www.euro6ix.net

Title:	Document Version:
Technical Report TR4.1A.10 **IX Support Tools**	0.4

Project Number:	Project Acronym:	Project Title:
IST-2001-32161	Euro6IX	European IPv6 Internet Exchanges Backbone

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type* - Security**:
31/12/2002	25/02/2003	R – PU

* Type: P - Prototype, R - Report, D - Demonstrator, O - Other
** Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

Responsible and Editor/Author:	Organization:	Contributing WP:
David Fernández	UPM	WP4

Authors (organizations) in alphabetical order:

Fermín Galán (UPM).

Abstract:

This annex to WP4 deliverable D4.1A summarizes the activities carried out inside the IX Support Tools subactivity during the first year. This subactivity has to do with the development and testing of applications and tools related with the IXs architecture and associated services in the context of Euro6IX network.

Work under this subactivity has focused during the first year on the IXs Route Server functionality. The general objective was to gain experience on the design, set-up and management of a route server based IPv6 IXs. The work carried out here complements and provides experimental feedback to WP2 and WP3 activities in this area, with the final object of experimenting with and deploying route server based IXs over Euro6IX network.

Keywords:

IX, BGP, IPv6, MRT, Route Server, RPSL, User Mode Linux, Zebra

Revision History

Revision	Date	Description	Author (Organization)
v0.1	15/12/2002	Document creation. Main contribution written.	Fermín Galán (UPM)
v0.2	26/12/2003	Introduction and Route Server sections added	David Fernández (UPM)
v0.3	3/1/2003	Final revision and corrections	David Fernández (UPM)
v0.4	25/02/2003	Logos added an PDF generated	Jordi Palet (Consulintel)

Executive Summary

This annex to WP4 deliverable D4.1A summarizes the activities carried out inside the IX Support Tools subactivity during the first year. This subactivity has to do with the development and testing of applications and tools related with the IXs architecture and associated services in the context of Euro6IX network.

Work under this subactivity has focused during the first year on the IXs Route Server functionality. The general objective was to gain experience on the design, set-up and management of a route server based IPv6 IXs. The work carried out here complements and provides experimental feedback to WP2 and WP3 activities in this area, with the final object of experimenting with and deploying route server based IXs over Euro6IX network.

The subactivity has been organized around three main activities:

- Collect and study information about available Route Server implementations for IPv6 (mainly based on open source code), as well as related functionalities like routing policy databases and associated tools.

- Select one or more implementations for further testing over laboratory testbeds, including the set-up of a routing policy registry and the use of RS tools to automate the configuration of routers.

- Design and implement an IX emulation environment to allow the experimentation of complex IPv6 BGP routing scenarios without involving the cost of maintaining large laboratory testbeds made of an important number of machines.

Most of the subactivity objectives have been fulfilled: several implementations of the basic components of a route server based IX have been tested and a first prototype of the IX emulation environment has been successfully used to test medium size IX scenarios (around twenty peering routers). However, the lack of support of IPv6 extensions to RPSL language in the routing policy databases implementations has prevented us from testing a complete scenario.

The document is organized as follow. After a brief introduction in Section 1, which presents the general and particular objectives of the subactivity, Section 2 summarizes the general architecture of a Route Server based IX. Later, Section 3 shows the results of the analysis and tests carried out with the route server implementations selected. Section 4 describes the IX emulation environment created to allow the testing of complex scenarios inside one machine, as well as the scenarios tested over it. Section 6 summarizes the results of the tests carried out with routing policies databases implementations. Finally, Section 7 comes to some conclusions and Section 8 presents some proposals for future activities under this area.

Table of Contents

Table of Figures

1. INTRODUCTION

This subactivity has to do with the development and testing of applications and tools related with IntereXchange points (IX) architecture and associated services in the context of Euro6IX network. As described in [1], IXs will include, apart from the basic layer 2 switching infrastructure, a set of associated services like routing support tools (route servers, looking glasses, etc), transition facilities, router hosting facilities, server farms infrastructure, basic services like DNS, etc. This subactivity will cover the development, integration or adaptation of applications or tools that support these services.

During the first year of the project, work under this subactivity has focused on the Route Server functionality. The general objective was to gain experience on the design, set-up and management of a route server based IPv6 IXs. The work carried out here complements and provides experimental feedback to WP2 and WP3 activities in this area, with the final object of experimenting with and deploying route server based IXs over Euro6IX network.

In particular, the specific objectives for the first year have been the following:

- Collect and study information about available Route Server implementations for IPv6 (mainly based on open source code), as well as related functionalities like routing policy databases and associated tools. Focus will be on knowing the degree of IPv6 support of the implementations investigated.

- Select one or more implementations for further testing over laboratory testbeds. Pilot scenarios will include, if available, the set-up of a routing policy registry and the use of RS tools to automate the configuration of routers.

- Design and implement an IX emulation environment that allows the experimentation of complex IPv6 BGP routing scenarios without involving the cost of maintaining a laboratory testbed made of an important number of machines. Basically, we will focus on how to emulate an IX scenario where an important number of ISPs (around fifty) are peering through a route server using a small number of computers (ideally only one).

The document is organized as follows. Section 2 summarizes the general architecture of a Route Server based IX. Section 3 shows the results of the analysis of the route server implementations selected for testing. Section 4 describes the IX emulation environment developed. Section 6 summarizes the results of the preliminary tests carried out with two routing policies database implementations. Finally, Section 7 comes to some conclusions and Section 8 presents proposals for future activities under this area.

2. ROUTE SERVER FUNCTIONALITY

In order to make the document readable, this section presents a brief introduction to the Route Server functionality and the architecture of a Route Server based IX. More detailed information about this subject can be found in [1] (where most of the text presented here comes from), in [2] or in the references mentioned in this section.

2.1 Route Server based IX Architecture

Maintaining a complete mesh topology of BGP peering sessions in an IX could be difficult to manage if the number of participants is high (some present IXs host more than 50 participants) or if some of the participants do not have enough technical expertise. Each router has to maintain a high number of peering sessions and configure a different set of filters for each one. Besides, when a new participant comes to the IX, almost all routers would have to modify its configuration to peer with the new participant. For all of these reasons, the scalability of IXs based on full mesh peering sessions is seriously compromised.

The main proposal to solve these scalability problems came as a result of the Routing Arbiter Project [3], started in 1993 as part of the initiatives to design the post-NSFNET Internet in USA around the Network Access Points (NAPs). This project gave rise to the ***Route Server*** concept.

