

# Virtual Laboratory

January 19, 2011

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>VMware as bridge to virtualization</b>	<b>5</b>
2.1	Introduction to virtualization . . . . .	5
2.1.1	Benefits of virtualization . . . . .	6
2.1.2	Virtualization programs . . . . .	7
2.2	Definition of VMware Server . . . . .	7
2.2.1	Introduction . . . . .	7
2.2.2	How VMware Server works . . . . .	8
2.2.3	VMware Server Virtual Switch . . . . .	8
2.2.3.1	VMware Network element . . . . .	8
2.2.3.2	The Virtual Switch . . . . .	10
<b>3</b>	<b>Quagga: Routing Software Suite</b>	<b>12</b>
3.1	Introduction . . . . .	12
3.2	System Architecture . . . . .	13
3.3	Supported Platform . . . . .	14
3.4	Installation . . . . .	15
3.4.1	Installation by source codes . . . . .	15
3.4.1.1	The Configure script and its options . . . . .	15
3.4.1.2	Build the Software . . . . .	17
3.4.1.3	Install the Software . . . . .	17
3.4.1.4	Checking Quagga files after installation . . . . .	17
3.4.2	Installing by terminal . . . . .	18
3.5	Using Quagga . . . . .	18
3.5.1	Quagga daemons . . . . .	18
3.5.2	Configuration files . . . . .	19
3.5.3	VTYSH . . . . .	20
3.5.4	Routers functionality comparison . . . . .	22
3.5.5	Commands . . . . .	23
3.5.5.1	Basic Commands . . . . .	23
3.5.5.2	Terminal Mode Commands . . . . .	25
3.5.6	Zebra . . . . .	25
3.5.6.1	Virtual Terminal Interfaces . . . . .	25
3.5.6.2	Interface Command . . . . .	26

3.5.7	RIP . . . . .	28
3.5.7.1	Virtual Terminal Interfaces . . . . .	29
3.5.7.2	RIP configuration . . . . .	29
3.5.7.3	RIP Version Control . . . . .	30
3.5.7.4	Access list RIP Routes . . . . .	31
3.5.7.5	RIP Authentication . . . . .	32
3.5.7.6	RIP Timers . . . . .	32
3.5.7.7	Show RIP Information . . . . .	33
3.5.8	OSPFv2 . . . . .	33
3.5.8.1	Virtual Terminal Interfaces . . . . .	33
3.5.8.2	OSPFv2 Configuration . . . . .	34
3.5.8.3	OSPF router . . . . .	34
3.5.8.4	OSPF area . . . . .	37
3.5.8.5	OSPF interface . . . . .	40
3.5.8.6	Showing OSPF information . . . . .	42
3.5.9	BGP . . . . .	42
3.5.9.1	Virtual Terminal Interfaces . . . . .	42
3.5.9.2	Configuring BGP . . . . .	43
3.5.9.3	BGP router . . . . .	43
3.5.9.4	BGP network . . . . .	44
<b>4</b>	<b>Generating traffic and monitoring</b>	<b>45</b>
4.1	Generating traffic . . . . .	45
4.1.1	Introduction to Generating Traffic . . . . .	45
4.1.2	Iperf . . . . .	45
4.1.2.1	What Iperf is . . . . .	45
4.2	Monitoring Traffic . . . . .	46
4.2.1	Introduction to Monitoring Traffic . . . . .	46
4.2.2	Wireshark . . . . .	47
4.2.2.1	What Wireshark is . . . . .	47
4.3	SNMP: Simple Network Management Protocol . . . . .	47
4.3.1	Introduction to SNMP . . . . .	47
4.3.2	SNMP and Quagga . . . . .	49
<b>5</b>	<b>Configuring a virtual laboratory (Testing)</b>	<b>50</b>
5.1	Case Study 1: Lan with static routes . . . . .	50
5.1.1	Scenario . . . . .	50
5.1.2	Configuration . . . . .	51
5.1.2.1	How to Create VMware Virtual Switch . . . . .	51
5.1.2.2	How to configure Quagga . . . . .	52
5.1.3	Testing . . . . .	55
5.1.3.1	Ping . . . . .	55
5.2	Case Study 2: Lan with RIP dynamic routing protocol . . . . .	57
5.2.1	Scenario . . . . .	57
5.2.2	Configuration . . . . .	58
5.2.2.1	How to Create VMware Virtual Switch . . . . .	58

5.2.2.2	How to configure Quagga . . . . .	58
5.2.3	Testing LAN . . . . .	64
5.2.3.1	Ping . . . . .	64
5.2.3.2	Wireshark . . . . .	66
5.2.3.3	Iperf . . . . .	67
5.3	Case study: Lan with OSPF . . . . .	67
5.3.1	Scenario . . . . .	68
5.3.2	Configuration . . . . .	68
5.3.2.1	How to Create VMware Virtual Switch . . . . .	69
5.3.2.2	How to configure Quagga . . . . .	69
5.3.3	Testing . . . . .	73
5.3.3.1	Ping . . . . .	73
5.3.3.2	Wireshark . . . . .	75
5.3.3.3	Iperf . . . . .	77
5.3.3.4	SNMP: SMUX configuration . . . . .	77
5.4	Case study 4: Advanced Lan with OSPF . . . . .	78
5.4.1	Scenario . . . . .	79
5.4.2	Configuration . . . . .	80
5.4.2.1	How to Create VMware Virtual Switch . . . . .	80
5.4.2.2	How to configure Quagga . . . . .	80
5.4.2.3	Configuring Cisco routers . . . . .	82
5.4.3	Testing . . . . .	84
<b>6</b>	<b>Conclusion</b> . . . . .	<b>89</b>
6.1	Virtual Network Laboratory Environment as Learning Enviroment . . . . .	89
6.2	Virtual Network Laboratory Environment as Working Enviroment . . . . .	91
<b>7</b>	<b>Future Implementation</b> . . . . .	<b>93</b>

# Chapter 1

## Introduction

The purpose of these thesis is to built a virtual networking laboratory environment for University of Evtek in Finland.

The use of it is to offer additional learning environment for studying various topics on computers network. It is not going to substitute a real laboratory, but actually it is going to use for amending.

We can catch the meaning of this thesis analysing directly from this words: *Virtual Network Laboratory Environment*.

*Environment* is a set of component, hardware, software that are connected in the same system. *Network Laboratory* is a place where it is possible to manage and configure network device like router, switch, hubs and creating either a Local Area Network (LAN) or Wide Area Network (WAN). *Virtual* means one or more software that execute programs like a real machine.

Combine these knowledge together allows to built a system in which it possible to create, modify, remove, a LAN, Subnet, o configure routers, switches, executing software like a real laboratory.

This document is going to show all the steps necessary to built this virtual laboratory.

As first step it is about to explain in deep all the software component. It starts introducing the VMware in Chapter 2, why it is necessary and its rules in the system. Here it will be introduce the concept of Layer 2 switching with its concerning application like Virtual VMware switches.

In the 3 Chapter it is going to describe Quagga. It is the main software of all the virtual lab. It provides implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms. In that chapter is about to describe how to install and what zebra is, the routing protocols running in it, and it is going to give different example.

In the Chapter 4 it will be described the use of packet generator using to test with real traffic the created network. As probably every network administrator knows there is no working network without monitoring. For that reason it is very useful to have a software for network monitoring to provide SNMP (Simple Network Managing Protocol). It will discuss as well in this chapter.

In the Chapter 5 it will be showed how to create a Virtual Network Laboratory Environment. There will be studied four case study of different level of difficult to test them. This chapter is important because it will demonstrate if it possible to reach the purpose of this thesis, how and the results obtained.

This thesis is going to conclude with future implementation and a take look closer to virtual lab as learning enviroment and virtual lab as working enviroment.

## Chapter 2

# VMware as bridge to virtualization

### 2.1 Introduction to virtualization

The term of virtualization is often used in this thesis. There is different definition about it, in this session it has choosen ones that it could be the more appropriate to indicate the role of virtualization in this contest.

It can be defined as a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, and many others.

Virtualization allows to run multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that one physical computer across multiple environments. Different virtual machines can run different operating systems and multiple applications on the same physical computer.

Virtualization is a combination of software and hardware engineering that creates Virtual Machines (VMs), an abstraction of the computer hardware that allows a single machine to act as if it where many machines.

A virtual machine can be defined as software container that can run its own operating systems and applications as if it were a physical computer. A virtual machine behaves exactly like a physical computer and contains it own virtual CPU, RAM hard disk and network interface card (NIC).

A virtual machine is exactly the tool needed to implement the virtual lab implemented in this project.

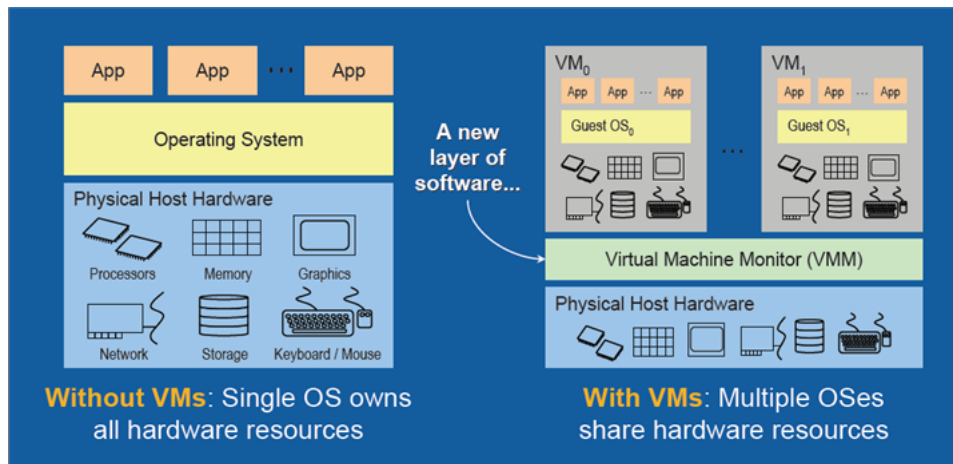


Figure 2.1: Non Virtual Machine and VM Configurations

As it has showed in the previous picture, a machine without virtual machine has its Operating System that owns all hardware resources.

Instead, a machine with a virtual machine installed on it can have multiple operating systems, each running its own virtual machine, share hardware resources. Virtualization enables multiple operating systems to run on the same physical platform.

A machine with a virtual machine has a new layer of software called Virtual Machine Monitor (VMM).

The VMM is the control system at the core of virtualization. It acts as the control and translation system between the VMs and the hardware.

The follows chapter it will explain the benefits of virtualization, which virtualization programs it has used and why.

### 2.1.1 Benefits of virtualization

Today's computer hardware was designed to run a single operating system and a single application. Virtualization allows to run multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that one physical computer across multiple environments. Therefore, different virtual machines can run different operating systems and multiple applications on the same physical computer.

These is a list of main advantages of using virtualization[1]:

- **Testing and development:** use of a VM enables rapid deployment by isolating the application in a known and controlled environment. Unknown factors such as mixed libraries caused by numerous installs can be eliminated. Severe crashes that required hours of re-installation now take moments by simply copying a virtual image.
- **Compatibility:** just like a physical computer, a virtual machine hosts its own guest operating system and applications, and has all the components found in a physical computer (motherboard, VGA card, network card controller, etc). As a result, virtual machines are completely compatible with all standard x86 operating systems, applications and device drivers, so you

can use a virtual machine to run all the same software that you would run on a physical x86 computer.

- **Isolation:** while virtual machines can share the physical resources of a single computer, they remain completely isolated from each other as if they were separate physical machines. If, for example, there are four virtual machines on a single physical server and one of the virtual machines crashes, the other three virtual machines remain available. Isolation is an important reason why the availability and security of applications running in a virtual environment is far superior to applications running in a traditional, non-virtualized system.
- **Encapsulation:** a virtual machine is essentially a software container that bundles or “encapsulates” a complete set of virtual hardware resources, as well as an operating system and all its applications, inside a software package. Encapsulation makes virtual machines incredibly portable and easy to manage. For example, you can move and copy a virtual machine from one location to another just like any other software file, or save a virtual machine on any standard data storage medium, from a pocket-sized USB flash memory card to an enterprise storage area networks (SANs).
- **Hardware Independence:** virtual machines are completely independent from their underlying physical hardware. For example, you can configure a virtual machine with virtual components (e.g., CPU, network card, SCSI controller) that are completely different from the physical components that are present on the underlying hardware. Virtual machines on the same physical server can even run different kinds of operating systems (Windows, Linux, etc).

### 2.1.2 Virtualization programs

Virtualization programs are software that can enable to create and run Virtual Machines. Among different products it has chosen VMware server 2.0.2.

The reason of this choice are that a VMware server offers a list of benefits that match with the needed of this project. First of all VMware Server provides multiple ways to configure a virtual machine for virtual networking<sup>1</sup>. Another reason is that it offers a Virtual Network Switch, and it works like a physical switch. So it offers an additional component for building a Virtual Laboratory.

Therefore VMware is a free use software and it is possible to download and install after registered in VMware site.

## 2.2 Definition of VMware Server

### 2.2.1 Introduction

VMware Server is a hosted virtualization platform that installs an application on a host and it allows to partition a physical host into multiple virtual machines.

A virtual machine is a isolated software container that can run its own operating systems and applications as if it were a physical computer. A virtual machine behaves exactly like a physical computer and contains its own virtual CPU, RAM hard disk and network interface card (NIC).

---

<sup>1</sup>VMware Server functions will be described deeply in the chapter 2.2



## 2.2.2 How VMware Server works

VMware Server installs and runs as an application on top of a host Windows or Linux operating system. A thin virtualization layer partitions the physical server so that multiple virtual machines can be run simultaneously on a single server. Computing resources of the physical server are treated as a uniform pool of resources that can be allocated to virtual machines in a controlled manner. VMware Server isolates each virtual machine from its host and other virtual machines, leaving it unaffected if another virtual machine crashes. Data does not leak across virtual machines and applications can only communicate over configured network connections. VMware Server encapsulates a virtual machine environment as a set of files, which are easy to back-up, move and copy.

To perform host and virtual machine configuration for VMware Server is used VMware Infrastructure Web Access (VI Web Access) 2.0[2]. It is a intuitive web-based interface that provides a simple and flexible tool for virtual machine management.

With a Web Access is possible to:

- Create, configure, and delete virtual machines.
- Add and remove virtual machines from the inventory.
- Perform power operations (start, stop, reset, suspend, and resume) on virtual machines.
- Monitor the operation of virtual machines.
- Generate a Web shortcut to customize the VI Web Access user interface for users, with the option to limit their view to the console or a single virtual machine.
- Generate a VMware Remote Console desktop shortcut that allows virtual machine users to interact directly with the guest operating system outside of a Web browser.

Therefore, VMware supports different Operating Systems like Windows (also the new Windows Server 2008), Linux ( Red Hat, Ubuntu,Cent OS...).

## 2.2.3 VMware Server Virtual Switch

VMware Server has been chosen in this project also because it offers Virtual Switch and a set of features to manage it.

First of all it will explain the component and the way to manage network connection for Virtual Machines.

### 2.2.3.1 VMware Network element

VMware Server provides[?] multiple ways you can configure a virtual machine for virtual networking:

- Bridged networking.
- Network address translation (NAT).
- Host-only networking.

On a Windows host, the software needed for bridged, NAT, and host-only networking configurations is installed when you install VMware Server. The New Virtual Machine wizard connects the virtual machine to the virtual network selected.

Now it will discuss the meaning and how the virtual networking components work.

**Bridged Networking** It configures a virtual machine as a unique identity on the network, separate from and unrelated to its host. Other computers on the network can communicate directly with the virtual machine. The virtual network adapter in the virtual machine connects to the physical network adapter in your host computer, allowing it to connect to the LAN used by the host computer.

Using bridged networking, a virtual machine must have its own identity on the network. For example, on a TCP/IP network, the virtual machine needs its own IP address.

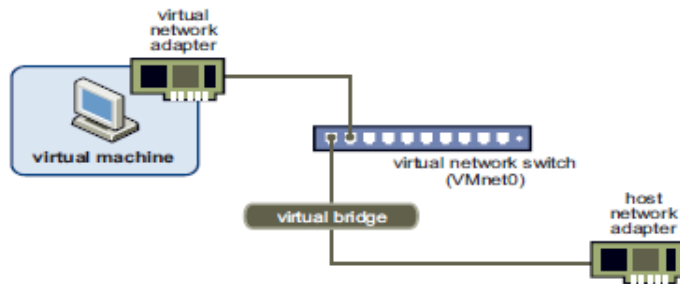


Figure 2.2: Bridged Networking Setup

**Network Address Translation (NAT)** NAT gives a virtual machine access to network resources by using the host computer's IP address. NAT is the easiest way to give a virtual machine access to the Internet or other TCP/IP network. NAT uses the host computer's dial-up networking or broadband connection. Therefore, a virtual machine shares the IP and MAC addresses of the host.

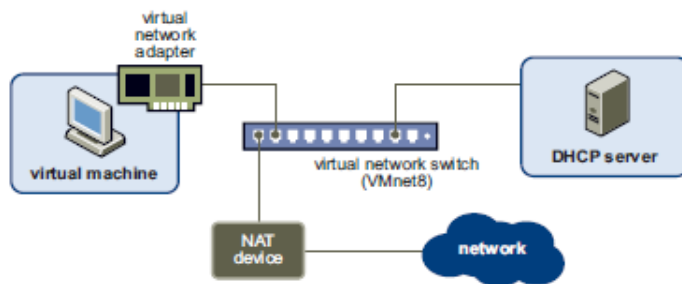


Figure 2.3: Network Address Translation Setup

Using a virtual machine does not have its own IP address on the external network. Instead, a separate private network is set up on the host computer. A virtual machine gets an address on that network from the VMware internal DHCP server. The VMware NAT device passes network data between one or more virtual machines and the external network, using a host network adapter that is visible to the host operating system. It identifies incoming data packets intended for each virtual machine and sends them to the correct destination.

**Host-Only Networking** It allows to configure a virtual machine to allow network access only to a private network on the host. Host-only networking creates a network that is completely contained within the host computer. With host-only networking, the virtual machine can communicate only with the host and other virtual machines in the host-only network. This can be useful if it is needed to set up an isolated virtual network.

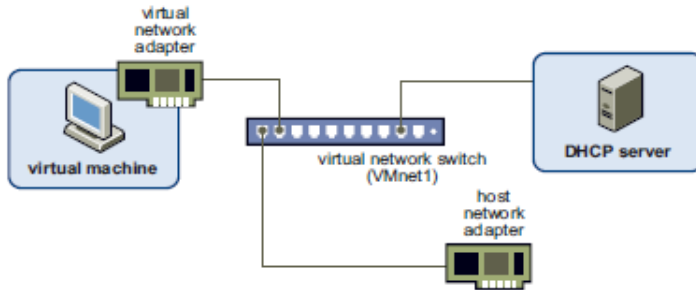


Figure 2.4: Host-Only Networking Setup

### 2.2.3.2 The Virtual Switch

In a physical network configuration, a switch is typically a hardware device which is used to provide a central point of network connectivity for network clients. The devices on a physical network are connected to a switch using twisted pair cabling. When a client sends a network packet to another device on the network, the switch checks the packet for the MAC address of the destination device and forwards the packet to the port to which that device is connected.

VMware Server includes a virtual network switch which, although entirely software based, performs the same task as physical switch, in that it allows virtual devices, such as virtual machines and other virtual network components, to be connected together to form a virtual network.

The virtual switch works like a physical switch, but it is used by virtual machines. Like a physical switch, a virtual switch allows to connect other networking components together. Virtual switches are created as needed by VMware Server, up to a total of 10 virtual switches on Windows and 255 on Linux. Virtual switches can be used in bridged, host-only, and NAT network configurations. It has possible to connect one or more virtual machines to a switch.

For that reasons, in this project, virtual switch are used to create switch-to-router connections, switch-to-switch connections and switch-to-host connections.

A few networks have default names and switches associated with them:

- The Bridged network uses VMnet0.
- The HostOnly network uses VMnet1.
- The NAT network uses VMnet8.

Additional virtual switches may be added, up to the allowed host platform maximum, using the names VMnet2, VMnet3, VMnet4, and so on to custom for the own needs.

A list of currently configured networks, and the switches to which they are assigned may viewed via the VI Web Access management interface to VMware Server 2.0 by selecting the host system

from the Inventory panel and locating the Networks panel in the Summary workspace. The following figure illustrates the Networks panel with five virtual switches configured.


Networks 		
Name ▲	VMnet	Type
Bridged	vmnet0	bridged
HostOnly	vmnet1	hostonly
NAT	vmnet8	nat
VMnet2	vmnet2	hostonly
VMnet3	vmnet3	hostonly

Figure 2.5: Virtual Switch

In the above example, there are five networks named Bridged, HostOnly and NAT. These are assigned to virtual switched vmnet0, vmnet1, vmnet8, vmnet2 and vmnet3 respectively.

The first three virtual switches are created by default by VMware Server, the others are created by administrator.

It has to remind that a virtual switch allows connection between one or more virtual machines. But virtual switch doesn't manage and route any traffic. For that reasons Quagga is a added value.

## Chapter 3

# Quagga: Routing Software Suite

### 3.1 Introduction

Quagga is actually the most important part of the project. It plays a central role : it is the core of the project. In fact all the functionality of this project works thanks to Quagga.

The concept of virtualization in this project is just beyond applications and operating systems, it will cover a new notion: *network virtualization*. And the tool to realize it is Quagga.

The concept of network virtualization can be defined as the process of combining hardware network resources and software network resources into a single environment. In a *real network environment* hardware network are routers, switches and workstations with their NIC, while software network are the OS of the hardware network. In a *virtual network environment* hardware network are workstations with their NIC, while software network is a container of software that can act and perform routing functionality.

Nowadays, in a real network environment TCP/IP networks are covering all of the world. The Internet has been deployed in many countries, companies, and to the home. The packets will pass many routers which have TCP/IP routing functionality.

In a virtual environment, Quagga installed acts as a dedicated router. With Quagga, the machine exchanges routing information with real routers (e.g. Cisco...) using routing protocols.

Quagga is a *routing software package* that provides a suite of TCP/IP based routing protocols. Precisely it provides implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms, particularly FreeBSD, Linux, Solaris and NetBSD[4]. Quagga is a fork of GNU Zebra which was developed by Kunihiro Ishiguro.

