by enumerating all neighbouring ASes with which routing information is exchanged. For each neighbour the routing policy is described in terms of exactly what is being sent (announced) and allowed in (accepted). It is important to note that this is exactly the part of the global policy over which an AS has direct control. Thus each aut-num object describes what can indeed be implemented and enforced locally by the AS concerned. Combined together all the aut-num objects provide the global routing graph and permit to deduce the exact routing policy between any two ASes.

While the aut-num objects describe how routing information is propagated, the route object describes a single route injected into the external routing mesh. The route object references the AS injecting (originating) the route and thereby indirectly provides contact information for the originating AS. This reference also provides the primary way of grouping routes into larger collections. This is necessary because describing routing policy on the level of single routes would be awkward to impractical given the number of routes in the Internet which is about 20,000 at the time of this writing. Thus routing policy is most often defined for groups of routes by originating AS. This method of grouping is well supported by current exterior routing protocols. The route object also references community objects described below to provide another method of grouping routes. Modification of aut-num object itself and the referencing by route objects is strictly protected to provide network operators control over the routing policy description and the routes originated by their ASes.

Sometimes even keeping track of groups of routes at the AS level is cumbersome. Consider the case of policies described at the transit provider level which apply transitively to all customers of the transit provider. Therefore another level of grouping is provided by the as-macro object which provides groups of ASes which can be referenced in routing policies just like single ASes. Membership of as-macro groups is also strictly controlled.

Sometimes there is a need to group routes on different criteria than ASes for purposes like statistics or local access policies. This is provided by the community object. A community object is much like an AS but without a routing policy. It just describes a group of routes. This is not supported at all by exterior routing protocols and depending on aggregation of routes may not be generally usable to define routing policies. It is suitable for local policies and non-routing related purposes.

These routing related objects will be described in detail in the sections below.

5. The Route Object

As stated in the previous chapter routing and address space allocation information are now clearly separated. This is performed with the introduction of the route object. The route object will contain all the information regarding a routing announcement.

All routing related attributes are removed from the inetnum object. Some old attributes are obsoleted: connect, routpr-l, bdryg-l, nsf-in, nsf-out, gateway). The currently useful routing attributes are moved to the route object: aut-sys becomes origin, ias-int will be encoded as part of the inet-rtr [15] object and comm-list simply moves. See [6] for detail of the "inetnum" object definition.

The information in the old inetnum object

```
inetnum:   192.87.45.0
netname:   RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Amsterdam, Netherlands
country:   NL
admin-c:   Daniel Karrenberg
tech-c:    Marten Terpstra
connect:   RIPE NSF WCW
aut-sys:   AS3333
comm-list: SURFNET
ias-int:   192.87.45.80  AS1104
ias-int:   192.87.45.6   AS2122
ias-int:   192.87.45.254 AS2600
rev-srv:   ns.ripe.net
rev-srv:   ns.eu.net
notify:    ops@ripe.net
changed:   tony@ripe.net 940110
source:    RIPE
```

will be distributed over two objects:

```
inetnum:   192.87.45.0
netname:   RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Amsterdam, Netherlands
country:   NL
admin-c:   Daniel Karrenberg
tech-c:    Marten Terpstra
rev-srv:   ns.ripe.net
rev-srv:   ns.eu.net
notify:    ops@ripe.net
changed:   tony@ripe.net 940110
source:    RIPE

route:       192.87.45.0/24
descr:       RIPE Network Coordination Centre
origin:      AS3333
comm-list:   SURFNET
changed:     dfk@ripe.net 940427
source:      RIPE
```

The route object is used to represent a single route originated into the Internet routing mesh. The actual syntax is given in Appendix D. However, there are several important aspects of the attributes worthy of note.

The value of the route attribute will be a classless address. It represents the exact route being injected into the routing mesh. The representation of classless addresses is described in [10].

The value of the origin attribute will be an AS reference of the form AS1234 referring to an aut-num object. It represents the AS injecting this route into the routing mesh. The "aut-num" object (see below) thus referenced provides all the contact information for this route.

Special cases: There can only be a single originating AS in each route object. However in todays Internet sometimes a route is injected by more than one AS. This situation is potentially dangerous as it can create conflicting routing policies for that route and requires coordination between the originating ASes. In the routing registry this is represented by multiple route objects.

This is a departure from the one route (net), one AS principle of the ripe-81 routing registry. The consequences for the different tools based in the routing registry will need to be evaluated and possibly additional consistency checking of the database is needed.

The examples below will illustrate the usage of the route object further. Suppose three chunks of address space of 2 different enterprises represented by the following inetnum objects:

Examples

```
inetnum:   193.0.1.0
netname:   ENT-1
descr:     Enterprise 1
 ...

inetnum:   193.0.8.0
netname:   ENT-2
descr:     Enterprise 2
 ...

inetnum:   193.0.9.0
netname:   ENT-2-SPEC
descr:     Enterprise 2
 ...
```

Supposing that the Enterprises have their own AS numbers straight application of routing without aggregation would yield:

```
route:       193.0.1.0/24
descr:       Enterprise 1
origin:      AS1
 ...

route:       193.0.8.0/24
descr:       Enterprise 2
origin:      AS2
 ...

route:       193.0.9.0/24
descr:       Enterprise 2
origin:      AS2
 ...
```

NB: This representation can be achieved by straight translation from the ripe-81 representation. See Appendix G for more details.

Homogeneous Aggregation

The two chunks of address space of Enterprise 2 can be represented by one aggregate route turning two route objects into one and potentially saving routing table space for one route.

```
route:       193.0.8.0/23
descr:       Enterprise 2
origin:      AS2
 ...
```

Note that AS2 can also decide to originate all routes mentioned so far, two 24-bit prefixes and one 23-bit prefix. This case would be represented by storing all three route objects in the database. In this particular example the additional routes will not add any functionality however and only increase the amount of routes announced unnecessarily.

Heterogeneous Aggregation

Consider the following case however:

```
route:       193.0.8.0/24
descr:       Enterprise 2
origin:      AS2
 ...

route:       193.0.9.0/24
descr:       Enterprise 2 / Special
origin:      AS2
comm-list:   SPECIAL
 ...
```

Now the prefix 193.0.9.0/24 belongs to community SPECIAL (this community may well not be relevant to routing) and the other prefix originated by AS2 does not. If AS2 aggregates these prefixes into the 193.0.8.0/23 prefix, routing policies based on the community value SPECIAL cannot be implemented in general, because there is no way to distinguish between the special and the not-so-special parts of AS2. If another AS has the policy to accept only routes to members of community SPECIAL it cannot implement it, because accepting the route to 193.0.8.0/23 would also route to 193.0.8.0/24 and not accepting this route would lose connectivity to the special part 193.0.9.0/24. We call aggregate routes consisting of components belonging to different communities or even different ASes "heterogeneous aggregates".

The major problem introduced with heterogeneous aggregates is that once the homogeneous more specific routes are withdrawn one cannot tell if a more specific part of the heterogeneous route has a different policy. However, it can be counter argued that knowing this policy is of little use since a routing policy based on the less specific heterogeneous aggregate only cannot be implemented. In fact, this displays a facet of CIDR itself in that one may actually trade off implementing slight policy variations over announcing a larger (albeit heterogeneous in terms of policy) aggregate to save routing table space.

However, it is still useful to be able to document these variations in policy especially when this homogeneous more specific route is just being withdrawn. For this one can use the "withdrawn" attribute. The withdrawn attribute can serve to both indicate that a less specific aggregate is in fact heterogeneous and also allow the general documenting of route withdrawal.

So there has to be a way for AS2 to document this even if it does not originate the route to 193.0.9.0/24 any more. This can be done with the "withdrawn" attribute of the route object. The aggregate route to 193.0.8.0/23 is now be registered as:

```
route:        193.0.8.0/23
descr:        Enterprise 2
origin:       AS2
 ...
```

With the two homogeneous routes marked as withdrawn from the Internet routing mesh but still preserving their original routing information.

```
route:        193.0.8.0/24
descr:        Enterprise 2
origin:       AS2
withdrawn:    940701
 ...

route:        193.0.9.0/24
descr:        Enterprise 2 / Special
origin:       AS2
comm-list:    SPECIAL
withdrawn:    940701
 ...
```

It should be noted that the date value used in the withdrawn attribute can only be in the past.

Proxy Aggregation

The next step of aggregation are aggregates consisting of more than one AS. This generally means one AS is aggregating on behalf of another. It is called proxy aggregation. Proxy aggregation should be done with great care and always be coordinated with other providers announcing the same route.

Consider the following:

```
route:        193.0.0.0/20
descr:        All routes known by AS1 in a single package
origin:       AS1
 ...


route:        193.0.1.0/24
descr:        Foo
origin:       AS1
withdrawn:    940310
 ...
```

```
route:       193.0.8.0/24
descr:       Bar
origin:      AS2
withdrawn:   940310
 ...


route:       193.0.9.0/24
descr:       Bar-2
origin:      AS2
withdrawn:   940310
comm-list:   SPECIAL
 ...
```

If AS1 announced no other routes to a single homed neighbouring AS, that neighbour can in general either take that route or leave it but not differentiate between AS1 and AS2.

Note: If the neighbor was previously configured to accept routes originating in AS2 but not in AS1 they lose connectivity to AS2 as well. This means that proxy aggregation has to be done carefully and in a well coordinated fashion. The information in the withdrawn route object can help to achieve that.