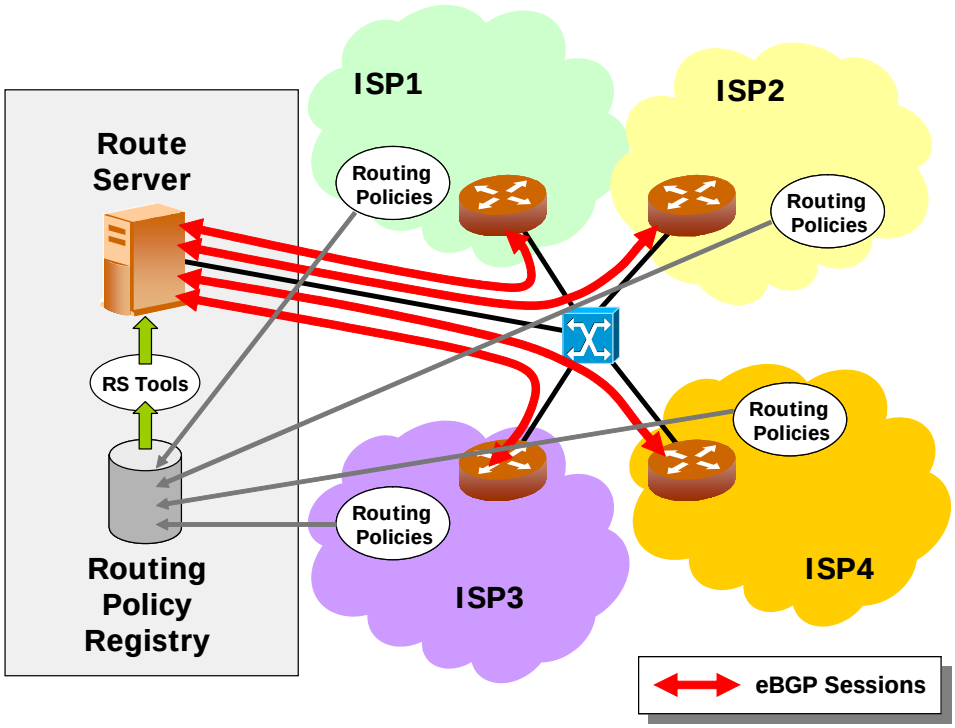

Figure 2-1: Peering through a Route Server

The Router Arbiter Project proposed the creation of a new system, named the ***Route Server (RS)***, which should centralize the interchange of routes between participants in an IX. Instead of peering among each other, providers peer only with the RS. Figure 2-1 shows the architecture of a route server based IX.

As stated in [4], “the Route Server facilitate routing exchange among IX-attached ISPs by gathering routing information from ISPs routers, processing the information based on the ISP's routing policy requirements, and passing the processed routing information to each ISP router. The RS uses the BGP-4 inter-domain routing protocol to exchange routing information with each ISP router”.

“The Route Server does not forward packets among the ISP routers attached to a connection point. Instead, it uses BGP's third-party routing information capabilities to pass routing information from one ISP to another, with the next hop pointing to the ISP router that advertises the route to the RS. Traffic is therefore exchanged directly among the ISP routers on the NAP, even though the routing information is provided by the Route Server”.

Basically a Route Server is a modified BGP4 daemon capable of maintaining several copies of the Routing Information Base (RIB), commonly named *Views*, one for each participant ISP.

One of the advantages of a RS based IX architecture is that routing policies can be centralized and applied by the RS. As shown in Figure 2-1, instead of directly applying its routing policies, ISPs store them in a common database named ***Routing Policy Registry***, using a ***Routing Policy Description Language***. The policies are processed using some tools (named ***RS Tools*** in the figure) and stored in the RS where they are used to filter routing updates and create particular “views” for each ISP with exactly the same content as if the ISP would have peered directly with other providers. For instance, the RS could give a different path towards a given destination to different ISPs, in case those paths were available and the policy requirements of the different ISPs stated it.

In summary, Route Servers facilitate and make safer the interconnection between participants at an IX. While typically global-scale backbone networks avoid the use of Route Servers, smaller regional and local ISPs, who often have fewer technical staff or use less restrictive routing policies, could benefit from its use as it simplifies the configuration and management of their routers.

As mentioned in [5], route servers can greatly accelerate the creation of new IXs by allowing each participating network to configure a single peering session with the route server, rather than a full mesh of individual adjacencies with each of the other participants. As new participants join the exchange, they receive the immediate benefit of full use of the exchange as soon as they contact the route server, rather than incrementally gaining use of the exchange as they configure each of many peering sessions.

Besides, route servers also increase the degree of safety to the participants, since they're configured with filters which ensure the accuracy of the routing information which passes through them, between participants. This filtering prevents the widespread propagation of network outages caused by participants' accidental missconfiguration of routers.

The next subsections describe with more detail the main components of a route server based IX.

2.2 Route Server

It is typically a UNIX based workstation running a modified version of a BGP daemon able to maintain multiple routing databases (views), one for each participating in the IX. Contrary to normal BGP implementations, RSs do not install the learned routes into its own routing tables; they just act as intermediaries between participants. They usually require an important amount of memory, although they are not critic in other aspects, as they do not switch data traffic, which goes directly between peer routers through L2 IX switches.

Route Servers typically have their own AS number and can be configured for each peering session to insert or not its own number on the AS_PATH attribute of routes exchanged. In the former case, the Route Server is said to work in transparent mode and, as the AS number is not propagated, identifiers from the AS numbers private range (64512 to 65535) can be used.

As the Route Server becomes a critic point of failure in the whole routing systems, it is typically possible group two or more Route Servers in a cluster to provide fault tolerance.

2.3 Routing Policy Specification Languages

Routing Policy Specification languages have been designed to express the routing policies of an AS in order to store them in routing policy databases. Besides, they allow the expression of general information about ASs (contact names, e-mails, etc), to facilitate the diagnostic and solution of global routing problems.

The language widely used nowadays is the ***Routing Policy Specification Language (RPSL)*** [6]. It was developed by the IETF Routing Policy System Working group and derived from a previous language defined by RIPE [7].

RPSL specifies the object-oriented data that contain pieces of policy and administrative information in attribute-value pairs. RPSL allows a network operator to specify routing policies at various levels in the Internet routing system such as those policies affecting Autonomous System (AS) articulation and router configuration. RPSL is designed to work in environments like the one shown in Figure 2-1, where router configurations can be generated from the description of the policy for one AS combined with the description of a router.

RPSL, together with some additions and improvements proposed by RIPE, is the language used in most of the public routing policy registries in Internet, like RIPE, ARIN, RADB or APNIC.

At present, RPSL does not support IPv6. Some limited extensions have been proposed to store IPv6 information in routing databases and facilitate the coordination of the 6BONE network. That is the case, for example, of the “inetnum6” or “ipv6site” object types, defined to store IPv6 addresses and information about IPv6 sites respectively, and implemented in 6BONE routing registry [8].

However, deeper modifications to RPSL are needed in order to completely describe IPv6 routing policies. A new version of RPSL, named RPSL Next Generation (RPSLng), is at present being discussed and defined in the context of RIPE Routing Working Group. A new draft has been recently posted [9]. However, there is neither a general agreement on it nor implementations available.

2.4 Routing Policy Databases

They are just the databases where the routing policies of ASs are stored. They can be queried using different tools, being the most commonly used the ***whois*** application. There are at presents more than 50 around the Internet operated by RIRs or organizations such as Verio, Cable and Wireless, and Merit. All together form what has been named *The Internet Routing Registry (IRR)* [10], a public repository of announced routes and routing policy in a common format that ISPs use to configure their backbone routers, analyze routing policies, and to build tools to help in the effort of coordinate interdomain routing.

Merit Network Inc. operates since 1995 the main routing policy database in the Internet, named RADB [10]. RABD was the database set up for the Routing Arbiter Project. Nowadays it stores routing policies form an important number of providers, but its main mission is "to mirror all component databases so as to provide the most complete view of the entire IRR". Currently, querying the Merit whois server, you can retrieve information from more that 34 IRR databases.