It also supports special BGP Route Reflector and Route Server behavior. In addition to traditional IPv4 routing protocols, Quagga also supports IPv6 routing protocols. With SNMP daemon which supports SMUX protocol, Quagga provides routing protocol MIBs. Currently, Quagga supports common unicast routing protocols. Multicast routing protocols such as BGMP, PIM-SM, PIM-DM may be supported in Quagga 2.0. MPLS support is going on. In the future, TCP/IP filtering control, QoS control, diffserv configuration will be added to Quagga. Quagga project's final goal is making a productive, quality, free TCP/IP routing software

A router can be generally defined as a protocol independent networking device that routes IP packets between IP subnets. Quagga can be considered a router because, as a router, is able to perform the following operations:

- It keeps a local routing table containing information about destination networks and the way they are reached. The incoming packet is analyzed and the destination address on the packet is compared to the routing table entries and the one with the longest match. A routing table entry includes destination network, subnet mask, next hop address, output interface and metric information.
- Routing table entries are either locally configured static routes or dynamically learned from other routers with a routing protocol. On large networks with altering states, dynamic routing is normally used. Routing protocol defines the message discussion and format, and decision methodology for routing table alterations.
- As a router has physical interfaces, quagga has his interface (physical if quagga is running on a real machine, virtual if quagga is running on virtual machine), using different data link layer encapsulation methods. Layer 2 header is stripped from the incoming frame to expose the network layer packet (containing the destination address). After the routing decision the packet should be send to the next hop address, using the data link layer encapsulation configured for the interface the packet is send by. Routers decapsulate and encapsulate packets from/to data link layer frames.
- For configuration purposes, a router offers a user interface for configuration download, updates and uploads.

Quagga uses routing information to update the kernel routing table so that the right data goes to the right place. The configuration can be dynamically changed and routing table information can be viewed from the Quagga terminal interface.

Therefore, it uses an advanced software architecture to provide multi server routing engine. Quagga has an interactive user interface for each routing protocol and supports common client commands.

## 3.2 System Architecture

Traditional routing software is made as a one process program which provides all of the routing protocol functionality. Quagga takes a different approach. It is made from a collection of several daemons that work together to build the routing table. There may be several protocol-specific routing daemons and zebra the kernel routing manager.

The *ripd* daemon handles the RIP protocol, while *ospfd* is a daemon which supports OSPF version 2. The *bgpd* supports the BGP-4 protocol. For changing the kernel routing table and for redistribution of routes between different routing protocols, there is a kernel routing table manager *zebra* daemon. It is possible to add a new routing protocol daemons to the entire routing system without affecting any other software. It is just necessary to run only the protocol daemon associated with routing protocols in use. Thus, user may run a specific daemon and send routing reports to a central routing console.

There is no need for these daemons to be running on the same machine. It is even possible to run several same protocol daemons on the same machine. This architecture creates new possibilities for the routing system.

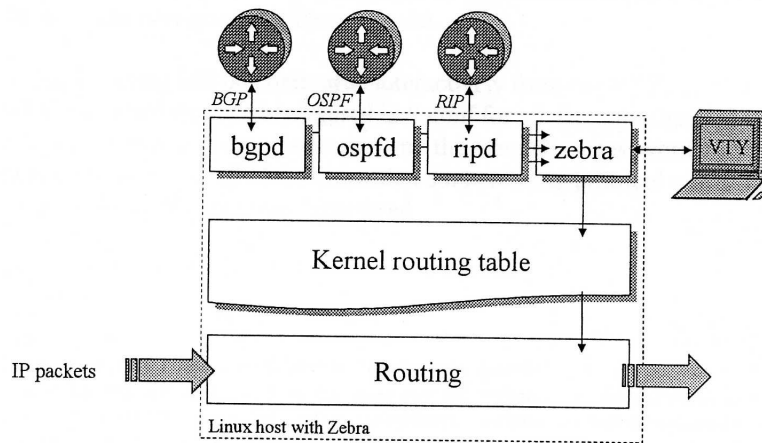


Figure 3.1: Quagga System Architecture

Multi-process architecture brings extensibility, modularity and maintainability. At the same time it also brings many configuration files and terminal interfaces. Each daemon has its own configuration file and terminal interface. When a static route is configured, it must be done in zebra configuration file. When BGP network is configured, it must be done in bgpd configuration file. This can be a very annoying thing. To resolve the problem, Quagga communities recently integrated user interface shell called vtysh. Vtysh connects to each daemon with UNIX domain socket and then works as a proxy for user input.

### 3.3 Supported Platform

Currently Quagga supports GNU/Linux, BSD and Solaris. Porting Quagga to other platforms is not too difficult. Protocol daemons are mostly platform independent.

The list of officially supported platforms are listed below. Note that Quagga may run correctly on other platforms, and may run with partial functionality on further platforms.

- GNU/Linux 2.4.x and higher.
- FreeBSD 4.x and higher.
- NetBSD 1.6 and higher.
- OpenBSD 2.5 and higher.
- Solaris 8 and higher.

In this project it has been used two distribution, CentOS (based on Red Hat Enterprise Linux) and Ubuntu server (based on Debian Linux).

## 3.4 Installation

There are two ways to install Quagga in a machine. The first by source codes which is suggested by Quagga support. The second is made by precompiled codes using the terminal.

### 3.4.1 Installation by source codes

There are three steps for installing the software:

- configuration
- compilation
- installation

The easiest way to get Quagga running is to issue the following commands:

```
% configure
% make
% make install
```

It follows steps will be describe how to use these script.

#### 3.4.1.1 The Configure script and its options

Quagga has an excellent configure script which automatically detects most host configurations. There are several additional configure options you can use to turn off IPv6 support, to disable the compilation of specific daemons, and to enable SNMP support.

‘**–enable-guile**’ Turn on compilation of the zebra-guile interpreter. You will need the guile library to make this. zebra-guile implementation is not yet finished. So this option is only useful for zebra-guile developers.

‘**–disable-ipv6**’ Turn off IPv6 related features and daemons. Quagga configure script automatically detects IPv6 stack. But sometimes you might want to disable IPv6 support of Quagga.

‘**–disable-zebra**’ Do not build zebra daemon.

‘**–disable-ripd**’ Do not build ripd.

‘**–disable-ripngd**’ Do not build ripngd.

‘**–disable-ospfd**’ Do not build ospfd.

‘**–disable-ospf6d**’ Do not build ospf6d.

‘**–disable-bgpd**’ Do not build bgpd.

‘**–disable-bgp-announce**’ Make bgpd which does not make bgp announcements at all. This feature is good for using bgpd as a BGP announcement listener.



‘**–enable-netlink**’ Force to enable GNU/Linux netlink interface. Quagga configure script detects netlink interface by checking a header file. When the header file does not match to the current running kernel, configure script will not turn on netlink support.

‘**–enable-snmp**’ Enable SNMP support. By default, SNMP support is disabled.

‘**–enable-opaque-lsa**’ Enable support for Opaque LSAs (RFC2370) in ospfd.

‘**–disable-ospfapi**’ Disable support for OSPF-API, an API to interface directly with ospfd. OSPF-API is enabled if `-enable-opaque-lsa` is set. ‘**–disable-ospfclient**’ Disable building of the example OSPF-API client.

‘**–enable-ospf-te**’ Enable support for OSPF Traffic Engineering Extension (Internet-draft) this requires support for Opaque LSAs. ‘**–enable-multipath=ARG**’ Enable support for Equal Cost Multipath. ARG is the maximum number of ECMP paths to allow, set to 0 to allow unlimited number of paths.

‘**–enable-rtadv**’ Enable support IPV6 router advertisement in zebra.

It could possible to specify any combination of the above options to the configure script. By default, the executable are placed in ‘/usr/local/sbin’ and the configuration files in ‘/usr/local/etc’. The ‘/usr/local/’ installation prefix and other directories may be changed using the following options to the configuration script.

‘**–prefix=prefix**’**Install** architecture-independent files in prefix [/usr/local].

‘**–sysconfdir=dir**’ Look for configuration files in dir [prefix/etc]. Note that sample configuration files will be installed here.

‘**–localstatedir=dir**’ Configure zebra to use dir for local state files, such as pid files and unix sockets.

This is an example how to configure quagga disabling ipv6:

```
./configure --disable-ipv6
```

**Least-Privilege support** Additionally, it is possible to configure zebra to drop its elevated privileges shortly after start up and switch to another user. The configure script will automatically try to configure this support. There are three configure options to control the behavior of Quagga daemons.

‘**–enable-user=user**’ Switch to user ARG shortly after start up, and run as user ARG in normal operation.

‘**–enable-group=group**’ Switch real and effective group to group shortly after start up.

‘**–enable-vty-group=group**’ Create Unix Vty sockets (for use with vtysh) with group ownership set to group. This allows one to create a separate group which is restricted to accessing only the Vty sockets, hence allowing one to delegate this group to individual users, or to run vtysh setgid to this group.

The default user and group which will be configured is 'quagga' if no user or group is specified. Note that this user or group requires write access to the local state directory and requires at least read access, and write access if you wish to allow daemons to write out their configuration, to the configuration directory (see `-sysconfdir`).

On systems which have the 'libcap' capabilities manipulation library (currently only Linux), the quagga system will retain only minimal capabilities required, further it will only raise these capabilities for brief periods. On systems without libcap, quagga will run as the user specified and only raise its uid back to uid 0 for brief periods.

### 3.4.1.2 Build the Software

After configuring the software, it is necessary to compile it for the system. Simply issue the command `make` in the root of the source directory and the software will be compiled. This is the command:

```
% make
```

### 3.4.1.3 Install the Software

Installing the software to the system consists of copying the compiled programs and supporting files to a standard location. After the installation process has completed, these files have been copied from your work directory to `/usr/local/bin`, and `/usr/local/etc`.

To install the Quagga suite, issue the following command at your shell prompt:

```
% make install
```

### 3.4.1.4 Checking Quagga files after installation

Quagga daemons have their own terminal interface or VTY. After installation, in `/etc/services` file there are the following entries.

```
zebrasrv 2600/tcp # zebra service
zebra    2601/tcp # zebra vty
ripd     2602/tcp # ripd vty (zebra)
ripngd   2603/tcp # ripngd vty (zebra)
ospfd    2604/tcp # ospfd vty (zebra)
bgpd     2605/tcp # bgpd vty (zebra)
ospf6d   2606/tcp # ospf6d vty (zebra)
ospfapi  2607/tcp # OSPF-API
isisd    2608/tcp # ISISd vty (zebra)
```

Figure 3.2: `/etc/services`

The daemons are placed in `/usr/local/sbin`. To start the daemons it is necessary to perform the follow task:

```
#zebra -d
#ripd -d
#ospfd -d
```

The configure files about the daemons are placed in `/usr/local/etc`.

### 3.4.2 Installing by terminal

This way to install Quagga is the suggestion way also better the installation guide obtaining from Quagga official website.

The first step is to open the Terminal, login as super-user and then is to follow the next steps.

- to see the Quagga version available:

```
root@stefano-ubuntu:/home/stefano# apt-cache policy quagga
quagga:
  Installato: (nessuno)
  Candidato: 0.99.15-1
  Tabella versione:
    0.99.15-1 0
    500 http://it.archive.ubuntu.com/ubuntu/ lucid/main Packages
```

- To download and install Quagga:

```
root@stefano-ubuntu:/home/stefano# apt-get install quagga
```

- To see the Quagga dependencies:

```
root@stefano-ubuntu:/home/stefano# apt-cache depends quagga
quagga
Dipende: libc6
Dipende: libcap2
Dipende: libpam0g
Dipende: libpcrc3
Dipende: libreadline6
Dipende: logrotate
Dipende: iproute
Dipende: debconf
Dipende: <debconf-2.0>
  cdebconf
  debconf
Pre-dipende: adduser
Consiglia: snmpd
Va in conflitto: <zebra>
Va in conflitto: <zebra-pj>
Sostituisce: <zebra>
Sostituisce: <zebra-pj>
```

## 3.5 Using Quagga

### 3.5.1 Quagga daemons

Before start using quagga is necessary to know that Quagga is composed of several daemons, one per routing protocol and another one called Zebra acting as the kernel routing manager. Each daemon has its own configuration file and terminal interface which can be accessed by telnet.

The following list are the daemons:

**zebra:** Interface declaration and static routing.

**bgpd:** BGP routing protocol.

**ospfd:** OSPF routing protocol.

**ospf6d:** OSPF IPv6 routing protocol.

**ripd:** RIP v2 routing protocol.

**ripngd:** RIP Ipv6 routing protocol.

It is important to know that the daemons must be activated after installing. In the following example there are been activated *zebra* and *rip* daemons:

- from the terminal perform this command

```
root@stefano-ubuntu:/home/stefano# vi /etc/quagga/daemons
```

- from the file remove the word “no” with “yes” to activate daemons as shown in the following imagine

```
#
zebra=yes
bgpd=no
ospfd=no
ospf6d=no
ripd=yes
ripngd=no
```

After made this changing is necessary restart the Quagga service typing from terminal the command:

```
root@stefano-ubuntu:/# /etc/init.d/quagga restart
```

You can check the Quagga daemons status:

```
#ps -ef | grep quagga
```

This is the output:

```
root@ubuntu:~# ps -ef | grep quagga
quagga    4406    1  0 14:08 ?        00:00:00 /usr/lib/quagga/zebra --daemon -
^ 127.0.0.1
quagga    4410    1  0 14:08 ?        00:00:00 /usr/lib/quagga/ripd --daemon -
^ 127.0.0.1
quagga    4414    1  0 14:08 ?        00:00:00 /usr/lib/quagga/ospfd --daemon -
^ 127.0.0.1
```

If a Quagga daemon doesn't stop properly, it is possible to kill it manually and start the quagga service:

```
#kill -9 "UID number"
#/etc/init.d/quagga start
```

### 3.5.2 Configuration files

It is necessary to create a configuration file to activate Quagga daemon.

Each daemon is associated with a specific file name:

**zebra:** zebra.conf.

**bgpd:** bgpd.conf.

**ospfd:** ospfd.conf.

**ospf6d:** ospf6d.conf.

**ripd:** ripd.conf.

**ripngd:** ripngd.conf.

These files are placed in `/etc/quagga/`. In this folder there are already created some daemons configuration samples.

To create the config files, it is necessary to copy the sample config files as follows: In this example, to activate zebra and ospfd daemons it is needed to create the `zebra.conf` and `ospfd.conf` files.

```
#cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
#cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf
```

Another way it is to create two empty files called `/etc/quagga/ospfd.conf` and `/etc/quagga/zebra.conf`. But in this case will not be possible to telnet a daemon, and it will be necessary to configure the telnet permissions with `vtsh`.

Finally, give user and group ownership to respectively `quagga` and `quaggavty` to the files inside the `/etc/quagga` directory:

```
#chown quagga.quaggavty /etc/quagga/*.conf #chmod 640 /etc/quagga/*.conf
```

Restart the Quagga service:

```
#!/etc/init.d/quagga restart
```

### 3.5.3 VTYSH

The way to access the daemons starting to configure the routing protocols is by telnetting the port number of each daemon. Each daemon has its configuration file and terminal interface.

**zebra:** 2601

**ripd:** 2602

**ripng:** 2603

**ospfd:** 2604

**bgpd:** 2605

**ospf6d:** 2606

This is the command to access, for example, the `ospfd` daemon:

```
#telnet localhost 2604
```

To be very honest it is not practical to configure every router by telnetting its daemons separately. For this reason recently it has been created `VTYSH` to configure every daemon in one single interface.

To use `vtsh`, it is necessary first to create its configuration file as follows:

```
#cp /usr/share/doc/quagga/examples/vtysh.conf.sample /etc/quagga/vtysh.conf
```

Apply correct permissions and restart Quagga:

```
#chown quagga.quaggavty /etc/quagga/*.conf
#chmod 640 /etc/quagga/*.conf
#/etc/init.d/quagga restart
```

This is an example of vtysh.conf:

```
?
? Sample configuration file for vtysh.
?
?service integrated-vtysh-config
hostname quagga-router
?username root password
```

In the example above the "service integrated-vtysh-config" setting has been disabled (recommended). In this case, when you save the config under vtysh, it will be stored in separate files depending on the protocols you activated. Below, an example where the Quagga configuration is saved under vtysh. (The zebra and ospfd daemons have been enabled).

```
#vtysh quagga-router
#write
```

Configuration saved to /etc/quagga/zebra.conf Configuration saved to /etc/quagga/ospfd.conf

If you activate "service integrated-vtysh-config", the configuration under vtysh will be saved in one file called Quagga.conf in the /etc/quagga/ directory. With this setting, when you access a daemon via telnet, the daemon will look first to the Quagga.conf file before looking for its own file. This means that, when you telnet a device, there can be a difference between what you see after the "show run" command and the content of the associated file, for example zebra.conf.

```
#vtysh quagga-router
#write
Configuration saved to /etc/quagga/Quagga.conf
```

It is recommended to disable "service integrated-vtysh-config" because if this setting is enabled and in case of a syntax error in the Quagga.conf file, this can lead to all your daemons being unable to start up. This will not be case when "service integrated-vtysh-config" is disabled because the configurations are stored in separate files.

Check that the default "vtysh\_enable=yes" setting are configured in your /etc/quagga/debian.conf file.

Then it's useful to add the "VTYSH\_PAGER=more" setting in your /etc/environment file, otherwise the screen will show an unfriendly "(END)" blinking in the left-down corner of the screen each time your enter a command and will need to press the "q" key to continue. Here what to do:

```
#echo VTYSH_PAGER=more > /etc/environment
```

Log off and log on to enable the environment setting. Now it possible to access to the Quagga router with the vtysh command:

```
#vtysh
Hello, this is Quagga(version 0.99.6).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
quagga-router#
```

To run a Quagga command from the Linux shell:

```
#vtysh -c "command"
```

For instance, `vtysh -c "show ip route"` will display the Quagga routing table.

To use Ping and traceroute to perform connectivity checks from the vtysh prompt. Of course, these two programs need to be installed on the Linux machine. Ping is generally installed by default but traceroute often not. To install traceroute:

```
#apt-get install traceroute
```

IP forwarding is required to transfer packets between the network interfaces of a Linux system. See a picture of the Linux kernel routing.

```
#echo "1" > /proc/sys/net/ipv4/ip_forward
```

The command above will add the "1" value inside the `/proc/sys/net/ipv4/ip_forward` file and thus activate the IP forwarding. To keep the IP forwarding after a Linux reboot:

```
#echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
```

It is possible to check the `ip_forwarding` status under the Quagga router:

```
#show ip forwarding
IP forwarding is on
```

In this case the IP forwarding is activated.

For the project in this thesis it has been used vtysh configuration.

### 3.5.4 Routers functionality comparison

This section will show a comparison of some characteristics among four routers: the first two are the best open source routers available, Quagga and Vyatta<sup>1</sup> and the remaining two are commercial devices and best-sellers from Cisco: a standard router from the 26xx family and a Layer Three Switch 3750.

---

<sup>1</sup>Vyatta is an open source routing software which is developed by the Vyatta company created in 2005. Vyatta's open, software-based approach to networking has created a complete network OS that can connect and secure physical networks as well as virtual and cloud computing infrastructures. Vyatta software and appliances offer users a flexible, affordable alternative to proprietary, hardware-based routers, firewalls, VPN concentrators and intrusion prevention devices.

	RFC	QUAGGA 0.99.6	VYATTA VC 3.0	CISCO 2651	CISCO 3750 L3
Software type		IOS-Like	JunOS-Like	Cisco IOS	Cisco IOS
Opensource		YES	YES	NO	NO
Installation		YES	YES	NO	NO
Static Routing	-	YES	YES	YES	YES
RIPv2	2453	YES	YES	YES	YES
OSPFv2	2328	YES	YES	YES	YES
BGPv4	1771/4271	YES	YES	YES	YES
NAT	2766	NO/Linux	YES	YES	NO
VRRP	3768	NO/Linux	YES	YES	NO
Access lists		YES	YES	YES	YES
Route maps		YES	YES	YES	YES
VPN IPsec	4301	NO/Linux (1)	YES	YES	YES
VPN SSL		NO/Linux (2)	NO/Linux (2)	NO	NO
FTP client	959	NO/Linux	YES	YES	YES
TFTP client	1350	NO	YES	YES	YES
Telnet server	854	YES (3)	YES	YES	YES
ssh server	4251	NO/Linux	YES	YES/NO	YES/NO
HTTP server		NO	YES	YES	YES
DHCP server	2131	NO/Linux	YES	YES	YES
DHCP relay		NO/Linux	YES	YES	YES
NTP server	1305	NO/Linux	NO	YES/NO	NO
NTP client	1305	NO/Linux	YES	YES	YES
SNMP	3412/1157	YES (4)	YES	YES	YES
Ping		YES (5)	YES	YES	YES
tracert		YES (5)	YES	YES	YES

Legend:

- YES: The protocol is supported.  
 YES/NO: Available on specific Cisco IOSs only.  
 BUGGED: There is a major bug on the protocol.  
 NO/Linux: Not supported on the Quagga router but can be enabled at the Linux level.  
 (1) Openswan can be used as an implementation of IPsec for Linux.  
 (2) OpenVPN can be used to build SSL VPNs on Linux.  
 (3) Each routing protocol daemon and the zebra kernel routing manager have to be telnetted separately.  
 (4) Quagga must be compiled with the SNMP functionality.  
 (5) Ping or traceroute must be installed at the Linux level to be used under Quagga.

Figure 3.3: Routers functionality comparison

## 3.5.5 Commands

### 3.5.5.1 Basic Commands

In this session it is explained the basic command to use quagga and configure it as a real router.

Once inside the terminal it is possible to invoke the following commands.

**hostname *hostname*.** Set hostname of the router.

**password *password*.** Set password for vty interface. If there is no password, a vty won't accept connections.

**enable password *password*.** Set enable password.



**log trap level.** These commands are deprecated and are present only for historical compatibility. The log trap command sets the current logging level for all enabled logging destinations, and it sets the default for all future logging commands that do not specify a level. The normal default logging level is debugging. The no form of the command resets the default level for future logging commands to debugging, but it does not change the logging level of existing logging destinations.

**log stdout level.** Enable logging output to stdout. If the optional second argument specifying the logging level is not present, the default logging level (typically debugging, but can be changed using the deprecated log trap command) will be used. The no form of the command disables logging to stdout. The level argument must have one of these values: emergencies, alerts, critical, errors, warnings, notifications, informational, or debugging. Note that the existing code logs its most important messages with severity errors.

**log file filename.** If you want to log into a file, please specify filename as in this example: log file /var/log/quagga/bgpd.log informational. If the optional second argument specifying the logging level is not present, the default logging level (typically debugging, but can be changed using the deprecated log trap command) will be used. The no form of the command disables logging to a file. Note: if do not configure any file logging, and a daemon crashes due to a signal or an assertion failure, it will attempt to save the crash information in a file named /var/tmp/quagga.<daemon name>.crashlog. For security reasons, this will not happen if the file exists already, so it is important to delete the file after reporting the crash information.

**log syslog.** Enable logging output to syslog. If the optional second argument specifying the logging level is not present, the default logging level (typically debugging, but can be changed using the deprecated log trap command) will be used. The no form of the command disables logging to syslog.

**log monitor.** Enable logging output to vty terminals that have enabled logging using the terminal monitor command. By default, monitor logging is enabled at the debugging level, but this command (or the deprecated log trap command) can be used to change the monitor logging level. If the optional second argument specifying the logging level is not present, the default logging level (typically debugging, but can be changed using the deprecated log trap command) will be used. The no form of the command disables logging to terminal monitors.