Aggregates with Holes

If we assume that the world of our example still consists of only three chunks of address space the aggregate above contains what are called holes, parts of an aggregate that are not reachable via the originator of the route. From the routing information itself one cannot tell whether these are holes and what part of the route falls inside one. The only way to tell is to send a packet there and see whether it gets to the destination, or an ICMP message is received back, or there is silence. On the other hand announcing aggregates with holes is quite legitimate. Consider a 16-bit aggregate with only one 24-bit prefix unreachable. The savings in routing table size by far outweigh the hole problem.

For operational reasons however it is very useful to register these holes in the routing registry. Consider the case where a remote network operator experiences connectivity problems to addresses inside an aggregate route. If the packets are getting to the AS announcing the aggregate and there are no more specific routes, the normal cause of action is to get in touch with the originating AS of the aggregate route and ask them to fix the problem. If the address falls into a hole this is futile. Therefore problem diagnosis can be sped up and unnecessary calls prevented by registering the holes in the routing registry. We do this by using the "hole" attribute. In our example the representation would be:

```
route:        193.0.0.0/20
descr:        All routes known by AS1
origin:       AS1
hole:         193.0.0.0/24
hole:         193.0.2.0/23
hole:         193.0.4.0/22
hole:         193.0.10.0/23
hole:         193.0.12.0/22
 ...
```

Note: there would also be two routes with the withdrawn attribute as displayed above (i.e. 193.0.8.0/24 and 193.0.9.0/24). It is not mandatory to document all holes. It is recommended all holes routed by another service provider are documented.

Multiple Proxy Aggregation

Finally suppose that AS2 decides to announce the same aggregate, as in the previous example, they would add the following route object to the registry:

```
route:        193.0.0.0/20
descr:        All routes known by AS2
origin:       AS2
hole:         193.0.0.0/24
hole:         193.0.2.0/23
hole:         193.0.4.0/22
hole:         193.0.10.0/23
hole:         193.0.12.0/22
 ...
```

Both AS1 and AS2 will be notified that there already is a route to the same prefix in the registry.

This multiple proxy aggregation is very dangerous to do if the sub-aggregates of the route are not the same. It is still dangerous when the sub-aggregates are consistent but connectivity to the sub-aggregates varies widely between the originators.

Route object update procedures

Adding a route object will have to be authorised by the maintainer of the originating AS. The actual implementation of this is outside the scope of this document. This guarantees that an AS guardian has full control over the registration of the routes it announces [11].

What is an Inter-AS network ?

An inter-AS network(3) exists for the purpose of passing traffic and routing information between

(3) Inter-AS IP networks are those networks are currently called FIXes, IXFs, DMZs, NAPs, GIX and many other acronyms.

different autonomous systems. The most simple example of an inter-AS network is a point-to-point link, connecting exactly two ASes. Each end of such a link is connected to an interface of router belonging to each of the autonomous systems. More complex examples are broadcast type networks with multiple interfaces connecting multiple ASes with the possibility of more than one connection per AS. Consider the following example of three routers 1, 2 and 3 with interfaces a through f connected by two inter-AS networks X and Y:

```
              X               Y
      a1b    ---    c2d      ---    e3f
```

Suppose that network X is registered in the routing registry as part of AS1 and net Y as part of AS3. If traffic passes from left to right prtraceroute will report the following sequence of interfaces and ASes:

```
a in AS1
c in AS1
e in AS3
```

The traceroute algorithm enumerates only the **receiving** interfaces on the way to the destination. In the example this leads to the passage of AS2 going unnoticed. This is confusing to the user and will also generate exceptions when the path found is checked against the routing registry.

For operational monitoring tools such as *prtraceroute* it is necessary to know which interface on an inter-AS network belongs to which AS. If AS information is not known about interfaces on an inter-AS network, tools like *prtraceroute* cannot determine correctly which ASes are being traversed.

All interfaces on inter-AS networks will are described in a separate object know as the 'inet-rtr' object [15].

6. The Autonomous System Object

Autonomous Systems

An Autonomous System (AS) is a group of IP networks operated by one or more network operators which has a single and clearly defined external routing policy.

An AS has a unique number associated with it which is used both in exchange of exterior routing information and as an identifier of the AS itself. Exterior routing protocols such as BGP and EGP are used to exchange routing information between ASes.

In routing terms an AS will normally use one or more interior gateway protocols (IGPs) in conjunction with some sort of common agreed metrics when exchanging network information within its own AS.

The term AS is often confused or even misused as a convenient way of grouping together a set of networks which belong under the same administrative umbrella even if within that group of networks there are various different routing policies. We provide the "community" concept for such use. ASes can strictly have only one single external routing policy.

The creation of an AS should be done in a conscious and well coordinated manner to avoid creating ASes for the sake of it, perhaps resulting in the worst case scenario of one AS per routing announcement. It should be noted that there is a limited number of AS numbers available. Also creating an AS may well increase the number of AS paths modern EGPs will have to keep track of. This aggravates what is known as "the routing table growth problem". This may mean that by applying the general rules for the creation and allocation of an AS below, some re-engineering may well be needed. However, this may be the only way to actually implement the desired routing policy anyway. The creation and allocation of an AS should be done with the following recommendations in mind:

- Creation of an AS is only required when exchanging routing information with other ASes. Some router implementations make use of an AS number as a form of tagging to identify the routing process. However, it should be noted that this tag does not need to be unique unless routing information is indeed exchanged with other ASes.

- For a simple case of customer networks connected to a single service provider, the IP network should normally be a member of the service providers AS. In terms of routing policy the IP network has exactly the same policy as the service provider and there is no need to make any distinction in routing information. This idea may at first seem slightly alien to some, but it highlights the clear distinction in the use of the AS number as a representation of routing policy as opposed to some form of administrative use.

- If a network operator connects to more than one AS with different routing policies then they need to create their own AS. In the case of multi-homed customer networks connected to two service providers there are at least two different routing policies to a given customer network. At this point the customer networks will be part of a single AS and this AS would be distinct from either of the service providers ASes. This allows the customer the ability of having a different representation of policy and preference to the different service providers. This is the ONLY case where a network operator should create its own AS number.

- As a general rule one should always try to populate the AS with as many routes as possible, providing all routes conform to the same routing policy.

Each AS is represented in the RIPE database by both an aut-num object and the route objects representing the routes originated by the AS. The aut-num object stores descriptive, administrative and contact information about the AS as well as the routing policies of the AS in relation to all neighbouring ASes.

The origin attributes of the route objects define the set of routes originated by the AS. Each route object can have exactly one origin attribute. Route objects can only be created and updated by the maintainer of the AS and not by those immediately responsible for the particular routes referenced therein. This ensures that operators, especially service providers, remain in control of AS routing announcements.

The AS object itself is used to represent a description of administrative details and the routing policies of the AS itself. The AS object definition is depicted as follows.

Example:

```
aut-num:  AS1104
descr:    NIKHEF-H Autonomous system
as-in:    from AS1213 100 accept AS1213
as-in:    from AS1913 100 accept AS1913
as-in:    from AS1755 150 accept ANY
as-out:   to AS1213 announce ANY
as-out:   to AS1913 announce ANY
as-out:   to AS1755 announce AS1104 AS1913 AS1213
tech-c:   Rob Blokzijl
admin-c:  Eric Wassenaar
guardian: as-guardian@nikhef.nl
changed:  ripe-dbm@ripe.net 920910
source:   RIPE
```

See Appendix A for a complete syntax definition of the "aut-num" object.

It should be noted that this representation provides two things:

- a set of routes.
- a description of administrative details and routing policies.

The set of routes can be used to generate network list based configuration information as well as configuration information for exterior routing protocols knowing about ASes. This means an AS can be defined and is useful even if it does not use routing protocols which know about the AS concept.

Description of routing policies between ASs with multiple connections — "interas-in/interas-out"

The following section is only relevant for ASes which use different policies on multiple links to the same neighboring AS. Readers not doing this may want to skip this section.

Description of multiple connections between ASs defines how two ASs have chosen to set different policies for the use of each or some of the connections between the ASs. This description is necessary only if the ASs are connected in more than one way and the routing policy and differs at these two connections.

Example:

```
                LINK1
   193.0.1.1 +----------+ 193.0.1.2
             |          |
AS1------AS2==          ==AS3-----AS4
             |          |
   193.0.1.5 +----------+ 193.0.1.6
                 LINK2
```

> Note: LINK here denotes the peer connection points between ASs. It is not necessarily just a serial link. It could be ethernet or any other type of connection as well. It can also be a peer session where the address is the same at one end and different at the other end.