2.5 Route Server Tools

The operation of a Route Server needs to be supported by a set of tools that help in tasks like the automatic generation of router configurations starting from databases contents, the analysis of routing policies or the diagnostic of global routing problems.

An initial set of tools were designed and developed inside the Router Arbiter Project named the ***RAToolSet***. They are basically the standard set of tools used worldwide. Later, as USC Information Sciences Institute -the initial developer of RAToolSet- discontinued its effort on them, a new project was created by RIPE, the ***Internet Routing Registry Toolset Project***, in order to continue its development and support. The set of tools has also been renamed to ***IRRToolSet*** [11].

As neither RPSLng is completely defined nor routing policy databases implement it, IRRTools available do not support IPv6.

3. EVALUATION OF ROUTE SERVER IMPLEMENTATIONS

After an investigation about available IPv6 BGP implementations that could be used to test route server functionality in the laboratory, three routing suites were selected and evaluated: MRTd [12], Zebra [13] and Cisco IOS [14]. They all share some characteristics:

- All of them implement a comprehensive set of Internet routing protocols (RIP, OSPF, BGP, etc.) and, in particular, they have support for IPv6 BGP, which we are mainly interested in. However not all support route server functionality, as it will be mentioned later.

- They all offer a User Interactive Interface (UII) that allows remote user access to the routing processes to modify the configuration (stored in a configuration file or in an EPROM) and monitor protocols behaviour. The access to this UII is performed through telnet sessions (a password is required to authenticate accesses). The commands used in the UII are very similar (in fact, Zebra and MRTd UIIs are based on Cisco IOS UII), with slight differences.

- They all support BGP route filtering (that is, they accept or deny routes that match certain conditions, such having a particular AS number in the as-path attribute) and other capabilities typically used in IXs.

The main differences between MRTd, Zebra and Cisco IOS related with their functionality are described in the following sections.

3.1 MRT

Tested version: 2.2.2a (released 14/08/2000).

MRT is released as a routing toolbox including a set of programs. The main one is ***mrtd***, a monolithic multi-threaded background daemon that implements several routing protocols. In particular, it implements BGP with IPv6 support. The mrtd daemon can act as a route server (configuring transparency options in BGP peerings), as well as a route reflector.

However, MRT is no longer supported (since last version was released) and its main disadvantages come from its obsolescence: some difficulties to install the software over modern systems (for example, GNU/Linux with glibc-2), instabilities and the lack of documentation, between other problems.

A useful simulation tool, ***bgpsim***, is provided with MRT. It is designed to help testing complex BGP scenarios, allowing establishing peering sessions, defining the prefixes and route attributes interchanged in it, and some stability conditions (for example, prefixes that flap within certain time interval).

MRT IPv6 capabilities have been satisfactory tested (with occasional stability problems) over FreeBSD 4.5 and GNU/Linux 2.4 IPv6 capable kernels. Other Unices are also supported (check documentation of the program), but not always IPv6 support is granted.

Although -due to its obsolescence- MRT has not been selected to be used either in the emulation environment described in Section 4 or in other testing environments, bgpsim tool has showed to be very useful to complement the scenarios based on other implementations, allowing the creation of fictitious routes or instabilities situations.

3.2 Zebra

Tested version: 0.93a (released 08/07/2002).

Zebra takes a modular approach in order to implement the routing functionality. Instead of being a single monolithic routing process, it is spited in several small processes; each one specialized in certain tasks. The main process is ***zebra***, the kernel routing manager; other processes implement each one a different routing protocol (RIP, OSPF, etc). We have focused our work on ***bgpd***, the BGP daemon process with IPv6 support. Additionally, another process, ***vsh***, integrates the UII for all routing processes. This modular architecture is shown in Figure 3-1.

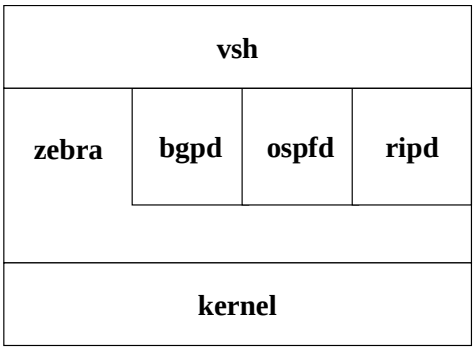

Figure 3-1: Zebra architecture

Unlike MRT or Cisco IOS, Zebra does not need that route prefixes announced in peering sessions are real routes (indeed, routes known by the kernel)[1]. This is very useful to avoid route flapping that may happen due to network topology stability problems.

Route server configurations, as well as route reflection, are possible using Zebra bgpd. Furthermore, bgpd allows to have several separate *instances* or *views* inside the same routing process (this is useful in scenarios where the same router is part of several ASs). The main difference between views and instances is that views do not install learned routes into kernel routing table.

Although it is theoretically possible to run the bgpd process alone, it is recommendable to run also the zebra process, because bgpd queries zebra to get certain parameters (for example, the IPv4 address of an interface in order to establish router-ID). So, zebra and bgpd should run together, for the sake of stability.

Zebra IPv6 capabilities have been tested satisfactory over FreeBSD 4.5 and GNU/Linux 2.4 IPv6 capable kernels. Other Unices are also supported (check documentation of the program), but not always IPv6 support is granted.

[1] MRT and Cisco IOS solve this problem configuring “fake” routes using loopback interfaces. These routes have a high weight attribute, in order to not being used to route actual traffic.

3.3 Cisco IOS

Tested version: 12.2 (released 04/05/2002)

IOS is the operating system used in Cisco routers. It is very specialized on routing tasks. Stability, reliability and high performance are its main characteristics. Additionally, IOS allows the easy integration of BGP with other routing processes (like OSPF or RIP distribution of routes learn by BGP into AS inner routers). IPv6 is supported without problems.

However, Cisco IOS can not act as a route server, although in can be used as route reflector for iBGP sessions. Nevertheless, as most other BGP implementations, it can act as a route server client in an IX. This fact, together with the availability of CISCO routers in our labs, led us to select Cisco implementation for route server tests.

3.4 Conclusions

The following table shows a comparison of characteristics of the three alternatives evaluated before. Criteria are: route server functionality, BGP client functionality, documentation and maintenance, stability and portability.

Program	Route Server Functionality	BGP Client Functionality	Documentation & Maintenance	Stability	Portability
MRTd	Route server capable.	Basic client functionality. Includes bgpsim simulator.	Poor documentation. Not maintained.	Good.	GNU/Linux, FreeBSD and other Unices.
Zebra	Route server capable.	Basic client functionality. Do not need announce actual route prefixes	Very poor documentation. Currently maintained.	Very good.	GNU/Linux, FreeBSD and other Unices.
Cisco IOS	No route server capable.	Basic client functionality.	Good documentation. Maintained.	Excellent.	Only in Cisco routers.

Figure 3-2: MRT, Zebra and CISCO ISO comparison

MRTd is not maintained software. This fact, together with a slight better functionality (for example, route server configuration is easier and the prefixes announced do not need to be real routes) and a better stability, makes Zebra the best option to implement the route server functionality in Euro6IX. However, MRT's bgpsim is a useful tool in order to created emulated scenarios.

As mentioned, Cisco IOS can not act as route server, but it is a good option to implement the clients of a route server in the boundary of the ASes.