**log facility facility.** This command changes the facility used in syslog messages. The default facility is daemon. The no form of the command resets the facility to the default daemon facility.

**log record-priority.** To include the severity in all messages logged to a file, to stdout, or to a terminal monitor (i.e. anything except syslog), use the log record-priority global configuration command. To disable this option, use the no form of the command. By default, the severity level is not included in logged messages. Note: some versions of syslogd (including Solaris) can be configured to include the facility and level in the messages emitted.

**service password-encryption.** Encrypt password.

**line vty.** Enter vty configuration mode.

**banner motd default.** Set default motd string.

**access-class *access-list*.** Restrict vty connections with an access list.

### 3.5.5.2 Terminal Mode Commands

In this section there are a list of common commands used in Terminal Mode.

**write terminal.** Displays the current configuration to the vty interface.

**write file.** Write current configuration to configuration file.

**configure terminal.** Change to configuration mode. This command is the first step to configuration.

**terminal length <0-512>.** Set terminal display length to <0-512>. If length is 0, no display control is performed.

**who.** Show a list of currently connected vty sessions.

**list.** List all available commands.

**show version.** Show the current version of Quagga and its build host information.

**show logging.** Shows the current configuration of the logging system. This includes the status of all logging destinations.

**logmsg *level message*.** Send a message to all logging destinations that are enabled for messages of the given severity.

## 3.5.6 Zebra

In this section it will explain how to access to zebra daemon configure terminal and a list of basic commands used to start configuring routers.

### 3.5.6.1 Virtual Terminal Interfaces

Virtual Terminal Interface (VTY) is a command line interface (CLI) for user interaction with the routing daemon. VTY stands for Virtual TeletYpe interface. It means it could possible to connect to the daemon via the telnet protocol.

To enable a VTY interface, a VTY password has to be setup. If there is no VTY password, one cannot connect to the VTY interface at all.

This is an example how to connect, for example, to zebra. It shows that the only way to connect directly to this daemon is by telnetting the port 2601.

Therefore, the VTY ask the User Access Verification saved in zebra.conf file. After this, like a real Cisco router, typing the question mark it will shows a list of command that a standard user can invoke. Then, typing the *enable* command and inserting the password is possible to access to the core to configure the virtual router.

```

root@ubuntu:/home/stefano# telnet localhost 2601
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.9).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
Router>
  echo      Echo a message back to the vty
  enable    Turn on privileged mode command
  exit      Exit current mode and down to previous mode
  help      Description of the interactive help system
  list      Print command list
  quit      Exit current mode and down to previous mode
  show      Show running system information
  terminal   Set terminal line parameters
  who       Display who is on vty
Router> enable
Password:
Router#

```

Figure 3.4: zebra VTY

### 3.5.6.2 Interface Command

In this session it will show a list of commands used in *configure terminal* mode like a Cisco router.

```

quagga-router# configure terminal
quagga-router(config)# interface eth0
quagga-router(config-if)# ip address 192.168.10.2/27
quagga-router(config-if)# no shutdown
quagga-router(config-if)# description Interface router quagga

```

Figure 3.5: example of sample interface configuration

The previous commands set up or down the current interface. The IPv4, the description are set for the interface.

Therefore it is possible to configure static routing. Static routing is a very fundamental feature of routing technology. It defines static prefix and gateway.

**ip route *network gateway*.** *Network* is destination prefix with format of A.B.C.D/M. *Gateway* is gateway for the prefix. It is taken as a IPv4 address gateway. Otherwise it is treated as an interface name.

Here some example of this command.

```

#ip route 10.0.0.0/8 10.0.0.2
#ip route 10.0.0.0/8 ppp0
#ip route 10.0.0.0/8 null0

```

First example defines 10.0.0.0/8 static route with gateway 10.0.0.2. Second one defines the same prefix but with gateway to interface ppp0. The third install a blackhole route.

**ip route *network netmask gateway*.** This is alternate version of above command. When network is A.B.C.D format, user must define netmask value with A.B.C.D format. gateway is same option as above command

```
#ip route 10.0.0.0 255.255.255.0 10.0.0.2
#ip route 10.0.0.0 255.255.255.0 ppp0
#ip route 10.0.0.0 255.255.255.0 null0
```

These statements are equivalent to those in the previous example.

**ip route *network gateway distance*.** This command installs the route with the specified distance.

Multiple next hop static route

```
#ip route 10.0.0.1/32 10.0.0.2
#ip route 10.0.0.1/32 10.0.0.3
#ip route 10.0.0.1/32 eth0
```

If there is no route to 10.0.0.2 and 10.0.0.3, and interface eth0 is reachable, then the last route is installed into the kernel.

If zebra has been compiled with multipath support, and both 10.0.0.2 and 10.0.0.3 are reachable, zebra will install a multipath route via both next hops, if the platform supports this. This is the command:

```
#show ip route
```

And this is the output:

```
S> 10.0.0.1/32 [1/0] via 10.0.0.2 inactive
                        via 10.0.0.3 inactive
      *                  is directly connected, eth0
#ip route 10.0.0.0/8 10.0.0.2
#ip route 10.0.0.0/8 10.0.0.3
#ip route 10.0.0.0/8 null0 255
```

This will install a multiform route via the specified next-hops if they are reachable, as well as a high-metric blackhole route, which can be useful to prevent traffic destined for a prefix to match less-specific routes (eg default) should the specified gateways not be reachable. For example:

```
#show ip route 10.0.0.0/8
```

And this is the output:

```
Routing entry for 10.0.0.0/8
  Known via "static", distance 1, metric 0
  10.0.0.2 inactive
  10.0.0.3 inactive
Routing entry for 10.0.0.0/8
  Known via "static", distance 255, metric 0
  directly connected, Null0
```

This is the command to set ipv6:

```
#ipv6 route network gateway
#ipv6 route network gateway distance
```

These behave similarly to their ipv4 counterparts.

The follows command are about the Cisco command “show”

```
show ip route
```

Display current routes which zebra holds in its database.

```
quagga-router# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

K>* 0.0.0.0/0 via 192.168.145.2, eth2
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.10.0/24 is directly connected, eth1
C>* 192.168.10.0/27 is directly connected, eth0
C>* 192.168.10.64/27 is directly connected, eth1
C>* 192.168.145.0/24 is directly connected, eth2
```

```
#show ipv6 route
```

```
#show interface
```

```
Interface eth0 is up, line protocol detection is disabled
index 2 metric 1 mtu 1500
flags: <UP,BROADCAST,MULTICAST>
HWaddr: 00:0c:29:9d:86:95
inet 192.168.10.2/27 broadcast 192.168.10.31
  0 input packets (0 multicast), 0 bytes, 0 dropped
  0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
  0 fifo, 0 missed
  0 output packets, 0 bytes, 0 dropped
  0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartbeat
  0 window, 0 collisions
Interface eth1 is up, line protocol detection is disabled
index 3 metric 1 mtu 1500
flags: <UP,BROADCAST,RUNNING,MULTICAST>
HWaddr: 00:0c:29:9d:86:9f
inet 192.168.10.66/24 broadcast 192.168.10.255
inet 192.168.10.65/27 broadcast 192.168.10.95
inet6 fe80::20c:29ff:fe9d:869f/64
  1 input packets (0 multicast), 243 bytes, 0 dropped
  0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
  0 fifo, 0 missed
  889 output packets, 38850 bytes, 0 dropped
  0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartbeat
  0 window, 0 collisions
```

```
#show ip forward
```

Display whether the host’s IP forwarding function is enabled or not. Almost any UNIX kernel can be configured with IP forwarding disabled. If so, the box can’t work as a router.

```
#show ipv6forward
```

Display whether the host’s IP v6 forwarding is enabled or not.

### 3.5.7 RIP

In this section it will explain how to access to ripd daemon configure terminal and a list of basic commands used to start configuring routing.

### 3.5.7.1 Virtual Terminal Interfaces

Very similar how explained in the section 3.5.6.1, the only way to connect directly to this daemon is by telnetting the port 2602.

Therefore, the VTY ask the User Access Verification saved in `ripd.conf` file. After this, like a real Cisco router, typing the question mark it will shows a list of command that a standard user can invoke. Then, typing the *enable* command and inserting the password is possible to access to the core to configure the virtual router.

```

root@ubuntu:~# telnet localhost 2602
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^I'.

Hello, this is Quagga (version 0.99.9).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
ripd>
  echo      Echo a message back to the vty
  enable    Turn on privileged mode command
  exit      Exit current mode and down to previous mode
  help      Description of the interactive help system
  list      Print command list
  quit      Exit current mode and down to previous mode
  show      Show running system information
  terminal   Set terminal line parameters
  who       Display who is on vty
ripd> enable
Password:
ripd#

```

Figure 3.6: ripd VTY

### 3.5.7.2 RIP configuration

Routing Information Protocol (RIP) is widely deployed interior gateway protocol. RIP was developed in the 1970s at Xerox Labs as part of the XNS routing protocol. RIP is a distance-vector protocol and is based on the Bellman-Ford algorithms. As a distance-vector protocol, RIP router send updates to its neighbors periodically, thus allowing the convergence to a known topology. In each update, the distance to any given network will be broadcasted to its neighboring router.

The daemon *ripd* supports RIP version 2 as described in RFC2453 and RIP version 1 as described in RFC1058.

The netmask features of *ripd* support both version 1 and version 2 of RIP. Version 1 of RIP originally contained no netmask information. In RIP version 1, network classes were originally used to determine the size of the netmask. Class A networks use 8 bits of mask, Class B networks use 16 bits of masks, while Class C networks use 24 bits of mask. Today, the most widely used method of a network mask is assigned to the packet on the basis of the interface that received the packet. Version 2 of RIP supports a variable length subnet mask (VLSM). By extending the subnet mask, the mask can be divided and reused. Each subnet can be used for different purposes such as large to middle size LANs and WAN links. Quagga *ripd* does not support the non-sequential netmasks that are included in RIP Version 2.

```
quagga-router(config)#router rip
```

The router rip command is necessary to enable RIP. To disable RIP, use the no router rip command. RIP must be enabled before carrying out any of the RIP commands.

```
quagga-router(config)#no router rip
```

Set the RIP enable interface by network. The interfaces which have addresses matching with network are enabled.

```
quagga-router(config)#network network
quagga-router(config)#no network network
```

Set the RIP enable interface by network. The interfaces which have addresses matching with network are enabled.

This group of commands either enables or disables RIP interfaces between certain numbers of a specified network address. For example, if the network for 10.0.0.0/24 is RIP enabled, this would result in all the addresses from 10.0.0.0 to 10.0.0.255 being enabled for RIP. The no network command will disable RIP for the specified network.

```
quagga-router(config)#network ifname
quagga-router(config)#no network ifname
```

Set a RIP enabled interface by ifname. Both the sending and receiving of RIP packets will be enabled on the port specified in the network ifname command. The no network ifname command will disable RIP on the specified interface.

```
quagga-router(config)#neighbor a.b.c.d
quagga-router(config)#neighbor a.b.c.d
```

Specify RIP neighbor. When a neighbor doesn't understand multicast, this command is used to specify neighbors. In some cases, not all routers will be able to understand multicasting, where packets are sent to a network or a group of addresses. In a situation where a neighbor cannot process multicast packets, it is necessary to establish a direct link between routers. The neighbor command allows the network administrator to specify a router as a RIP neighbor. The no neighbor a.b.c.d command will disable the RIP neighbor.

Below is very simple RIP configuration. Interface eth0 and interface which address match to 10.0.0.0/8 are RIP enabled.

```
quagga-router(config)#router rip
quagga-router(config-router)#network 10.0.0.0/8 network eth0
```

### 3.5.7.3 RIP Version Control

RIP can be configured to send either Version 1 or Version 2 packets. The default is to send RIPv2 while accepting both RIPv1 and RIPv2 (and replying with packets of the appropriate version for REQUESTS / triggered updates). The version to receive and send can be specified globally, and further overridden on a per-interface basis if needs be for send and receive separately (see below).

It is important to note that RIPv1 can not be authenticated. Further, if RIPv1 is enabled then RIP will reply to REQUEST packets, sending the state of its RIP routing table to any remote routers that ask on demand. For a more detailed discussion on the security implications of RIPv1 see RIP Authentication.

```
quagga-router(config)# version version
```

Set RIP version to accept for reads and send. version can be either ‘1’ or ‘2’.

Disabling RIPv1 by specifying version 2 is strongly encouraged, see section RIP Authentication. This may become the default in a future release.

Default: send Version 2, and accept either version.

```
quagga-router(config)#no version
```

Reset the global version setting back to the default.

```
quagga-router(config)#ip rip send version [1-2]
```

This interface command overrides the global rip version setting, and selects which version of RIP to send packets with, for this interface specifically. Choice of RIP Version 1, RIP Version 2, or both versions. In the latter case, where ‘1 2’ is specified, packets will be both broadcast and multicast.

#### 3.5.7.4 Access list RIP Routes

RIP routes can be filtered by a distribute-list.

```
quagga-router(config)#distribute-list access_list direct ifname
```

To apply access lists to the interface with a distribute-list command; *access\_list* is the access list name. While *direct* is ‘in’ or ‘out’. If direct is ‘in’ the access list is applied to input packets.

The distribute-list command can be used to filter the RIP path. distribute-list can apply access-lists to a chosen interface. First, one should specify the access-list. Next, the name of the access-list is used in the distribute-list command. For example, in the following configuration ‘eth0’ will permit only the paths that match the route 10.0.0.0/8

```
router rip
network 192.168.10.0/27
network 192.168.10.64/27
distribute-list 1 in eth1
?
access-list 1 deny 192.168.10.36
```

distribute-list can be applied to both incoming and outgoing data.

```
distribute-list prefix prefix_list (in|out) ifname
```

To apply prefix lists to the interface with a distribute-list command; *prefix\_list* is the prefix list name. Next is the direction of “in” or “ou”. If direct is ‘in’ the access list is applied to input packets.



### 3.5.7.5 RIP Authentication

RIPv2 allows packets to be authenticated via either an insecure plain text password, included with the packet, or via a more secure MD5 based HMAC (keyed-Hashing for Message Authentication), RIPv1 can not be authenticated at all, thus when authentication is configured ripd will discard routing updates received via RIPv1 packets.

However, unless RIPv1 reception is disabled entirely, see section RIP Version Control, RIPv1 REQUEST packets which are received, which query the router for routing information, will still be honored by ripd, and ripd WILL reply to such packets. This allows ripd to honour such REQUESTs (which sometimes is used by old equipment and very simple devices to bootstrap their default route), while still providing security for route updates which are received.

In short: enabling authentication prevents routes being updated by unauthenticated remote routers, but still can allow routes (I.e. the entire RIP routing table) to be queried remotely, potentially by anyone on the Internet, via RIPv1.

To prevent such unauthenticated querying of routes disable RIPv1, See section RIP Version Control.

```
quagga-router(config)#interface eth0
quagga-router(config-if)#ip rip authentication mode md5
quagga-router(config-if)#no ip rip authentication mode md5
```

Set the interface with RIPv2 MD5 authentication.

```
quagga-router(config-if)#ip rip authentication mode text
quagga-router(config-if)#no ip rip authentication mode text
```

To set the interface with RIPv2 simple password authentication.

```
quagga-router(config-if)#ip rip authentication string string
quagga-router(config-if)#no ip rip authentication string string
```

RIP version 2 has simple text authentication. This command sets authentication string. The string must be shorter than 16 characters.

```
quagga-router(config-if)#ip rip authentication key-chain key-chain
quagga-router(config-if)#no ip rip authentication key-chain key-chain
```

### 3.5.7.6 RIP Timers

RIP protocol has several timers. User can configure those timer's values by timers basic command.

```
quagga-router(config-router)#timers basic [update timeout garbage]
```

The default settings for the timers are as follows:

- The update timer is 30 seconds. Every update timer seconds, the RIP process is awakened to send an unsolicited Response message containing the complete routing table to all neighboring RIP routers.

- The timeout timer is 180 seconds. Upon expiration of the timeout, the route is no longer valid. However, it is retained in the routing table for a short time so that neighbors can be notified that the route has been dropped.
- The garbage collect timer is 120 seconds. Upon expiration of the garbage-collection timer, the route is finally removed from the routing table.

The `timers basic` command allows the the default values of the timer listed above to be changed.

```
quagga-router(config-router)#no timers basic
```

The `no timers basic` command will reset the timers to the default settings listed above.

### 3.5.7.7 Show RIP Information

This is the command to display RIP routes:

```
quagga-router#show ip rip
```

This is the output:

```
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

      Network          Next Hop          Metric From          Tag Time
C(i) 192.168.10.0/24   0.0.0.0           1 self              0
C(i) 192.168.10.0/27   0.0.0.0           16 self             0 01:50
R(n) 192.168.10.32/27  192.168.10.1      16 192.168.10.1     0 01:50
C(i) 192.168.10.64/27 0.0.0.0           1 self              0
```

The command displays all RIP routes. For routes that are received through RIP, this command will display the time the packet was sent and the tag information. This command will also display this information for routes redistributed into RIP.

```
quagga-router#show ip protocols
```

The previous command displays current RIP status. It includes RIP timer, filtering, version, RIP enabled interface and RIP peer information.

## 3.5.8 OSPFv2

In this section it will explain how to access to ospfv2 daemon configure terminal and a list of basic commands used to start configuring routing.

### 3.5.8.1 Virtual Terminal Interfaces

Very similar how explained in the section 3.5.6.1, the only way to connect directly to this daemon is by telnetting the port 2604.

Therefore, the VTY ask the User Access Verification saved in `ospfd.conf` file. After this, like a real Cisco router, typing the question mark it will shows a list of command that a standard user can invoke. Then, typing the `enable` command and inserting the password is possible to access to the core to configure the virtual router.

```

root@ubuntu:~# telnet localhost 2604
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^I'.

Hello, this is Quagga (version 0.99.9).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
ospfd>
  echo      Echo a message back to the vty
  enable    Turn on privileged mode command
  exit      Exit current mode and down to previous mode
  help      Description of the interactive help system
  list      Print command list
  quit      Exit current mode and down to previous mode
  show      Show running system information
  terminal   Set terminal line parameters
  who       Display who is on vty
ospfd> enable
ospfd#

```

Figure 3.7: Ospf VTY

### 3.5.8.2 OSPFv2 Configuration

OSPF (Open Shortest Path First) version 2 is a routing protocol which is described in RFC2328, OSPF Version 2. OSPF is an IGP (Interior Gateway Protocol). Compared with RIP, OSPF can provide scalable network support and faster convergence times. OSPF is widely used in large networks such as ISP (Internet Service Provider) backbone and enterprise networks.

### 3.5.8.3 OSPF router

To start OSPF process you have to specify the OSPF router. As of this writing, ospfd does not support multiple OSPF processes.

```

quagga-router(config)#router ospf
quagga-router(config)#: no router ospf

```

Enable or disable the OSPF process. Ospf does not yet support multiple OSPF processes. So it is not possible to specify an OSPF process number.

```

quagga-router(config-router)#ospf router-id a.b.c.d
quagga-router(config-router)#no ospf router-id

```

This sets the router-ID of the OSPF process. The router-ID may be an IP address of the router. However it must be unique within the entire OSPF domain to the OSPF speaker, bad things will happen if multiple OSPF speakers are configured with the same router-ID. If one is not specified then ospfd will obtain a router-ID automatically from zebra.

```

quagga-router(config-router)#ospf abr-type type
quagga-router(config-router)#no ospf abr-type type

```

The key-word type can be `cisco|ibm|shortcut|standard`. The "Cisco" and "IBM" types are equivalent.

The OSPF standard for ABR behavior does not allow an ABR to consider routes through non-backbone areas when its links to the backbone are down, even when there are other ABRs in attached non-backbone areas which still can reach the backbone - this restriction exists primarily to ensure routing-loops are avoided.

With the "Cisco" or "IBM" ABR type, the default in this release of Quagga, this restriction is lifted, allowing an ABR to consider summaries learned from other ABRs through non-backbone areas, and hence route via non-backbone areas as a last resort when, and only when, backbone links are down.

Note that areas with fully-adjacent virtual-links are considered to be "transit capable" and can always be used to route backbone traffic, and hence are unaffected by this setting (see OSPF virtual-link).

More information regarding the behavior controlled by this command can be found in RFC 3509. Alternative implementations of OSPF Area Border Routers.

Moreover though the definition of the ABR (Area Border Router) in the OSPF specification does not require a router with multiple attached areas to have a backbone connection, it is actually necessary to provide successful routing to the inter-area and external destinations. If this requirement is not met, all traffic destined for the areas not connected to such an ABR or out of the OSPF domain, is dropped. This document describes alternative ABR behaviors implemented in Cisco and IBM routers.

```
quagga-router(config-router)#ospf rfc1583compatibility
quagga-router(config-router)#no ospf rfc1583compatibility
```

RFC2328, the successor to RFC1583, suggests according to section G.2 (changes) in section 16.4 a change to the path preference algorithm that prevents possible routing loops that were possible in the old version of OSPFv2. More specifically it demands that inter-area paths and intra-area path are now of equal preference but still both preferred to external paths.

This command should not be set normally.

```
quagga-router(config-router)#log-adjacency-changes [detail]
quagga-router(config-router)#no log-adjacency-changes [detail]
```

Configures ospfd to log changes in adjacency. With the optional detail argument, all changes in adjacency status are shown. Without detail, only changes to full or regressions are shown.

```
quagga-router(config-router)#passive-interface interface
quagga-router(config-router)#no passive-interface interface
```

Do not speak OSPF interface on the given interface, but do advertise the interface as a stub link in the router-LSA (Link State Advertisement) for this router. This allows one to advertise addresses on such connected interfaces without having to originate AS-External/Type-5 LSAs (which have global flooding scope), as would occur if connected addresses were redistributed into OSPF (see section Redistribute routes to OSPF). This is the only way to advertise non-OSPF links into stub areas.

```
quagga-router(config-router)#timers throttle spf delay
quagga-router(config-router)#no timers throttle spf
```

This command sets the initial delay, the initial-holdtime and the maximum-holdtime between when SPF is calculated and the event which triggered the calculation. The times are specified in milliseconds and must be in the range of 0 to 600000 milliseconds.

The delay specifies the minimum amount of time to delay SPF calculation (hence it affects how long SPF calculation is delayed after an event which occurs outside of the holdtime of any previous SPF calculation, and also serves as a minimum holdtime).