It may be that AS2 wants to use LINK2 only for traffic towards AS4. LINK1 is used for traffic to AS3 and as backup to AS4, should LINK2 fail. To implement this policy, one would use the attribute "interas-in" and "interas-out." This attribute permits ASs to describe their local decisions based on its preference such as multi-exit-discriminators (MEDs) as used in some inter-domain routing protocols (BGP4, IDRP) and to communicate those routing decisions. This information would be useful in resolving problems when some traffic paths changed from traversing AS3's gateway in Timbuktu rather than the gateway in Mogadishu. The exact syntax is given in Appendix A. However, if we follow this example through in terms of AS2 we would represent this policy as follows:

Example:

```
aut-num: AS2
as-in: from AS3 10 accept AS3 AS4
as-out: to AS3 announce AS1 AS2
interas-in: from AS3 193.0.1.1/32 193.0.1.2/32 (pref=5) accept AS3
interas-in: from AS3 193.0.1.1/32 193.0.1.2/32 (pref=15) accept AS4
interas-in: from AS3 193.0.1.5/32 193.0.1.6/32 (pref=10) accept AS4
 ...
```

Here we see additional policy information between two ASs in terms of the IP addresses of the connection. The parentheses and keyword are syntactic placeholders to add the readability of the attributes. If pref=MED is specified the preference indicated by the remote AS via the multi-exit-discriminator metric such as BGP is used. Of course this type on inter-AS policy should always be bilaterally agreed upon to avoid asymmetry and in practice there may need to be corresponding interas-out attributes in the policy representation of AS3.

The interas-out attribute is similar to interas-in as as-out is to as-in. The one *major* difference being that interas-out allows you to associate an outgoing metric with each route. It is important to note that this metric is just passed to the peer AS and it is at the peer AS's discretion to use or ignore it. A special value of IGP specifies that the metric passed to the receiving AS will be derived from the IGP of the sending AS. In this way the peer AS can choose the optimal link for its traffic as determined by the sending AS.

If we look at the corresponding interas-out for AS3 we would see the following:

Example:

```
aut-num: AS3
as-in: from AS2 10 accept AS1 A2
as-out: to AS2 announce AS3 AS4
interas-out: to AS2 193.0.1.2/32 193.0.1.1/32 (metric-out=5) announce AS3
interas-out: to AS2 193.0.1.2/32 193.0.1.1/32 (metric-out=15) announce AS4
interas-out: to AS2 193.0.1.6/32 193.0.1.5/32 (metric-out=10) announce AS4
 ...
```

Descriptions of interas policies do not replace the global policy described in as-in, as-out and other policy attributes which should be specified too. If the global policy mentions more routes than the combined local policies then local preferences for these routes are assumed to be equal for all links.

Any route specified in interas-in/out and not specified in as-in/out is assumed not accepted/announced between the ASes concerned. Diagnostic tools should flag this inconsistency as an error. It should be noted that if an interas-in or interas-out policy is specified then it is mandatory to specify the corresponding global policy in the as-in or as-out line. Please note there is no relevance in the cost associated with as-in and the preferences used in interas-in.

The interaction of interas-in/interas-out with as-in/as-out

Although formally defined above, the rules associated with policy described in terms of interas-in and interas-out with respect to as-in and as-out are worthy of clarification for implementation.

When using interas-in or interas-out policy descriptions, one must always make sure the set of policies described between two ASes is always equal to or a sub-set of the policy described in the global as-in or as-out policy. When a sub-set is described remember the remaining routes are implicitly shared across all connections. It is an error for the interas policies to describe a superset of the global policies, i.e. to announce or accept more routes than the global policies.

When defining complex interas based policies it is advisable to ensure that any possible ambiguities are not present by explicitly defining your policy with respect to the global as-in and as-out policy.

If we look at a simple example, taking just in-bound announcements to simplify things. If we have the following global policy:

```
aut-num: AS1
as-in: from AS2 10 accept AS100 OR {10.0.0.0/8}
```

Suppose there are three peerings between AS1 and AS2, known as L1-R1, L2-R2 and L3-R3 respectively. The actual policy of these connections is to accept AS100 equally on these three links and just route 10.0.0.0/8 on L3-R3. The simple way to mention this exception is to just specify an interas policy for L3-R3:

```
interas-in: from AS2 L3 R3 (pref=100) accept {10.0.0.0/8}
```

The implicit rule that all routes not mentioned in interas policies are accepted on all links with equal preference ensures the desired result.

The same policy can be written explicitly as:

```
interas-in: from AS2 L1 R1 (pref=100) accept AS100
interas-in: from AS2 L2 R2 (pref=100) accept AS100
interas-in: from AS2 L3 R3 (pref=100) accept AS100 OR {10.0.0.0/8}
```

Whilst this may at first sight seem obvious, the problem arises when not all connections are mentioned. For example, if we specified only an interas-in line for L3-R3 as below:

```
aut-num: AS1
as-in: from AS2 10 accept AS100 OR {10.0.0.0/8}
interas-in: from AS2 L3 R3 (pref=100) accept AS100 OR {10.0.0.0/8}
```

then the policy for the other links according to the rules above would mean they were equal to the global policy minus the sum of the local policies (i.e. ((AS100 OR {10.0.0.0/0}) / (AS100 OR {10.0.0.0/0})) = empty) which in this case would mean nothing is accepted on connections L1-R1 and L2-R2 which is incorrect.

Another example: If we only registered the policy for link L2-R2:

```
interas-in: from AS2 L2 R2 (pref=100) accept AS100
```

The implicit policy for both L1-R1 and L3-R3 would be as follows:

```
interas-in: from AS2 L1 R1 (pref=100) accept {10.0.0.0/8}
interas-in: from AS2 L3 R3 (pref=100) accept {10.0.0.0/8}
```

This is derived as the set of global policies minus the set of interas-in policies (in this case just accept AS100 as it was the L2-R2 interas-in policy we registered) with equal cost for the remaining connection. This again is clearly not what was intended.

We strongly recommend that you always mention all policies for all interas connections explicitly, to avoid these possible errors. One should always ensure the set of the interas policies is equal to the global policy. Clearly if interas policies differ in complex ways it is worth considering splitting the AS in question into separate ASes. However, this is beyond the direct scope of this document.

It should also be noted there is no direct relationship between the cost used in as-in and the preference used in interas-in.

How to describe the exclusion policy of a certain AS — "as-exclude"

Some ASes have a routing policy based on the exclusion of certain routes if for whatever reason a certain AS is used as transit. Whilst, this is in general not good practice as it makes implicit assumptions on topology with asymmetry a possible outcome if not coordinated, this case needs to be accommodated within the routing policy representation.

The way this is achieved is by making use of the "as-exclude" attribute. The precise syntax of this attribute can be found in Appendix A along with the rest of the defined syntax for the "aut-num" object. However, some explanation of the use of this attribute is useful. If we have the following example topology.

Example:

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

With a simple corresponding policy like so:

Example:

```
aut-num: AS1
as-in:  from AS2 100 accept ANY
as-out: to AS2 announce AS1
as-exclude: exclude AS4 to ANY
 ....
```

We see an interesting policy. What this says in simple terms is AS1 doesn't want to reach anything if it transits AS4. This can be a perfectly valid policy. However, it should be realised that if for whatever reason AS2 decides to route to AS3 via AS4 then immediately AS1 has no connectivity to AS3 or if AS1 is running default to AS2 packets from AS1 will still flow via AS4. The important point about this is that whilst AS1 can advise its neighbours of its policy it has no direct control on how it can enforce this policy to neighbours upstream.

Another interesting scenario to highlight the unexpected result of using such an "as-exclude" policy. If we assume in the above example AS2 preferred AS4 to reach AS3 and AS1 did not use default routing then as stated AS1 would have no connectivity to AS3. Now lets suppose that for example the link between AS2 and AS4 went down for some reason. Like so:

Example:

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

Suddenly AS1 now has connectivity to AS3. This unexpected behavior should be considered when created policies based on the "as-exclude" attribute.

The second problem with this type of policy is the potential of asymmetry. In the original example we saw the correct policy from AS1's point of view but if ASes with connectivity through AS4 do not use a similar policy you have asymmetric traffic and policy. If an AS uses such a policy they must be aware of the consequences of its use. Namely that the specified routes which transit the AS (i.e. routing announcements with this AS in the AS path information) in question will be excluded. If not coordinated this can easily cause asymmetry or even worse loss of connectivity to unknown ASes behind (or in front for that matter) the transit AS in question. With this in mind this attribute can only be viewed as a form of advisory to other service providers. However, this does not preclude its use with policy based tools if the attribute exists.

By having the ability to specify a route keyword based on any of the four notations given in the syntax it allows the receiving AS to specify what routes it wishes to exclude through a given transit AS to a network granularity.

7. AS Macros

It may be difficult to keep track of each and every new AS that is represented in the routing registry. A convenient way around this is to define an ‘AS Macro’ which essentially is a convenient way to group ASes. This is done so that each and every AS guardian does not have to add a new AS to it’s routing policy as described by the as-in and as-out attributes of it’s AS object.

However, it should be noted that this creates an implicit trust on the guardian of the AS-Macro.

An AS-Macro can be used in <routing policy expressions> for the "as-in" and "as-out" attributes in the aut-num object. The AS-Macro object is then used to derive the list or group of ASes.

A simple example would be something like:

Example:

```
aut-num: AS786
as-in:   from AS1755 100 accept AS-EBONE AND NOT AS1104
as-out   to AS1755 announce AS786
 .....
```

Where the as-macro object for AS-EBONE is as follows:

```
as-macro:  AS-EBONE
descr:     ASes routed by EBONE
as-list:   AS2121 AS1104 AS2600 AS2122
as-list:   AS1103 AS1755 AS2043
guardian:  guardian@ebone.net
 ......
```

So the policy would be evaluated to:

```
aut-num: AS786
as-in:   from AS1755 100 accept (AS2121 OR AS1104 OR AS2600 OR AS2122
as-in:   from AS1755 100 accept AS1103 OR AS1755 OR AS2043) AND NOT AS1104
 ......
```

It should be noted that the above examples incorporates the rule for line wrapping as defined in Appendix A for policy lines. See Appendix C for a definition on the AS-Macro syntax.