4. IX EMULATION ENVIRONMENT

4.1 Requirements

In order to test and evaluate route server functionality using the implementations mentioned before and to study and analyze complex IXs scenarios (like the ones being discussed inside Euro6IX project) we decided to create an IX emulation environment, with the following requirements:

- It should be able to emulate complex scenarios made of tenths of peering routers interchanging a big amount of prefixes -with or without a route server- using few machines (ideally a single box).
- It should allow the connection of peers running on external equipment (like a Cisco router).
- Route server and BGP implementations (like Zebra) should be tested as they would be running on a real implementation. For them, the emulation scenario should be as transparent as possible.
- The environment should be highly configurable and flexible, capable of emulating different complex scenarios.
- The emulation environments should be made, if possible, using open source tools, avoiding license cost.

Basically we looked for a flexible environment to test route server based IXs scenarios without involving the high investment in resources needed to create it using real equipment.

4.2 Analysis of alternatives

Before reaching the final implementation using UML (described in section 4.3), several alternatives were considered in order to build the emulation system that fulfills the requirements pointed above. The next subsections briefly describe each alternative evaluated, together with the problems they present.

4.2.1 Several routing process in the same box

The first alternative studied consists in running several BGP routing processes inside the same box, each one simulating the peer router of a complex routing scenario. For example, five Zebra bgpd processes could be run inside the same PC host, four of them acting as a boundary router of an ASs and a fifth acting as route server.

However, in this simple approach to emulate routing scenarios, network resources sharing problems arise. In particular, all BGP processes try to bind to the default port (TCP 179). This is not a problem in a real case, where each process runs in a different box (Figure 4-1), but when running them in the same host, processes can not share the same port. The same situation happens with the UII port used to establish management telnet sessions to the BGP processes.

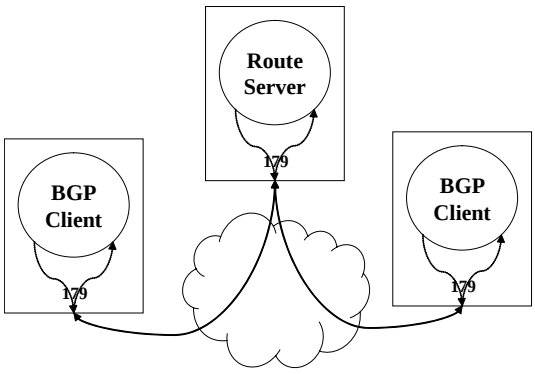

Figure 4-1: Real BGP scenario

Note that the problem can not be solved using different network interfaces and/or alias of existing interfaces (that is, having each BGP process bound to a different interface, so there is no conflict opening TCP sockets with the same port value), as it is the case of other applications (see, for example, the emulation environment designed for DNSSEC tests in Annex 5 of this deliverable). Although this solution could be theoretically possible, it does not work with the implementations tested.

The solution to the problem could be that every BGP process uses a different TCP ports to establish peering sessions with neighbors, as shown in Figure 4-2.

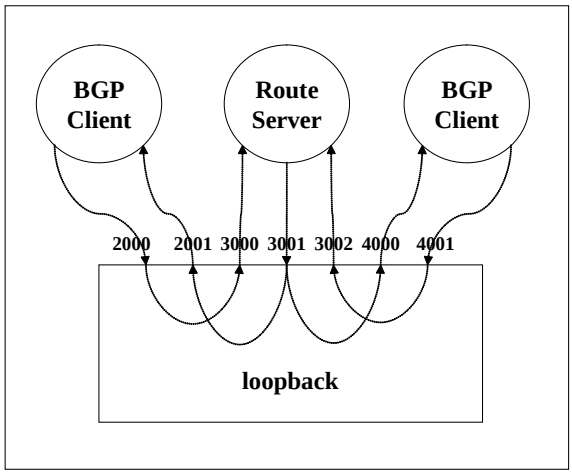

Figure 4-2: BGP scenario using non-default ports

By default, BGP processes use the same port (TCP 179) to listen to and connect with its neighbors. This behavior can be overridden:

- In Zebra, the listening port can be established globally (indeed, the same listen port to all neighbors) and the connection port can be configured for each neighbor peer.

- MRT's bgpsim does not allow changing the default port. However, modifications of the source code of the program (MRT is an open source project), allows to set listening and connection ports (globally) to a different value.

Anyway, at the end changing default ports is not a good solution, due to several reasons:

- Low flexibility and difficulty of configuration: it is hard to keep coherence between all peer routers configuration (the connection port of a peer must match with the listening port of its neighbor).

- It does not allow the integration of other BGP implementations that do not allow changing the port used.

- The requirement of transparency is not accomplished (processes would use port number that do not use in a real situation).

4.2.2 FreeBSD jails

Jails [15] are a FreeBSD operating system extension that allows running *jailed* processes in the user process space. A jailed process is much like a normal process, except that it runs in a virtual separate environment (including a different root filesystem and one or more network interfaces). Local communication with other processes in the system using standard interprocesses communications mechanisms (for example, using software signals or through shared memory) is forbidden; although communications using TCP/IP through the jail network interface/s is allowed.

The main utility of jails is to run secure services, that is, services than even when they are compromised by attackers, they do not compromise the security of the system where they run.

Jails can be used to create the emulation system we look for, as it allows running several BGP processes in the same physical box, each one inside a separate jail and with TCP/IP connectivity between them. Thus, there is no problem using the same TCP port for all the process, because of each one is jailed in a separate network interface and no conflict is possible.

Two parameters are needed when running a jailed process: a root filesystem for the process and an IP address. As root filesystem, a specifically created directory for each jail can be given (a minimum jail environment takes about 125 Mbytes, although several jails can share read-only filesystem to save disk space); the IP address must already exits in the host environment. Several IP addresses can be configured in the same physical interface (as alias) and each one used to jail a process, without having problems sharing the same physical interface (the kernel jail keep things all right).

Jails were successfully used to create BGP emulation environments, with several BGP daemons running in different jails and establishing peering sessions among them using IPv4. However, at the time of writing this document, IPv6 interfaces are not yet supported when jailing a process. The JailNG project [16] has announced it will develop this feature in the future, but no dates are available.

4.3 UML emulation

User Mode Linux (UML [17]) is a virtualization tool that allows to run a completely functional Linux kernel as a process (UML kernel) on the top of the real kernel (the standard kernel that communicates with the hardware), as shown in Figure 4-3. The host machine, where UMLs run, has to run a Linux kernel version equal or newer than 2.4.6. Each UML kernel has its own resources, like memory address space, processes, root filesystem and network devices.

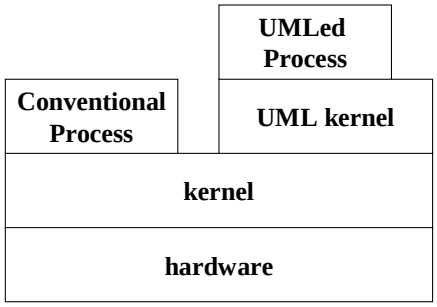

Figure 4-3: UML kernel

UML can be used in the same way FreeBSD jails were used for the development of a BGP emulation system: to run several BGP processes in the same physical box, each one in a separate UML. There is no conflict using network resources for this process, due to the fact that each UML behaves like a complete separated virtual machine, with its own independent network devices.