Consecutive SPF calculations will always be separated by at least 'hold-time' milliseconds. The hold-time is adaptive and initially is set to the initial-holdtime configured with the above command. Events which occur within the holdtime of the previous SPF calculation will cause the holdtime to be increased by initial-holdtime, bounded by the maximum-holdtime configured with this command. If the adaptive hold-time elapses without any SPF-triggering event occurring then the current holdtime is reset to the initial-holdtime. The current holdtime can be viewed with `show ip ospf`, where it is expressed as a multiplier of the initial-holdtime.

```
quagga-router(config)#router ospf
quagga-router(config-router)#timers throttle spf 200 400 10000
```

In this example, the delay is set to 200ms, the initial holdtime is set to 400ms and the maximum holdtime to 10s. Hence there will always be at least 200ms between an event which requires SPF calculation and the actual SPF calculation. Further consecutive SPF calculations will always be separated by between 400ms to 10s, the hold-time increasing by 400ms each time an SPF-triggering event occurs within the hold-time of the previous SPF calculation.

This command super cedes the `timers spf` command in previous Quagga releases.

```
quagga-router(config-router)#max-metric router-lsa [on-startup|on-shutdown]
quagga-router(config-router)#max-metric router-lsa administrative
quagga-router(config-router)#no max-metric router-lsa [on-startup|on-shutdown]
```

This enables RFC3137, OSPF Stub Router Advertisement support, where the OSPF process describes its transit links in its router-LSA as having infinite distance so that other routers will avoid calculating transit paths through the router while still being able to reach networks through the router.

This support may be enabled administratively (and indefinitely) or conditionally. Conditional enabling of max-metric router-lsas can be for a period of seconds after start up and/or for a period of seconds prior to shutdown.

Enabling this for a period after start up allows OSPF to converge fully first without affecting any existing routes used by other routers, while still allowing any connected stub links and/or redistributed routes to be reachable. Enabling this for a period of time in advance of shutdown allows the router to gracefully excuse itself from the OSPF domain.

Enabling this feature administratively allows for administrative intervention for whatever reason, for an indefinite period of time. Note that if the configuration is written to file, this administrative form of the stub-router command will also be written to file. If `ospfd` is restarted later, the command will then take effect until manually reconfigured.

Configured state of this feature as well as current status, such as the number of second remaining till on-startup or on-shutdown ends, can be viewed with the `show ip ospf` command.

```
quagga-router(config-router)#auto-cost reference-bandwidth <1-4294967>
quagga-router(config-router)#no auto-cost reference-bandwidth
```

This sets the reference bandwidth for cost calculations, where this bandwidth is considered equivalent to an OSPF cost of 1, specified in Mbits/s. The default is 100Mbit/s (i.e. a link of bandwidth 100Mbit/s or higher will have a cost of 1. Cost of lower bandwidth links will be scaled with reference to this cost).

This configuration setting **MUST** be consistent across all routers within the OSPF domain.

```
quagga-router(config-router)#network a.b.c.d/m area a.b.c.d
quagga-router(config-router)#network a.b.c.d/m area <0-4294967295>
quagga-router(config-router)#no network a.b.c.d/m area a.b.c.d
quagga-router(config-router)#no network a.b.c.d/m area <0-4294967295>
```

This command specifies the OSPF enabled interface(s). If the interface has an address from range 192.168.1.0/24 then the command below enables ospf on this interface so router can provide network information to the other ospf routers via this interface.

```
quagga-router(config)#router ospf
quagga-router(config-router)#network 192.168.1.0/24 area 0.0.0.0
```

Prefix length in interface must be equal or bigger (ie. smaller network) than prefix length in network statement. For example statement above doesn't enable ospf on interface with address 192.168.1.1/23, but it does on interface with address 192.168.1.129/25.

#### 3.5.8.4 OSPF area

This section will explain other command to set ospf area.

```
quagga-router(config-router)#area a.b.c.d range a.b.c.d/m
quagga-router(config-router)#area <0-4294967295> range a.b.c.d/m
quagga-router(config-router)#no area a.b.c.d range a.b.c.d/m
quagga-router(config-router)#no area <0-4294967295> range a.b.c.d/m
```

Summarize intra area paths from specified area into one Type-3 summary-LSA announced to other areas. This command can be used only in ABR and ONLY router-LSAs (Type-1) and network-LSAs (Type-2) (ie. LSAs with scope area) can be summarized. Type-5 AS-external-LSAs can't be summarized, their scope is AS. Summarizing Type-7 AS-external-LSAs isn't supported yet by Quagga.

```
quagga-router(config)#router ospf
quagga-router(config-router)#network 192.168.1.0/24 area 0.0.0.0
quagga-router(config-router)#network 10.0.0.0/8 area 0.0.0.10
```

With configuration above one Type-3 Summary-LSA with routing info 10.0.0.0/8 is announced into backbone area if area 0.0.0.10 contains at least one intra-area network (ie. described with router or network LSA) from this range.

```

quagga-router(config-router)#area a.b.c.d range IPV4_PREFIX not-advertise
quagga-router(config-router)#no area a.b.c.d range IPV4_PREFIX not-advertise

```

Instead of summarizing intra area paths filter them, intra area paths from this range are not advertised into other areas. This command makes sense in ABR only.

```

quagga-router(config-router)#area a.b.c.d range IPV4_PREFIX substitute IPV4_PREFIX OSPF
quagga-router(config-router)#no area a.b.c.d range IPV4_PREFIX substitute IPV4_PREFIX

```

Substitute summarized prefix with another prefix.

```

quagga-router(config)#router ospf
quagga-router(config-router)#network 192.168.1.0/24 area 0.0.0.0
quagga-router(config-router)#network 10.0.0.0/8 area 0.0.0.10
quagga-router(config-router)#area 0.0.0.10 range 10.0.0.0/8

```

One Type-3 summary-LSA with routing info 11.0.0.0/8 is announced into backbone area if area 0.0.0.10 contains at least one intra-area network (ie. described with router-LSA or network-LSA) from range 10.0.0.0/8. This command makes sense in ABR only.

```

quagga-router(config-router)#area a.b.c.d virtual-link a.b.c.d
quagga-router(config-router)#area <0-4294967295> virtual-link a.b.c.d
quagga-router(config-router)#no area a.b.c.d virtual-link a.b.c.d
quagga-router(config-router)#no area <0-4294967295> virtual-link a.b.c.d
quagga-router(config-router)#area a.b.c.d shortcut
quagga-router(config-router)#area <0-4294967295> shortcut
quagga-router(config-router)#no area a.b.c.d shortcut
quagga-router(config-router)#no area <0-4294967295> shortcut

```

Configure the area as Shortcut capable. See RFC3509. This requires that the 'abr-type' be set to 'shortcut'.

```

quagga-router(config-router)#area a.b.c.d stub
quagga-router(config-router)#area <0-4294967295> stub
quagga-router(config-router)#no area a.b.c.d stub
quagga-router(config-router)#no area <0-4294967295> stub

```

Configure the area to be a stub area. That is, an area where no router originates routes external to OSPF and hence an area where all external routes are via the ABR(s). Hence, ABRs for such an area do not need to pass AS-External LSAs (type-5s) or ASBR-Summary LSAs (type-4) into the area. They need only pass Network-Summary (type-3) LSAs into such an area, along with a default-route summary.

```

quagga-router(config-router)#area a.b.c.d stub no-summary
quagga-router(config-router)#area <0-4294967295> stub no-summary
quagga-router(config-router)#no area a.b.c.d stub no-summary
quagga-router(config-router)#no area <0-4294967295> stub no-summary

```

Prevents an ospfd ABR from injecting inter-area summaries into the specified stub area.

```
quagga-router(config-router)#area a.b.c.d default-cost <0-16777215>
quagga-router(config-router)#no area a.b.c.d default-cost <0-16777215>
```

Set the cost of default-summary LSAs announced to stubby areas.

```
quagga-router(config-router)#area a.b.c.d export-list NAME
quagga-router(config-router)#area <0-4294967295> export-list NAME
quagga-router(config-router)#no area a.b.c.d export-list NAME
quagga-router(config-router)#no area <0-4294967295> export-list NAME
```

Filter Type-3 summary-LSAs announced to other areas originated from intra- area paths from specified area.

```
quagga-router(config)#router ospf
quagga-router(config-router)#network 192.168.1.0/24 area 0.0.0.0
quagga-router(config-router)#network 10.0.0.0/8 area 0.0.0.10
quagga-router(config-router)#area 0.0.0.10 export-list foo
quagga-router(config)#access-list foo permit 10.10.0.0/16
quagga-router(config)#access-list foo deny any
```

With example above any intra-area paths from area 0.0.0.10 and from range 10.10.0.0/16 (for example 10.10.1.0/24 and 10.10.2.128/30) are announced into other areas as Type-3 summary-LSA's, but any others (for example 10.11.0.0/16 or 10.128.30.16/30) aren't.

This command is only relevant if the router is an ABR for the specified area.

```
quagga-router(config-router)#area a.b.c.d import-list NAME
quagga-router(config-router)#area <0-4294967295> import-list NAME
quagga-router(config-router)#no area a.b.c.d import-list NAME
quagga-router(config-router)#no area <0-4294967295> import-list NAME
```

Same as export-list, but it applies to paths announced into specified area as Type-3 summary-LSAs.

```
quagga-router(config-router)#area a.b.c.d filter-list prefix NAME in
quagga-router(config-router)#area a.b.c.d filter-list prefix NAME out
quagga-router(config-router)#area <0-4294967295> filter-list prefix NAME in
quagga-router(config-router)#area <0-4294967295> filter-list prefix NAME out
quagga-router(config-router)#no area a.b.c.d filter-list prefix NAME in
quagga-router(config-router)#no area a.b.c.d filter-list prefix NAME out
quagga-router(config-router)#no area <0-42949672> filter-list prefix NAME in
quagga-router(config-router)#no area <0-42949672> filter-list prefix NAME out
```

Filtering Type-3 summary-LSAs to/from area using prefix lists. This command makes sense in ABR only.

```
quagga-router(config-router)#area a.b.c.d authentication
quagga-router(config-router)#area <0-4294967295> authentication
quagga-router(config-router)#no area a.b.c.d authentication
quagga-router(config-router)#no area <0-4294967295> authentication
```



Specify that simple password authentication should be used for the given area.

```
quagga-router(config-router)#area a.b.c.d authentication message-digest
quagga-router(config-router)#area <0-4294967295> authentication message
```

Specify that OSPF packets must be authenticated with MD5 HMACs within the given area. Keying material must also be configured on a per-interface basis (see `ip ospf message-digest-key`).

MD5 authentication may also be configured on a per-interface basis (see `ip ospf authentication message-digest`). Such per-interface settings will override any per-area authentication setting.

### 3.5.8.5 OSPF interface

Set OSPF authentication key to a simple password. After setting `AUTH_KEY`, all OSPF packets are authenticated. `AUTH_KEY` has length up to 8 chars.

```
quagga-router(config)#interface eth0
quagga-router(config-if)#ip ospf authentication-key AUTH_KEY
quagga-router(config-if)#no ip ospf authentication-key
```

Simple text password authentication is insecure and deprecated in favor of MD5 HMAC authentication (see `ip ospf authentication message-digest`).

```
quagga-router(config-if)#ip ospf authentication message-digest
```

Specify that MD5 HMAC authentication must be used on this interface. MD5 keying material must also be configured (see `ip ospf message-digest-key`). Overrides any authentication enabled on a per-area basis (see `area authentication message-digest`).

Note that OSPF MD5 authentication requires that time never go backwards (correct time is NOT important, only that it never goes backwards), even across resets, if `ospfd` is to be able to promptly reestablish adjacency with its neighbors after restarts/reboots. The host should have system time be set at boot from an external or non-volatile source (eg battery backed clock, NTP, etc.) or else the system clock should be periodically saved to non-volatile storage and restored at boot if MD5 authentication is to be expected to work reliably.

```
quagga-router(config-if)#ip ospf message-digest-key KEYID md5 KEY Interface
quagga-router(config-if)#no ip ospf message-digest-key
```

Set OSPF authentication key to a cryptographic password. The cryptographic algorithm is MD5.

`KEYID` identifies secret key used to create the message digest. This ID is part of the protocol and must be consistent across routers on a link.

`KEY` is the actual message digest key, of up to 16 chars (larger strings will be truncated), and is associated with the given `KEYID`.

```
quagga-router(config-if)#ip ospf cost <1-65535>
quagga-router(config-if)#no ip ospf cost
```

Set link cost for the specified interface. The cost value is set to router-LSA's metric field and used for SPF calculation.

```
quagga-router(config-if)#ip ospf dead-interval <1-65535>
quagga-router(config-if)#ip ospf dead-interval minimal hello-multiplier <2-20>
quagga-router(config-if)#no ip ospf dead-interval
```

Set number of seconds for RouterDeadInterval timer value used for Wait Timer and Inactivity Timer. This value must be the same for all routers attached to a common network. The default value is 40 seconds.

If 'minimal' is specified instead, then the dead-interval is set to 1 second and one must specify a hello-multiplier. The hello-multiplier specifies how many Hellos to send per second, from 2 (every 500ms) to 20 (every 50ms). Thus one can have 1s convergence time for OSPF. If this form is specified, then the hello-interval advertised in Hello packets is set to 0 and the hello-interval on received Hello packets is not checked, thus the hello-multiplier need not be the same across multiple routers on a common link.

```
quagga-router(config-if)#ip ospf hello-interval <1-65535>
quagga-router(config-if)#no ip ospf hello-interval
```

Set number of seconds for HelloInterval timer value. Setting this value, Hello packet will be sent every timer value seconds on the specified interface. This value must be the same for all routers attached to a common network. The default value is 10 seconds.

This command has no effect if ip ospf dead-interval minimal is also specified for the interface.

```
quagga-router(config-if)#ip ospf network
quagga-router(config-if)#no ip ospf network
```

Set explicitly network type for specified interface.

```
quagga-router(config-if)#ip ospf priority <0-255>
quagga-router(config-if)#no ip ospf priority
```

Set RouterPriority integer value. The router with the highest priority will be more eligible to become Designated Router. Setting the value to 0, makes the router ineligible to become Designated Router. The default value is 1.

```
quagga-router(config-if)#ip ospf retransmit-interval <1-65535>
quagga-router(config-if)#no ip ospf retransmit interval
```

Set number of seconds for RxmtInterval timer value. This value is used when retransmitting Database Description and Link State Request packets. The default value is 5 seconds.

```
quagga-router(config-if)#ip ospf transmit-delay
quagga-router(config-if)#no ip ospf transmit-delay
```

Set number of seconds for InfTransDelay value. LSAs' age should be incremented by this value when transmitting. The default value is 1 seconds.

### 3.5.8.6 Showing OSPF information

These are the command to get information from quagga vtysh about OSPF configurations.

```
quagga-router#show ip ospf
```

Show information on a variety of general OSPF and area state and configuration information.

```
quagga-router#show ip ospf interface [INTERFACE]
```

Show state and configuration of OSPF the specified interface, or all interfaces if no interface is given.

```
quagga-router#show ip ospf neighbor
quagga-router#show ip ospf neighbor INTERFACE
quagga-router#show ip ospf neighbor detail
quagga-router#show ip ospf neighbor INTERFACE detail
quagga-router#show ip ospf database
quagga-router#show ip ospf database (asbr-summary|external|network|router)
quagga-router#show ip ospf database (asbr-summary|external|network|router)
quagga-router#show ip ospf database (asbr-summary|external|network|router)
quagga-router#show ip ospf database (asbr-summary|external|network|router)
quagga-router#show ip ospf database (asbr-summary|external|network|router)
quagga-router#show ip ospf database (asbr-summary|external|network|router)
quagga-router#show ip ospf database max-age
quagga-router#show ip ospf database self-originate
quagga-router#show ip ospf route
```

Show the OSPF routing table, as determined by the most recent SPF calculation

## 3.5.9 BGP

In this section it will explain how to access to bgpd daemon configure terminal and a list of basic commands used to start configuring routing.

### 3.5.9.1 Virtual Terminal Interfaces

Very similar how explained in the section 3.5.6.1, the only way to connect directly to this daemon is by telnetting the port 2605.

Therefore, the VTY ask the User Access Verification saved in ospfd.conf file. After this, like a real Cisco router, typing the question mark it will shows a list of command that a standard user can invoke. Then, typing the *enable* command and inserting the password is possible to access to the core to configure the virtual router

```

root@ubuntu:~# telnet localhost 2605
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^I'.

Hello, this is Quagga (version 0.99.9).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
bgpd>
  echo      Echo a message back to the vty
  enable    Turn on privileged mode command
  exit      Exit current mode and down to previous mode
  help      Description of the interactive help system
  list      Print command list
  quit      Exit current mode and down to previous mode
  show      Show running system information
  terminal   Set terminal line parameters
  who       Display who is on vty
bgpd> enable
Password:
bgpd#

```

Figure 3.8: bgpd VTY

### 3.5.9.2 Configuring BGP

BGP stands for a Border Gateway Protocol. The latest BGP version is 4. It is referred as BGP-4. BGP-4 is one of the Exterior Gateway Protocols and de-fact standard of Inter Domain routing protocol. BGP-4 is described in RFC1771, a Border Gateway Protocol 4 (BGP-4).

Many extensions have been added to RFC1771. RFC2858, Multiprotocol Extensions for BGP-4 provides multiprotocol support to BGP-4.

### 3.5.9.3 BGP router

First of all you must configure BGP router with router bgp command. To configure BGP router, you need AS number. AS number is an identification of autonomous system. BGP protocol uses the AS number for detecting whether the BGP connection is internal one or external one.

```
quagga-router(config)#router bgp asn
```

Enable a BGP protocol process with the specified asn. After this statement you can input any BGP Commands. You can not create different BGP process under different asn without specifying multiple-instance (see section Multiple instance).

```
quagga-router(config)#no router bgp asn
```

Destroy a BGP protocol process with the specified asn.

```
quagga-router(config-router)#bgp router-id A.B.C.D
```

This command specifies the router-ID. If bgpd connects to zebra it gets interface and address information. In that case default router ID value is selected as the largest IP Address of the interfaces. When router zebra is not enabled bgpd can't get interface information so router-id is set to 0.0.0.0. So please set router-id by hand.

```
quagga-router(config-router)#distance bgp <1-255> <1-255> <1-255>
```

This command change distance value of BGP. Each argument is distance value for external routes, internal routes and local routes.

```
quagga-router(config-router)#distance <1-255> A.B.C.D/M
quagga-router(config-router)#distance <1-255> A.B.C.D/M word
```

This command set distance value to

#### 3.5.9.4 BGP network

This command adds the announcement network.

```
quagga-router(config-router)#network A.B.C.D/M
```

This configuration example says that network 10.0.0.0/8 will be announced to all neighbors. Some vendors' routers don't advertise routes if they aren't present in their IGP routing tables; bgp doesn't care about IGP routes when announcing its routes.

```
quagga-router(config)#router bgp 1
quagga-router(config-router)#network 10.0.0.0/8
quagga-router(config-router)#no network A.B.C.D/M
quagga-router(config-router)#aggregate-address A.B.C.D/M
```

This command specifies an aggregate address.

```
quagga-router(config-router)#aggregate-address A.B.C.D/M as-set
```

This command specifies an aggregate address. Resulting routes include AS set.

```
quagga-router(config-router)#aggregate-address A.B.C.D/M summary-only
```

This command specifies an aggregate address. Aggregated routes will not be announce.

```
quagga-router(config-router)#no aggregate-address A.B.C.D/M
```

# Chapter 4

## Generating traffic and monitoring

### 4.1 Generating traffic

#### 4.1.1 Introduction to Generating Traffic

This section will describe what generating traffic means and his use in this project.

The next step after creating a virtual laboratory is to test that laboratory. In the previous chapter it has been discussed how to build a virtual laboratory and the tools to realize it. Once created a virtual lab it could be very useful to test it, to see what happen when there is traffic incoming.

For this use it has been necessary to generate traffic that goes throw the virtual network. The tool to realize it are called packet generator.

Packet generators create a diserate chunk of communication in a predefined format. A packet is a data block containing a header that includes destination address. All network communication that accur accross a packet-switched system trasmit packets. These packets are then reassembled by a recurring system from at the destination.

The purpose of a packet generator is to permit users or network specialist to construct a packet (e.g.: WAN packets, VOIP packets,...) from one or more specific protocol stack areas for the purpose of testing security, communication affectiveness, or source-to-destination accurency.

A packet generator or packet builder is a type of software that generates random packets or allows the user to construct detailed custom packets. This software sends specific packets out a single or multiple network interfaces.

#### 4.1.2 Iperf

##### 4.1.2.1 What Iperf is

Iperf[?] is a commonly used network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them. Iperf allows the user to set various parameters that can be used for testing a network, or alternately for optimizing or tuning a network. So it can be defined as a tool to measure bandwidth and the quality of a network link. The network link is delimited by two hosts running Iperf. Iperf has a client and server functionality,so means that the hosts of this link acts as client and server respectively. Therefore, it can measure the

throughput between the two ends, either unidirectional or bi-directional. It is open source software and runs on various platforms including Linux, Unix and Windows.

Iperf is a modern tool for network performance measurement written in C++.

The quality of a link can be tested as follows:

- Latency (response time or RTT): can be measured with the Ping command.
- Jitter (latency variation): can be measured with an Iperf UDP test.
- Datagram loss: can be measured with an Iperf UDP test.
- The bandwidth: can be measured through TCP tests.

In addition, the difference between TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) is that TCP uses processes to check that the packets are correctly sent to the receiver whereas with UDP the packets are sent without any checks but with the advantage of being quicker than TCP. Iperf uses the different capacities of TCP and UDP to provide statistics about network links.

When used for testing UDP capacity, Iperf allows the user to specify the datagram size and provides results for the datagram throughput and the packet loss.

When used for testing TCP capacity, Iperf measures the throughput of the payload. One thing to note is that Iperf uses 1024\*1024 for megabytes and 1000\*1000 for megabits.

There is a Graphical user interface (GUI) front end available called jperf, it can be associated with Iperf to provide a graphical frontend written in Java.

Typical Iperf output contains a timestamped report of the amount of data transferred and the throughput measured.

Iperf is significant as it is a standardized tool that can be run over any network and output standardized performance measurements. Thus it can be used for comparison of wired and wireless networking equipment and technologies in an unbiased way. As it is open source, the measurement methodology can be scrutinized by user.

## 4.2 Monitoring Traffic

### 4.2.1 Introduction to Monitoring Traffic

This section will explain why it is important to monitor traffic and which tool has been used to perform it.

After creating a virtual network is really useful to monitor traffic that Quagga his routing protocol have generated to detect network connections and bandwidth usages. This approach allows users to learn for example TCP/IP protocols.

The tool used to perform the monitoring is Wireshark[?]. The following section is going to describe it.

## 4.2.2 Wireshark

### 4.2.2.1 What Wireshark is

Wireshark (formerly known as Ethereal<sup>1</sup>) is a free and open-source software protocol analyzer, or “packet sniffer” application, used for network troubleshooting, analysis, software and protocol development, and education. Wireshark has all of the standard features of a protocol analyzer.

It runs on various Unix-like operating system including Linux, Mac OS, BSD , and Microsoft Windows. Wireshark is very similar to tcpdump, but has a graphical front-end, and many more information sorting and filtering options. Wireshark allows the user to see all traffic being passed over the network by putting the network interface into promiscuous mode.