8. The Community Object

A community is a group of routes that cannot be represented by an AS or a group of ASes. It is in some circumstances useful to define a group of routes that have something in common. This could be a special access policy to a supercomputer centre, a group of routes used for a specific mission, or a disciplinary group that is scattered among several autonomous systems. Also these communities could be useful to group routes for the purpose of network statistics.

Communities do not exchange routing information, since they do not represent an autonomous system. More specifically, communities do not define routing policies, but access or usage policies. However, they can de used as in conjunction with an ASes routing policy to define a set of routes the AS sets routing policy for.

Communities should be defined in a strict manner, to avoid creating as many communities as there are routes, or even worse. Communities should be defined following the two rules below;

- Communities must have a global meaning. Communities that have no global meaning, are used only in a local environment and should be avoided.

- Communities must not be defined to express non-local policies. It should be avoided that a community is created because some other organisation forces a policy upon your organisation. Communities must only be defined to express a policy defined by your organisation.

Community examples

There are some clear examples of communities:

BACKBONE —
: all customers of a given backbone service provider even though they can have various different routing policies and hence belong to different ASes. This would be extremely useful for statistics collection.

HEPNET —
: the High Energy Physics community partly shares infrastructure with other organisations, and the institutes it consists of are scattered all over Europe, often being part of a non HEPNET autonomous system. To allow statistics, access or part of a routing policy , a community HEPNET, consisting of all routes that are part of HEPNET, conveniently groups all these routes.

NSFNET —
: the National Science Foundation Network imposes an acceptable use policy on routes that wish to make use of it. A community NSFNET could imply the set of routes that comply with this policy.

MULTI —
: a large multinational corporation that does not have its own internal infrastructure, but connects to the various parts of its organisations by using local service providers that connect them all together, may decide to define a community to restrict access to their networks,

> only by networks that are part of this community. This way a corporate network could be defined on shared infrastructure. Also, this community could be used by any of the service providers to do statistics for the whole of the corporation, for instance to do topology or bandwidth planning.

Similar to Autonomous systems, each community is represented in the RIPE database by both a community object and community tags on the route objects representing the routes belonging to the community. The community object stores descriptive, administrative and contact information about the community.

The community tags on the route objects define the set of routes belonging to a community. A route can have multiple community tags. The community tags can only be created and updated by the "guardian" of the community and not by those directly responsible for the particular network. This ensures that community guardians remain in control of community membership.

Here's an example of how this might be represented in terms of the community tags within the network object. We have an example where the route 192.16.199.0/24 has a single routing policy (i.e. that of AS 1104), but is part of several different communities of interest. We use the tag "comm-list" to represent the list of communities associated with this route. NIKHEF-H uses the service provider SURFNET (a service provider with customers with more than one routing policy), is also part of the High Energy Physics community as well as having the ability to access the Supercomputer at CERN(4).

Example:

```
route:      192.16.199.0/24
descr:      Local Ethernet
descr:      NIKHEF section H
origin:     AS1104
comm-list: HEPNET CERN-SUPER SURFNET
changed:    ripe-dbm@ripe.net 920604
source:     RIPE
```

In the above examples some communities have been defined. The community object itself will take the following format:

(4) The community 'CERN-SUPER', is somewhat national, but is intended as an example of a possible use of an access policy constraint.

Example:

```
community:  SURFNET
descr:      Dutch academic research network
authority:  SURFnet B.V.
guardian:   comm-guardian@surfnet.nl
admin-c:    Erik-Jan Bos
tech-c:     Erik-Jan Bos
changed:    ripe-dbm@ripe.net 920604
source:     RIPE
```

For a complete explanation of the syntax please refer to Appendix B.

9. Representation of Routing Policies

Routing policies of an AS are represented in the autonomous system object. Initially we show some examples, so the reader is familiar with the concept of how routing information is represented, used and derived. Refer to Appendix A, for the full syntax of the "aut-num" object.

The topology of routing exchanges is represented by listing how routing information is exchanged with each neighbouring AS. This is done separately for both incoming and outgoing routing information. In order to provide backup and back door paths a relative cost is associated with incoming routing information.

Example 1:

```
AS1------AS2
```

This specifies a simple routing exchange of two presumably isolated ASes. Even if either of them has routing information about routes in ASes other than AS1 and AS2, none of that will be announced to the other.

```
aut-num:   AS1
as-out:    to AS2 announce AS1
as-in:     from AS2 100 accept AS2

aut-num:   AS2
as-out:    to AS1 announce AS2
as-in:     from AS1 100 accept AS1
```

The number 100 in the in-bound specifications is a relative cost, which is used for backup and back door routes. The absolute value is of no significance. The relation between different values within the same AS object is. A lower value means a lower cost. This is consciously similar to the cost based preference scheme used with DNS MX RRs.

Example 2:

Now suppose that AS2 is connected to one more AS, besides AS1, and let's call that AS3:

```
AS1------AS2------AS3
```

In this case there are two reasonable routing policies:

a) AS2 just wants to exchange traffic with both AS1 and AS3 itself without passing traffic between AS1 and AS3.

b) AS2 is willing to pass traffic between AS3 and AS1, thus acting as a transit AS

Example 2a:

In the first case AS1's representation in the routing registry will remain unchanged as will be the part of AS2's representation describing the routing exchange with AS1. A description of the additional routing exchange with AS3 will be added to AS2's representation:

```
aut-num:   AS1
as-out:    to AS2 announce AS1
as-in:     from AS2 100 accept AS2

aut-num:   AS2
as-out:    to AS1 announce AS2
as-in:     from AS1 100 accept AS1
as-out:    to AS3 announce AS2
as-in:     from AS3 100 accept AS3

aut-num:   AS3
as-out:    to AS2 announce AS3
as-in:     from AS2 100 accept AS2
```

Note that in this example, AS2 keeps full control over its resources. Even if AS3 and AS1 were to allow each others routes in from AS2, the routing information would not flow because AS2 is not announcing it(5).

(5) Of course AS1 and AS3 could just send traffic to each other to AS2 even without AS2 announcing the routes, hoping that AS2 will forward it correctly. Such questionable practices however are beyond the scope of this document.

Example 2b:

If contrary to the previous case, AS1 and AS3 are supposed to have connectivity to each other via AS2, all AS objects have to change:

```
aut-num:   AS1
as-out:    to AS2 announce AS1
as-in:     from AS2 100 accept AS2 AS3

aut-num:   AS2
as-out:    to AS1 announce AS2 AS3
as-in:     from AS1 100 accept AS1
as-out:    to AS3 announce AS2 AS1
as-in:     from AS3 100 accept AS3

aut-num:   AS3
as-out:    to AS2 announce AS3
as-in:     from AS2 100 accept AS1 AS2
```

Note that the amount of routing information exchanged with a neighbour AS is defined in terms of routes belonging to ASes. In BGP terms this is the AS where the routing information originates and the originating AS information carried in BGP could be used to implement the desired policy. However, using BGP or the BGP AS-path information is not required to implement the policies thus specified. Configurations based on route lists can easily be generated from the database. The AS path information, provided by BGP can then be used as an additional checking tool as desired.

The specification understands one special expression and this can be expressed as a boolean expression:

ANY -

means any routing information known. For output this means that all routes an AS knows about are announced. For input it means that anything is accepted from the neighbour AS.

Example 3:

AS4 is a stub customer AS, which only talks to service provider AS123.

```
                    |
                    |
          -----AS123------AS4
                    |
                    |
```

```
aut-num: AS4
as-out:  to AS123 announce AS4
as-in:   from AS123 100 accept ANY

aut-num: AS123
as-in:   from AS4 100 accept AS4
as-out:  to AS4 announce ANY
<further neighbours>
```

Since AS4 has no other way to reach the outside world than AS123 it is not strictly necessary for AS123 to send routing information to AS4. AS4 can simply send all traffic for which it has no explicit routing information to AS123 by default. This strategy is called default routing. It is expressed in the routing registry by adding one or more default tags to the autonomous system which uses this strategy. In the example above this would look like:

```
aut-num: AS4
as-out:  to AS123 announce AS4
default: AS123 100

aut-num: AS123
as-in:   from AS4 100 accept AS4
<further neighbours>
```

Example 4:

AS4 now connects to a different operator, AS5. AS5 uses AS123 for outside connectivity but has itself no direct connection to AS123. AS5 traffic to and from AS123 thus has to pass AS4. AS4 agrees to act as a transit AS for this traffic.

```
             |
             |
        -----AS123------AS4-------AS5
             |
             |
```

```
aut-num:    AS4
as-out:     to AS123 announce AS4 AS5
as-in:      from AS123 100 accept ANY
as-out:     to AS5 announce ANY
as-in:      from AS5 50 accept AS5

aut-num:    AS5
as-in:      from AS4 100 accept ANY
as-out:     to AS4 announce AS5

aut-num:    AS123
as-in:      from AS4 100 accept AS4 AS5
as-out:     to AS4 announce ANY
<further neighbours>
```

Now AS4 has two sources of external routing information. AS5 which provides only information about its own routes and AS123 which provides information about the external world. Note that AS4 accepts information about AS5 from both AS123 and AS5 although AS5 information cannot come from AS123 since AS5 is connected only via AS4 itself. The lower cost of 50 for the announcement from AS5 itself compared to 100 from AS123 ensures that AS5 is still believed even in case AS123 will unexpectedly announce AS5.