UML is weightier than FreeBSD jails (resources needed to maintain the entire kernel that supports the BGP process are greater than the ones needed to maintain it in a jail). However, UML emulation is very accurate, so the processes run as they are in a real host. The goal of transparency is fully accomplished.

4.3.1 Building and running UML

The process to build an UML kernel (the kernel UML virtual machines will run) is very similar to the process of building a Linux patched kernel. The steps are: get the sources of the Linux kernel (usually from [18]), patching them with the UML patch (available from [17]), configuring it (using menuconfig or xconfig interfaces), and finally compiling it. Configuring UML is as flexible as configuring a normal Linux kernel.

Note that, in order to get IPv6 networking support under UML virtual machines, this option must be enabled in the UML kernel configuration before compilation. In other words, IPv6 capabilities of the UML kernel capabilities are exactly the same as in any conventional Linux kernel.

The result is an executable file (usually named ‘linux’) that contains the UML kernel image. When this file is executed, it boots the kernel using the root file system specified as parameter (in the same way that a boot loader passes parameters to a kernel at boot time). For example:

```
[bash$] linux ubd0=/uml_fs/root_fs eth0=daemon
```

where ‘linux’ is the name of the UML kernel executable file and ‘udb0=/uml_fs/root_fs eth0=daemon’ are the parameters passed to the UML kernel (the root file system and the network interface the UML will use).

Several parameters can be passed when starting a UML virtual machine. Some of them are the same than normal kernels accept (like ‘root’ parameter, that establishes the root filesystem); others are specific of UML and have to do with the emulation of physical devices inside the virtual machine. The most significant are:

- **eth*n***, enables a virtual Ethernet device inside UML (see section about virtual networking in [17]).

- **ubd*n***, is used to associate a block device with a file in the underlying filesystem.

- **mem**, sets the "physical" memory that kernel allocates for the system. By default, this value is about 30 Mbytes.

As mentioned, in order to boot an UML kernel, a root filesystem is required. Filesystems are stored in files, whose pathname is given in the ubd0 parameter when booting the UML kernel. There are several root filesystems available at [17], based on different Linux distributions (Debian, RedHat, etc); alternatively, they can be build using the UML builder tool.

Usually, root filesystems are very large files (in the order of hundreds of Mbytes). Each UML needs its own separate filesystem (in the same way that jails), although some optimizations can be made to share parts of the filesystem between UMLs (configuration and tuning of the filesystem can be performed before the UML booting process, mounting the UML filesystem in a point of the filesystem of the real host, as it would be a usual block device).

4.3.2 Virtual networking

In order to use UML as the basis of the emulation scenario, network support is needed to allow BGP processes running inside UMLs to establish peer sessions among them. Fortunately, virtual networking techniques are supported by UML in order to establish communications between them and, if needed, with external processes through the networks connected to the physical host.

To allow virtual networking inside a UML, a network interface must be created by giving the eth*n* option (usually eth0) when booting UML kernels. This option creates a virtual Ethernet device inside UML. The particular behavior of the interface and the way it interacts with the host side of the interface is specified by means of a parameter in the command line (see [19] for more information).

UML virtual networking can be configured on several ways. The two most interesting ways for our purposes are the following:

- **Using the physical host as a router.** Physical host environment can act as a router (layer 3 interconnection). Communications between UMLs and between UMLs and external networks is done through the host, as shown in Figure 4-4.

  To connect each UML with the host environment, ***tap*** interfaces are used. *TUN/TAP* or *ethertap* transports can be used for this task. Each UML have to configure the host as default gateway and IP forwarding must be turn on in the host.

  Note that, in this case, there is no direct connection between UMLs: all packets are forwarded through the host. From the point of view of the BGP configuration of the processes running in the UML, multihop option must be configured for these peering sessions.

  Due to the fact that the host environment is the gateway between UMLs and external networks, it can perform additional task, like firewalling or NAT.

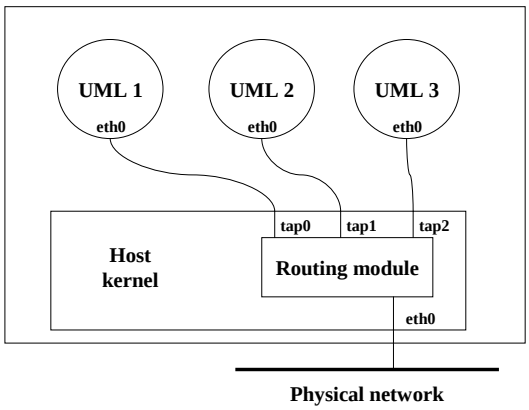

Figure 4-4: Virtual network with host as router

- **Using physical host as a bridge.** Physical host environment can act as a bridge (layer 2 interconnection), as shown in Figure 4-5. UMLs behave as they are directly connected to the external network, so the host does not act as an IP router. Note that this method is more transparent to the BGP process than the former: no need to use multihop configuration in peering sessions. External peers (like Cisco routers) connected to the network can directly peer with BGP process inside UMLs as they would be in real hosts.

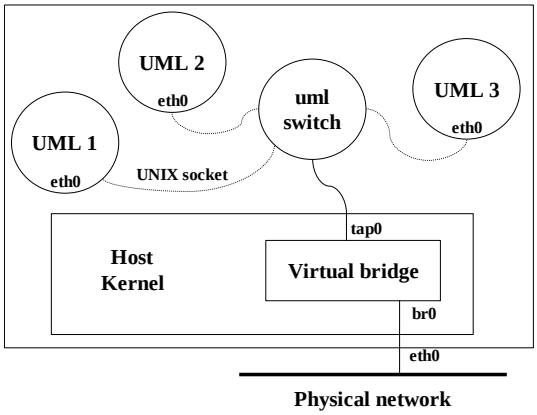

Figure 4-5: Virtual network with host as bridge

In order to configure this virtual networking system, firstly, a two ports virtual bridge must be set in the host (one port is the physical network interface of the host and the other a tap device to connect to the UMLs). Secondly, the *uml_switch* daemon has to be started bound to the tap device. Each UML will connect to the *uml_switch* at boot time (though UNIX sockets) and use it to send and receive packets from/to the bridge.

UML virtual networking support is very powerful and flexible, allowing to configure complex scenarios. For example, we can have several isolated networks inside the same physical box, connected through the host acting as router (as shown in Figure 4-6).

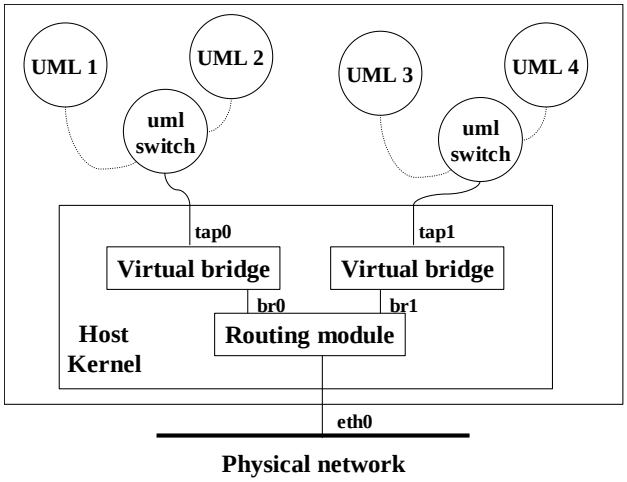

Figure 4-6: Multiple networks in the same physical host

Due to the fact that our focus is on IPv6, in our case eth0 interfaces on the UMLs will only have IPv6 addresses[2]. SSH access will be enabled from the real host to each UML in order to perform management tasks.