## 4.3 SNMP: Simple Network Management Protocol

### 4.3.1 Introduction to SNMP

SNMP[7] is a protocol for network management. It is used for collecting information from, and configuring, network devices, such as servers, printers, hubs, switches, and routers on an Internet Protocol (IP) network. SNMP can collect information such as a server’s CPU level, Server chassis Temperature. The SNMP protocol was designed to provide a simple method of centralizing the management of TCP/IP-based networks .

SNMP[8] is based on the manager/agent model consisting of an SNMP manager, an SNMP agent, a database of management information, managed SNMP devices and the network protocol. The SNMP manager provides the interface between the human network manager and the management system. The SNMP agent provides the interface between the manager and the physical device(s) being managed.

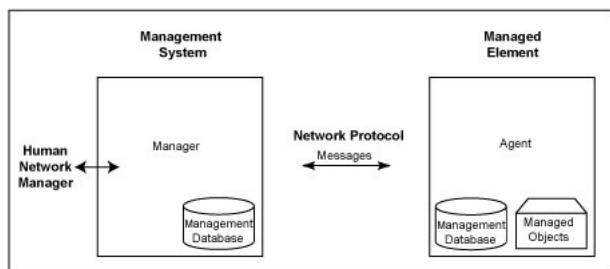


Figure 4.1: SNMP uses a manager/agent architecture. Alarm messages (Traps) are sent by the agent to the manager.

The manager and agent use a Management Information Base (MIB) and a relatively small set of commands to exchange information. The MIB is organized in a tree structure with individual variables, such as point status or description, being represented as leaves on the branches. A long numeric tag or object identifier (OID) is used to distinguish each variable uniquely in the MIB and in SNMP messages.

Snmp uses five basic messages to communicate between the manager and the agent:

<sup>1</sup>Originally named Ethereal, in May 2006 the project was renamed Wireshark due to trademark issues.



- Get.
- GetNext.
- GetResponse.
- Set and Trap.

The Get and GetNext messages allow the manager to request information for a specific variable. The agent, upon receiving a Get or GetNext message, will issue a GetResponse message to the manager with either the information requested or an error indication as to why the request cannot be processed.

A Set message allows the manager to request a change be made to the value of a specific variable in the case of an alarm remote that will operate a relay. The agent will then respond with a GetResponse message indicating the change has been made or an error indication as to why the change cannot be made.

The Trap is a change-of-state message allows the agent to spontaneously inform the manager in case of event.

This messages are issued by the SNMP manager; only the trap message is initiated by an agent.

Each snmp element manages specific objects with each object having specific characteristics. Each object/characteristic has a unique object identifier (OID) consisting of numbers separated by decimal points (e.g., 1.3.6.1.4.1.). These object identifiers naturally form a tree as shown in the below illustration. The MIB lists the unique object identifier (OID) of each managed element in an SNMP network.

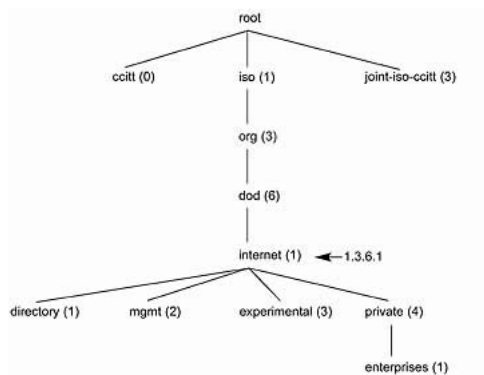


Figure 4.2: The branch of the MIB object identifier tree

When an SNMP manager wants to know the value of an object/characteristic, such as the state of an alarm point, the system name, or the element uptime, it will assemble a GET packet that includes the OID for each object/characteristic of interest. The element receives the request and looks up each OID in its code book (MIB). If the OID is found (the object is managed by the element), a response packet is assembled and sent with the current value of the object / characteristic included. If the OID is not found, a special error response is sent that identifies the unmanaged object.

UDP<sup>2</sup> is the IP transport layer protocol that supports SNMP messages. Unlike TCP, UDP is a connectionless protocol. A UDP host places messages on the network without first establishing a connection with the recipient. UDP does not guarantee message delivery, but it's a lightweight protocol that can transport a large number of status messages without using too many network resources.

### 4.3.2 SNMP and Quagga

In the chapter 3 it has been explained that Quagga acts as a router. Every router can be implemented with SNMP. Quagga itself does not support SNMP agent (server daemon) functionality but is able to connect to a SNMP agent using the SMUX protocol[9] and make the routing protocol MIBs available through it.

There are several SNMP agent which support SMUX. In this paper it will use the latest version of *net-snmp* which was formerly known as *ucd-snmp*. It is free and open software and as binary package for most Linux distributions. The *net-snmp* has to be compiled with `-with-mib-modules=smux` to be able to accept connections from Quagga.

A separate connection has then to be established between between the SNMP agent (*snmpd*) and each of the Quagga daemons. This connections each use different OID numbers and passwords. This OID number is not the one that is used in queries by clients, it is solely used for the intercommunication of the daemons.

The following OID numbers are used for the interprocess communication of *snmpd* and the Quagga daemons. SNMP has not been implemented in all daemons yet.

```
(OIDs below .iso.org.dod.internet.private.enterprises)
zebra      .1.3.6.1.4.1.3317.1.2.1 .gnome.gnomeProducts.zebra.zserv
bgpd      .1.3.6.1.4.1.3317.1.2.2 .gnome.gnomeProducts.zebra.bgpd
ripd      .1.3.6.1.4.1.3317.1.2.3 .gnome.gnomeProducts.zebra.ripd
ospfd     .1.3.6.1.4.1.3317.1.2.5 .gnome.gnomeProducts.zebra.ospfd
ospf6d    .1.3.6.1.4.1.3317.1.2.6 .gnome.gnomeProducts.zebra.ospf6d
```

Figure 4.3: OID number for communication with Quagga daemons

The following OID numbers are used for querying the SNMP daemon by a client:

```
zebra      .1.3.6.1.2.1.4.24 .iso.org.dot.internet.mgmt.mib-2.ip.ipForward
ospfd     .1.3.6.1.2.1.14 .iso.org.dot.internet.mgmt.mib-2.ospf
bgpd      .1.3.6.1.2.1.15 .iso.org.dot.internet.mgmt.mib-2.bgp
ripd      .1.3.6.1.2.1.23 .iso.org.dot.internet.mgmt.mib-2.rip2
ospf6d    .1.3.6.1.3.102 .iso.org.dod.internet.experimental.ospfv3
```

Figure 4.4: OID numbers for querying SNMP daemons

The following syntax is understood by the Quagga daemons for configuring SNMP:

```
quagga-router#smux peer oid
quagga-router#smux peer oid
quagga-router#smux peer oid password
quagga-router#no smux peer oid password
```

---

<sup>2</sup>UDP stands for User Datagram Protocol

## Chapter 5

# Configuring a virtual laboratory (Testing)

This section will show different scenario how to configure a virtual labs. There will be different case study for different level of difficult.

To perform all the labs it has been used one physical machine with the following requisites:

- Windows XP operating System
- Service Pack 3
- Pentium Dual-Core 2.60 GHz
- 2 GB Ram

### 5.1 Case Study 1: Lan with static routes

In this section it will be shown a case study where the connection among routers is allowed configuring static routing protocol.

First of all it will be described the scenario, after how to configure Quagga to work as router and then it will be tested the Lan.

#### 5.1.1 Scenario

The following figure will shows the scenario that it will be created using quagga software. In this virtual laboratory there are three routers. To do that it is necessary three virtual machines, one for each router. Notice that to perform this Lan in a real environment they are necessary three physical router, two physical switch plus four cables. In a virtual network environment only one physical PC is needed.

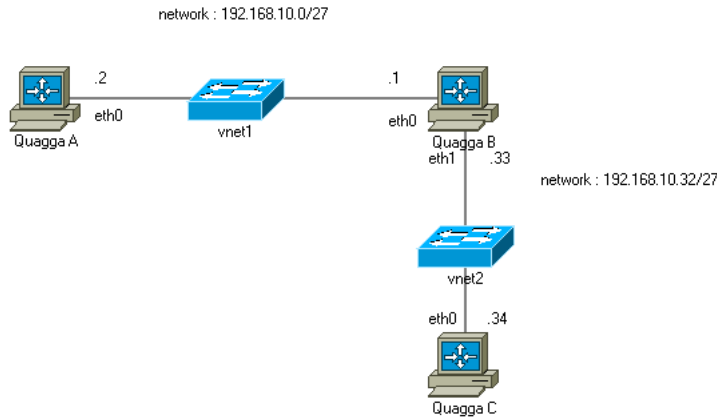


Figure 5.1: scenario with static routes

The virtual machines are created using VMware Server 2.0. They are three and the operating system running are Ubuntu Server 8.04 for Quagga A and Quagga B and CentOS for Quagga C.

Every virtual machines have one processor with 2.6 GHz, 256 MB of RAM and 8 GB of Hard Disk.

The next section will explain how to configure the virtual machines to build a Lan.

### 5.1.2 Configuration

This section will show how to configure the virtual machines to create a simple Lan. After installing the Linux operating system<sup>1</sup>, the steps to create a virtual laboratory are :

- create a VMware virtual switch and configure it
- configure Quagga

Now it's time to perform the previous two points explaining in detail what to do.

#### 5.1.2.1 How to Create VMware Virtual Switch

In this section it will be described how to create a Virtual Switch using VMware.

The Virtual Switch is set using the application Manage Virtual Switch. It is an editor that allows to create a new Network Adapter and associate it to a virtual switch.

<sup>1</sup>Quagga runs only on Linux system, so the Virtual Machines have to use Linux Operating System

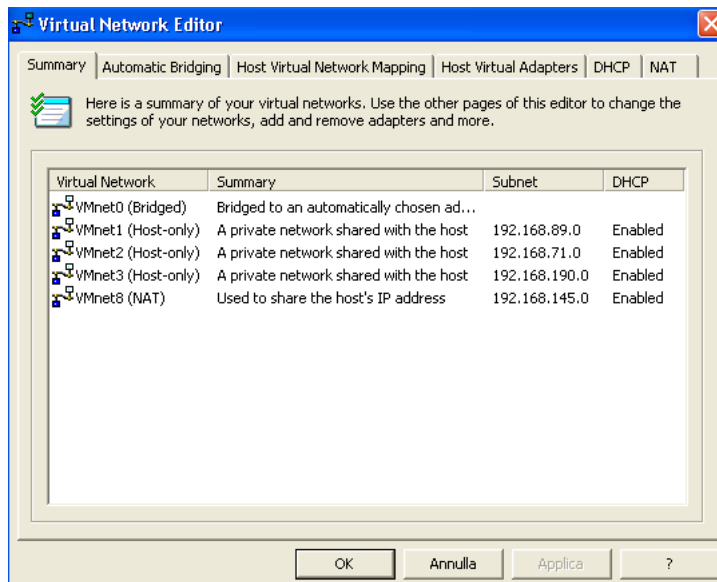


Figure 5.2: Virtual Switch Editor

To do that it is necessary to go to “Host Virtual Adapters”, click on “Add”, go to Host “Virtual Network Mapping” and associate one of the VMnet to the adapter created.

By default there are already three Virtual Switches : VMnet0, VMnet1 and VMnet8.

For this labs it has been used the VMnet1 to connect QuaggaA to QuaggaB and VMnet2 to connect QuaggaB to QuaggaC

### 5.1.2.2 How to configure Quagga

This section will start explaining how to configure Quagga.

First of all to set the hostname of the Quagga routers<sup>2</sup> it is necessary to access to the file in this way:

```
root@ubuntu:/#vim /etc/quagga/vtysh.con
```

And add this sentence to the file of each Quagga routers:

```
hostname QuaggaA
```

To configure Quagga on QuaggaA it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaA :

<sup>2</sup>The term Quagga router is used in this thesis to specify the virtual machine where quagga software is running and is working as router.

```
QuaggaA#configute terminal
QuaggaA(config)#interface eth0
QuaggaA(config-if)#ip address 192.168.10.2/27
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#interface eth1
QuaggaA(config-if)#ip address 192.168.10.65/27
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#end
QuaggaA#
```

To configure Quagga on QuaggaB it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaB :

```
QuaggaB#configute terminal
QuaggaB(config)#interface eth0
QuaggaB(config-if)#ip address 192.168.10.1/27
QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#interface eth1
QuaggaB(config-if)#ip address 192.168.10.33/27
QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#end
QuaggaB#
```

To configure Quagga on QuaggaC it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaC :

```
QuaggaC#configute terminal
QuaggaC(config)#interface eth0
QuaggaC(config-if)#ip address 192.168.10.34/27
QuaggaC(config-if)#no shutdown
QuaggaC(config-if)#link-detect
QuaggaC(config-if)#end
QuaggaC#
```

In this situation it is possible for QuaggaA to reach the interface eth0 of QuaggaB but not the interface eth1 of the same router. Moreover QuaggaB cannot reach the interface eth0 of QuaggaA but not the interface eth1 of QuaggaA. This means that each router can reach each other only if they are directly connected. Otherwise a router does not know how to reach a router not connected with him. For that reason it is necessary to set ip route static. This means that, like a Cisco router, it needs to specify in which interface forward traffic to reach another network. All this information must be set manually from network administrator, in fact a routing table of a router will not update dynamically in every change of network.

The following command will specify to Quagga A that to reach the network 192.168.10.32/27 has to forward packets to 192.168.10.1:

```
QuaggaA#configure terminal
QuaggaA(config)#ip route 192.168.10.32/27 192.168.10.1
```

The following command will specify to Quagga B that to reach the network 192.168.10.64/27 has to forward packets to 192.168.10.2:

```
QuaggaB#configure terminal
QuaggaB(config)#ip route 192.168.10.64/27 192.168.10.2
```

In this situation, on QuaggaC do not need to set any ip static routing. That because QuaggaC has only one interface and it will forward all the packet to QuaggaB.

Doing what described before allows all the router to reach each other.

Now it's useful to watch all the configuration of the routers or their routing table in case of troubleshooting. Now there will show a set of instruction to perform troubleshooting.

To see the routing table, for example of QuaggaB, this is the command:

```
QuaggaB#show ip route
```

And this is the output:

```
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.10.0/27 is directly connected, eth0
C>* 192.168.10.32/27 is directly connected, eth1
S>* 192.168.10.64/27 [1/0] via 192.168.10.2, eth0
```

Figure 5.3: Output show ip route

It is really interesting to know that in the routing table appear that QuaggaB reaches the network 192.168.10.64/27 forwarding packets through 192.168.10.2 and its configuration was set with a static configuration.

To see the status of the interfaces:

```
QuaggaB#show interface eth0
```

And this is the output:

```

Interface eth0 is up, line protocol is up
  index 2 metric 1 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:0c:29:84:6f:86
  inet 192.168.10.1/27 broadcast 192.168.10.31
  inet6 fe80::20c:29ff:fe84:6f86/64
    221 input packets (0 multicast), 18653 bytes, 0 dropped
    0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
    0 fifo, 0 missed
    121 output packets, 10986 bytes, 0 dropped
    0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartbeat
    0 window, 0 collisions

```

Figure 5.4: Output show ip interface

### 5.1.3 Testing

This section will try explain wich test has been performed to test the functionality of the case study previously described. In other words the Lan that it has been created can amend a Lan configured with real Cisco device. And this project try to ask itself if this Lan can offers features like all the other Lans created with real devices. To try to answer these questions it has used the ping utility. It tests the reachability among the virtual machine.

#### 5.1.3.1 Ping

This section will show the reachability among the virtual machines, actually among the Quagga routers using ping<sup>3</sup> utility. In other words if a Quagga router cannot reach onother router something on the its configuration or on the purpose of the project are wrong.

For that reason from each virtual machine is used to ping all the other virtual machines. First of all it will be showed the ping results of the Quagga routers directly connected.

The first test show the ping between eth0 of QuaggaA and eth0 of QuaggaB from vttysh console.

This is the command:

```
QuaggaA#ping 192.168.10.1
```

And this is the output.

```

PING 192.168.10.1 (192.168.10.1) 56(84) bytes of data:
64 bytes from 192.168.10.1: icmp_seq=1 ttl=64 time=0.208 ms
64 bytes from 192.168.10.1: icmp_seq=2 ttl=64 time=0.160 ms
64 bytes from 192.168.10.1: icmp_seq=3 ttl=64 time=0.442 ms
64 bytes from 192.168.10.1: icmp_seq=4 ttl=64 time=0.179 ms

--- 192.168.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.160/0.247/0.442/0.114 ms

```

Figure 5.5: Ping Output

<sup>3</sup>Ping is a computer network administration utility used to test the reachability of a host on an Internet Protocol (IP) network and to measure the round-trip time for messages sent from the originating host to a destination computer.



The figure above shows that the eth0 of both QuaggaA and QuaggaB can reach each other. This output establish that the connection works correctly. Moreover, it is possible to notice that the round-trip time<sup>4</sup> has 0.160 as min value, 0.247 as average value and 0.442 as max value.

The following figure will show the reachability between eth1 of QuaggaB and eth0 of QuaggaC. This is the command:

```
QuaggaB#ping 192.168.10.34
```

```
PING 192.168.10.34 (192.168.10.34) 56(84) bytes of data:
64 bytes from 192.168.10.34: icmp_seq=1 ttl=64 time=0.187 ms
64 bytes from 192.168.10.34: icmp_seq=2 ttl=64 time=0.169 ms
64 bytes from 192.168.10.34: icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from 192.168.10.34: icmp_seq=4 ttl=64 time=0.194 ms

--- 192.168.10.34 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.169/0.185/0.194/0.016 ms
```

Figure 5.6: Ping Output

The figure above shows that eth1 of QuaggaB can reach the eth0 of QuaggaC. This establish that the connection works properly.

Now it will show the ping results of the Quagga routers that are not directly connected. To allow this ping it is necessary that in someway the Quagga routers exchange among them routing information. To test it the ping will be done from eth0 of QuaggaA to eth1 to QuaggaB.

This is the command:

```
QuaggaA#ping 192.168.10.34
```

And this is the output:

```
PING 192.168.10.34 (192.168.10.34) 56(84) bytes of data:
64 bytes from 192.168.10.34: icmp_seq=1 ttl=63 time=0.622 ms
64 bytes from 192.168.10.34: icmp_seq=2 ttl=63 time=0.300 ms
64 bytes from 192.168.10.34: icmp_seq=3 ttl=63 time=0.618 ms
64 bytes from 192.168.10.34: icmp_seq=4 ttl=63 time=0.565 ms

--- 192.168.10.34 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.300/0.526/0.622/0.133 ms
```

Figure 5.7: Ping Output

The figure above shows that QuaggaA can reach QuaggaC. This means that the packets from QuaggaA are forwarded through QuaggaB and it forward these packets to QuaggaC. A further and useful tool to understand it is by traceroute<sup>5</sup>. This is the command:

```
QuaggaA#traceroute 192.168.10.34
```

<sup>4</sup>Round-trip time (RTT)[11], also called round-trip delay, is the time required for a signal pulse or packet to travel from a specific source to a specific destination and back again.

<sup>5</sup>Traceroute is a computer network tool for measuring the route path and transit times of packets across an Internet Protocol (IP) network. The traceroute tool is available on practically all Unix-like operating systems.

And this is the output:

```
tracert to 192.168.10.34 (192.168.10.34), 30 hops max, 40 byte packets
 1 (192.168.10.1)  0.986 ms  0.963 ms  0.922 ms
 2 (192.168.10.34) 4.079 ms 4.044 ms 4.031 ms
```

Figure 5.8: Traceroute output

From the figure above it is possible to notice that the packets is sent to 192.168.10.1 (QuaggaB) before reaching 192.168.10.34 (QuaggaC). This means that the static routing table has been configured correctly and it is working properly.

## 5.2 Case Study 2: Lan with RIP dynamic routing protocol

In this section it will be shown a case study where the routing is done by RIP<sup>6</sup> dynamic protocol. First of all it will be described the scenario, after how to configure Quagga to work as router and then it will be tested the Lan.

### 5.2.1 Scenario

The following picture shows the scenario that it will be created with Quagga. It is possible to notice that it is a simple LAN configuration, in fact there are only three routers and one workstation. In a real network environment<sup>7</sup> to create this Lan are necessary three physical router, three physical switch and one physical workstation plus six Ethernet cable to connect all the devices together. In a virtual network environment only one physical PC is needed.

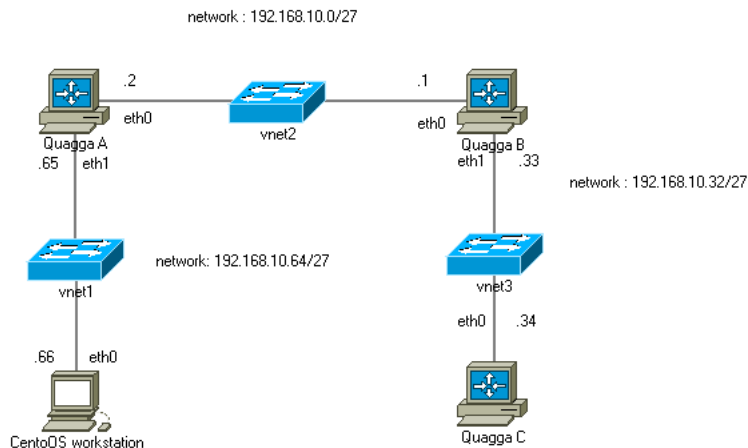


Figure 5.9: Scenario with RIP dynamic routing

<sup>6</sup>RIP is a dynamic routing protocol used in local and wide area networks.

<sup>7</sup>The concept of real and virtual network environment is explained in the Chapter 3.1

The virtual machines are created using VMware Server 2.0. They are four and the operating system running are Ubuntu Server 8.04 for Quagga A and Quagga B, CentOS for Quagga C and the Workstation.

Every virtual machines have one processor with 2.6 GHz, 256 MB of RAM and 8 GB of Hard Disk.

The next section will explain how to configure the virtual machines to build a Lan.

## 5.2.2 Configuration

This section will show how to configure the virtual machines to create a simple Lan. After installing the Linux operating system<sup>8</sup>, the steps to create a virtual laboratory are :

- create a VMware virtual switch and configure it
- configure Quagga

Now it's time to perform the previous two points explaining in detail what to do.

### 5.2.2.1 How to Create VMware Virtual Switch

The Virtual Switch are the same as explained in the Chapter 5.1.2.1.

The virtual switch VMnet1 connects the workstation with QuaggaA, VMnet2 connects QuaggaA and QuaggaB, VMnet3 QuaggaB and QuaggaC.

### 5.2.2.2 How to configure Quagga

After installed Quagga as explained in the section 3.4.2, it is going to explain how to configure Quagga to acts as a real router.