In this example too, default routing can be used by AS5 much like in the previous example. AS4 can also use default routing towards AS123:

```
aut-num:    AS4
as-out:     to AS123 announce AS4 AS5
default:    AS123 11
as-in:      from AS5 50 accept AS5

Note no announcements to AS5, they default to us.

aut-num:    AS5
as-out:     to AS4 announce AS5
default:    AS4 100

aut-num:    AS123
as-in:      from AS4 100 announce AS4 AS5
<further neighbours>
```

Note that the relative cost associated with default routing is totally separate from the relative cost associated with in-bound announcements. The default route will never be taken if an explicit route is known to the destination. Thus an explicit route can never have a higher cost than the default route. The relative cost associated with the default route is only useful in those cases where one wants to configure multiple default routes for redundancy.

Note also that in this example the configuration using default routes has a subtly different behavior than the one with explicit routes: In case the AS4-AS5 link fails AS4 will send traffic to AS5 to AS123 when using the default configuration. Normally this makes not much difference as there will be no answer and thus little traffic. With certain datagram applications which do not require acknowledgments however, significant amounts of traffic may be uselessly directed at AS123. Similarly default routing should not be used if there are stringent security policies which proscribe any traffic intended for AS5 to ever touch AS123.

Once the situation gets more complex using default routes can lead to unexpected results or even defeat the routing policies established when links fail. As an example consider how Example 5a) below could be implemented using default routing. Therefore, generally it can be said that default routing should only be used in very simple topologies.

Example 5:

In a different example AS4 has a private connection to AS6 which in turn is connected to the service provider AS123:

```
          |
          |
-----AS123------AS4
          |          |
          |          |
          |          |
     AS6 ---------+
```

There are a number of policies worth examining in this case:

a) AS4 and AS6 wish to exchange traffic between themselves exclusively via the private link between themselves; such traffic should never pass through the backbone (AS123). The link should never be used for transit traffic, i.e. traffic not both originating in and destined for AS4 and AS6.

b) AS4 and AS6 wish to exchange traffic between themselves via the private link between themselves. Should the link fail, traffic between AS4 and AS6 should be routed via AS123. The link should never be used for transit traffic.

c) AS4 and AS6 wish to exchange traffic between themselves via the private link between themselves. Should the link fail, traffic between AS4 and AS6 should be routed via AS123. Should the connection between AS4 and AS123 fail, traffic from AS4 to destinations behind AS123 can pass through the private link and AS6's connection to AS123.

d) AS4 and AS6 wish to exchange traffic between themselves via the private link between themselves. Should the link fail, traffic between AS4 and AS6 should be routed via AS123. Should the backbone connection of either AS4 or AS6 fail, the traffic of the disconnected AS should flow via the other AS's backbone connection.

Example 5a:

```
aut-num:   AS4
as-in:     from AS123 100 accept NOT AS6
as-out:    to AS123 announce AS4
as-in:     from AS6 50 accept AS6
as-out:    to AS6 announce AS4

aut-num:   AS123
as-in:     from AS4 100 accept AS4
as-out:    to AS4 announce ANY
as-in:     from AS6 100 accept AS6
as-out:    to AS6 announce ANY
<further neighbours>

aut-num:    AS6
as-in:      from AS123 100 accept NOT AS4
as-out:     to AS123 announce AS6
as-in:      from AS4 50 accept AS4
as-out:     to AS4 announce AS6
```

Note that here the configuration is slightly inconsistent. AS123 will announce AS6 to AS4 and AS4 to AS6. These announcements will be filtered out on the receiving end. This will implement the desired policy. Consistency checking tools might flag these cases however.

Example 5b:

```
aut-num:   AS4
as-in:     from AS123 100 accept ANY
as-out:    to AS123 announce AS4
as-in:     from AS6 50 accept AS6
as-out:    AS6 AS4

aut-num:   AS123
as-in:     AS4 100 AS4
as-out:    AS4 ANY
as-in:     AS6 100 AS6
as-out:    AS6 ANY
<further neighbours>

aut-num:   AS6
as-in:     from AS123 100 accept ANY
as-out:    to AS123 announce AS6
as-in:     from AS4 50 accept AS4
as-out:    to AS4 announce AS6
```

The thing to note here is that in the ideal operational case, 'all links working' AS4 will receive announcements for AS6 from both AS123 and AS6 itself. In this case the announcement from AS6 will be preferred because of its lower cost and thus the private link will be used as desired. AS6 is configured as a mirror image.

Example 5c:

The new feature here is that should the connection between AS4 and AS123 fail, traffic from AS4 to destinations behind AS123 can pass through the private link and AS6's connection to AS123.

```
aut-num:  AS4
as-in:    from AS123 100 accept ANY
as-out:   to AS123 announce AS4
as-in:    from AS6 50 accept AS6
as-in:    from AS6 110 accept ANY
as-out:   to AS6 AS4

aut-num:  AS123
as-in:    from AS4 1 accept AS4
as-out:   to AS4 announce ANY
as-in:    from AS6 1 accept AS6
as-in:    from AS6 2 accept AS4
as-out:   to AS6 announce ANY
<further neighbours>

aut-num:  AS6
as-in:    from AS123 100 accept ANY
as-out:   to AS123 AS6 announce AS4
as-in:    from AS4 50 accept AS4
as-out:   to AS4 announce ANY
```

Note that it is important to make sure to propagate routing information for both directions in backup situations like this. Connectivity in just one direction is not useful at all for almost all applications.

Note also that in case the AS6-AS123 connection breaks, AS6 will only be able to talk to AS4. The symmetrical case (5d) is left as an exercise to the reader.

10. Future Extensions

We envision that over time the requirements for describing routing policy will evolve. The routing protocols will evolve to support the requirements and the routing policy description syntax will need to evolve as well. For that purpose, a separate document will describe experimental syntax definitions for policy description. This document [14] will be updated when new objects or attributes are proposed or modified.

11. References

[1] Bates, T., Jouanigot, J-M., Karrenberg, D., Lothberg, P., Terpstra, M., "Representation of IP Routing Policies in the RIPE Database", RIPE-81, February 1993.

[2] Merit Network Inc.,"Representation of Complex Routing Policies of an Autonomous System", DRAFT, March, 1994.

[3] PRIDE Tools Release 1.
See **ftp.ripe.net:pride/tools/pride-tools-1.tar.Z**.

[4] Merit Inc. RRDB Tools.
See **rrdb.merit.edu:pub/meritrr/***

[5] The Network List Compiler.
See **dxcoms.cern.ch:pub/ripe-routing-wg/nlc-2.2d.tar**

[6] Lord, A., Terpstra, M., "RIPE Database Template for Networks and Persons", RIPE-119, October, 1994.

[7] Karrenberg, D., "RIPE Database Template for Domains", RIPE-49, April 1992.

[8] Lougheed, K., Rekhter, Y., "A Border Gateway Protocol 3 (BGP-3)", RFC1267, October 1991.

[9] Rekhter, Y., Li, T., "A Border Gateway Protocol 4 (BGP-4)", RFC-1654, May 1994.

[10] Bates, T., Karrenberg, D., Terpstra, M., "Support for Classless Internet Addresses in the RIPE Database", RIPE-121, October 1994.

[11] Karrenberg, D., "Authorisation and Notification of Changes in the RIPE Database", RIPE-120, October 1994.

[12] Bates, T., "Support of Guarded fields within the RIPE Database", ripe-117, July 1994.

[13] Estrin, D., Li, T., Rekhter, Y., Varadhan, K., Zappala, D., "Source Demand Routing: Packet Format and Forwarding Specification (Version 1)", INTERNET-DRAFT, draft-ietf-sdr-sdrp-04.txt, March 1994.

[14] Joncheray, L., "Experimental Objects and attributes for the Routing Registry", RIPE-182, October1994.

[15] Bates, T., "Specifying an ‘Internet Router’ in the Routing Registry", RIPE-122, October 1994.

[16] Bates, T., Karrenberg, D., Terpstra, M., "RIPE Database Transition Plan", RIPE-123, October 1994.

12. Author's Addresses

Tony Bates
RARE/PRIDE Project
c/o RIPE Network Coordination Centre
Kruislaan 409
NL-1098 SJ Amsterdam
The Netherlands
+31 20 592 5064
T.Bates@ripe.net

Elise Gerich
The University of Michigan
Merit Computer Network
1075 Beal Avenue
Ann Arbor, MI 48109
USA
+1 313 936 2120
epg@merit.edu

Laurent Joncheray
The University of Michigan
Merit Computer Network
1075 Beal Avenue
Ann Arbor, MI 48109
USA
+1 313 936 2065
lpj@merit.edu

Jean-Michel Jouanigot
CERN, European Laboratory for Particle Physics
CH-1211 Geneva 23
Switzerland
+41 22 767 4417
Jean-Michel.Jouanigot@cern.ch

Daniel Karrenberg
RIPE Network Coordination Centre
Kruislaan 409
NL-1098 SJ Amsterdam
The Netherlands
+31 20 592 5065
D.Karrenberg@ripe.net

Marten Terpstra
PRIDE Project
c/o RIPE Network Coordination Centre
Kruislaan 409
NL-1098 SJ Amsterdam
The Netherlands
+31 20 592 5064
M.Terpstra@ripe.net

Jessica Yu
The University of Michigan
Merit Computer Network
1075 Beal Avenue
Ann Arbor, MI 48109
USA
+1 313 936 2655
jyy@merit.edu

Appendix A — Syntax for the aut-num object.