4.3.3 Performance

The cost of the flexibility and transparency that UML offers, compared to other mechanisms, comes at the price of a higher resource consumption and performance penalty. To run each bgpd process that emulates a peering partner, a separate UML virtual machine must be started. However, much of the functionality of the entire kernel started for each UML is not used. Other mechanisms, like FreeBSD jails, are lighter in this aspect and do not need as much resources as UML.

The cost of UML based emulation can be analyzed further in terms of four performance overheads: CPU usage, physical memory, virtual networking and storage capability:

- **CPU usage.** The most CPU intensive task is to boot UMLs, especially if several ones are booted simultaneously. However, once the boot process has finished, CPU usage decreases to low percentages (about 10% or less in a Pentium III/900 MHz, even with a significant number of UMLs running –around twenty-).

  Processes created inside a UML kernel are mapped into the actual kernel process space (each process boot by a UML has its own PID on the actual kernel) and the scheduler is efficient enough sleeping processes to not waste CPU cycles. Note that, in our application, which will be the most common situation: a BGP process consumes low CPU resources (mainly, sending a keepalive message each several seconds and low intensive calculations on the route table), being a significant part of the time is slept.

- **Physical memory.** RAM is usually the limiting factor for UML, unless CPU intensive processes are run inside UML (which it is not our case). Each UML kernel has its own memory space (by default about 30 Mbytes), allocated in the virtual memory system of the actual kernel. In order to avoid suffering a significant performance penalty due to the trashing of memory pages in and out swap memory, the total allocated memory for the UMLs plus the memory used by the host should not excess the physical available RAM memory. Anyway, a certain degree of swapping can be tolerated.

[2] For debugging, a IPv4 private address can be established additionally,

- **Virtual networking.** Virtual networking support needs additional resources (routing, virtual bridging or uml_switch daemon) that a normal network does not need. However, the most appreciable impact of virtual networking is a higher than usual latency when establishing connections. However, once established there is no significant overhead. Note that BGP processes establish peer sessions (implemented through a TCP socket) only one time (in spite of the number of keepalive and update messages transmitted), so this is not a great problem.

- **Storage capability.** As each UML needs her own filesystem and, in general, filesystems have a significant size (the master filesystem used in our simulations has a size of about 630 Mbytes), disk space consumed could be high if the number of UML started is important. However, as the use of UML we do is very specific, a lot of files are not used and they could be removed for the master filesystem.

The conclusion of this quick performance analysis is that the penalty of using UML can be easily overcome increasing the hardware resources in the emulation machine. Fortunately, two relatively cheap resources, RAM memory and hard disk storage capacity, are the most important ones. That means that even with standard PCs, complex scenarios can be emulated. For example, to implement the scenarios described in the next section, a PC Pentium III 900 MHz with 512 Mbytes of RAM memory was used, being able to run a maximum of thirty simultaneous UMLs (about 18.5 Gbytes to storage filesystem) with a good global performance result.

4.4 Test scenarios using UML

Among the alternatives evaluated (several BGP process in the same box, FreeBSD jails and UML), UML option was selected as it was the only one fulfilling all the requirements. UML based solution offers a high degree of transparency and a good flexibility for the definition of different scenarios. If complemented with the creation of a set of scripts (of which a basic version is already available) to automate the management of the UMLs and simulation processes, it leads to a very powerful emulation environment.

Using this powerful tool, several BGP interesting scenarios have been successfully tested. We present in the following sections the most interesting ones: a basic route server based IX scenario, and an extended one including route injection using bgpsim.

As mentioned before, UML virtual networking is so flexible that can deal with much more complex scenarios. For example, an interesting scenario that could be emulated in the future could be the complete Euro6IX network, including all its IXs and the relation with other IPv6 networks.

4.4.1 Simulation scripts

A set of *Bash* scripts have been developed to automate and ease the management tasks associated with the UML based emulation environment. All this scripts run from the host. In order to communicate with UMLs from the host, Secure Shell (SSH) or mconsole (a UML tool that allows direct communication with UML kernel through a UNIX socket) are used.

Basically, the scripts developed allow to:

- Configure the virtual network bridge.
- Create a UML filesystem from a master filesystem.
- Boot a UML from a particular filesystem.

- Check UMLs with a keepalive ping.
- Halt UMLs (all of them or a particular one).
- Start a BGP emulation process inside a particular UML using a configuration file (filename is given as parameter of the script).
- Stop BGP simulation process on a particular UML.

These elemental scripts are modular and flexible, so more complex scripts implementing a whole emulation experiment (as the ones described below) can be built using them as basic building blocks.

4.4.2 Basic route server based IX scenario

In this scenario (shown in Figure 4-7), twenty one UMLs are started running each one a Zebra bgpd process. One of them acts as a route server and the other twenty as BGP clients peering with the route server. Each client simulates the border router of a different AS peering on the IX. The IX itself is considered another AS (it has its own AS number). As usual, external BGP (eBGP) is used to establish peering session between route server and clients.

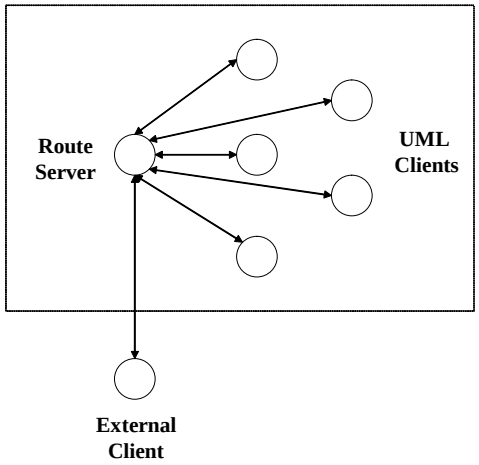

Figure 4-7: Route server peering scenario

Each client announces a set of prefixes (different tests have been done with different numbers of prefixes, from 1 to 50 routes) to the route server and receives all the prefixes announced by other clients. Route server acts as a route distribution point, being configured in transparent mode (that is, its AS number is not incorporated to the routes).

The scenario is emulated using UML host bridged virtual networking. In this way, external clients connected to the physical network of the host (such Cisco routers) can be configured to peer with the route server transparently.

4.4.3 Advanced route server based IX scenario

This emulation scenario (shown in Figure 4-8) is just a more complex version of the previous one, including route injections using bgpsim tool. Each AS is emulated using two UMLs: one to run the bgpd process and the other to run a copy of bgpsim tool (both are connected using iBGP). In total, twenty one UMLs are run in this scenario (emulating one route server and ten ASs).

With more detail, the role of the two UMLs that emulate each AS is the following:

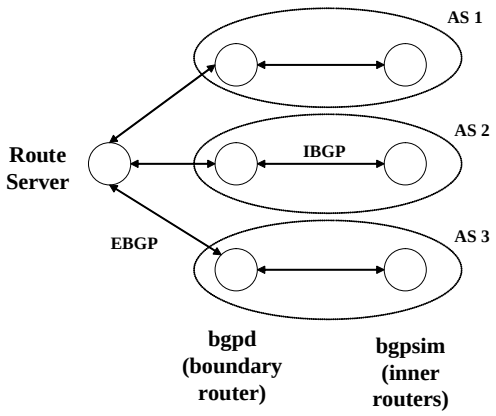

Figure 4-8: AS interconnection scenario

- UML running MRTd bgpsim. This process emulates all inner routers of the AS. Due to the simulation features of bgpsim, several set of prefixes can be fine defined, each one with its own attributes and stability conditions (for example, a set of routes that flaps up and down each hour). This process established an iBGP peering session with the Zebra bgpd.