Starting with the configuration of Quagga A, it will show how to set the different routers to allow them to communicate each other.

To configure Quagga on QuaggaA it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the hostname it is necessary to execute this command:

```
quagga-router#configure terminal
quagga-router(config)#hostname QuaggaA
QuaggaA(config)#
```

To set the ip address to the router it has been executed this command:

```
QuaggaA#configure terminal
QuaggaA(config)#interface eth0
QuaggaA(config-if)#ip address 192.168.10.2/27
```

---

<sup>8</sup>Quagga runs only on Linux system, so the Virtual Machines have to use Linux Operating System

```

QuaggaA(config-if)#link-detect9
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#exit
QuaggaA(config)#interface eth1
QuaggaA(config-if)#ip address 192.168.10.65/27
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#no shutdown

```

To configure Quagga on QuaggaB it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the hostname it is necessary to execute this command:

```

quagga-router#configure terminal
quagga-router(config)#hostname QuaggaB
QuaggaB(config)#

```

To set the ip address to the router it has been executed this command:

```

QuaggaB#configure terminal
QuaggaB(config)#interface eth0
QuaggaB(config-if)#ip address 192.168.10.1/27
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#exit
QuaggaB(config)#interface eth1
QuaggaB(config-if)#ip address 192.168.10.33/27
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#no shutdown

```

To configure Quagga on Quagga C it is necessary to open the terminal, then execute this command from super user :

```
root@localhost:/#vtysh
```

This command allows to access to the terminal of the router.

To set the hostname it is necessary to execute this command:

```

quagga-router#configure terminal
quagga-router(config)#hostname QuaggaC
QuaggaC(config)#

```

To set the ip address to the router it has been executed this command:

---

<sup>9</sup>This command allows protocol detection

```
QuaggaC#configure terminal
QuaggaC(config)#interface eth0
QuaggaC(config-if)#ip address 192.168.10.34/27
QuaggaC(config-if)#link-detect
QuaggaC(config-if)#no shutdown
```

In this case it is already possible to ping QuaggaB from QuaggaC or QuaggaA to QuaggaB because they are on the same subnet and they are directly connect each other.

To ping QuaggaB from QuaggaC:

```
QuaggaC#ping 192.168.10.33
```

To ping QuaggaC from Quagga B:

```
QuaggaB#ping 192.168.10.34
```

The ping will be unsuccessful trying to ping QuaggaA<sup>10</sup> or the eth0 of QuaggaB from QuaggaC. That's because they are not on the same network and QuaggaC does not know in which interface to route traffic to reach QuaggaA. For that reason it is necessary to set a dynamic routing using RIP protocol in every router.

To set RIP on QuaggaA:

```
QuaggaA#configure terminal
QuaggaA(config)#router rip
QuaggaA(config-router)#network 192.168.10.0/27
QuaggaA(config-router)#network 192.168.10.64/27
```

To set RIP on QuaggaB:

```
QuaggaB#configure terminal
QuaggaB(config)#router rip
QuaggaB(config-router)#network 192.168.10.0/27
QuaggaB(config-router)#network 192.168.10.32/27
```

To set RIP on Quagga C:

```
QuaggaA#configure terminal
QuaggaA(config)#router rip
QuaggaA(config-router)#network 192.168.10.32/27
```

Now it is necessary to ping the interfaces of the routers to see if they can reach each other. Actually with this setting the ping test can work still if a router pings only the router directly connected. That's due to the virtual machine does not know where to route the packets.

In this situation, there are different task to solve this problem. Here a list :

---

<sup>10</sup>To ping Quagga means ping the interface (eth0, eth1,...) configured in that router

1. configure the network setting of virtual machines inserting the ip address, the netmask assigned to Quagga routers and as gateway the ip address of the interface of the router directly connected to which forward the packets<sup>11</sup>. For example in case the case of QuaggaC these are the network configuration :
  - ip address: 192.168.10.34
  - netmask: 255.255.255.224
  - default gateway: 192.168.10.33
2. add on the routing table of the linux virtual machine the exactly route where to reach other network.
3. add a static routing in which direction forward packets for the specific network

The easy way and less confusing is to perform the point 1 obtaining that the virtual interface of the virtual machine and the interface of the Quagga router are set with the same value.

Doing what described before it is possible to reach network not directly connected with a router. So for example, QuaggaC can reach QuaggaA and viceversa. In fact the QuaggaC knows that can forward the traffic to reach. This is an output of a successful ping:

```

QuaggaA# ping 192.168.10.34
PING 192.168.10.34 (192.168.10.34) 56(84) bytes of data.
64 bytes from 192.168.10.34: icmp_seq=1 ttl=63 time=12.3 ms
64 bytes from 192.168.10.34: icmp_seq=2 ttl=63 time=2.12 ms
64 bytes from 192.168.10.34: icmp_seq=3 ttl=63 time=0.554 ms
64 bytes from 192.168.10.34: icmp_seq=4 ttl=63 time=1.16 ms
64 bytes from 192.168.10.34: icmp_seq=5 ttl=63 time=0.385 ms

--- 192.168.10.34 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.385/3.309/12.320/4.546 ms

```

Figure 5.10: Output ping

It's now really useful and common to check the configuration of the routers in case of troubleshooting. The commands to perform it are similar to Cisco commands.

To see the configuration on for example Quagga B it is necessary to perform the following command :

```
QuaggaB#show run
```

This is the output:

---

<sup>11</sup>On a Linux OS server Ubuntu it is not necessary to specify any gateway. On CentOS it is necessary. It could depend of the distribution.

```

?
interface eth0
 ip address 192.168.10.1/27
 ipv6 nd suppress-ra
?
interface eth1
 ip address 192.168.10.33/27
 ipv6 nd suppress-ra
?
interface eth2
 ipv6 nd suppress-ra
?
interface lo
end
?
router rip
 network 192.168.10.0/27
 network 192.168.10.32/27
?
ip forwarding
?

```

Figure 5.11: Output show run command

To see the routing table it is necessary to perform this command.

```
QuaggaB#show ip route
```

This is the output.

```

Codes: K - kernel route, C - connected, S - static, R - RIP, D - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.10.0/27 is directly connected, eth0
C>* 192.168.10.32/27 is directly connected, eth1
R>* 192.168.10.64/27 [120/2] via 192.168.10.2, eth0, 03:19:35

```

Figure 5.12: Output show ip route command

From the picture above it is possible to notice that the networks 192.168.10.0 and 192.168.10.32 are connected to the router QuaggaB, so it can reach this network directly. The network 192.168.10.64 is known to QuaggaB via the gateway 192.168.10.2 through the interface eth0. Notice also that this information is gathered by RIP routing.

To understand further how quagga software works, it is possible to compare the route information gathered from QuaggaB with the route information gathered from the Linux virtual machine ( in this case an Ubuntu Server 8.04). This is the output entering the command `route -n` on the terminal of the virtual machine.

```

root@ubuntu:~# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.10.64 192.168.10.2 255.255.255.224 UG 2 0 0 eth0
192.168.10.32 0.0.0.0 255.255.255.224 U 0 0 0 eth1
192.168.10.0 0.0.0.0 255.255.255.224 U 0 0 0 eth0

```

Figure 5.13: Output route -n command

From the figure above it is possible to notice that the information displayed are exactly the same from QuaggaB routing table.

To check the information about interfaces it has been used the command:

```
QuaggaA#show interface eth0
```

And this is the output:

```
Interface eth0 is up, line protocol is up
  index 2 metric 1 mtu 1500
  flags: <UP,BROADCAST,RUNNING,MULTICAST>
  HWaddr: 00:0c:29:9d:86:95
  inet 192.168.10.2/27 broadcast 192.168.10.31
  inet6 fe80::20c:29ff:fe9d:8695/64
    2504 input packets (0 multicast), 320540 bytes, 0 dropped
    0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
    0 fifo, 0 missed
    3994 output packets, 353283 bytes, 0 dropped
    0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartbeat
    0 window, 0 collisions
```

Table 5.1: Output show interface command

As a Cisco router command it is possible to perform the question mark to get help from the console. Here a list of command executed in configuration terminal:

```
access-list  Add an access list entry
bgp          BGP information
debug       Debugging functions (see also 'undebug')
dump        Dump packet
enable      Modify enable password parameters
end         End current mode and change to enable mode
exit        Exit current mode and down to previous mode
hostname    Set system's network name
interface   Select an interface to configure
ip          IP information
ipv6       IPv6 information
key         Authentication key management
line        Configure a terminal line
list        Print command list
log         Logging control
no          Negate a command or set its defaults
password    Assign the terminal connection password
route-map   Create route-map or enter route-map command mode
router      Enable a routing process
router-id   Manually set the router-id
service     Set up miscellaneous service
table       Configure target kernel routing table
username
```

Figure 5.14: Output question mark

As everyone knows a Cisco router terminal has two level of authentication. Actually only connecting with the daemons by telnet it is possible to perform the same level of access. With vtysh itself does not offer any level of authentication. The login to vtysh is offered using the authentication setting used in /etc/pam.d folder of a Linux operating system. The first step is to remove or comment the line *username root nopassword* in the /etc/quagga/vtysh.conf file. This command allows user to login to vtysh without a password. The second step is to go to /etc/pam.d/quagga



file and write there the favorite level of login. One easy way is to associate the login to vtsh with the same credential of the login to the Linux virtual machine user. This is the commands to perform this task:

```
account      required      pam_unix.so
```

Writing the command above to `/etc/pam.d/quagga` file allows to write the same password of the Linux login to vtsh login. This is an example of the obtaining output:

```
root@ubuntu:/# vtsh
Password:

Hello, this is Quagga (version 0.99.9).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

QuaggaA# _
```

Figure 5.15: vtsh with authentication

### 5.2.3 Testing LAN

This section will try to test the functionality of the case study previously described. In other words the Lan that it has been created can amend a Lan configured with real Cisco device. And this project try to ask itself if this Lan can offers features like all the other Lans created with real devices. To try to answer these questions it has used some tools to test and point out the Lan. First of all it is going to check the reachability among the virtual machine using utility. Then other tools like Wireshark and Iperf are used to check deeply the functionality of the network.

#### 5.2.3.1 Ping

This section will show the reachability among the virtual machines, actually among the Quagga routers using ping utility.

For that reason from each virtual machine is used to ping all the other virtual machines. First of all it will be showed the ping results of the Quagga routers directly connected.

The first test show the ping between eth0 of QuaggaA and eth0 of QuaggaB from vtsh console.

This is the command:

```
QuaggaA#ping 192.168.10.1
```

And this is the output.

```
PING 192.168.10.1 (192.168.10.1) 56(84) bytes of data:
64 bytes from 192.168.10.1: icmp_seq=1 ttl=64 time=0.394 ms
64 bytes from 192.168.10.1: icmp_seq=2 ttl=64 time=0.195 ms
64 bytes from 192.168.10.1: icmp_seq=3 ttl=64 time=0.196 ms
64 bytes from 192.168.10.1: icmp_seq=4 ttl=64 time=0.207 ms

--- 192.168.10.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.195/0.248/0.394/0.084 ms
```

Figure 5.16: Ping Output

The figure above shows that the eth0 of both QuaggaA and QuaggaB can reach each other. This output establish that the connection works correctly. Moreover, it is possible to notice that the round-trip time has 0.195 as min value, 0.248 as average value and 0.394 as max value.

The following figure will show the reachability between eth1 of QuaggaB and eth0 of QuaggaC. This is the command:

```
QuaggaB#ping 192.168.10.34
```

```
PING 192.168.10.34 (192.168.10.34) 56(84) bytes of data:
64 bytes from 192.168.10.34: icmp_seq=1 ttl=64 time=0.229 ms
64 bytes from 192.168.10.34: icmp_seq=2 ttl=64 time=0.269 ms
64 bytes from 192.168.10.34: icmp_seq=3 ttl=64 time=0.156 ms
64 bytes from 192.168.10.34: icmp_seq=4 ttl=64 time=0.156 ms

--- 192.168.10.34 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.156/0.202/0.269/0.050 ms
```

Figure 5.17: Ping Output

The figure above shows that eth1 of QuaggaB can reach the eth0 of QuaggaC. This establish that the connection works properly.

Now it will show the ping results of the Quagga routers that are not directly connected. To allow this ping it is necessary that in someway the Quagga routers exchange among them routing information. To test it the ping will be done from eth0 of QuaggaA to eth1 to QuaggaB.

This is the command:

```
QuaggaA#ping 192.168.10.34
```

And this is the output:

```
PING 192.168.10.34 (192.168.10.34) 56(84) bytes of data:
64 bytes from 192.168.10.34: icmp_seq=1 ttl=63 time=0.632 ms
64 bytes from 192.168.10.34: icmp_seq=2 ttl=63 time=0.297 ms
64 bytes from 192.168.10.34: icmp_seq=3 ttl=63 time=0.316 ms
64 bytes from 192.168.10.34: icmp_seq=4 ttl=63 time=0.507 ms

--- 192.168.10.34 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.297/0.438/0.632/0.138 ms
```

Figure 5.18: Ping Output

The figure above shows that QuaggaA can reach QuaggaC. This means that the packets from QuaggaA are forwarded through QuaggaB and it forward these packets to QuaggaC. A further and useful tool to understand it is by traceroute. This is the command:

```
QuaggaA#traceoute 192.168.10.34
```

And this is the output:

```

tracert to 192.168.10.34 (192.168.10.34), 30 hops max, 40 byte packets
 1  (192.168.10.1)  0.300 ms  0.278 ms  0.265 ms
 2  (192.168.10.34) 0.847 ms  0.813 ms  0.794 ms

```

Figure 5.19: Traceroute output

From the figure above it is possible to notice that the packets is sent to 200.0.0.1 (QuaggaB) before reaching 10.2.0.3 (QuaggaC). This means that the RIP routing table is working properly.

### 5.2.3.2 Wireshark

Wireshark, as explained in the session 4.2.2, is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

It could be considered as a measuring device used to examine what's going on inside a network cable.

Wireshark is used in this project to establish that the virtual network enviroment created in this section works, and to evaluate its functions and if it has the requirement to be used for amending in a real enviroment network.

Wireshark helps to evaluate a virtual network enviroment because it offers the following features:

- to troubleshoot network problems
- to examine security problems
- to debug protocol implementations
- to learn network protocol internals

First of all wireshark allows to see the OSPF traffic for updating the routing table of Quagga routers. It confims that Quagga routers exchange OSPF information among them. The following picture will show the OPSF packets captured on the interface eth0 of QuaggaA.

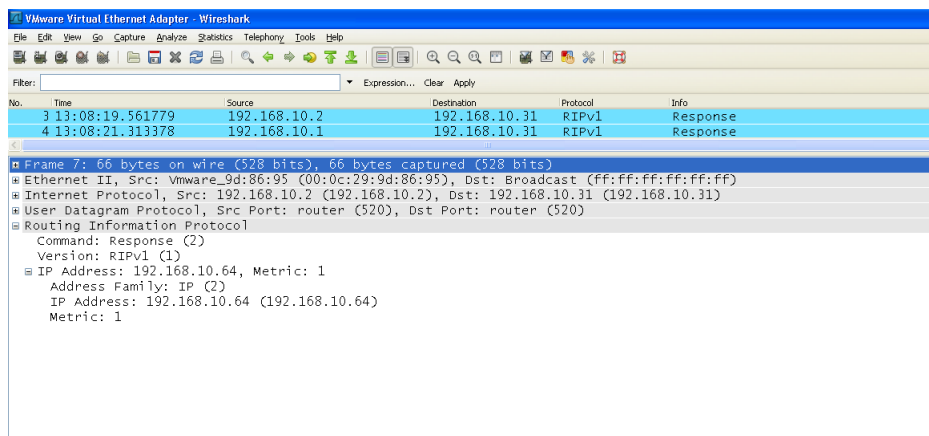


Figure 5.20: OSPF capture

This is the proof that the Quagga routers sends each other RIPv1 packets and this is the reason that allows routers not directly connected to communicate.

Exploring the packets showed in the previous figure will appear different kind of packet that QuaggaA sends to Multicast group. Actually it is possible to notice that QuaggaA (ip: 192.168.10.2) sends to broadcast information about routing table that inform the ip address 192.168.10.64 is out one of its interface.

This information establish that the Quagga routers are working like real routers.

### 5.2.3.3 Iperf

Iperf is a program for measuring throughput, jitter and datagram loss. Iperf was developed to simplify TCP performance tuning by making it easy to measure maximum throughput and bandwidth. When used with UDP, iperf can also measure datagram loss and delay (jitter). Iperf can be run over any kind of IP network, including local Ethernet LANs, Internet access links, and Wi-Fi networks. To use iPerf, it is necessary to install two components: an iPerf server (which listens for incoming test requests) and an iPerf client (which launches test sessions). Iperf is available as open source or executable binaries for many operating systems, including Win32, Linux, FreeBSD, MacOS X, OpenBSD, and Solaris.

To measure Lan performance, it has been installed Iperf on the Quagga routers. It has been tested on QuaggaA and QuaggaC.

First it is necessary to start up iperf in server mode on QuaggaA, then run it from QuaggaC in client mode with these commands from linux terminal:

```
root@QuaggaA:/# iperf -s
root@QuaggaC:/# iperf -c 200.0.0.2
```

By default, iperf clients establish a single TCP session to the iPerf server listening to Port 5001 at the specified destination.

By default, Iperf runs a 10 second test, measuring total bytes transmitted (e.g., 270 megabytes) and the resulting estimated bandwidth (e.g., 226 Mbps). Test length can be controlled by specifying time (-t seconds) or number of buffers (-n buffers). You can also view test results at regular intervals (-i seconds). To determine max TCP throughput, iPerf tries to send just data as quickly as it can from client to server. Default data is sent from an 8 KB buffer, using the operating system's default TCP window size. To mimic a specific TCP application, you can tell your iPerf client to send data from a specified file (-F filename) or enter it interactively (-I). iPerf can also be used to measure UDP datagram throughput, loss, and delay. Unlike TCP tests, UDP tests do not send traffic as quickly as possible. Instead, iPerf tries to send 1 Mbps of traffic, packaged in 1470 byte UDP datagrams (fits into one Ethernet frame). This rate can be increased by supplying a target bandwidth parameter, specified in Kbps or Mbps (-b #K or -b #M).

## 5.3 Case study: Lan with OSPF

In this section it will be shown a case study where the connection among routers is allowed configuring OSPF dynamic routing.

First of all it will be described the scenario, after how to configure Quagga to work as router and then it will be tested the Lan.

### 5.3.1 Scenario

The following figure will show the scenario that it will be created using quagga software. In this virtual laboratory there are three routers plus a workstation. To do that it is necessary three virtual machines, one for each router. Notice that to perform this Lan in a real environment they are necessary three physical router, three physical switch plus six cables. In a virtual network environment only one physical PC is needed.

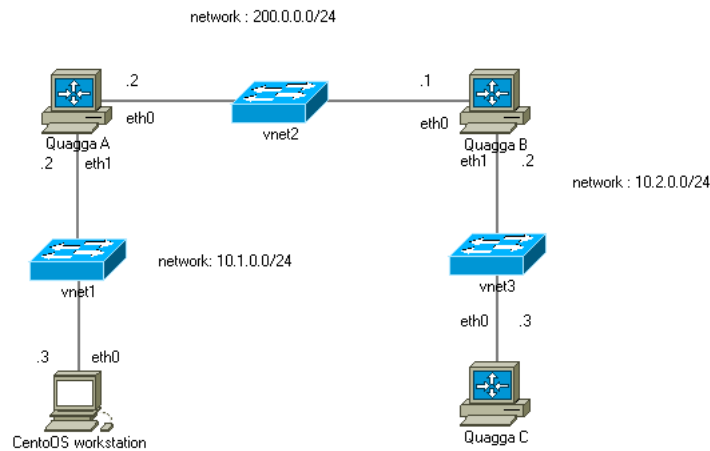


Figure 5.21: Scenario with static routes

The virtual machines are created using VMware Server 2.0. They are four and the operating system running are Ubuntu Server 8.04 for QuaggaA and QuaggaB, CentOS for QuaggaC and Workstation.

Every virtual machines have one processor with 2.6 GHz, 256 MB of RAM and 8 GB of Hard Disk.

The next section will explain how to configure the virtual machines to build a Lan.

### 5.3.2 Configuration

This section will show how to configure the virtual machines to create a simple Lan. After installing the Linux operating system<sup>12</sup>, the steps to create a virtual laboratory are :

- create a VMware virtual switch and configure it
- configure Quagga

Now it's time to perform the previous two points explaining in detail what to do.

<sup>12</sup>Quagga runs only on Linux system, so the Virtual Machines have to use Linux Operating System

### 5.3.2.1 How to Create VMware Virtual Switch

The Virtual Switch are the same as explained in the Chapter 5.1.2.1.

The virtual switch VMnet1 connects the workstation with QuaggaA, VMnet2 connects QuaggaA and QuaggaB, VMnet3 QuaggaB and QuaggaC.

### 5.3.2.2 How to configure Quagga

This section will start explaining how to configure Quagga.

First of all to set the hostname of the Quagga routers it is necessary to access to the file in this way:

```
root@ubuntu:/#vim /etc/quagga/vtysh.con
```

And add this sentence to the file of each Quagga routers:

```
hostname QuaggaA
```

To configure Quagga on QuaggaA it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaA :

```
QuaggaA#configure terminal
QuaggaA(config)#interface eth0
QuaggaA(config-if)#ip address 200.0.0.2/24
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#interface eth1
QuaggaA(config-if)#ip address 10.1.0.2/24
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#end
QuaggaA#
```

To configure Quagga on QuaggaB it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaB :

```
QuaggaB#configure terminal
QuaggaB(config)#interface eth0
QuaggaB(config-if)#ip address 200.0.0.1/24
```

```

QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#interface eth1
QuaggaB(config-if)#ip address 10.2.0.2/24
QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#end
QuaggaB#

```

To configure Quagga on QuaggaC it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaC :

```

QuaggaC#configure terminal
QuaggaC(config)#interface eth0
QuaggaC(config-if)#ip address 10.2.0.3/24
QuaggaC(config-if)#no shutdown
QuaggaC(config-if)#link-detect
QuaggaC(config-if)#end
QuaggaC#

```

In this situation it is possible for QuaggaA to reach the interface eth0 of QuaggaB but not the interface eth1 of the same router. Moreover QuaggaB cannot reach the interface eth0 of QuaggaA but not the interface eth1 of QuaggaA. This means that each router can reach each other only if they are directly connected. Otherwise a router does not know how to reach a router not connected with its. In this case study is chosen to test OSPF routing protocol. To simply the network only one area is used.