Here is a summary of the tags associated with aut-num object itself and their status. The first column specifies the attribute, the second column whether this attribute is mandatory in the aut-num object, and the third column whether this specific attribute can occur only once per object [single], or more than once [multiple]. When specifying multiple lines per attribute, the attribute name must be repeated. See [6] the example for the *descr:* attribute.

```
aut-num:       [mandatory]         [single]
as-name:       [optional]          [single]
descr:         [mandatory]         [multiple]
as-in:         [optional]          [multiple]
as-out:        [optional]          [multiple]
interas-in:    [optional]          [multiple]
interas-out:   [optional]          [multiple]
as-exclude:    [optional]          [multiple]
default:       [optional]          [multiple]
tech-c:        [mandatory]         [multiple]
admin-c:       [mandatory]         [multiple]
guardian:      [mandatory]         [single]
remarks:       [optional]          [multiple]
notify:        [optional]          [multiple]
mnt-by:        [optional]          [multiple]
changed:       [mandatory]         [multiple]
source:        [mandatory]         [single]
```

Each attribute has the following syntax:

aut-num:

The autonomous system number. This must be a uniquely allocated autonomous system number from an AS registry (i.e. the RIPE NCC, the Inter-NIC, etc).

Format:

```
AS<positive integer between 1 and 65535>
```

Example:

```
aut-num: AS1104
```

Status: mandatory, only one line allowed

as-name:

The name associated with this AS. This should as short but as informative as possible.

Format:

Text consisting of capitals, dashes ("-") and digits, but must start with a capital.

Example:

```
as-name: NIKHEF-H
```

Status: single, only one line allowed

descr:

A short description of the Autonomous System.

Format:
`free text`

Example:

```
descr: NIKHEF section H
descr: Science Park Watergraafsmeer
descr: Amsterdam
```

Status: mandatory, multiple lines allowed

as-in:

A description of accepted routing information between AS peers.

Format:

from `<aut-num> <cost>` **accept** `<routing policy expression>`

The keywords **from** and **accept** are optional and can be omitted.

`<aut-num>` refers to your AS neighbour.

`<cost>` is a positive integer used to express a relative cost of routes learned. The lower the cost the more preferred the route.

`<routing policy expression>` can take the following formats.

1. A list of one or more **ASes**, **AS Macros**, **Communities** or **Route Lists**.

   A Route List is a list of routes in prefix length format, separated by commas, and surrounded by curly brackets (braces, i.e. '{' and '}').

   Examples:

   ```
   as-in: from AS1103 100 accept AS1103
   as-in: from AS786  105 accept AS1103
   as-in: from AS786   10 accept AS786 HEPNET
   as-in: from AS1755 110 accept AS1103 AS786
   as-in: from AS3333 100 accept {192.87.45.0/16, 128.141.0.0/16}
   ```

2. A set of **KEYWORDS**. The following KEYWORD is currently defined:

   ANY this means anything the neighbour AS knows.

3. A logical expression of either 1 or 2 above The current logical operators are defined as:

   AND
   OR
   NOT

   This operators are defined as true BOOLEAN operators even if the operands themselves do not appear to be BOOLEAN. Their operations are defined as follows:

```
Operator        Operation       Example

   OR           UNION           AS1 OR AS2
                                |
                                +-> all routes in AS1 or AS2.

   AND          INTERSECTION    AS1 AND HEPNET
                                |
                                +-> a route in AS1 and belonging
                                    to community HEPNET.

   NOT          COMPLEMENT      NOT AS3
                                |
                                +-> any route except AS3 routes.
```

Rules are grouped together using parenthesis i.e "(" and ")".

The ordering of evaluation of operators and there association is as follows:

```
Operator         Associativity

   ()            left to right
  NOT            right to left
  AND            left to right
   OR            left to right
```

NOTE: if no logical operator is given between ASes, AS-macros, Communities, Route Lists and KEYWORDS it is implicitly evaluated as an **'OR'** operation. The OR can be left out for conciseness. However, please note the operators are still evaluated as below so make sure you include parentheses whenever needed. To highlight this here is a simple example. If we denoted a policy of for example; from AS1755 I accept all routes except routes from AS1, A2 and AS3 and you enter the following as-in line.

```
as-in: from AS1755 100 accept NOT AS1 AS2 AS3
```

This will be evaluated as:

```
as-in: from AS1755 100 accept NOT AS1 OR AS2 OR AS3
```

Which in turn would be evaluated like this:

```
(NOT AS1) OR AS2 OR AS3
-> ((ANY except AS1) union AS2) union AS3)
--> (ANY except AS1)
```

This is clearly incorrect and not the desired result. The correct syntax should be:

```
as-in: from AS1755 100 accept NOT (AS1 AS2 AS3)
```

Producing the following evaluation:

```
NOT (AS1 OR AS2 OR AS3)
-> (ANY) except (union of AS1, AS2, AS3)
```

Which depicts the desired routing policy.
Note that can also be written as below which is perhaps somewhat clearer:

```
as-in: from AS1755 100 accept ANY AND NOT (AS1 OR AS2 OR AS3)
```

Examples:

```
as-in: from AS1755 100 accept ANY AND NOT (AS1234 OR AS513)
as-in: from AS1755 150 accept AS1234 OR {35.0.0.0/8}
```

A rule can be wrapped over lines providing the associated <aut-num>, <cost> values and **from** and **accept** keywords are repeated and occur on consecutive lines.

Example:

```
as-in: from AS1755 100 accept ANY AND NOT (AS1234 AS513)
```

and

```
as-in: from AS1755 100 accept ANY AND NOT (
as-in: from AS1755 100 accept AS1234 AS513)
```

are evaluated to the same result. Please note that the ordering of these continuing lines is significant.

Status: optional, multiple lines allowed

as-out:

A description of generated routing information sent to other AS peers.

Format:

to <aut-num> **announce** <routing policy expression

The **to** and **announce** keywords are optional and can be omitted.

`<aut-num>` refers to your AS neighbour.

`<routing policy expression>` is explained in the **as-in** attribute definition above.

Example:

```
as-out: to AS1104 announce AS978
as-out: to AS1755 announce ANY
as-out: to AS786 announce ANY AND NOT (AS978)
```

Status: optional, multiple lines allowed

interas-in:

Describes incoming local preferences on an inter AS connection.

Format:

from <aut-num> <local-rid> <neighbor-rid> <preference> **accept** <routing policy expression>

The keywords **from** and **accept** are optional and can be omitted.

<aut-num> is an autonomous system as defined in **as-in**.

<local-rid> contains the IP address of the border router in the AS describing the policy. IP address must be in prefix length format.

<neighbor-rid> contains the IP address of neighbor AS's border router from which this AS accept routes defined in the <routing policy expression>. IP addresses must be in prefix length format.

<preference> is defined as follows:

(**<pref-type>**=<value>)

It should be noted the parenthesis "(" and ")" and the "**<pref-type>**" keyword must be present for this preference to be valid.

<pref-type> currently only supports "pref". It could be expanded to other type of preference such as TOS/QOS as routing technology matures.

<value> can take one of the following values:

<cost>

<cost> is a positive integer used to express a relative cost of routes learned. The lower the cost the more preferred the route. This <cost> value is only comparable to other **interas-in** attributes, not to **as-in** attributes.

MED

This indicates the AS will use the MUTLI_EXIT_DISCRIMINATOR (MED) metric, as implemented in BGP4 and IDRP, sent from its neighbor AS.

NOTE: Combinations of MED and <cost> should be avoided for the same destinations.

CAVEAT: The pref-type values may well be enhanced in the future as more inter-ASs routing protocols introduce other metrics.

Any route specified in interas-in and not specified in as-in is assumed not accepted between the ASes concerned. Diagnostic tools should flag this inconsistency as an error. It should be noted that if an interas-in policy is specified then it is mandatory to specify the corresponding global policy in the as-in line. Please note there is no relevance in the cost associated with as-in and the preferences used in interas-in.

<routing policy expression> is an expression as defined in as-in above.

Examples:

```
interas-in: from AS1104 192.87.45.254/32 192.87.45.80/32 (pref=10) accept AS786 AS987
interas-in: from AS1104 192.87.45.254/32 192.87.45.79/32 (pref=20) accept AS987
interas-in: from AS1103 192.87.45.254/32 192.87.45.32/32 (pref=MED) accept ANY
```

Status: optional, multiple lines allowed

interas-out:

Format:

`to` `<aut-num> <local-rid> <neighbor-rid> [<metric>]` **`announce`** `<routing policy expression>`

The keywords **`to`** and **`announce`** are optional and can be omitted.

The definitions of `<aut-num>`, `<local-rid> <neighbor-rid>`, and `<routing policy expression>` are identical to those defined in **interas-in**.

`<metric>` is optional and is defined as follows:

(**`<metric-type>`**`=<value>`)

It should be noted the parenthesis "(" and ")" and the keywords of "**<metric-type>**" must be present for this metric to be valid.