- UML running Zebra bgpd. This process simulates an AS external router, that propagates prefixes (learned from the iBGP peer) out to the route server and propagates to the inner routers of the AS routes learned from the route server. Additionally to the iBGP established with the MRTd bgpsim process, it maintains an eBGP peering session with the route server.

Filtering capabilities have also been tested in this scenario. For example, the Zebra bgpd process of an AS was configured to filter route prefixes coming from other particular AS, so prefixes with that AS number in the as-path attribute were not propagated. An interesting possibility, that will be investigated in the future, consists in centralizing filtering tasks is the route server, defining what routers should not propagate following general filter policies (common to all participants in the IX) or particular ones for each AS.

5. ROUTING POLICIES DATABASES TEST

Apart from the route server entity, the full implementation of a route server based IX depends on the existence of a routing policies database. Databases used for that purpose should fulfill the following basic requirements:

- To store all routing information about an AS, including its route prefixes, its peering relations and the route filters applied. This information should be expressed in RPSL objects.

- Mechanisms to query and update the objects stored in the database. In particular, tools as IRRTools should be able to query the database to build automatic configurations for route server and clients.

- IPv6 capable. Unfortunately, as mentioned before, although RPSL for IPv4 is widely used, there is still a lack of standardization around IPv6 extensions to RPSL (named RPSLng).

As a direct consequence of the last point mentioned, no IPv6 routing policies database implementations exist nowadays. However, in order to gain experience in the configuration and management of this databases, we decided to investigate which IPv4 policy routing implementations were available and make some basic tests with them.

After a brief investigation, we selected two open source databases currently available for evaluation: IRRd and RIPE Database. Tests consisting on installing the databases and checking their basic functionality (creating a database, querying and updating, etc) have been done using both implementations. More detailed tests will be performed when RPSLng support is available on any of them.

5.1 IRRd

Tested version: 2.1.4 (released 6/02/2002; recent versions are available).

IRRd [20] is a light database implemented by Merit to run its RADB routing registry. It includes two main functionalities:

- **Stand alone routing registry,** to store local routing information (for example, to maintain routing information within a particular organization).

- **Mirror,** to mirror the global Internet Routing register (maintained by RIPE, ARIN and other Regional Internet Registries).

Once installed, IRRd runs as background process. In starts from the shell command line (or from the usual Unix boot scripts in /etc/rc.d) and can be stopped with a kill signal. Configuration is done through a configuration file or through a UII (User Interactive Interface) accessed by telnet.

IRRd allows defining several databases. Each one is stored in plain-text files (usually in a pathname inside /var directory). The syntax used in these files is RPSL. The behavior of each database can be configured. For example, updating of database can be allowed or disallowed and access lists (for queering, updating or both) can be defined.

IRRd can perform two operations over the stored RSPL objects: queries and updates. Deletion is considered as a particular update case.

- **Queries.** IRRd listens on whois port (usually port 43) and answers queries to the database from any other host in the Internet. Of course, that host has to be allowed in the access list configuration. Multiple command mode (response several queries using the same TCP connection) is supported.

- **Email updating.** IRRd can process email updates. In order to do that, we need to configure the mailer process (usually sendmail) to deliver updates (usually emailed to 'auto-dbm@irrdhost') to irr_rpsl_submit, and external plug-in of the IRRd database that handless the update (parsing the email and checking syntax).

- **Direct updates.** TCP connection to port 8888 (by default) can be used to update the database. In this case, inetd must be configured to run irr_rpsl_submit whenever a connection attempt is done to port 8888.

- **Mirror updates.** If the database is configured as a mirror of another database, updates are performed automatically.

IRRd has been tested in FreeBSD 4.5, although it works also over Linux and Solaris.

5.2 RIPE Database

Tested version: 3.1.1 (releases 8/10/2002).

RIPE Database [21] is the database server used by RIPE (the European Regional Internet Registry). It relies on a MySQL database running on the same host to store the RPSL objects, as shown in Figure 5-1.

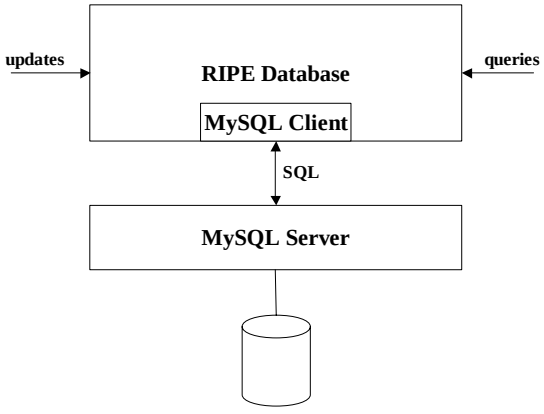

Figure 5-1: RPSL Database architecture

Although querying the MySQL database directly is possible in theory, we lose the syntactic and semantic RPSL checking that RIPE Database performs. Additionally, the querying and updating RPSL database through whois and email is more simple that using the SQL language to do the same task.

RIPE Database allows the usual operations: querying and updating (by email or direct connection, thanks to plug-ins). It has been tested in GNU/Linux 2.4, although it works also over FreeBSD and Solaris.

5.3 IPv6 support

At present, IPv6 support on both databases is very limited:

- IRRd. Current version does not have IPv6 support. Support for the 'ipv6-site' and 'inet6num' objects used inside 6BONE, as well as support for RPSLng draft standards, has been announced for future versions [22].

- RIPE Database. The 'inet6num' and 'ipv6-site' objects are supported. RPSLng is not supported (no announcement or date available).

5.4 RPSL related tools

One of the main goals of the routing policy databases is to use the routing information stored in them to perform different analysis on this data (automated router configuration, routing policies analysis, and maintenance). To do so, a suitable toolbox has to be used, like the IRRToolSet, which is the main suite of policy analysis tools to operate with routing policies in RPSL used nowadays.

A brief description of the main tools included in IRRToolSet follows:

- **RtConfig**, that analyzes the routing policies registered in the Internet Routing Registry (IRR) and produces router configuration files,

- **Prtraceroute**, which compares the result of a traceroute with policies registered in the routing policy database

- **Prpath,** which enumerates a list of paths between an AS and a specified destination.

IRRToolSet does not support IPv6 yet, as the routing policy databases it relies on do not support IPv6.

6. SUMMARY AND CONCLUSIONS

This document has presented the activities carried out in the context of A4.2 around the Route Server functionality present on some IXs. These activities have been coordinated and complement other activities in progress inside WP2 around the same subject. Their final goal is the design, installation and experimentation of route server based IXs inside Euro6IX project.

As it has been shown, route server based IX architectures play an important role to improve the management and scalability of IXs. Its use is limited nowadays: they are mainly used on small and medium size IXs, and typically only include the basic functionality of route redistribution between peering partners. However, we believe that several routing functions already implemented in the border routers peering on an IX could be moved to the route server, and be implemented there in a more efficient way, securing and simplifying AS border router management.