The following command will specify to Quagga A that to reach the network 10.2.0.0/24 has to forward packets to 200.0.0.1/24:

```

QuaggaA#configure terminal
QuaggaA(config)#router ospf
QuaggaA(config-router)#network 10.1.0.0/24 area 0
QuaggaA(config-router)#network 200.0.0.0/24 area 0

```

The following command will specify to Quagga B that to reach the network 10.1.0.0/24 has to forward packets to 200.0.0.2/24:

```

QuaggaB#configure terminal
QuaggaB(config)#router ospf
QuaggaB(config-router)#network 10.2.0.0/24 area 0
QuaggaB(config-router)#network 200.0.0.0/24 area 0

```

In this situation, on QuaggaC does not need to set any ip static routing. That because QuaggaC has only one interface and it will forward all the packet to QuaggaB.

Doing what described before allows all the router to reach each other.

Now it's useful to watch all the configuration of the routers or their routing table in case of troubleshooting. Now there will show a set of instruction to perform troubleshooting.

To see the routing table, for example of QuaggaB, this is the command:

```
QuaggaB#show ip route
```

And this is the output:

```
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

O>* 10.1.0.0/24 [110/20] via 200.0.0.2, eth0, 00:20:55
O 10.2.0.0/24 [110/10] is directly connected, eth1, 00:35:08
C>* 10.2.0.0/24 is directly connected, eth1
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.10.0/27 is directly connected, eth0
C>* 192.168.10.32/27 is directly connected, eth1
O 200.0.0.0/24 [110/10] is directly connected, eth0, 00:34:34
C>* 200.0.0.0/24 is directly connected, eth0
```

Figure 5.22: Output show ip route

It is really interesting to see that in the routing table above appear that for OSPF configuration the network 10.1.0.0/24 is reached via 200.0.0.2, the network 10.2.0.0/24 is directly connected and so on.

This description is really similar to any Cisco devices and clear to understand.

OSPF has further command. This is the command to see a router's neighbor:

```
QuaggaB#show ip ospf neighbor
```

And this is the output:

Neighbor	ID	Pri	State	Dead Time	Address	Interface
10.2.0.3		1	ExStart/Backup	38.860s	10.2.0.3	eth1:10.2.0.1
0	0	0				
10.1.0.2		1	Full/Backup	39.644s	200.0.0.2	eth0:200.0.0.1
0	0	0				

Figure 5.23: Output show ip ospf neighbor

To see OSPF database information summary this is the command :

```
QuaggaB#show ip ospf database
```

And this is the output:



```

OSPF Router with ID (200.0.0.1)

      Router Link States (Area 0.0.0.0)

Link ID        ADU Router      Age  Seq#          CkSum  Link count
10.1.0.2       10.1.0.2        1762 0x80000004    0x73ff 2
200.0.0.1      200.0.0.1       1793 0x80000005    0x1fd9 2

      Net Link States (Area 0.0.0.0)

Link ID        ADU Router      Age  Seq#          CkSum
200.0.0.1      200.0.0.1       1793 0x80000001    0x4ea1

```

Figure 5.24: Output ip ospf database

A very interesting command is the following:

```
QuaggaB#show ip ospf route
```

The command above shows briefly the OSPF network routing table. This is the output:

```

===== OSPF network routing table =====
N   10.1.0.0/24          [20] area: 0.0.0.0
    via 200.0.0.2, eth0
N   10.2.0.0/24          [10] area: 0.0.0.0
    directly attached to eth1
N   200.0.0.0/24        [10] area: 0.0.0.0
    directly attached to eth0

===== OSPF router routing table =====
===== OSPF external routing table =====

```

Figure 5.25: Output show ip ospf route

The following command will show statics of the different interfaces to see if a interface is up or down, the ip address, the cost, the delay, the designated router and so on.

```
QuaggaB#show ip ospf interface eth0
```

And this is the output:

```

eth0 is up
  ifindex 2, MTU 1500 bytes, BW 0 Kbit <UP,BROADCAST,RUNNING,MULTICAST>
  Internet Address 200.0.0.1/24, Broadcast 200.0.0.255, Area 0.0.0.0
  MTU mismatch detection:enabled
  Router ID 200.0.0.1, Network Type BROADCAST, Cost: 10
  Transmit Delay is 1 sec, State DR, Priority 1
  Designated Router (ID) 200.0.0.1, Interface Address 200.0.0.1
  Backup Designated Router (ID) 10.1.0.2, Interface Address 200.0.0.2
  Multicast group memberships: OSPFAllRouters OSPFDesignatedRouters
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s, Retransmit 5
  Hello due in 3.497s
  Neighbor Count is 1, Adjacent neighbor count is 1

```

Figure 5.26: Output show ip ospf interface

### 5.3.3 Testing

This section will try to test the functionality of the case study previously described. In other words the Lan that it has been created can amend a Lan configured with real Cisco device. And this project try to ask itself if this Lan can offers features like all the other Lans created with real devices. To try to answer these questions it has used some tools to test and point out the Lan. First of all it is going to check the reachability among the virtual machine using utility. Then other tools like Wireshark, Iperf and SNMP are used to check deeply the functionality of the network.

#### 5.3.3.1 Ping

This section will show the reachability among the virtual machines, actually among the Quagga routers using ping utility. In other words if a Quagga router cannot reach another router something on the its configuration or on the purpose of the project are wrong.

For that reason from each virtual machine is used to ping all the other virtual machines. First of all it will be showed the ping results of the Quagga routers directly connected.

The first test show the ping between eth0 of QuaggaA and eth0 of QuaggaB from vttysh console. This is the command:

```
QuaggaA#ping 200.0.0.1
```

And this is the output.

```
PING 200.0.0.1 (200.0.0.1) 56(84) bytes of data:
64 bytes from 200.0.0.1: icmp_seq=1 ttl=64 time=1.21 ms
64 bytes from 200.0.0.1: icmp_seq=2 ttl=64 time=0.188 ms
64 bytes from 200.0.0.1: icmp_seq=3 ttl=64 time=0.227 ms
64 bytes from 200.0.0.1: icmp_seq=4 ttl=64 time=0.192 ms

--- 200.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.188/0.454/1.210/0.436 ms
```

Figure 5.27: Ping Output

The figure above shows that the eth0 of both QuaggaA and QuaggaB can reach each other. This output establish that the connection works correctly. Moreover, it is possible to notice that the round-trip time<sup>13</sup> has 0.188 as min value, 0.454 as average value and 1.210 as max value.

The following figure will show the reachability between eth1 of QuaggaB and eth0 of QuaggaC. This is the command:

```
QuaggaB#ping 10.2.0.3
```

---

<sup>13</sup>Round-trip time (RTT)[11], also called round-trip delay, is the time required for a signal pulse or packet to travel from a specific source to a specific destination and back again.

```

PING 10.2.0.3 (10.2.0.3) 56(84) bytes of data.
64 bytes from 10.2.0.3: icmp_seq=1 ttl=64 time=0.195 ms
64 bytes from 10.2.0.3: icmp_seq=2 ttl=64 time=0.161 ms
64 bytes from 10.2.0.3: icmp_seq=3 ttl=64 time=0.140 ms
64 bytes from 10.2.0.3: icmp_seq=4 ttl=64 time=0.172 ms

--- 10.2.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.140/0.167/0.195/0.019 ms

```

Figure 5.28: Ping Output

The figure above shows that eth1 of QuaggaB can reach the eth0 of QuaggaC. This establish that the connection works properly.

Now it will show the ping results of the Quagga routers that are not directly connected. To allow this ping it is necessary that in someway the Quagga routers exchange among them routing information. To test it the ping will be done from eth0 of QuaggaA to eth1 to QuaggaB.

This is the command:

```
QuaggaA#ping 10.2.0.3
```

And this is the output:

```

PING 10.2.0.3 (10.2.0.3) 56(84) bytes of data.
64 bytes from 10.2.0.3: icmp_seq=1 ttl=63 time=0.535 ms
64 bytes from 10.2.0.3: icmp_seq=2 ttl=63 time=0.314 ms
64 bytes from 10.2.0.3: icmp_seq=3 ttl=63 time=0.379 ms
64 bytes from 10.2.0.3: icmp_seq=4 ttl=63 time=0.303 ms

--- 10.2.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.303/0.382/0.535/0.095 ms

```

Figure 5.29: Ping Output

The figure above shows that QuaggaA can reach QuaggaC. This means that the packets from QuaggaA are forwarded through QuaggaB and it forward these packets to QuaggaC. A further and useful tool to understand it is by traceroute<sup>14</sup>. This is the command:

```
QuaggaA#traceroute 10.2.0.3
```

And this is the output:

```

traceroute to 10.2.0.3 (10.2.0.3), 30 hops max, 40 byte packets
 1  (200.0.0.1)  0.286 ms  0.291 ms  0.255 ms
 2  (10.2.0.3)  1.068 ms  1.129 ms  1.197 ms

```

Figure 5.30: Traceroute output

From the figure above it is possible to notice that the packets is sent to 200.0.0.1 (QuaggaB) before reaching 10.2.0.3 (QuaggaC). This means that the OSPF routing table is working properly.

<sup>14</sup>Traceroute is a computer network tool for measuring the route path and transit times of packets across an Internet Protocol (IP) network. The traceroute tool is available on practically all Unix-like operating systems.

### 5.3.3.2 Wireshark

Wireshark, as explained in the session 4.2.2, is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

It could be considered as a measuring device used to examine what's going on inside a network cable.

Wireshark is used in this project to establish that the virtual network environment created in this section works, and to evaluate its functions and if it has the requirement to be used for amending in a real environment network.

Wireshark helps to evaluate a virtual network environment because it offers the following features:

- to troubleshoot network problems
- to examine security problems
- to debug protocol implementations
- to learn network protocol internals

First of all wireshark allows to see the OSPF traffic for updating the routing table of Quagga routers. It confirms that Quagga routers exchange OSPF information among them. The following picture will show the OPSF packets captured on the interface eth0 of QuaggaA.

No.	Time	Source	Destination	Protocol	Info
1	11:38:56.637374	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
2	11:38:56.801458	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
3	11:39:06.647599	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
4	11:39:06.810691	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
5	11:39:16.656872	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
6	11:39:16.811141	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
7	11:39:26.657283	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
8	11:39:26.820345	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
9	11:39:36.666535	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
10	11:39:36.820813	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
11	11:39:46.666993	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
12	11:39:46.831015	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
13	11:39:56.677190	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
14	11:39:56.840227	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
15	11:40:06.686391	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
16	11:40:06.840728	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
17	11:40:16.686865	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
18	11:40:16.849976	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
19	11:40:26.697148	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
20	11:40:26.850365	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
21	11:40:36.706283	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	Hello Packet
22	11:40:36.860159	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet
23	11:40:39.267537	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	LS Update
24	11:40:39.267581	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	LS Update
25	11:40:39.267618	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	LS Update
26	11:40:39.650799	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	LS Acknowledge
27	11:40:44.277271	200.0.0.1	ospf-all.mcast.nOSPF	OSPF	LS Update
28	11:40:44.670490	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	LS Acknowledge

Ethernet II, Src: 200.0.0.1 (00:0c:29:84:6f:86), Dst: IPv4mcast\_00:00:05 (01:00:5e:00:00:05)  
 Internet Protocol, Src: 200.0.0.1 (200.0.0.1), Dst: ospf-all.mcast.net (224.0.0.5)  
 Open Shortest Path First  
 OSPF Header  
 OSPF Version: 2  
 Message Type: Hello Packet (1)  
 Packet Length: 48  
 Source OSPF Router: 10.2.0.2 (10.2.0.2)  
 Area ID: 0.0.0.0 (Backbone)  
 Packet Checksum: 0x588f [correct]

Figure 5.31: OSPF capture

This is the proof that the Quagga routers sends each other OSPF *Hello Packet* and this is the reason that allows routers not directly connected to communicate.

Exploring the packets showed in the previous figure will appear different kind of packet that QuaggaA sends to Multicast group. The first series of packets are Hello packet and are showed in the next figure.

No.	Time	Source	Destination	Protocol	Info
2	11:38:56.801458	200.0.0.2	ospf-all.mcast.nOSPF	OSPF	Hello Packet

Figure 5.32: OSPF Hello Packets

The Hello packet is used to discover neighbors and form DR/BDR<sup>15</sup> relationship and exchange neighbor capabilities[12].

The next figure will show another types of packets sent from Quagga router. They are *LS Update* and *LS Acknowledge*.

<sup>15</sup>DR stands for Designated Router. BDR stands for Backup Designated Router

No.	Time	Source	Destination	Protocol	Info
25	11:40:39.267618	200.0.0.1	ospf-all.mcast.nOSPF		LS Update
26	11:40:39.650799	200.0.0.2	ospf-all.mcast.nOSPF		LS Acknowledge
27	11:40:44.277271	200.0.0.1	ospf-all.mcast.nOSPF		LS Update
28	11:40:44.670490	200.0.0.2	ospf-all.mcast.nOSPF		LS Acknowledge

Figure 5.33: LS Update and LS acknowledge

LS Update is used to send the entire LSA to the neighbor who requested the particular LSA through the link request packet. This packet is also used in flooding. LS Acknowledge is used to acknowledge the receipt of the link-state update packet.

This information establish that the Quagga routers are working like real routers.

### 5.3.3.3 Iperf

Iperf is a program for measuring throughput, jitter and datagram loss. Iperf was developed to simplify TCP performance tuning by making it easy to measure maximum throughput and bandwidth. When used with UDP, iperf can also measure datagram loss and delay (jitter). Iperf can be run over any kind of IP network, including local Ethernet LANs, Internet access links, and Wi-Fi networks. To use iPerf, it is necessary to install two components: an iPerf server (which listens for incoming test requests) and an iPerf client (which launches test sessions). Iperf is available as open source or executable binaries for many operating systems, including Win32, Linux, FreeBSD, MacOS X, OpenBSD, and Solaris.

To measure Lan performance, it has been installed Iperf on the Quagga routers. It has been tested on QuaggaA and QuaggaC.

First it is necessary to start up iperf in server mode on QuaggaA, then run it from QuaggaC in client mode with these commands from linux terminal:

```
root@QuaggaA:/# iperf -s
root@QuaggaC:/# iperf -c 200.0.0.2
```

By default, iperf clients establish a single TCP session to the iPerf server listening to Port 5001 at the specified destination.

By default, Iperf runs a 10 second test, measuring total bytes transmitted (e.g., 270 megabytes) and the resulting estimated bandwidth (e.g., 226 Mbps). Test length can be controlled by specifying time (-t seconds) or number of buffers (-n buffers). You can also view test results at regular intervals (-i seconds). To determine max TCP throughput, iPerf tries to send just data as quickly as it can from client to server. Default data is sent from an 8 KB buffer, using the operating system's default TCP window size. To mimic a specific TCP application, you can tell your iPerf client to send data from a specified file (-F filename) or enter it interactively (-I). iPerf can also be used to measure UDP datagram throughput, loss, and delay. Unlike TCP tests, UDP tests do not send traffic as quickly as possible. Instead, iPerf tries to send 1 Mbps of traffic, packaged in 1470 byte UDP datagrams (fits into one Ethernet frame). This rate can be increased by supplying a target bandwidth parameter, specified in Kbps or Mbps (-b #K or -b #M).

### 5.3.3.4 SNMP: SMUX configuration

To enable SMUX protocol support, Quagga must have been build with the `-enable-snmp` option.

A separate connection has then to be established between between the SNMP agent (snmpd) and each of the Quagga daemons. This connections each use different OID numbers and passwords. Be aware that this OID number is not the one that is used in queries by clients, it is solely used for the intercommunication of the daemons.

In the following example the ospfd daemon will be connected to the snmpd daemon using the password "quagga\_ospfd". For testing it is recommending to take exactly the below snmpd.conf as wrong access restrictions can be hard to debug.

```
/etc/snmp/snmpd.conf:
#
# example access restrictions setup
#
com2sec readonly default public
group MyROGroup v1 readonly
view all included .1 80
access MyROGroup "" any noauth exact all none none
#
# the following line is relevant for Quagga
#
smuxpeer .1.3.6.1.4.1.3317.1.2.5 quagga_ospfd

/etc/quagga/ospf:
!
!
smux peer .1.3.6.1.4.1.3317.1.2.5 quagga_ospfd
!
```

Figure 5.34: SNMP configuration

After restarting snmpd and quagga, a successful connection can be verified in the syslog and by querying the SNMP daemon:

```
snmpd[12300]: [smux_accept] accepted fd 12 from 127.0.0.1:36255
snmpd[12300]: accepted smux peer: \
oid GNOME-PRODUCT-ZEBRA-MIB::ospfd, quagga-0.96.5

# snmpwalk -c public -v1 localhost .1.3.6.1.2.1.14.1.1
OSPF-MIB::ospfRouterId.0 = IPAddress: 192.168.10.1
```

Figure 5.35: SNMP output

## 5.4 Case study 4: Advanced Lan with OSPF

In this section it will be shown a case study where will be tested the connection among Quagga routers and Cisco routers[?]. The purpose of this case study is to verify the compatibility and the effective exchange of packets between a Quagga routers and Cisco devices. This could allow to build a laboratory that consists of a communication between virtual environment and real environment.

The connections is allowed configuring OSPF dynamic routing in both Quagga routers and Cisco devices.

First of all it will be described the scenario, after how to configure Cisco routers and Quagga. Then the configuration files of the routers.

### 5.4.1 Scenario

The following figure will show the scenario. In this virtual laboratory there are two Quagga routers, a Cisco switch 3750 and a Cisco router 2651. The Cisco 3750 and QuaggaA are located in OSPF area 0, while Cisco 2651 and QuaggaB are located in area 1. Quagga A is a border router because it is in area 0 or backbone area and in area 1.

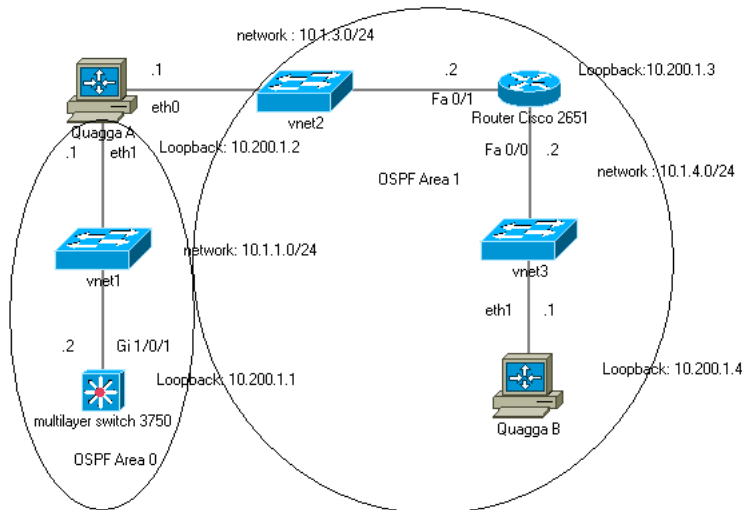


Figure 5.36: Scenario with static routes

In this lab there are two physical machines. In each machine there is configured a virtual machine. The two virtual machines are created using VMware Server 2.0. The operating system running in both machines are CentOS for QuaggaA and QuaggaB.

Every virtual machine has one processor with 2.6 GHz, 256 MB of RAM and 8 GB of Hard Disk.

The next section will explain how to configure the virtual machines and the Cisco routers to build a Lan.



## 5.4.2 Configuration

This section will show how to configure the virtual machines to create a simple Lan. After installing the Linux operating system<sup>16</sup>, the steps to create a virtual laboratory are :

- create a VMware virtual switch and configure it
- configure Quagga
- configure Cisco devices

Now it's time to perform the previous two points explaining in detail what to do.

### 5.4.2.1 How to Create VMware Virtual Switch

The Virtual Switch are created as explained in the Chapter 5.1.2.1.

The virtual switch VMnet1 connects the Cisco 3750 with QuaggaA, it's configured as Bridged Networking. This configuration allows a virtual machine to communicate with other machine using the physical NIC of the host where it is running. In this case, it allows a virtual machine like QuaggaA to link a Cisco switch. VMnet2 connects QuaggaA and QuaggaB, it's configured as Bridged Networking as well because this two virtual machine are running on two different physical machines. VMnet3 connects QuaggaB and QuaggaC using Bridged Networking for the same reasons.

### 5.4.2.2 How to configure Quagga

This section will start explaining how to configure Quagga.

First of all to set the hostname of the Quagga routers it is necessary to access to the file in this way:

```
root@ubuntu:/#vim /etc/quagga/vtysh.con
```

And add this sentence to the file of each Quagga routers:

```
hostname QuaggaA
```

To configure Quagga on QuaggaA it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaA :

```
QuaggaA#config terminal
QuaggaA(config)#interface eth0
QuaggaA(config-if)#ip address 10.1.3.1/24
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#interface eth1
```

---

<sup>16</sup>Quagga runs only on Linux system, so the Virtual Machines have to use Linux Operating System

```

QuaggaA(config-if)#ip address 10.1.1.1/24
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#interface lo
QuaggaA(config-if)#ip address 10.200.1.2/32
QuaggaA(config-if)#no shutdown
QuaggaA(config-if)#link-detect
QuaggaA(config-if)#end
QuaggaA#

```

To configure Quagga on QuaggaB it is necessary to open the terminal, then execute this command from super user :

```
root@ubuntu:/#vtysh
```

This command allows to access to the terminal of the router.