<metric-type> currently only supports "metric-out". It could be expanded to other type of preference such as TOS/QOS as routing technology matures.
`<value>` can take one of the following values:

<num-metric>

`<num-metric>` is a pre-configured metric for out-bound routes. The lower the cost the more preferred the route. This `<num-metric>` value is literally passed by the routing protocol to the neighbor. It is expected that it is used there which is indicated by `pref=MED` on the corresponding interas-in attribute. It should be noted that whether to accept the outgoing metric or not is totally within the discretion of the neighbor AS.

IGP

This indicates that the metric reflects the ASs internal topology cost. The topology is reflected here by using MED which is derived from the AS's IGP metric.

NOTE: Combinations of IGP and `<num-metric>` should be avoided for the same destinations.

CAVEAT: The metric-out values may well be enhanced in the future as more interas protocols make use of metrics.

Any route specified in interas-out and not specified in as-out is assumed not announced between the ASes concerned. Diagnostic tools should flag this inconsistency as an error. It should be noted that if an interas-out policy is specified then it is mandatory to specify the corresponding global policy in the as-out line.

Examples:

```
interas-out: to AS1104 192.87.45.254/32 192.87.45.80/32 (metric-out=10) announce AS23 AS10
interas-out: to AS1104 192.87.45.254/32 192.87.45.80/32 (metric-out=15) announce AS10
interas-out: to AS1103 192.87.45.254/32 192.87.45.80/32 (metric-out=IGP) announce ANY
```

Status: optional, multiple lines allowed

as-exclude:

A list of transit ASes to ignore all routes from.

Format:

exclude <aut-num> **to** <exclude-route-keyword>

Keywords **exclude** and **to** are optional and can again be omitted.

`<aut-num>` refers to the transit AS in question.

an `<exclude-route-keyword>` can be **ONE** of the following.

1. <aut-num>
2. AS macro
3. Community
4. ANY

Examples:

`as-exclude: exclude AS690 to HEPNET`

This means exclude any HEPNET routes which have a route via AS690.

`as-exclude: exclude AS1800 to AS-EUNET`

This means exclude any AS-EUNET routes which have a route via AS1800.

`as-exclude: exclude AS1755 to AS1104`

This means exclude any AS1104 route which have a route via AS1755.

`as-exclude: exclude AS1104 to ANY`

This means exclude **all** routes which have a route via AS1104.

Status: optional, multiple lines allowed

default:

An indication of how default routing is done.

Format:

`<aut-num> <relative cost> <default-expression>`

where `<aut-num>` is the AS peer you will default route to,

and `<relative cost>` is the relative cost is a positive integer used to express a preference for default. There is no relationship to the cost used in the **as-in** tag. The AS peer with the lowest cost is used for default over ones with higher costs.

`<default-expression>` is optional and provides information on how a default route is selected. It can take the following formats:

1. **static**. This indicates that a default is statically configured to this AS peer.

2. A route list with the syntax as described in the as-in attribute. This indicates that this list of routes is used to generate a default route. A special but valid value in this is the special route used by some routing protocols to indicate default: `0.0.0.0/0`
3. **default**. This is the same as `{0.0.0.0/0}`. This means that the routing protocol between these two peers generates a true default.

Examples:

```
default: AS1755 10
default: AS786   5 {140.222.0.0/16, 192.87.45.0/24}
default: AS2043 15 default
```

Status: optional, multiple lines allowed

tech-c:

Full name or uniquely assigned NIC-handle of a technical contact person. This is someone to be contacted for technical problems such as misconfiguration.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Example:

```
tech-c: John E Doe
tech-c: JED31
```

Status: mandatory, multiple lines allowed

admin-c:

Full name or uniquely assigned NIC-handle of an administrative contact person. In many cases this would be the name of the guardian.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Example:

```
admin-c: Joe T Bloggs
admin-c: JTB1
```

Status: mandatory, multiple lines allowed

guardian:

Mailbox of the guardian of the Autonomous system.

Format:

`<email-address>`

The `<email-address>` should be in RFC822 domain format wherever possible.

Example:

```
guardian: as1104-guardian@nikhef.nl
```

Status: mandatory, only one line and e-mail address allowed

remarks:

Remarks/comments, to be used only for clarification.

Format:

`free text`

Example:

```
remarks: Multihomed AS talking to AS1755 and AS786
remarks: Will soon connect to AS1104 also.
```

Status: optional, multiple lines allowed

notify:

The notify attribute contains an email address to which notifications of changes to this object should be sent. See also [11].

Format:

```
<email-address>
```

The `<email-address>` should be in RFC822 domain syntax wherever possible.

Example:

```
notify: Marten.Terpstra@ripe.net
```

Status: optional, multiple lines allowed

mnt-by:

The mnt-by attribute contains a registered maintainer name. See also [11].

Format:

```
<registered maintainer name>
```

Example:

```
mnt-by: RIPE-DBM
```

Status: optional, multiple lines allowed

changed:

Who changed this object last, and when was this change made.

Format:

```
<email-address> YYMMDD
```

`<email-address>` should be the address of the person who made the last change. `YYMMDD` denotes the date this change was made.

Example:

```
changed: johndoe@terabit-labs.nn 900401
```

Status: mandatory, multiple lines allowed

source:

Source of the information.

This is used to separate information from different sources kept by the same database software. For RIPE database entries the value is fixed to `RIPE`.

Format:

```
RIPE
```

Status: mandatory, only one line allowed

Appendix B — Syntax details for the community object.

Here is a summary of the tags associated with community object itself and their status. The first column specifies the attribute, the second column whether this attribute is mandatory in the community object, and the third column whether this specific attribute can occur only once per object [single], or more than once [multiple]. When specifying multiple lines per attribute, the attribute name must be repeated. See [6] the example for the *descr:* attribute.

```
community:      [mandatory]          [single]
descr:          [mandatory]          [multiple]
authority:      [mandatory]          [single]
guardian:       [mandatory]          [single]
tech-c:         [mandatory]          [multiple]
admin-c:        [mandatory]          [multiple]
remarks:        [optional]           [multiple]
notify:         [optional]           [multiple]
mnt-by:         [optional]           [multiple]
changed:        [mandatory]          [multiple]
source:         [mandatory]          [single]
```

Each attribute has the following syntax:

community:

Name of the community. The name of the community should be descriptive of the community it describes.

Format:

Upper case text string which cannot start with `"AS"` or any of the `<routing policy expression>` KEYWORDS. See Appendix A.

Example:

```
community: WCW
```

Status: mandatory, only one line allowed

descr:

A short description of the community represented.

Format:

`free text`

Example:

```
descr: Science Park Watergraafsmeer
descr: Amsterdam
```

Status: mandatory, multiple lines allowed

authority:

The formal authority for this community. This could be an organisation, institute, committee, etc.

Format:

`free text`

Example:

```
authority:  WCW LAN Committee
```

Status: mandatory, only one line allowed

guardian:

Mailbox of the guardian of the community.

Format:

```
<email-address>
```

The `<email-address>` should be in RFC822 domain format wherever possible.

Example:

```
guardian: wcw-guardian@nikhef.nl
```

Status: mandatory, only one line and email address allowed

tech-c:

Full name or uniquely assigned NIC-handle of an technical contact person for this community.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Example:

```
tech-c: John E Doe
tech-c: JED31
```

Status: mandatory, multiple lines allowed

admin-c:

Full name or uniquely assigned NIC-handle of an administrative contact person. In many cases this would be the name of the guardian.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Example:

```
admin-c: Joe T Bloggs
admin-c: JTB1
```

Status: mandatory, multiple lines allowed

remarks:

Remarks/comments, to be used only for clarification.

Format:

```
free text
```

Example:

```
remarks: Temporary community
remarks: Will be removed after split into ASes
```

Status: optional, multiple lines allowed

notify:

The notify attribute contains an email address to which notifications of changes to this object should be send. See also [11].

Format:

```
<email-address>
```

The <email-address> should be in RFC822 domain syntax wherever possible.

Example:

```
notify: Marten.Terpstra@ripe.net
```

Status: optional, multiple lines allowed

mnt-by:

The mnt-by attribute contains a registered maintainer name. See also [11].

Format:

```
<registered maintainer name>
```

Example:

```
mnt-by: RIPE-DBM
```

Status: optional, multiple lines allowed

changed:

Who changed this object last, and when was this change made.

Format:

```
<email-address> YYMMDD
```

<email-address> should be the address of the person who made the last change. YYMMDD denotes the date this change was made.

Example:

```
changed: johndoe@terabit-labs.nn 900401
```

Status: mandatory, multiple lines allowed

source:

Source of the information.

This is used to separate information from different sources kept by the same database software. For RIPE database entries the value is fixed to RIPE.

Format:

```
RIPE
```

Status: mandatory, only one line allowed

Appendix C — AS Macros syntax definition.

Here is a summary of the tags associated with as-macro object itself and their status. The first column specifies the attribute, the second column whether this attribute is mandatory in the as-macro object, and the third column whether this specific attribute can occur only once per object [single], or more than once [multiple]. When specifying multiple lines per attribute, the attribute name must be repeated. See [6] the example for the *descr:* attribute.

```
as-macro:      [mandatory]          [single]
descr:         [mandatory]          [multiple]
as-list:       [mandatory]          [multiple]
guardian:      [mandatory]          [single]
tech-c:        [mandatory]          [multiple]
admin-c:       [mandatory]          [multiple]
remarks:       [optional]           [multiple]
notify:        [optional]           [multiple]
mnt-by:        [optional]           [multiple]
changed:       [mandatory]          [multiple]
source:        [mandatory]          [single]
```

Each attribute has the following syntax:

as-macro:

The name of a macro containing at least two Autonomous Systems grouped together for ease of administration.

Format:

```
AS-<string>
```

The `<string>` should be in upper case and not contain any special characters.

Example:

```
as-macro: AS-EBONE
```

Status: mandatory, only one line allowed

descr:

A short description of the Autonomous System Macro.

Format:

```
free text
```

Example:

```
descr:  Macro for EBONE connected ASes
```

Status: mandatory, multiple lines allowed

as-list:

The list of ASes or other AS macros that make up this macro. It should be noted that recursive use of AS macros is to be encouraged.

Format:

```
<aut-num> <as-macro> ...
```

See Appendix A for `<aut-num>` definition.

Example:

```
as-list: AS786 AS513 AS1104
as-list: AS99 AS-NORDUNET
```

Status: mandatory, multiple lines allowed

guardian:

Mailbox of the guardian of this AS macro.

Format:

`<email-address>`

The `<email-address>` should be in RFC822 domain format wherever possible.

Example:

```
guardian: as-ebone-guardian@ebone.net
```

Status: mandatory, only one line and e-mail address allowed

tech-c:

Full name or uniquely assigned NIC-handle of a technical contact person for this macro. This is someone to be contacted for technical problems such as misconfiguration.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Examples:

```
tech-c: John E Doe
tech-c: JED31
```

Status: mandatory, multiple lines allowed

admin-c:

Full name or uniquely assigned NIC-handle of an administrative contact person. In many cases this would be the name of the guardian.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Examples:

```
admin-c: Joe T Bloggs
admin-c: JTB1
```

Status: mandatory, multiple lines allowed

remarks:

Remarks/comments, to be used only for clarification.

Format:

`free text`

Example:

```
remarks: AS321 will be removed from this Macro shortly
```

Status: optional, multiple lines allowed

notify:

The notify attribute contains an email address to which notifications of changes to this object should be send. See also [11].

Format:
`<email-address>`

The `<email-address>` should be in RFC822 domain syntax wherever possible.

Example:

```
notify: Marten.Terpstra@ripe.net
```

Status: optional, multiple lines allowed

mnt-by:

The mnt-by attribute contains a registered maintainer name. See also [11].

Format:
`<registered maintainer name>`

Example:

```
mnt-by: RIPE-DBM
```

Status: optional, multiple lines allowed

changed:

Who changed this object last, and when was this change made.

Format:
`<email-address> YYMMDD`

`<email-address>` should be the address of the person who made the last change. `YYMMDD` denotes the date this change was made.

Example:

```
changed: johndoe@terabit-labs.nn 900401
```

Status: mandatory, multiple lines allowed

source:

Source of the information.

This is used to separate information from different sources kept by the same database software. For RIPE database entries the value is fixed to `RIPE`.

Format:
`RIPE`

Status: mandatory, only one line allowed

Appendix D — Syntax for the "route" object.

There is a summary of the tags associated with route object itself and their status. The first column specifies the attribute, the second column whether this attribute is mandatory in the community object, and the third column whether this specific attribute can occur only once per object [single], or more than once [multiple]. When specifying multiple lines per attribute, the attribute name must be repeated. See [6] the example for the *descr:* attribute.

```
route:          [mandatory]         [single]
descr:          [mandatory]         [multiple]
origin:         [mandatory]         [single]
hole:           [optional]          [multiple]
withdrawn:      [optional]          [single]
comm-list:      [optional]          [multiple]
remarks:        [optional]          [multiple]
notify:         [optional]          [multiple]
mnt-by:         [optional]          [multiple]
changed:        [mandatory]         [multiple]
source:         [mandatory]         [single]
```

Each attribute has the following syntax:

route:

Route being announced.

Format:

Classless representation of a route with the RIPE database known as the "prefix length" representation. See [10] for more details on classless representations.

Examples:

`route: 192.87.45.0/24`

This represents addressable bits 192.87.45.0 to 192.87.45.255.

`route: 192.1.128.0/17`

This represents addressable bits 192.1.128.0 to 192.1.255.255.

Status: mandatory, only one line allowed

origin:

The autonomous system announcing this route.

Format:

`<aut-num>`

See appendix A for `<aut-num>` syntax.

Example:

`origin: AS1104`

Status: mandatory, only one line allowed

hole:

Denote the parts of the address space covered this route object to which the originator does not provide connectivity. These holes may include routes that are being currently routed by another provider (e.g., a customer using that space has moved to a different service provider). They may also include space that has not yet been assigned to any customer.

Format:

Classless representation of a route with the RIPE database known as the "prefix length" representation. See [10] for more details on classless representations. It should be noted that this sub-aggregate must be a component of that registered in the route object.

Example:

```
hole: 193.0.4.0/24
```

Status: optional, multiple lines allowed

withdrawn:

Used to denote the day this route has been withdrawn from the Internet routing mesh. This will be usually be used when a less specific aggregate route is now routed the more specific (i.e. this route) is not need anymore.

Format:

`YYMMDD`

`YYMMDD` denotes the date this route was withdrawn.

Example:

```
withdrawn: 940711
```

Status: optional, one line allowed.

comm-list:

List of one or more communities this route is part of.

Format:

`<community> <community> ...`

See Appendix B for `<community>` definition.

Example:

```
comm-list: HEP LEP
```

Status: optional, multiple lines allowed

remarks:

Remarks/comments, to be used only for clarification.

Format:

`free text`

Example:

```
remarks: Multihomed AS talking to AS1755 and AS786
remarks: Will soon connect to AS1104 also.
```

Status: optional, multiple lines allowed

notify:

The notify attribute contains an email address to which notifications of changes to this object should be send. See also [11].

Format:
`<email-address>`

The `<email-address>` should be in RFC822 domain syntax wherever possible.

Example:

```
notify: Marten.Terpstra@ripe.net
```

Status: optional, multiple lines allowed

mnt-by:

The mnt-by attribute contains a registered maintainer name. See also [11].

Format:
`<registered maintainer name>`

Example:

```
mnt-by: RIPE-DBM
```

Status: optional, multiple lines allowed

changed:

Who changed this object last, and when was this change made.

Format:
`<email-address> YYMMDD`

`<email-address>` should be the address of the person who made the last change. `YYMMDD` denotes the date this change was made.

Example:

```
changed: johndoe@terabit-labs.nn 900401
```

Status: mandatory, multiple lines allowed

source:

Source of the information.

This is used to separate information from different sources kept by the same database software. For RIPE database entries the value is fixed to `RIPE`.

Format:
`RIPE`

Status: mandatory, only one line allowed

Appendix E - List of reserved words

The following list of words are reserved for use within the attributes of the AS object. The use of these words is solely for the purpose of clarity. All keywords must be lower case.

```
accept
announce
exclude
from
to
transit
```

Examples of the usage of the reserved words are:

as-in: from <neighborAS> accept <route>

as-out: to <neighborAS> announce <route>

as-exclude: exclude <ASpath> to <destination>

as-transit: transit <ASpath> to <destination>

default: from <neighborAS> accept <route>

default: to <neighborAS> announce <route>

Note: that as-transit is an experimental attribute. See section 10.

Appendix F - Motivations for RIPE-81++

This appendix gives motivations for the major changes in this proposal from ripe-81.

The main goals of the routing registry rework are:

SPLIT
: Separate the allocation and routing registry functions into different database objects. This will facilitate data management if the Internet registry and routing registry functions are separated (like in other parts of the world). It will also make more clear what is part of the routing registry and who has authority to change allocation vs. routing data.

CIDR
: Add the possibility to specify classless routes in the routing registry. Classless routes are being used in Internet production now. Aggregation information in the routing registry is necessary for network layer troubleshooting. It is also necessary because aggregation influences routing policies directly.

CALLOC
: Add the possibility to allocate address space on classless boundaries in the *allocation* registry. This is a way to preserve address space.

CLEAN
: To clean up some of the obsolete and unused parts of the routing registry.

The major changes are now discussed in turn:

Introduce Classless Addresses

CIDR, CALLOC

Introduce route object.

SPLIT, CIDR and CALLOC.

Delete obsolete attributes from inetnum.

CLEAN.

Delete RIPE-DB and LOCAL from routing policy expressions.

CLEAN

Allow multiple ASes to originate the same route

Because it is being done. CIDR. Made possible by SPLIT.

Appendix G — Transition strategy from RIPE-81 to RIPE-81++

Transition from the routing registry described by ripe-81 to the routing registry described in this document is a straightforward process once the new registry functions have been implemented in the database software and are understood by the most commonly used registry tools. The routing related attributes in the classful inetnum objects of ripe-81 can be directly translated into new routing objects. Then these attributes can be deleted from the inetnum object making that object if conform to the new schema.

Proposed transition steps:

1) Implement classless addresses and new object definition in the database software.

2) Make common tools understand the new schema and prefer it if both old and new are present.

3) Invite everyone to convert their data to the new format. This can be encouraged by doing conversions automatically and proposing them to maintainers.

4) At a flag day remove all remaining routing information from the inetnum objects. Before the flag day all usage of obsoleted inetnum attributes has to cease and all other routing registry functions have to be taken over by the new objects and attributes.