The main activities during this year have focused on the experimentation with available implementations of the basic route server components (route server, databases, tools, etc), as well as on the creation of an IX emulation environment. Tests have mainly focused on Zebra's BGP implementation, as it has demonstrated to be the most stable and up-to-date among the implementations evaluated.

The IX emulation environment developed has shown to be very useful in order to emulate BGP complex scenarios without incurring in the management complexity and the cost of maintaining testing environments made of a high number of machines. This tool will greatly simplify the process of testing new functionalities or scenarios in the future, not necessarily tied to route server functionality, as the tool is flexible enough to emulate other general networking scenarios.

7. FUTURE WORK

Future work around the activities presented in this document can be proposed in the following areas (not necessarily to be carried out inside A4.2):

- **Experimentation with RPSLng**

  Study and experimentation with the RPSL IPv6 extensions being proposed in RPSLng drafts should be initiated as soon as possible, probably in the context of WP2, in order to gain experience in its use and contribute to its standardization process and implementation. A good exercise could consist in the description of Euro6IX routing policies using RPSLng, in order to provide feedback about problems detected and missing functionality. This task should be done in cooperation with 6NET project, where this exercise has been already proposed [2], RIPE Routing Working Group and other active RPSL initiatives.

- **Study about Route Server functions**

  Further investigations are needed about what functions can be incorporated to the route server of an IX, with the goal of simplifying router management and configuration tasks inside the IX. As the route server is a central point and all the routing information flows through it, it seems very sensible to incorporate to it several functions that nowadays are being done in the ISP routers or simply not done. For example:

  - General route filters to discard routes to bogus networks (addresses not assigned yet, private addresses, etc) or ASs (private or non existent AS numbers). In this case, instead of configuring the same filters on every ISP border router, the route server could do it for all of them. As a consequence, only the IX management team should be responsible of maintaining the route filters up-to-date, changing them as new addresses or AS numbers are assigned by RIRs.

  - Checking that networks announced are originated in the proper ASs. That is, route server could check, by querying the database, if the networks announced by every AS originate in the proper AS, in order to avoid the typical situation where an AS announces networks belonging to other AS (due to configuration errors or even malicious behavior).

  - Limit the minimum length of the prefixes interchanged. The route server could be used to force the IX routing policy of discarding prefixes longer than an agreed minimum length.

  - Monitoring facilities and statistics (as it is already the case nowadays).

- **Definition of Testing Scenarios**

  Although complex scenarios have already been tested using the emulation environment, they have been used basically as a proof-of-the-concept tests. Additional work has to be invested in defining the precise scenarios that will allow a deeper study of route server based IXs architectures and implementations.

As a minimum, a complete scenario including routing policy databases and associated tools should be defined (and tested as soon as all pieces are available). This scenario can be used as a testbed for the new functionalities proposed in the study described above.

Besides, some scenarios to measure the performance, reliability and scalability of route server implementations could be defined. For example, in order to get real performance figures, a route server could be installed in an isolated machine, and one or more emulated environments with tenths of BGP peers could be used to stress the route server implementation injection high number of prefixes and some route instabilities. Also, redundant route server configurations could be tested, in order to test the reliability in case of failures.

Finally, we think that the emulation environment could be used also as a valuable tool for other subactivities. For example, the tool could play an important role in defining and testing multihoming solutions in the context of A4.1 subactivities.

- **Improvement of the IX Emulation Environment**

  Now we have a basic IX emulation environment made of a basic set of scripts that could be considered as a proof-of-concept implementation. Work will be invested in the future in order to improve the flexibility and easy the use of the environment, as well as to:

  - Include monitoring tools in the emulated environment, for example, AS-Path or similar tools.
  - Have a deeper evaluation of the scalability of the emulation environment, testing scenarios with higher numbers of emulated routers and bigger routing tables.
  - Design a simple language for defining scenarios topology, together with the associated tools to translate tests definitions into the scripts that emulate them.

- **Routing Policy Databases**

  Availability of RPSLng compliant policy routing databases is a key point for the activities described here. A deeper study should be done to know when they will be available. Adaptation of one of the existent implementations could even be proposed as a new activity inside A4.2 if none is available during the project timeframe.

- **Security in IXs**

  Due to its importance for the correct operation of IP networks, a significant effort is being invested at present in securing routing protocols. For this purpose, a new IETF working group -Routing Protocol Security Requirements (rpsec) [23]- has been recently created to study the general requirements on this area, and several proposals are being circulated to increase the security of specific routing protocols.

  In particular, there are proposals to improve the security of BGP protocols: Secure BGP [24] and soBGP [25]. It would be interesting to analyze how these proposals match the route server based IX architecture and test them over the emulated scenarios, in case implementations are available.

8. REFERENCES

[1] Specification of the Internal Network Architecture of each IX point. Deliverable 2.1, Euro6IX project. July 2002.

[2] Contribute and report on discussion on IPv6 support in RIPE database. Deliverable 3.3.1, 6NET project. October 2002.

[3] Router Arbiter Project. See: http://www.isi.edu/ra/

[4] Route Server Technical Overview. See: http://www.rsng.net/overview.html

[5] Packet Clearing House. See: http://www.pch.net/technology/

[6] Routing Policy Specification Language (RPSL). C. Alaettinoglu et al. RFC 2622. June 1999.

[7] Representation of IP Routing Policies in a Routing Registry. T. Bates et al. RFC 1786. March 1995.

[8] 6BONE Registry Database. See: http://www.6bone.net/RIPE-registry.html

[9] RPSLng Draft. See: http://www.ripe.net/ripe/mail-archives/rpslng/2002/msg00090.html

[10] Internet Routing Registry. See: http://www.irr.net

[11] Internet Routing Registry Toolset Project. See: http://www.ripe.net/ripencc/pub-services/db/irrtoolset/index.html

[12] MRTd. http://www.merit.edu/mrt

[13] Zebra. http://www.zebra.org

[14] Cisco IOS. http://www.cisco.com

[15] Using Jails in FreeBSD for Fun and Profit. Paco Hope. Login: the magazine of Usenix & Sage. June 2002, volume 27, number 3. See: http://www.usenix.org/publications/login/2002-06/pdfs/hope.pdf

[16] JailNG. http://www.watson.org/~robert/freebsd/jailng/

[17] User Mode Linux Kernel Home Page. See: http://user-mode-linux.sourceforge.net

[18] Kernel.org. The Linux Kernel Archives. See: http://kernel.org

[19] UML Virtual Networking. See: http://user-mode-linux.sourceforge.net/networking.html

[20] Internet Routing Registry Daemon (IRRd). See: http://www.irrd.net

[21] RIPE Database Reference Manual RIPE Database User Manual.

[22] IRRd mailing list. http://www.merit.edu/mail.archives/irrd-discuss/msg00325.html

[23] Routing Protocol Security Requirements (rpsec) IETF Working Group. See: http://www.ietf.org/html.charters/rpsec-charter.html

[24] Secure BGP Project (S-BGP). See: http://www.net-tech.bbn.com/sbgp/sbgp-index.html

[25] Secure Origin BGP (soBGP). James Ng. Internet Draft. See: http://www.ietf.org/internet-drafts/draft-ng-sobgp-bgp-extensions-00.txt