To set the ip address of QuaggaB :

```

QuaggaB#configute terminal
QuaggaB(config)#interface eth0
QuaggaB(config-if)#ip address 10.1.4.1/24
QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#interface lo
QuaggaB(config-if)#ip address 10.200.1.4/32
QuaggaB(config-if)#no shutdown
QuaggaB(config-if)#link-detect
QuaggaB(config-if)#end
QuaggaB#

```

The following command will specify OSPF networks on Quagga A :

```

QuaggaA#configure terminal
QuaggaA(config)#router ospf
QuaggaA(config-router)#network 10.1.1.0/24 area 0
QuaggaA(config-router)#network 10.1.3.0/24 area 1
QuaggaA(config-router)#network 10.200.1.2/32 area 0
QuaggaA(config-roouter)#area 1 stub

```

The following command will specify OSPF networks on Quagga B:

```

QuaggaB#configure terminal
QuaggaB(config)#router ospf
QuaggaB(config-router)#network 10.1.4.0/24 area 1
QuaggaB(config-router)#network 10.200.1.2/32 area 1
QuaggaB(config-roouter)#area 1 stub

```

Because of the stub area, the network 10.200.1.1 should not be propagated on the Quagga B router and the Cisco 2651. For this reason it is redistributed into OSPF on the Quagga A router. This is the command in QuaggaA :

```
QuaggaA#configure terminal
QuaggaA(config)#ip route 10.200.1.1/32 10.1.1.2
```

### 5.4.2.3 Configuring Cisco routers

The Cisco router 2651 is configured as showed in the following figure:

```
!
version 12.2
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Cisco-2651
!
enable secret 5 $1$9uXg$2b7DcEw/XLIusySnD9vro/
!
ip subnet-zero
!
call rsvp-sync
!
interface Loopback1
ip address 10.200.1.3 255.255.255.255
!
interface FastEthernet0/0
ip address 10.1.4.2 255.255.255.0
duplex auto
speed auto
!
interface FastEthernet0/1
ip address 10.1.3.2 255.255.255.0
duplex auto
speed auto
!
router ospf 1
log-adjacency-changes
area 1 stub
network 10.1.3.0 0.0.0.255 area 1
network 10.1.4.0 0.0.0.255 area 1
network 10.200.1.3 0.0.0.0 area 1
!
ip classless
!
!
dial-peer cor custom
!
line con 0
line aux 0
line vty 0 4
no login
!
end
```

Figure 5.37: Cisco router 2651 configuration

The Cisco switch 3750 is configured as showed in the following figure:

```

!
version 12.2
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname 3750_L3
!
enable secret 5 $1$.NvH$HJZXybguidf6xCg/6pgqL0
!
no aaa new-model
switch 1 provision ws-c3750-24ts
ip subnet-zero
ip routing
!
ip multicast-routing distributed
!
no file verify auto
spanning-tree mode pvst
spanning-tree extend system-id
!
vlan internal allocation policy ascending
!
!
interface Loopback1
ip address 10.200.1.1 255.255.255.255
!
interface FastEthernet1/0/1
!
interface FastEthernet1/0/2
!
interface FastEthernet1/0/3
!
interface FastEthernet1/0/4
!
interface FastEthernet1/0/5
!
interface FastEthernet1/0/6
!
interface FastEthernet1/0/7
!
interface FastEthernet1/0/8
!
interface FastEthernet1/0/9
!
interface FastEthernet1/0/10
!
interface FastEthernet1/0/11
!
interface FastEthernet1/0/12
!
interface GigabitEthernet1/0/1
!
interface GigabitEthernet1/0/2
!
interface Vlan1
ip address 10.1.1.2 255.255.255.0
!
router ospf 1
log-adjacency-changes
network 10.1.1.0 0.0.0.255 area 0
!
ip classless
ip http server
ip http secure-server
!
control-plane
!
line con 0
line vty 0 4
no login
line vty 5 15
no login
!
end

```

Figure 5.38: Cisco switch 3750 configuration

### 5.4.3 Testing

Now it's useful to watch all the configuration of the routers or their routing table in case of troubleshooting. Now there will show a set of instruction to perform troubleshooting. These section will demonstrate that there is connectivity between Quagga routers and Cisco devices.

From QuaggaA, this is the command to see information about interface eth0

```
QuaggaA0#show interface eth0
```

And this is the output:

```
Interface eth0 is up, line protocol is up
index 2 metric 1 mtu 1500
flags: <UP,BROADCAST,RUNNING,MULTICAST>
HWaddr: 00:02:11:22:33:99
inet 10.1.3.1/24 broadcast 10.1.3.255
inet6 fe80::201:66ff:1122:3399/64
    2598 input packets (0 multicast), 213456 bytes, 0 dropped
    0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
    0 fifo, 0 missed
    2491 output packets, 229815 bytes, 0 dropped
    0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartbeat
    0 window, 0 collisions
```

Figure 5.39: Output interface eth0

To see the routing table of QuaggaA, this is the command:

```
QuaggaB#show ip route
```

And this is the output:

```
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 10.200.1.2/32 is directly connected, lo
C>* 10.1.3.0/24 is directly connected, eth0
C>* 10.1.1.0/24 is directly connected, eth1
S>* 10.200.1.1/32 [1/0] via 10.1.1.2, eth1
O 10.1.1.0/24 [110/10] is directly connected, eth1, 03:30:30
O 10.1.3.0/24 [110/10] is directly connected, eth0, 03:30:39
O 10.200.1.2/32 [110/10] is directly connected, lo, 03:30:39
O>* 10.1.4.0/24 [110/11] via 10.1.3.2, eth0, 03:30:29
O>* 10.200.1.3/32 [110/11] via 10.1.3.2, eth0, 03:30:29
O>* 10.200.1.4/32 [110/21] via 10.1.3.2, eth0, 03:30:29
```

Figure 5.40: Output show ip route

It is really interesting to see that in the routing table above appear that for OSPF configuration the network 10.1.4.0/24 is reached via 10.1.3.2, the network 10.1.3.0/24 is directly connected and so on.

This description is really similat to any Cisco devices and clear to understand. OSPF has further command. This is the command to see OSPF information:

```
QuaggaB#show ip ospf interface
```

And this is the output:

```

eth0 is up
  ifindex 2, MTU 1500 bytes, BW 0 Kbit <UP,BROADCAST,RUNNING,MULTICAST>
  Internet Address 10.1.3.1/24, Broadcast 10.1.3.255, Area 0.0.0.1 [Stub]
  MTU mismatch detection:enabled
  Router ID 10.200.1.2, Network Type BROADCAST, Cost: 10
  Transmit Delay is 1 sec, State Backup, Priority 1
  Designated Router (ID) 10.1.4.2, Interface Address 10.1.3.2
  Backup Designated Router (ID) 10.200.1.2, Interface Address 10.1.3.1
  Multicast group memberships: OSPFAllRouters OSPFDesignatedRouters
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s, Retransmit 5
  Hello due in 6.220s
  Neighbor Count is 1, Adjacent neighbor count is 1
eth1 is up
  ifindex 3, MTU 1500 bytes, BW 0 Kbit <UP,BROADCAST,RUNNING,MULTICAST>
  Internet Address 10.1.1.1/24, Broadcast 10.1.1.255, Area 0.0.0.0
  MTU mismatch detection:enabled
  Router ID 10.200.1.2, Network Type BROADCAST, Cost: 10
  Transmit Delay is 1 sec, State DR, Priority 1
  Designated Router (ID) 10.200.1.2, Interface Address 10.1.1.1
  Backup Designated Router (ID) 10.200.1.1, Interface Address 10.1.1.2
  Multicast group memberships: OSPFAllRouters OSPFDesignatedRouters
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s, Retransmit 5
  Hello due in 5.460s
  Neighbor Count is 1, Adjacent neighbor count is 1
lo is up
  ifindex 1, MTU 16436 bytes, BW 0 Kbit <UP,LOOPBACK,RUNNING>
  Internet Address 10.200.1.2/32, Area 0.0.0.0
  MTU mismatch detection:enabled
  Router ID 10.200.1.2, Network Type LOOPBACK, Cost: 10
  Transmit Delay is 1 sec, State Loopback, Priority 1
  No designated router on this network
  No backup designated router on this network
  Multicast group memberships: <None>
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s, Retransmit 5
  Hello due in inactive
  Neighbor Count is 0, Adjacent neighbor count is 0

```

Figure 5.41: Output show ip ospf interface

To see OSPF database information summary this is the command :

```
QuaggaB#show ip ospf database
```

And this is the output:

```

OSPF Router with ID (10.200.1.2)

  Router Link States (Area 0.0.0.0)

Link ID      ADV Router   Age   Seq#           CkSum  Link count
10.200.1.1  10.200.1.1   1496  0x8000001c    0x0c47  1
10.200.1.2  10.200.1.2   67    0x80000010    0xdfa2  2

  Net Link States (Area 0.0.0.0)

Link ID      ADV Router   Age   Seq#           CkSum
10.1.1.1    10.200.1.2   67    0x80000004    0xc009

  Summary Link States (Area 0.0.0.0)

Link ID      ADV Router   Age   Seq#           CkSum  Route
10.1.3.0    10.200.1.2   909  0x80000008    0x5e0a  10.1.3.0/24
10.1.4.0    10.200.1.2   79    0x80000009    0x5b0a  10.1.4.0/24
10.200.1.3  10.200.1.2   1790 0x80000008    0x039b  10.200.1.3/32
10.200.1.4  10.200.1.2   259  0x8000000a    0x5938  10.200.1.4/32

  Router Link States (Area 0.0.0.1 [Stub])

Link ID      ADV Router   Age   Seq#           CkSum  Link count
10.1.4.2    10.1.4.2     736  0x80000010    0x8742  3
10.200.1.2  10.200.1.2   59    0x8000000c    0xe98b  1
10.200.1.4  10.200.1.4   470  0x8000000a    0x5927  2

  Net Link States (Area 0.0.0.1 [Stub])

Link ID      ADV Router   Age   Seq#           CkSum
10.1.3.2    10.1.4.2     736  0x80000007    0xea43
10.1.4.2    10.1.4.2     736  0x80000008    0xf930

  Summary Link States (Area 0.0.0.1 [Stub])

Link ID      ADV Router   Age   Seq#           CkSum  Route
0.0.0.0     10.200.1.2   179  0x80000009    0xcfb0  0.0.0.0/0
10.1.1.0    10.200.1.2   1309 0x80000007    0x94d8  10.1.1.0/24
10.200.1.2  10.200.1.2   309  0x80000009    0x1f82  10.200.1.2/32

  AS External Link States

```

Figure 5.42: Output ip ospf database

A very interesting command is the following:

```
QuaggaB#show ip ospf route
```

The command above shows briefly the OSPF network routing table. This is the output:

```

===== OSPF network routing table =====
N      10.1.1.0/24      [10] area: 0.0.0.0
      directly attached to eth1
N      10.1.3.0/24      [10] area: 0.0.0.1
      directly attached to eth0
N      10.1.4.0/24      [11] area: 0.0.0.1
      via 10.1.3.2, eth0
N      10.200.1.2/32   [10] area: 0.0.0.0
      directly attached to lo
N      10.200.1.3/32   [11] area: 0.0.0.1
      via 10.1.3.2, eth0
N      10.200.1.4/32   [21] area: 0.0.0.1
      via 10.1.3.2, eth0

===== OSPF router routing table =====
===== OSPF external routing table =====

```

Figure 5.43: Output show ip ospf route

From the Cisco device 2651 it is possible to see its routing table noticing that it has exchanged information with Quagga routers. This is the command to see the ip route:

```
CiscoRouter2651#show ip route
```

And this is the output:

```
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
        D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
        N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
        E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
        i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
        ia - IS-IS inter area, * - candidate default, U - per-user static route
        o - ODR, P - periodic downloaded static route

Gateway of last resort is 10.1.3.1 to network 0.0.0.0

    10.0.0.0/8 is variably subnetted, 6 subnets, 2 masks
C       10.1.4.0/24 is directly connected, FastEthernet0/0
C       10.1.3.0/24 is directly connected, FastEthernet0/1
C       10.200.1.3/32 is directly connected, Loopback1
O       10.200.1.4/32 [110/11] via 10.1.4.1, 02:06:38, FastEthernet0/0
O IA    10.1.1.0/24 [110/11] via 10.1.3.1, 02:06:38, FastEthernet0/1
O IA    10.200.1.2/32 [110/11] via 10.1.3.1, 02:06:38, FastEthernet0/1
O*IA   0.0.0.0/0 [110/2] via 10.1.3.1, 02:06:40, FastEthernet0/1
```

Figure 5.44: Show ip route output

The figure above shows that the Cisco router can reach the network 10.1.1.0/24 via 10.1.3.1. This means that there has been connection between Cisco router and QuaggaA.

To see OSPF database information summary this is the command :

```
CiscoRouter2651#show ip ospf database
```

And this is the output:

```
OSPF Router with ID (10.1.4.2) (Process ID 1)

  Router Link States (Area 1)

Link ID      ADV Router   Age   Seq#           Checksum Link count
10.1.4.2     10.1.4.2     1800  0x8000000D    0x008D3F 3
10.200.1.2   10.200.1.2   414   0x80000009    0x00EF88 1
10.200.1.4   10.200.1.4   821   0x80000007    0x005F24 2

  Net Link States (Area 1)

Link ID      ADV Router   Age   Seq#           Checksum
10.1.3.2     10.1.4.2     1800  0x80000004    0x00F040
10.1.4.2     10.1.4.2     1800  0x80000005    0x00FF2D

  Summary Net Link States (Area 1)

Link ID      ADV Router   Age   Seq#           Checksum
0.0.0.0      10.200.1.2   595   0x80000006    0x00D5AD
10.1.1.0     10.200.1.2   79    0x80000004    0x009AD5
10.200.1.2   10.200.1.2   1790  0x80000006    0x00257F
```

Figure 5.45: Ip ospf database output

A further and useful tool to understand it is by traceroute. This is the command:

```
QuaggaB#traceroute 10.200.1.1
```



And this is the output:

```
traceroute to 10.200.1.1 (10.200.1.1), 30 hops max, 40 byte packets
 1  10.1.4.2          0.918 ms          0.960 ms          0.951 ms
 2  10.1.3.1          0.416 ms          0.336 ms          0.340 ms
 3  10.1.1.2          0.790 ms          *                  0.834 ms
```

Figure 5.46: Traceroute output

From the figure above it is possible to notice that the packets is sent to 10.1.4.2 (Cisco router 2651), then 10.1.3.1(QuaggaA), and then again 10.1.1.2 (Cisco Switch 3750) before reaching the loopback interface 10.200.1.1 of Cisco Switch 3750. This means that the OSPF routing table is working properly.

# Chapter 6

## Conclusion

The purpose of this thesis is to build a Virtual Network Laboratory Environment. This concept has been realized with Quagga. It has reached its stage before all the components of this lab like VMware, Quagga and the tools to test it like IPerf and Wireshark. Actually it has reached that a virtual machine can act as a router if there is installed Quagga, and more Quagga routers can communicate among them using static routing or dynamic routing protocols like RIP, OSPF. Moreover, a Quagga router can communicate with a real Cisco device exchanging information. All these results allow to wonder the following consideration:

1. Virtual Network Laboratory Environment can be used from any student to exercise their networking skills.
2. Virtual Network Laboratory Environment can be used in a business system.

These two points will be discussed in the next sections.

### 6.1 Virtual Network Laboratory Environment as Learning Environment

Nowadays a student that wants to exercise and learn how to configure routers has to use the Cisco laboratory installed in a room of his/her University. Sometimes happen that the labs can be busy because other students are using it, and the free routers available are not enough to perform exercises. So the student must wait. The reasons can be different, one because a lab can host a limit of routers for space reasons and then because a router has a cost for a University. A Virtual Network Laboratory Environment allows students to create and test Cisco labs in a single physical machine. This allows students to save time and University to save money.

Using a Virtual Network Laboratory Environment a student can learn how to configure Cisco routers, he/she can also learn how to set static routing and dynamic routing protocols like RIP, OSPF and so on. A student can also learn how to perform troubleshooting for networks. For example, a teacher can use Wireshark, which allows to see in detail the packets, on the implemented lab to study TCP/IP protocols.

But it is necessary to remark that a virtual lab cannot replace a laboratory and this is not its purpose. The goal is to improve the ways a student can learn.

Once a student has performed a labs, he has available a network. This means that he could enhance his knowledge not only in networking but also in other matter. A network consists of routers, workstation but also server. So a student can for example perform a server farm or starting building a DMZ (demilitarized zone). This allows student for example to start configuring a web server or a proxy server.

The following figure could be an example of advanced laboratory:

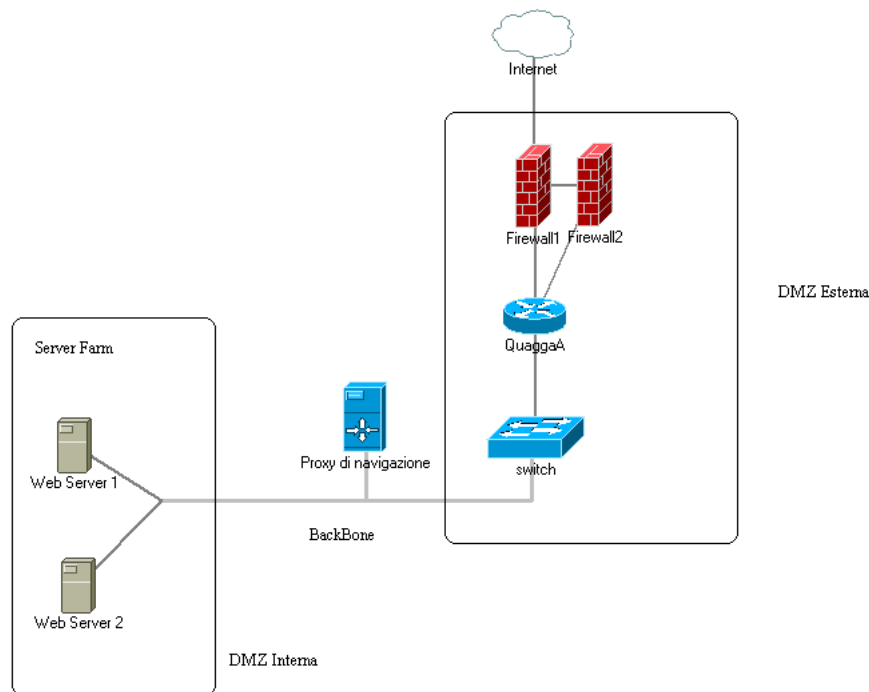


Figure 6.1: Advanced laboratory: DMZ

The figure above shows that once a student has created a network, he/she can create and configure for example a Firewall, or start to configure a web server, create some web pages using php, asp and so on. It could for example configure a proxy that manages traffic through the Internet. A student can do all these things using only one physical machine and two or three virtual machines. He could use Quagga to manage the routing and VMware to configure the servers.

Once a student has created a DMZ, it can be tested using penetration tests. He/she could perform some techniques of exploit of DB or some denial of services.

It is possible to notice that a student or a group of students can learn different levels of knowledge: he/she can start from networking, and then from server configuration, then again for web programming and at the end for penetration tests.

Actually a Virtual Network Laboratory Environment has some limits. Its performance depends on its hardware capacity. For example a virtual machine with only 250 MB of RAM cannot execute

quagga router and web server and proxy server and so on.

So the first limit of virtual machine is the hardware capacity of the physical machine.

## 6.2 Virtual Network Laboratory Environment as Working Environment

In this section, the thesis tries to wonder if a Virtual Network Laboratory Environment can help or give some advantages if used in working environment. This term is used to identify the all the possible way to implement the lab in an enterprise or a company.

Honestly, it is difficult to think that a enterprise can needs to substitute its Cisco router with Quagga routers even it can save money. The reason is the heavy amount of traffic used in a big company. It is not possible to say the same with a virtual machines yet. The main limit will be how fast a machine can forward packets. But it could be used to help to manage traffic in a virtual network in VMware Infrastructure or Xen.

Morover, processor speed can influence routing updates, routing convergence, route lookups. This is one reason why Cisco, Juniper, Foundry cost because route lookups are cached in hardware and take nanoseconds to find versus a PC which might take quite a few milliseconds.

In a small business that has a normal ADSL modem used to connect to Internet and some workstation together, a Virtual Network Laboratory Environment could be useful. If this company wants to manage its small traffic, enhance its security can implement a physical machine running for example an Ubuntu Server. On this machine can run Quagga router to manage the traffic between the workstation, and can run a Linux Firewall to protect the network and an IPS/IDS to verify what kind of packets are entering in the network.

The following figure shows an indea how Quagga could be implemented in a small business network. The roles Firewall, IPS/IDS, and Quagga router are performed from a single physical workstation.

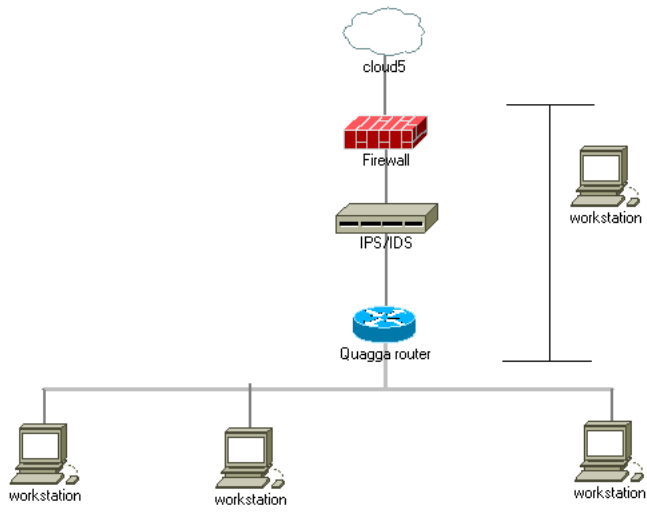


Figure 6.2: Small business network

## Chapter 7

# Future Implementation

Actually Quagga still does not support MPLS, but MPLS support is going on. In the future, could be implemented TCP/IP filtering control, QoS control, diffserv configuration to Quagga with the purpose to make a productive, quality, free TCP/IP routing software. A MPLS support could be very useful for student, which could learn how to build a MPLS network.

At the moment there is no VOIP support for Quagga router. It could be interesting for a student to configure Quagga router to support VOIP traffic, using for example same softphone for testing calls.

Recently it has been implemented the Multi-Router Looking Glass. It is a Web-based utility that can be used to display the interfaces and routes recognized by zebra. MRLG is really nothing more than a Web Interface to the zebra shell with a limited set of command. In the features it could be improved with more commands to execute by web-interface.

Today, there are no real performance comparison between Quagga router and other physical router, so it could be interesting build two different networks, ones with Quagga router and the other with for example Cisco router. Then tested the performance of the two networks to calculate the throughput, packet loss and latency.

# Bibliography

- [1] Burger T., “The Advantages of Using Virtualization Technology in the Enterprise”, 2008 <http://software.intel.com/en-us/articles/the-advantages-of-using-virtualization-technology-in-the-enterprise/>
- [2] “VMware Server 2.0 download”, [http://downloads.vmware.com/d/info/datacenter\\_downloads/vmware\\_server/2\\_0](http://downloads.vmware.com/d/info/datacenter_downloads/vmware_server/2_0)
- [3] “VMware Server User’s Guide”, Chapter 11, <http://www.vmware.com/pdf/vmserver2.pdf>
- [4] Kunihiro Ishiguro, “Quagga”, <http://www.quagga.net/docs/docs-info.php>
- [5] XJperf, “Iperf”, <http://code.google.com/p/xjperf/>
- [6] “Wireshark”, <http://www.wireshark.org/>
- [7] RFC 1157, “A Simple Network Management Protocol (SNMP)”, <http://tools.ietf.org/html/rfc1157#page-2>
- [8] DenHartog M., “The Fast Track Introduction to SNMP Alarm Monitoring”, 2008, <http://www.dpstele.com/white-papers/snmp-tutorial/index.php>
- [9] RFC1227, “SNMP MUX Protocol and MIB” , <http://www.ietf.org/rfc/rfc1227.txt>
- [10] Net-SNMP home page, <http://www.net-snmp.org/>
- [11] Definition of Round-trip time, <http://searchnetworking.techtarget.com/definition/round-trip-time>
- [12] “Troubleshooting IP Routing Protocols”, <http://cisco.iphelp.ru/faq/5/ch08lev1sec1.html>
- [13] “OSPF Advanced”, [http://openmaniak.com/quagga\\_case3.php](http://openmaniak.com/quagga_case3.php)
- [14] “Wireshark captures packets”, [http://www.wireshark.org/docs/wsug\\_html\\_chunked/ChapterIntroduction.html#ChIntroWhatIs](http://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs)