Design And Implementation Of Virtual Network Testbeds For Routing Protocols

Julius A. Bankole

BSc., University of Ibadan, (Nigeria), 1998
MSc., Beijing University of Posts & Telecommunications, (China), 2003

Project Report Submitted In Partial Fulfillment

Of The Requirements For The Degree Of

Master Of Science

in

Mathematical, Computer, And Physical Sciences

(Computer Science)

The University Of Northern British Columbia

April 2009

Abstract

In this project, we present the design and implementation of virtual network testbeds for studying routing changes. A virtual network testbed is a computer network that is completely created in software, while routing changes directly impact on the reliability and the reachability information of the network. We used testbeds to emulate a small and a large-scale network on a single Linux machine. These emulated networks allow the study of network behavior and operations which are examined using two routing protocols: Routing Information Protocol (`RIP`) and Open Shortest Path First (`OSPF`). We implemented a fifteen-node network to study `RIP`, and a model of the `GÉANT` network to examine `OSPF` in virtual network testbeds. Each testbed represents an autonomous system (`AS`) or an intra-domain environment. Therefore, these environments provided us with the opportunities to evaluate routing changes in an AS. We used the testbeds to compare the routing of the original network with the new routing of the missing links and routers to see what changes occur. The `GÉANT` network is the large-scale network used for investigations in this project. We then used our emulation results of the large-scale network to compare with the simulation work for the same network topology — the `GÉANT` network, and confirmed that our emulation studies also identified important links and routers in the same network.

Acknowledgments

First of all, I give thanks to God, the Almighty. The one who was, who is, and who is to come for preserving me throughout this academic period.

My sincere thanks and appreciation to my supervisor Dr. David Casperson for his mentoring and guidance. I really appreciate his doggedness and willingness to defy all odds and supervise me to completing this Masters' degree at UNBC. His patience, dedication to duty and attention to details have really helped me to improve my writing style. I am deeply indebted to his "touch of class".

I wish to thank members of my supervisory committee: Dr. Charles Brown and Dr. Matt Reid for reviewing my project report and their invaluable remarks. They were always there to provide me guidance and support. Their useful guidance, prompt and constructive feedback have been of tremendous help to my training at UNBC. Many thanks to Dr. Reid for helping me with technical assistance regarding experimental processes and reporting.

Most importantly, I would like to thank my darling wife Abisola and kids — Temmy and Tolu. Their love, support and belief in me never waned. They provided a pillar of strength that nurtured the environment for me to complete this M.Sc. In addition, I wish to extend gratitude to my family, friends and mentors for their help, encouragements and prayers.

Contents

List of Figures

Glossary

Abbreviations, Acronyms

AS	Autonomous System
BGP	Border Gateway Protocol
C-BGP	a BGP routing solver for large scale simulation of ASes
COW	Copy-On-Write
EGP	Exterior Gateway Protocol
GÉANT	a pan-European computer network for research and education
IGP	Interior Gateway Protocol
ISP	Internet Service Provider
OPNET	Optimized Network Engineering Tools
ICMP	Internet Control Message Protocol
OSPF	Open Shortest Path First
LAN	Local Area Network
RIP	Routing Information Protocol
SSFNET	Scalable Simulation Framework Network Models
TCP/IP	Transmission Control Protocol/Internet Protocol
UML	User Mode Linux
VELNET	Virtual Environment for Learning Networking
VNUML	Virtual Network User Mode Linux
WAN	Wide Area Network
XML	The eXtensible Markup Language

Chapter 1

Introduction

This project uses emulation techniques to investigate the impact of link and router failures on routing changes of networks. In this chapter, we explain our motivations for the study, discuss our responses to these motivations and provide an outline for this project report.

1.1 Preamble

The phenomenal growth of the Internet has led to the deployment of many network applications such as Voice or Video over IP (VoIP), electronic mail, Web browsing, e-voting, and e-shopping, to name a few. While the Internet has been designed for a best-effort service, many of these new applications and services require a better guarantee of services. End users may sometimes find the Internet service to be unreliable. There are many factors leading to this poor performance, such as link bandwidth, the efficiency of the application software, and robustness of the routing protocol. Routing protocols are a critical component of the Internet and their aim is to ensure that there is efficient traffic flow from source to destination. In this project, we use a virtualization tool to create testbeds and examine the routing changes of networks on these testbeds. This routing investigation enables network operators and researchers to gain further understanding of the routing protocols reactions to events such as traffic re-distribution and routing instability in the network. Routing instability is the rapid fluctuation of network reachability information: an important problem that directly affects the service reliability of the Internet.

The Internet is a network of networks. It is made up of a collection of over 21,000 domains or Autonomous Systems (ASs). An AS can be an Internet Service Provider (ISP), a university campus network, or a company network. An AS is made up of a collection of routers that are interconnected. Previous research has focused on the inter-connection of ASs and less attention has been paid to the intra-connection (i.e., intra-domain routing). In this project, we concentrate on intra-domain routing, and use it to study routing changes in our testbeds. The complex nature of a physical network often makes it difficult to carry out studies on how link and router changes affect the distribution of traffic across the network. Hence, we use virtual networks to emulate physical networks in this project.

This project investigates how to use virtual networks for studying routing changes in complex network environments. We use an emulation method to model routing of ASs, for a fifteen-node network and the GÉANT network — a pan-European backbone that connects Europe's national research and education networks. We study and evaluate the impact of link and router failures versus routing changes in these networks.

1.2 Motivations

In this section, we explain why the study of intra-domain routing is important, and what motivated us to carry out this particular research. There are four principal motivations: cost, networking administration training, student experimentation, and the possibility to offer particular practical advice.

Firstly, sometimes there is a need to quickly test a network configuration, e.g., a firewall rule set, but setting up the configuration on real equipment is too time consuming (e.g., physically wiring and installing multiple operating systems), and very expensive (e.g., multiple hosts and switches). We need a cheaper and more convenient testbed that can be used for this test. Therefore, we need to design virtual networks; and these networks can be used to carry out this test at little or no cost.

Secondly, students and network designers often need to obtain practical experience by learning how to design, build and maintain computer networks. CISCO offers users simulation software for this purpose, however, the experiences gained are restricted to CISCO products only. This is insufficient for a thorough grasp of the expected technical intricacies. In addition, network administration often involves activities like network addressing, assignment of routing protocols and routing table configurations. We provide a network emulation environment for conducting and testing these activities.

Thirdly, a number of situations frequently arise that require the use of more than one computer. Faculty and researchers often want to have extra full-fledged machines to aid their teaching and research work. In communities with limited funding, such as universities, the possibility of having as many full-fledged computer systems as necessary to create real networks for experimentation purposes is less likely. Therefore, creating effective virtual network testbeds will be a suitable alternative to assist faculty, researchers and other users with limited budgets, instead of investing in physical equipment.

Finally, investigating the problem of intra-domain routing in any network is very important. This is because many of the new applications and services on the Internet often demand service reliability. We use an emulation method to model a real network and evaluate the impact of changes to links and routers on the traffic distribution. In doing this, our results identify which links or routers in this network model need to be maintained. We also compare the results obtained from both simulation and emulation models of our selected network.

In the next section, we give a summary of how we address these motivations.

1.3 Contributions

In response to our motivations and the need for examining networks' routing changes, we develop two virtual networks. We use these virtual network testbeds to implement two dynamic routing protocols: Routing Information Protocol (`RIP`) and Open Shortest Path First (`OSPF`). We also provide sufficient documentation in this project report to allow prospective students and network administrators to make use of the models. In this project report, we aim to make the following contributions:

- The first contribution of this project is to develop virtual network testbeds that can be used and re-configured by students and network administrators. These testbeds will enhance learning and testing of network applications and services without requiring a real network. The designed virtual network testbeds can serve as working templates with which students can practise and modify for specific network configurations.

- The second contribution of this project is to implement a realistic network topology by emulation of the `GÉANT` network and by viewing it as an AS. The network topology of `GÉANT` is taken from the work in [4, 21]. Next, we present the techniques of how to configure routers and use the UML-utilities to implement the switches and routers on the virtual network testbed in our specification scripts. See Appendix A and Appendix C for these scripts.

- The third contribution of this project is a demonstration of a practical configuration of the routing protocol - RIP. We create and use a virtual network testbed to configure an RIP dæmon from Quagga [11]. We use the RIP dæmon to show how to detect the link failures, understand path selection using hop count, and dynamically adjust the routes.

- The fourth contribution of this project is to use case studies for investigating intra-domain routing using the OSPF dæmon from [11]. We model our network after the network used in a similar work conducted in [20, 21]. The first case study provides the measurements of link failures against the total routing cost at the head nodes of the links while the second provides the measurements of router failures against total routing cost in the GÉANT network. These case studies provide us with a better understanding of the links whose loss produce higher routing cost and the routers whose loss yields the largest total routing costs. This project report includes details of our configuration experiences, networking administration, and virtual networking experiments.

1.4 Overview of the project

The rest of the project report is organized as follows. Chapter 2 examines a summary of techniques, background information and literature review of related works that are used in this project. Chapter 3 provides reports on modelling of a simple network that is configured with the RIP routing protocol and discusses experimental results. In Chapter 4, we model the GÉANT network, conduct two case studies on this network, and provide the experimental results of our findings. Lastly, Chapter 5 presents the project report summary, our conclusions and a discussion of future work.

Chapter 2

Background and Literature Review

In this chapter, we provide background information, summary of techniques and literature review of related works that are necessary for this project. In Section 2.2, we give an overview of virtualization technologies and briefly discuss how network virtualization techniques have been used successfully in the teaching context. Section 2.3 contains the overview of the principles of User Mode Linux (`UML`) for designing virtual networks. Section 2.4 compares benefits and drawbacks of simulation and emulation techniques. In Section 2.5, we discuss different types of routing protocols that are connected to this project. Finally, Section 2.6 reviews previous research work that has been done using network virtualization techniques.

2.1 Introduction

We need to understand how virtualization technologies can support our investigations of link and router failures in the network. Virtualization techniques are often used to combine hardware and software resources, and are used to model a network for experimental purposes in this project. In addition to virtualization techniques, the principles of UML enable us to model a complex network. We make a comparison of emulation and simulation techniques, and present the major difference between the two techniques. We limit the focus of our virtualization techniques to network virtualization, and use this concept to investigate the performance of emulation techniques. The emulation techniques enable us to study routing changes when there are link and router failures in any network.

2.2 Virtualization technologies

In this section, we briefly explain the concept of virtualization in the context of computing. We also provide some examples of previous work using network virtualization techniques.

Virtualization is the term used to describe the abstraction of computer resources, and is often defined as the technique for the mapping of virtual resources to real resources. The user of the virtual resources is partially, or sometimes totally, detached from the real resources [32]. Virtualization technology hides the physical characteristics of the computing resources from the way that other systems, applications or end users communicate with those resources. Examples of various types of virtualization technologies include the following: virtual memory, redundant array of independent disks, network virtualization and storage virtualization. More on virtualization techniques can be reviewed in [2] and [32]. In this project, we limit our discussion of virtualization to network virtualization only.

Network virtualization is the technique of combining hardware and software network resources and network functionality into a single, software-based administrative entity: this is sometimes referred to as a virtual network. Network virtualization often includes platform virtualization, and occasionally combines with resource virtualization. Some previous research uses network virtualization as a tool for teaching computer networks and system administration [13, 14]. In [13, 14], Kneale *et al.* develop a tool called `VELNET`, which is a virtual environment for learning networking. `VELNET` is made up of one or more host machines and operating systems, commercial virtual machine software, virtual machines and their operating systems, and a virtual network connecting the virtual machines and remote desktop display software. Yuichiro *et al.* in [26] design a system that offers students a learning environment for LAN construction and troubleshooting. Their system reproduces virtual networks that consist of about ten Linux servers, clients, routers and switching hubs on one physical machine.

2.3 UML-based virtual networks

In this section, we explain principles and applications of `UML` in the context of networking. This `UML` technique is described as a port of a Linux kernel that allows running one or more instances of a complete Linux environment [17]. These instances are run as user-level processes on a physical host machine.

These user-level processes provide us with the virtualization of machines, routers and other nodes on a network. Within the `UML` process, an instance or a process of that `UML` communicates with the `UML` kernel which in turn talks with the host kernel in the same way that any user or application would. This `UML` technique allows a Linux kernel to be run in user space and possesses all of the features of a complete Linux machine. With `UML`, additional virtual machines or nodes can be created using the hardware of a single Linux machine. Therefore, it is possible to carry out multiple tasks and experiments on these virtual machines using a single computer system. Figure 2.1 shows the description of process space using `UML` approach.

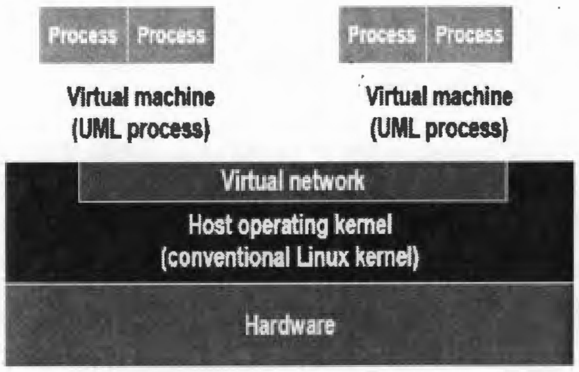

Process
Process
Process
Process
Virtual machine (UML process)
Virtual machine (UML process)
Virtual network
Host operating kernel (conventional Linux kernel)
Hardware

Figure 2.1: The architecture of the UML [9]

A virtualization tool, Virtual Networking User Mode Linux (VNUML) [8] allows us to easily create simple and complex network emulation scenarios based on UML virtualization software. The Linux machines that run over the host using UML virtualization software are called "virtual machines" or simply "UMLs".

In UML, a filesystem uses the *copy-on-write* (COW) technique to save disk space and to share a single filesystem when a number of virtual machines are run. This technique, COW, allows multiple UML processes/nodes to share a host file as a filesystem without interfering with each other's read-write operations [5]. In this mechanism, COW, the data objects are not copied until a write is made. When writing occurs, the data object is copied and non-shared afterward. Each process stores changes to the filesystem inside its own COW file. This technique allows the filesystem to be shared among all processes or virtual machines, it is also possible to revert to the original filesystem contents by simply deleting a COW file in case problems occur. Our virtualization tool, VNUML, uses COW to perform write functions while it uses the host filesystem as read-only. This COW mechanism is used in all UML-based networks in order to reduce frequent access to host memory.

With the aid of UML, virtual networks of different sizes can be created [5]. This UML technique is used to design and test networks of complex topologies and different configurations. Therefore, network designers can use the principles of UML to model virtual networks, and implement new communication protocols on these virtual networks.

2.4 Simulation versus emulation of networks

In this section, we compare experimental techniques of simulation and emulation for any network. We discuss benefits and drawbacks of these techniques.

Network designers often employ three experimental techniques in the design and validation of new and existing networking ideas. These techniques are: simulation, emulation and live network testing. All of these techniques have their strengths and weaknesses, and should not be viewed as competing methods.

Network simulation usually allows a repeatable and controlled environment for network experimentation. The simulation environments make it possible to predict outcomes of running a set of network devices on a complex network by using an internal model that is specific to the simulator. The set of initial parameters assumed for the simulators determines the model behavior of each simulation. Such environments often include simulation tools such as **OPNET** [27], **ns2** [6, 7] and **SSFNet** [3]. The fundamental drawback with simulators is that simulated devices often have limited functionalities, and the predicted behavior may not be close to that of the real system.

Network emulation reproduces features and the behavior of the real network devices. The emulation environment is made up of the software and hardware platform that provides the benefit of testing the same pieces of software that will be used on real devices. In sharp contrast to simulation systems, emulators allow the network being tested to undergo the same packet exchanges and state changes that would occur in the real world. Simulators, on the other hand, are concerned with the abstract model of the system being simulated and are often used to evaluate the performance of the protocols and algorithms.

A fundamental difference between simulation and emulation is that while the former runs in simulated time; the latter must run in real time, showing the close resemblance of the real world devices. The emulation environments closely reproduce the features and behaviors of real world devices. In an emulation environment, the network that is being tested often undergoes the same packet exchanges and state changes that usually occur in real world.

2.5 Routing in the Internet

In Section 2.3, we discussed how we can use UML to create virtual networks. In this section, we will briefly describe routing in the Internet and different types of routing protocols to regulate packets' routes in a network.

Routing is the process of determining the paths or routes that packets take on their trip from the source to the destination node. On the Internet, routing protocols are used to select the end-to-end path taken by a datagram, or packet, between the source and destination. In Chapter 1, we define the Internet as a collection of domains or ASs. Each AS is a collection of routers that are under the same administrative and technical control. An AS runs the same routing protocol among its multiple subnets. A routing algorithm within an AS is called an Interior Gateway Protocol (IGP) while an algorithm for routing between ASs is called an Exterior Gateway Protocol (EGP) [10, 15, 19, 25, 30].

Routing protocols specify how routers communicate with each other and disseminate information that allows them to select routes between any two nodes on a network.

Readers who are not familiar with routing protocols are encouraged to read [10, 15, 19, 25, 30].

2.5.1 Intra-AS routing: RIP

In this section, we explain one of the intra-AS protocols. The Routing Information Protocol (RIP) is the earliest intra-AS routing protocol. This RIP protocol uses a "hop" count as a cost metric, which is the term used to describe the number of subnets traversed along the shortest path from the source router to the destination subnet. The maximum cost of a path in RIP is fifteen. This number limits the use of RIP to smaller ASs. This protocol, RIP, is a distance vector protocol based on the Bell-Ford algorithm [10], and is based on a shortest path computation. A distance-vector routing protocol requires that a router periodically inform its directly attached neighbors of topology changes, and perform a routing calculation. The result of the calculation is distributed back to the attached neighbors. The primary goal of this protocol, like other intra-AS protocols, is to find the shortest path to the chosen destination based on a selected metric.

Normally, each router has a RIP table often called a routing table. Routers use their routing tables to decide the next hop to which they should forward a packet. The routers configured with this protocol, RIP, exchange advertisements approximately every thirty seconds. If a router fails to hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable; that is, it is either the neighbor (router) has died or the link has gone down. When this occurs, RIP modifies the local routing table and then propagates this information by sending advertisements to neighboring routers that are still reachable.

2.5.2 Intra-AS routing: OSPF

Similar to the previous section, here we discuss another intra-AS protocol: Open Shortest Path First (OSPF). This protocol is the successor to RIP, and was developed to handle limitations of RIP. This routing protocol, OSPF uses cost as the routing metric, and uses link-state information that is based on the Dijkstra least-cost algorithm [10]. This algorithm computes the shortest path to all subnets based on cost and selects the source node as the root node for cost computation. The network administrator assigns a cost to each link. The OSPF protocol floods the network with link state advertisements (LSAs), unlike RIP where a node only exchanges information with its neighbors. At periodic intervals, OSPF protocols use a "HELLO" message to check whether the routers are operational or not. This protocol is also a dynamic routing protocol.

Each router periodically sends an LSA across the network. This message is sent to provide information on a router's adjacencies or to update others when a router's state changes. By comparing adjacencies to link states, failed routers can be detected quickly, and the network's topology can be updated appropriately. From the topological database generated from LSAs, each router calculates a shortest-path tree, with itself as root. The shortest-path tree, in turn, yields a routing table.

2.5.3 Inter-AS routing: BGP

Here, we briefly explain an inter-AS protocol. Border Gateway Protocol (BGP) is the routing protocol for interconnecting different ASs. This protocol, BGP, is a path vector protocol, and does not use traditional IGP metrics. It makes routing decisions based on path, network policies and/or rule sets. This BGP protocol maintains a table of IP networks or "prefixes" which show network reachability among ASs. Because the

Internet is made up of a collection of ASs and it is used everywhere, BGP is critical to the proper functioning of the Internet. This BGP protocol is the core routing protocol of the Internet.

2.6 Related works

In this section, we present a review of some work on network virtualization. From the literature [9, 16, 18, 22–24, 28], some of the previous research concentrates on providing the concepts and implementation methods of virtualization, while some focus on producing commercial software.

Liu *et al.* [16] and Ham *et al.* [28] discuss the concepts and implementation approaches for designing a virtual network testbed. Both [16] and [28] only provide good insight regarding the concepts and implementation methods for virtualization without providing necessary hands-on learning experiences.

Massino uses an emulator called `Netkit` in his PhD thesis [22, 23] to study inter-domain routing policies on a network. Mottola uses a virtualization approach in his PhD thesis [18] to study simulation of mobile ad hoc networks. This approach is used for testing publish-subscribe middleware on mobile ad-hoc networks. Steffen *et al.* also use a UML-based network to set up an environment for automated software regression tests [24]. Software regression tests are carried out before the release of new official software to eliminate bugs during software development, hence Steffen *et al.* design an automated testing framework for the regression tests. Similarly, Galan *et al.* use virtualization techniques to design and implement an IP multimedia subsystem testbed [9]. This testbed is used for development and functional validation of multimedia services for next generation networks.

Previous work on network routing protocols for some ISPs can be found in [21, 29]. In [21], Quoitin *et al.* develop an open source routing solver, C-BGP. This is an efficient solver for BGP and is used for exchanging routing information across domains in the Internet. This solver can be used with large-scale topologies to predict the effect of link and router failures in an AS. C-BGP is also used by ISP operators for conducting case studies on the routing information collected in their network. C-BGP is used to collect the BGP updates for the work on page 16 of [21] and in Section 2.5.9 of [20]. It is also used to study inter domain traffic engineering techniques and to model the network of ISPs.

In [29], Watson *et al.* conduct an experimental study of an operational OSPF network for a period of one year. This network is a mid-size regional ISP that is running an intra-domain routing protocol, OSPF. The network is characterized by routing instability, different traffic levels and changes in the routing updates. They find out that the information from external routing protocols leads to significant levels of instability within OSPF.

Clearly, there is a substantial interest in network virtualization for purposes that range from testing new protocols, configuring networks, studying routing changes and traffic distributions to experimenting with new network designs.

Chapter 3

Modelling RIP Routing

This chapter discusses the design, description of the implementation, and results of our experimental studies with RIP on a fifteen-node virtual network testbed. In Section 3.1, we present an overview, some background information, and the problem. Section 3.3 describes how to model a network, explains its implementation and configurations, and defines how to validate a virtual network. Section 3.4 consists of our experimental studies and results. Section 3.5 contains the limitation of RIP and conclusions drawn from our experimental studies.

3.1 Introduction

Network simulation and emulation have been indispensable tools for understanding the performance of network systems. In this project, we focused on the use of emulation testbeds for studying various types of networking environments. We designed a testbed to demonstrate that emulation techniques produce reasonable results in a small network. In our experiments, we tested the reliability of RIP to dynamically learn and fill the routing table with a route to all subnets in the network. This feature of RIP routing protocol allowed us to examine routing changes in the network caused by link and router failures.

Routing changes affect the network reachability information, and are such an important problem that it directly affects the service reliability of AS. While much research has been conducted on inter-domain routing, the study of intra-domain routing has been quite limited. Inter-domain routing simply refers to the routing of inter-connected networks, while intra-domain routing refers to the routing within a network or an ISP. Most network operators do not have sufficient understanding of this problem. Some network operators often complain that they do not know to what extent intra-domain protocol can cause changes in their networks. There is a lack of understanding of the causes of these routing changes because it is difficult to detect in their live networks.

In our efforts to investigate this problem, we use an intra-domain protocol, RIP, to determine how routing is performed within a virtual network testbed. We explained briefly the concepts of intra-AS routing in Section 2.5.1.

In this chapter, our first goal is to use an emulation technique to design a fifteen-node virtual network testbed. The second goal is to use a routing protocol, RIP, to configure a small network and use this network to understand how a datagram or packet efficiently travels from source to destination on the testbed. The third goal is to use our virtual network testbed to investigate routing changes caused by link and router failures.

Most especially, the purpose of this set of experiments is to determine how sensitive a network is to link and router failures, which is critical to understanding the reliability of a network.

3.2 Experimental setup

In this section, we describe hardware and software components of the computer system that was used to design the virtual network testbed for this project. All the experiments were performed on testbeds that are built on a Toshiba Satellite Pro P300 notebook. This notebook is an Intel®Core, consists of a dual-processor P4250 running at 1.5GHz, has 32KB/32KB L1 Cache and 3MB L2 Cache. The RAM is 2GB DDR2 running at 667MHz and a 250.0 billion bytes S-ATA disk.

The computer system is a dual-booting type with pre-installed Windows™Vista and Ubuntu 7.10 operating systems. We modified a kernel of Ubuntu 7.10 by patching it with **skas3** for better performance. Next, we installed `VNUML 1.8` [8] on the modified host kernel of Ubuntu 7.10 for easy creation, execution, and release of virtual networking scenarios. The typical use of `VNUML` consists of: step 1 to create the scenarios, step 2 to execute commands as many times as desired or needed, and step 3 to release or destroy the scenarios. More information on the use of `VNUML` can be obtained from [8].

3.3 Modelling of a fifteen-node virtual network

In this section, we explain necessary steps to model and test a virtual network. We also discuss methodologies for implementing the topology of a virtual network testbed. Finally, we validate this virtual network testbed by testing for network connectivity on a `RIP`-configured testbed. This validation is done to verify whether or not `RIP` is functioning properly on the testbed.

3.3.1 Topology of the virtual network

We describe the topology of our virtual network testbed in this section. In order to build a network or a virtual network, we found it useful to produce a detailed map representing the network before proceeding to write the configuration files.

Firstly, we designed the topology of our virtual network to consist of a total of fifteen nodes that include nine virtual routers, six host machines, and eighteen links or sub-networks. Secondly, we assigned appropriate IP addresses to the routers and the links, and subsequently proceeded to write configuration files for each router in the network. We selected this network topology to demonstrate that emulation techniques produce reasonable results in a small network where the expected results are known. Figure 3.1 shows a detailed map of the implemented network topology.

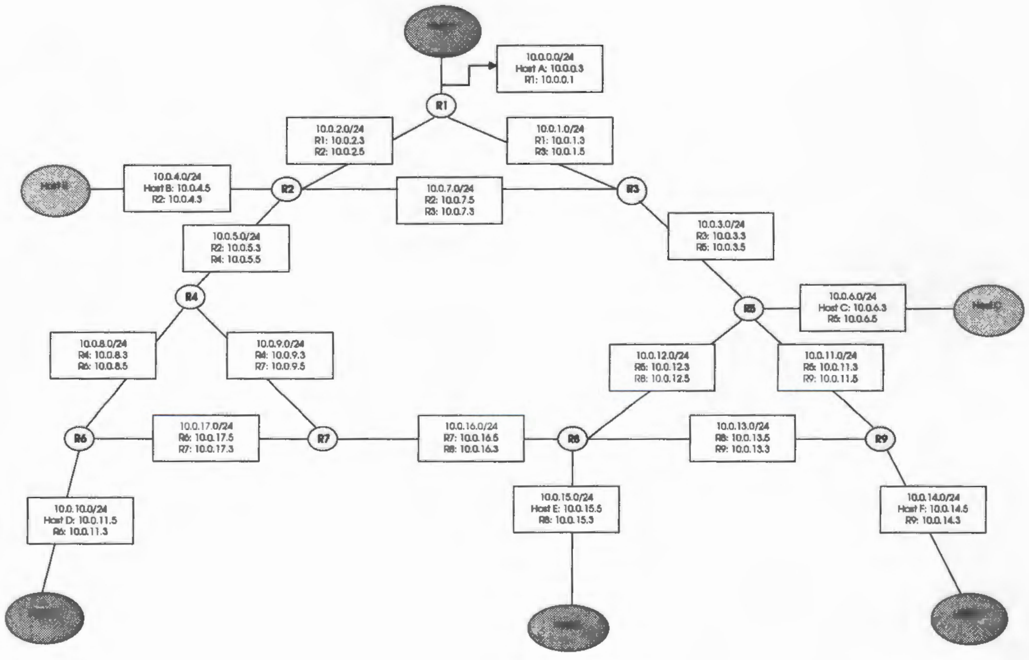


Figure 3.1: A fifteen-node network topology

3.3.2 Implementation and configuration with RIP

Before we conducted our experiments for this project, there were two fundamental tasks that we needed to carry out for preparing an enabling environment on a virtual network testbed. The first task was to create the network, and the second task was to configure each router in the network. In this section, we discuss an implementation of the network topology described in Section 3.3.1 using VNUML, and describe how we used VNUML to create networking scenarios. We also explain how we used Quagga to configure each router in the networking scenarios with a dynamic routing protocol. Quagga is open source, and is obtained from [11]. Finally, we set up and produced a fifteen-node virtual network testbed that was configured with a dynamic routing protocol, RIP. Below are the basic steps for the implementation and configuration of the fifteen-node virtual network.

1. We installed the VNUML tool on the LINUX environment of the host machine. This tool and the installation procedures can be downloaded from [8]. The VNUML tool is designed to easily create simple and complex networking scenarios.

2. Next, we installed Quagga in the system-wide `/etc/` directory of the host machine. Quagga is a routing software package that provides TCP/IP-based routing services and protocol dæmons. A machine installed with Quagga served as a dedicated router.

3. We encoded the network topology specified in Figure 3.1 in an XML file. The purpose of this file was to include specifications for creating a fifteen-node virtual network testbed. We ensured that the XML file specifications conformed to the VNUML DTD that comes with the VNUML tool. Details of the XML specifications are included in Appendix A.

4. We then created the VNUML session and individual machines by running the

commands in Figure 3.2. When we were finished with the networking scenario, we killed the scenario processes by running the commands specified in Figure 3.3. A screen shot of a fifteen-node virtual network testbed is shown in Figure 3.4. Each of the windows or machines in Figure 3.4 represents a node on the virtual network testbed.

5. The network created in step four had strictly local connectivity, but this particular network ignored the global network topology. This type of connectivity means that only adjacent routers could communicate with each other. To globally connect the network, we then configured each router in the network with `RIP` by creating these files: `zebra.conf`, `ripd.conf` and `vtysh.conf` in `/etc/quagga` directory. These three configuration files were created and designated for each router. A sample of each configuration file for router, R1, is included in Appendix B. See Appendix B for more details.

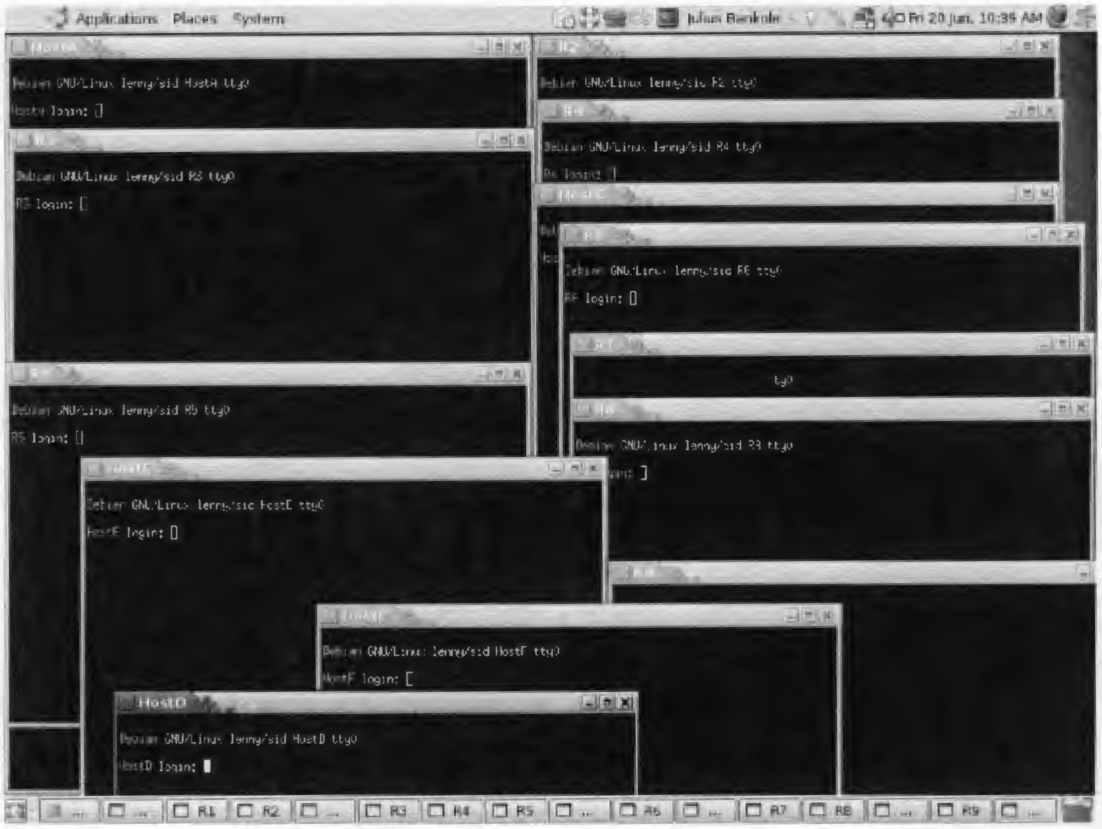
6. We then started and stopped the `RIP` dæmon by running commands as shown in Figure 3.5. A piece of code from `XML` specifications for starting and stopping the `ripd` dæmon is shown in Figure 3.6. See the `XML` file in Appendix A for more details. For the purpose of our experiments as described in Section 3.4, we verified that each router could connect to the local host. We achieved this goal by running the command, `telnet localhost ripd`, on each router to confirm that a telnet session was possible from each router. The telnet session for the router-R1 console is shown in Figure 3.7.

```
vnumlparser.pl -t /usr/share/vnuml/RIP15nodes.xml -v -u root
```

Figure 3.2: Commands for creating a virtual network for a fifteen-node topology.

```
vnumlparser.pl -d /usr/share/vnuml/RIP15nodes.xml -v
```

Figure 3.3: Commands for releasing a network scenario for a fifteen-node topology.



```
Debian GNU/Linux lenny/sid HostA tty0
HostA login:

Debian GNU/Linux lenny/sid R2 tty0

Debian GNU/Linux lenny/sid R4 tty0
R4 login:

Debian GNU/Linux lenny/sid R3 tty0
R3 login:

Debian GNU/Linux lenny/sid R6 tty0
R6 login:

tty0

Debian GNU/Linux lenny/sid R5 tty0
R5 login:

Debian GNU/Linux lenny/sid R8 tty0
login:

Debian GNU/Linux lenny/sid HostE tty0
HostE login:

Debian GNU/Linux lenny/sid HostF tty0
HostF login:

Debian GNU/Linux lenny/sid HostD tty0
HostD login:
```

Figure 3.4: Screen shot of a fifteen-node virtual network testbed

```
sudo vnumlparser.pl -x start@RIP15nodes.xml # Starting
```

```
sudo vnumlparser.pl -x stop@RIP15nodes.xml  # Stopping
```

Figure 3.5: Commands for starting and stopping RIP protocols

3.3.3 Validating the virtual network

The validation test is used to confirm that there is network connectivity in the virtual network. Without a routing protocol, a router knows neighbors or routes that are directly connected to it. When we configured the testbed with the RIP routing

```
    <filetree root="/etc/quagga" seq="start">R1</filetree>
    .....
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ripd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
```

Figure 3.6: XML code for starting and stopping **zebra** and **ripd** dæmons

```
R1:\~# telnet localhost ripd
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.7).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification
Password:(zebra)

ripd>
```

Figure 3.7: An example of a telnet session with the ripd dæmon.

protocol, each router could obtain the routing information of distant neighbors. Each router contains a RIP table known as a routing table. The routing table has three main columns among others: the first is the destination subnet, the second is the gateway or identity of the next router along the shortest path to the destination subnet and the third indicates the "metric" or the number of hops to the destination. An example of a routing table is discussed and shown in Section 3.4.1.

We used the ping command to test whether or not a particular host was reachable across the virtual network. A computer network tool, ping, is used to test whether a particular host is reachable across an IP network and to self test the network interface card of the router. The ping command sends an ICMP echo request to the stated destination address and the TCP/IP software at the destination then replies to the **ping** echo request packet with a similar packet, called an ICMP echo reply. If the network is connected and functional, it reports the number of packets transmitted, the percentage of packet loss, and the round-trip time. If the network is not connected, the **ping** command replies that the "Network is unreachable". The **ping** command estimates the round-trip time in milliseconds, records packet loss, and displays a statistical summary when it is finished.

When we tested router R1 in our virtual network with the **ping** command, we obtained the results as shown in Figure 3.8. The results from this test displayed information about the network and confirmed that the **ping** command was working. In the first case, the nodes that were not immediate neighbors were not reachable when RIP protocol was not running in the network. In the second case there was a global network connectivity when the network was configured with RIP protocol. An example of a session testing for connectivity from a router, R1 console to a distant Host F is shown in Figure 3.8.

```
                        pinging: R1 console
R1:~# ping 10.0.14.5 -c1 #Host F without RIP
connect: Network is unreachable

R1:~# ping 10.0.14.5 -c1 #Host F with RIP
PING 10.0.14.5 (10.0.14.5) 56(84) bytes of data.
64 bytes from 10.0.14.5: icmp_seq=1 ttl=61 time=0.589 ms

--- 10.0.14.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.589/0.589/0.589/0.000 ms
```

Figure 3.8: Pinging from R1 to Host F

In this section, we carried out the fundamental steps necessary to confirm that that we have successfully created a network testbed, before conducting further experiments. Most especially, XML code has to be specified in such a way that routers can be started and stopped easily without affecting the networking scenario in the VNUML session. At this point, our virtual network was validated, and we could proceed.

3.4 Experiments on a fifteen-node network testbed

In this section, we describe two experiments on the fifteen-node virtual network testbed. The goal of the first experiment was to use an emulation technique to understand routing changes in a small network for single-link failures. The goal of the second experiment was to use the same emulation technique to study routing changes for single-router failures in the same network. We will use the information obtained from these experiments as a background preparation towards our project's primary goal of making comparison for a large scale network using emulation and simulation technologies in Chapter 4.

3.4.1 Single-link failures with `RIP`

In this experiment, we investigated the effects of single-link failures on the virtual network testbed. We observed routing changes in the routing tables when single-link failures occurred.

We emulated all the single-link failures in the network, and observed the effect of the failures on each router configured with `RIP` in the network. We removed each link in the network sequentially, and recorded routing changes at the head node of a link in the network. Several tests were conducted by disabling each link in the network for this experiment and routing changes were recorded. This experiment was performed by specifying a command in each router as follows — `R1:~# ifconfig eth1 down`. Interfaces on the links for each router could be `eth0`, `eth1`, `eth2` and so on. The removal of each interface has a corresponding effect on the updates of the routing table for each router in the network.

We recorded the number of routing entries generated in these networking scenarios. This recording was carried out by running a network command **route** directly on each router to collect routing entries in the routing table. This command displays all the **RIP** routes in the network. We collected results from the routing tables for each router in the network. Examples of the results obtained from one of the experiments when the network was fully functional, and when an interface **eth3** of link R2-R4 was removed (before the network restabilized) are shown in Figure 3.9 and Figure 3.10. We then compared the results of routing changes in both cases obtained from their respective routing tables — the fully functioning network, and the missing links network scenarios before restabilizations — to see what changes occur.

```
R2 console
Case 1: Full Links with 19 routing entries in the network.
R2:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
192.168.0.8     *               255.255.255.252 U     0      0     0 eth0
10.0.4.0        *               255.255.255.0   U     0      0     0 eth2
10.0.5.0        *               255.255.255.0   U     0      0     0 eth3
10.0.6.0        10.0.7.3        255.255.255.0   UG    3      0     0 eth4
10.0.7.0        *               255.255.255.0   U     0      0     0 eth4
10.0.16.0       10.0.5.5        255.255.255.0   UG    3      0     0 eth3
10.0.0.0        10.0.2.3        255.255.255.0   UG    2      0     0 eth1
10.0.17.0       10.0.5.5        255.255.255.0   UG    3      0     0 eth3
10.0.1.0        10.0.2.3        255.255.255.0   UG    2      0     0 eth1
10.0.2.0        *               255.255.255.0   U     0      0     0 eth1
10.0.3.0        10.0.7.3        255.255.255.0   UG    2      0     0 eth4
10.0.12.0       10.0.7.3        255.255.255.0   UG    3      0     0 eth4
10.0.13.0       10.0.7.3        255.255.255.0   UG    4      0     0 eth4
10.0.15.0       10.0.7.3        255.255.255.0   UG    4      0     0 eth4
10.0.8.0        10.0.5.5        255.255.255.0   UG    2      0     0 eth3
10.0.9.0        10.0.5.5        255.255.255.0   UG    2      0     0 eth3
10.0.10.0       10.0.5.5        255.255.255.0   UG    3      0     0 eth3
10.0.11.0       10.0.7.3        255.255.255.0   UG    3      0     0 eth4
```

Figure 3.9: Case 1: Routing table for router R2 with full links — 19 routing entries

```
R2 console
Case 2: Link removal before network restabilized.
R2:~# ifconfig eth3 down # R2-R4 link removed.
R2:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
192.168.0.8     *               255.255.255.252 U     0      0     0 eth0
10.0.4.0        *               255.255.255.0   U     0      0     0 eth2
10.0.6.0        10.0.7.3        255.255.255.0   UG    3      0     0 eth4
10.0.7.0        *               255.255.255.0   U     0      0     0 eth4
10.0.0.0        10.0.2.3        255.255.255.0   UG    2      0     0 eth1
10.0.1.0        10.0.2.3        255.255.255.0   UG    2      0     0 eth1
10.0.2.0        *               255.255.255.0   U     0      0     0 eth1
10.0.3.0        10.0.7.3        255.255.255.0   UG    2      0     0 eth4
10.0.12.0       10.0.7.3        255.255.255.0   UG    3      0     0 eth4
10.0.14.0       10.0.7.3        255.255.255.0   UG    4      0     0 eth4
10.0.11.0       10.0.7.3        255.255.255.0   UG    3      0     0 eth4
```

Figure 3.10: Case 2 — Routing table for R2 before network restabilized — 12 routing entries

For explanation purposes, we selected one of the links — link R2-R4 — to illustrate the outcome of our single-link failure analysis. For further analysis, we can select any other link in the network to explain effects of single-link failures.

When the virtual network was fully functional in case one, the expected outcome is to obtain nineteen routing entries at the head node of the link from this network. However, when we experimented with the removal of the selected link R2-R4, the number of routing entries changed from nineteen to eleven. This result clearly shows that the removal of link R2-R4 leads to a 37% decrease in the number of routing entries. The summarized results of these experiments are shown graphically in Figure 3.11. In Figure 3.11, the links of the virtual network are shown on the x-axis while the head node routing changes in the network are shown on the y-axis.

In most instances, it is observed that a single-link failure caused many changes in the routing tables. About 50% of the links in this fifteen-node network will cause about

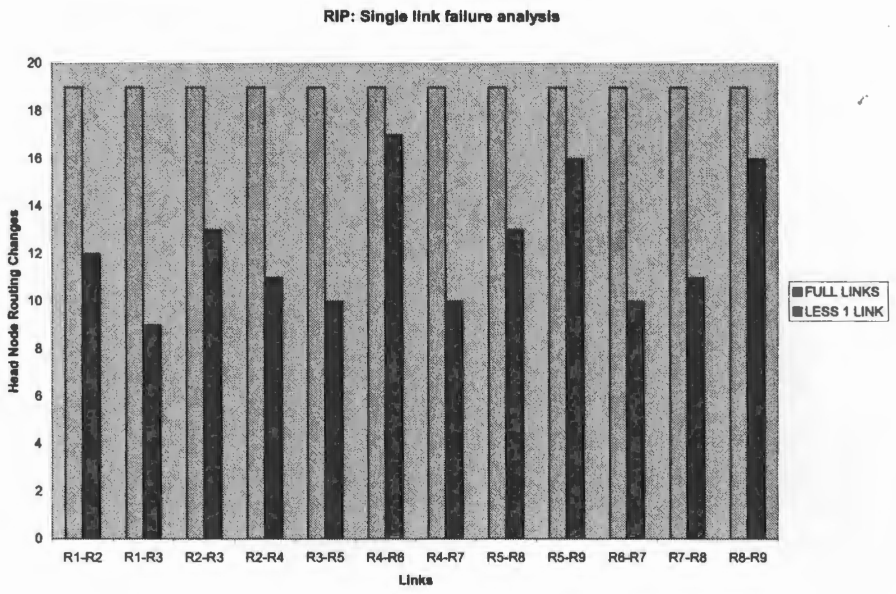


Figure 3.11: Single link removal analysis for RIP

a 42% decrease in the number of routing changes. These observations demonstrate that when the link connectivity of the network goes down, RIP will modify the local routing table and record the routing entries for the routers that are still connected in the network. This result conforms to the concepts explained regarding intra-AS routing and matches our hypothesis in Section 2.5.1.

3.4.2 Single-router failures with RIP

In this experiment, we investigated the impact of single-router failures in the virtual network. We recorded routing changes in the network as shown in the routing tables for single-router failures.

The concepts and implementations of RIP routing protocol explain that if a router does not hear from its neighbor at least once in every 180 seconds, that neighbor is considered to be no longer reachable [15]. This outcome means the neighbor is dead or the connectivity is lost. Therefore, RIP modifies the local routing table and propagates this information by sending advertisements to neighboring routes that are still reachable.

Firstly, we recorded the number of routing entries when all the routers were in place in the network and the network was fully functional. Secondly, we removed each router in the network, sequentially, then recorded the resulting routing changes for each router. When a particular router was missing, we collected and recorded the routing entries from each of the routers that were still active in the network. We compared the routing entries collected from the routing tables of the original network, with routing entries of the modified network, and reported changes observed for each missing router.

For our original and unmodified network, each router had nineteen routing entries in their routing table. When we modified the network by removing each router sequentially, each router produced had eighteen or nineteen routing entries depending on whether the router is a cut point. We now provide more detail.

Firstly, for routers — R3, R4, and R7 — there was no difference in their routing tables both before and after their removal. This outcome occurred because each router considers its neighbor dead after 180 seconds and adjusts its routes based on the next available router. As an example, we show the results for R5 in Figure 3.12 for when the virtual network is fully functioning and in Figure 3.13 for when router R3 was missing from the network.

Secondly, we observed only slight difference of one missing routing entry in these routers: R1, R2, R5, R6, R8, and R9. This difference occurred because these routers were cut points of the network topology shown in Figure 3.1. As an example, we show the results for R5 in Figure 3.12 for when the virtual network is fully functioning and in Figure 3.14 for when router R1 was missing from the network. The loss of one routing entry was as a result of network non-reachability due to a cut point on the network topology. Otherwise, the behaviors of all these routers were the same, and it reflected that effects of missing routers were not important in the RIP routers.

```
R5 console
R5:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
192.168.0.24    *               255.255.255.252 U     0      0      0 eth0
10.0.4.0        10.0.12.5       255.255.255.0   UG    5      0      0 eth4
10.0.5.0        10.0.12.5       255.255.255.0   UG    4      0      0 eth4
10.0.6.0        *               255.255.255.0   U     0      0      0 eth2
10.0.7.0        10.0.3.3        255.255.255.0   UG    2      0      0 eth1
10.0.16.0       10.0.12.5       255.255.255.0   UG    2      0      0 eth4
10.0.0.0        10.0.3.3        255.255.255.0   UG    3      0      0 eth1
10.0.17.0       10.0.12.5       255.255.255.0   UG    3      0      0 eth4
10.0.1.0        10.0.3.3        255.255.255.0   UG    2      0      0 eth1
10.0.2.0        10.0.3.3        255.255.255.0   UG    3      0      0 eth1
10.0.3.0        *               255.255.255.0   U     0      0      0 eth1
10.0.12.0       *               255.255.255.0   U     0      0      0 eth4
10.0.13.0       10.0.11.5       255.255.255.0   UG    2      0      0 eth3
10.0.14.0       10.0.11.5       255.255.255.0   UG    2      0      0 eth3
10.0.15.0       10.0.12.5       255.255.255.0   UG    2      0      0 eth4
10.0.8.0        10.0.12.5       255.255.255.0   UG    4      0      0 eth4
10.0.9.0        10.0.12.5       255.255.255.0   UG    3      0      0 eth4
10.0.10.0       10.0.12.5       255.255.255.0   UG    4      0      0 eth4
10.0.11.0       *               255.255.255.0   U     0      0      0 eth3
```

Figure 3.12: Routing table for the router-R5 with fully functional network

In summary, we obtained results obtained for the removal of each router. These results are summarized graphically in Figure 3.15. In Figure 3.15, the routers of the virtual network are shown on the x-axis while the number of routing changes in the network are shown on the y-axis. From the obtained results for a single-router failure analysis, we observed that a network configured with RIP always produced the same number of routing table entries at each router when a particular router was removed. The

```
                                        R5 console
R5:~# route (missing router R3)
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
192.168.0.20    *               255.255.255.252 U     0      0      0 eth0
10.0.4.0        10.0.12.5       255.255.255.0   UG    5      0      0 eth4
10.0.5.0        10.0.12.5       255.255.255.0   UG    4      0      0 eth4
10.0.6.0        *               255.255.255.0   U     0      0      0 eth2
10.0.7.0        10.0.12.5       255.255.255.0   UG    5      0      0 eth4
10.0.16.0       10.0.12.5       255.255.255.0   UG    2      0      0 eth4
10.0.0.0        10.0.12.5       255.255.255.0   UG    6      0      0 eth4
10.0.17.0       10.0.12.5       255.255.255.0   UG    3      0      0 eth4
10.0.1.0        10.0.12.5       255.255.255.0   UG    6      0      0 eth4
10.0.2.0        10.0.12.5       255.255.255.0   UG    5      0      0 eth4
10.0.3.0        *               255.255.255.0   U     0      0      0 eth1
10.0.12.0       *               255.255.255.0   U     0      0      0 eth4
10.0.13.0       10.0.12.5       255.255.255.0   UG    2      0      0 eth4
10.0.14.0       10.0.11.5       255.255.255.0   UG    2      0      0 eth3
10.0.15.0       10.0.12.5       255.255.255.0   UG    2      0      0 eth4
10.0.8.0        10.0.12.5       255.255.255.0   UG    4      0      0 eth4
10.0.9.0        10.0.12.5       255.255.255.0   UG    3      0      0 eth4
10.0.10.0       10.0.12.5       255.255.255.0   UG    4      0      0 eth4
10.0.11.0       *               255.255.255.0   U     0      0      0 eth3
```

Figure 3.13: Routing tables when the router-R3 was missing

slight difference in the routing table entries was because that particular router was a cut point of the network graph. This observation conforms to the concepts explained regarding dead neighbors or routers, and matches our hypothesis in Section 2.5.1.

3.5 Conclusions

In this section, we explain our conclusions for the experiments conducted in the small network with RIP configurations. We use the two experiments that we performed on single-link and single-router failures in Section 3.4.

The first experiment enabled us to study the routing changes caused by the removal of links in a virtual network configured with RIP routing protocol. When we exper-

```
R5 console
R5:~# route (missing router R1)
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref Use Iface
192.168.0.20    *               255.255.255.252 U     0      0     0 eth0
10.0.4.0        10.0.3.3        255.255.255.0   UG    3      0     0 eth1
10.0.5.0        10.0.3.3        255.255.255.0   UG    3      0     0 eth1
10.0.6.0        *               255.255.255.0   U     0      0     0 eth2
10.0.7.0        10.0.3.3        255.255.255.0   UG    2      0     0 eth1
10.0.16.0       10.0.12.5       255.255.255.0   UG    2      0     0 eth4
10.0.17.0       10.0.12.5       255.255.255.0   UG    3      0     0 eth4
10.0.1.0        10.0.3.3        255.255.255.0   UG    2      0     0 eth1
10.0.2.0        10.0.3.3        255.255.255.0   UG    3      0     0 eth1
10.0.3.0        *               255.255.255.0   U     0      0     0 eth1
10.0.12.0       *               255.255.255.0   U     0      0     0 eth4
10.0.13.0       10.0.12.5       255.255.255.0   UG    2      0     0 eth4
10.0.14.0       10.0.11.5       255.255.255.0   UG    2      0     0 eth3
10.0.15.0       10.0.12.5       255.255.255.0   UG    2      0     0 eth4
10.0.8.0        10.0.3.3        255.255.255.0   UG    4      0     0 eth1
10.0.9.0        10.0.12.5       255.255.255.0   UG    3      0     0 eth4
10.0.10.0       10.0.12.5       255.255.255.0   UG    4      0     0 eth4
10.0.11.0       *               255.255.255.0   U     0      0     0 eth3
```

Figure 3.14: Routing tables when the router-R1 was missing

imented with the removal of links as shown in Figure 3.11, it shows that there was a reduction of an average of 32%, with a range of 11% to 53% in the number of routing entries in the routing tables. In addition, the removal of different links leads to different routing changes in the network. We were able to identify links that had the most effects on network when they were removed. These experiments explain the importance of certain links to the routing changes in the network. Failure of each link has corresponding effects on the routing information of the network and, eventually, the routing changes. As an example, four links — R1-R3, R2-R4, R3-R5, R4-R7 and R6-R7 — in Figure 3.11 are critical for the day-to-day running of the network; any failure of such links has considerable effects on the routing changes.

The second experiment evaluated the effects of router failures in a virtual network configured with RIP routing protocol. We observed there was no difference in the

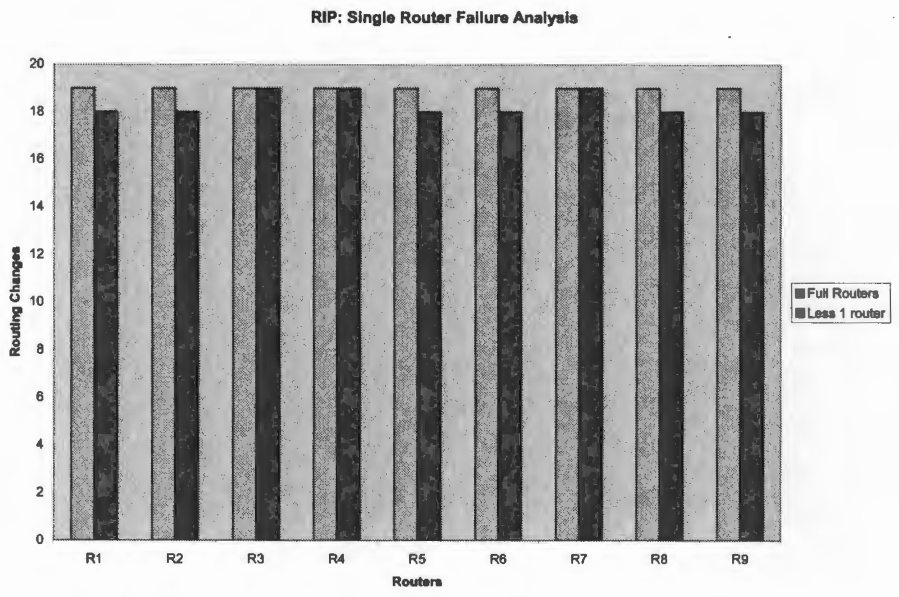


Figure 3.15: Single router removal analysis for RIP

number of routing changes with single-router failures — except for those with cut points — as shown in Figure 3.15. This evaluation confirmed the true behavior of RIP [15]: After waiting for 180 seconds, a router considers its immediate neighbor to be dead and takes the next available router as its next hop and adjusts its routes. When there was a single-router failure in the network, it was equivalent to the simultaneous failure of all their connecting links. The example in Figure 3.13 shows that when router R3 was missing from the network, there was simultaneous failure of all the three connecting links to this router. These single-router failure experiments show that they are quite different from single-link failure experiments. In the case of single-link failure experiments, the network waited for some time to stabilize while in the case of single-router failure experiment the network did not wait for stabilization; the next available router immediately became its next hop and adjusted the routes.

Finally, we are able to use this protocol in the virtual network to understand the routing changes caused by the link and router removal in a small network. But for large network and complex networks `RIP` is probably wholly inadequate. This routing protocol does compute new routes after any change in the network topology, but in some cases it does so very slowly, by counting to infinity. `RIP` prevents routing loops from continuing indefinitely by implementing a hop count limit. This limit ensures that anything more than fifteen hops away is considered unreachable by `RIP`. The drawback of `RIP` in [15] explains the choice of network operators and researchers for improved routing protocols from the link state family that can detect and correct router failures in their network.

Chapter 4

Modelling the Routing of an Autonomous System

This chapter contains the experimental work and results of modelling the routing of an AS, more specifically the GÉANT network. The GÉANT network was a pan-European backbone that connects Europe's national research and education networks.

We present the goals of this chapter in Section 4.1. In Section 4.2, we explain our network topology and describe how we modelled the GÉANT network using an emulation method, and validated the functionality of this virtual network. We conducted two case studies in the network, which are described in Section 4.3. The first case study investigated the effects of single-link failures in the virtual network, and the second case study examined the effects of single-router failures. In Section 4.4, we compare our emulation studies and a simulation work by Quoitin *et al.* in [20, 21]. Both simulation and emulation methods are used to study the routing changes in the GÉANT network. Lastly, in Section 4.5 we present our conclusions drawn from the two case studies.

4.1 Introduction

In this chapter, we aim to use emulation techniques to design a model of the GÉANT network. This emulated network topology is similar to that of the simulated model of the GÉANT network investigated in [20, 21]. Our first goal in this study was to use emulation techniques to design a model of the GÉANT network testbed. The second goal was to use a routing protocol, OSPF, to configure a large scale network on the testbed, referred to as the GÉANT. The third goal was to use our virtual GÉANT network testbed to study routing changes caused by link and router failures. Lastly, the final goal was to demonstrate that emulation techniques produce reasonable results for a large scale network, which are consistent with the results obtained by Quoitin *et al.* in [20, 21]. The results collected from our emulated study are directly compared to the results from Quoitin *et al.* simulation work.

4.2 Modelling of the GÉANT network

In this section, we describe how to use an emulation technique to model the GÉANT network. The GÉANT was a transit network: a pan-European computer network for research and education. The GÉANT network is the large-scale network we used for our investigations. The GÉANT network was a multi-gigabit European computer network project for research and education. Maintaining the GÉANT network project involved network testing and development of new technologies and networking research. Figure 4.1 shows the overview of the GÉANT backbone network. GÉANT2 is the successor to GÉANT, and its development began in November 2000 and officially ended in April 2005. See more details regarding GÉANT and GÉANT2 projects in [4, 31]. Later in the sub-sections, we briefly describe the implementation and configuration of this network with OSPF routing protocol.

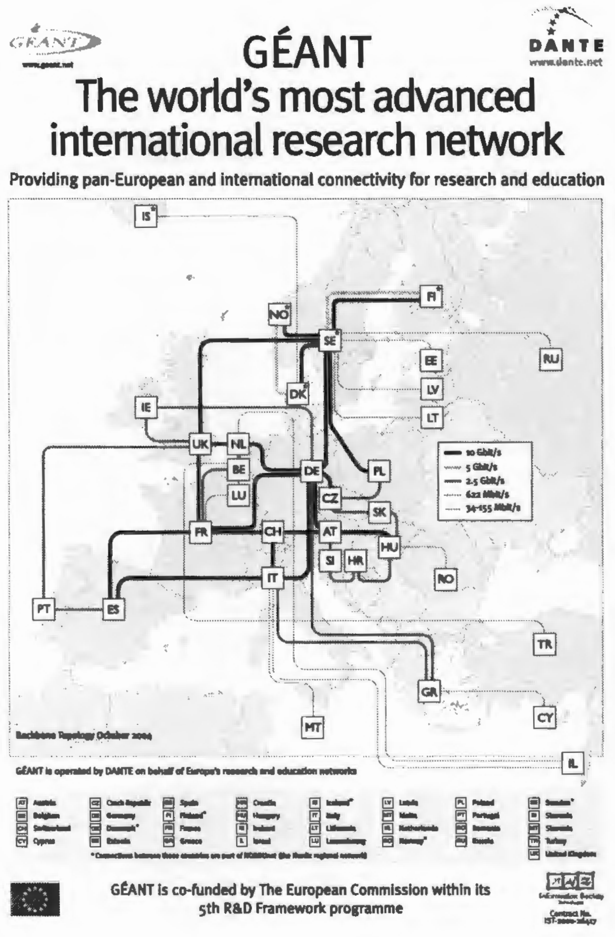

GÉANT
The world's most advanced international research network
Providing pan-European and international connectivity for research and education
DANTE
www.dante.net
IS
NO
FI
SE
RU
EE
LV
LT
DK
IE
UK
NL
BE
LU
DE
PL
CZ
SK
FR
CH
AT
HU
SI
HR
IT
RO
PT
ES
TR
GR
MT
CY
IL
10 Gbit/s
5 Gbit/s
2.5 Gbit/s
622 Mbit/s
34-155 Mbit/s
GÉANT is co-funded by The European Commission within its 5th R&D Framework programme

Figure 4.1: The GÉANT backbone network: Courtesy of DANTE

4.2.1 Topology of the GÉANT network

We modelled a network with similar topology to that used in [20, 21], which was the three-layer topology of the GÉANT captured from a one-day IS-IS trace of 24 November, 2004. Our model of the GÉANT network has a complex topology which includes the twenty-three routers, thirty-eight links, and two hosts. These two hosts are used for testing and validation purposes, they are not routers and have no effect on the topology.

The emulated GÉANT network model was designed for our investigations of the impact of router and link failures in a large scale network. The network graph of the emulation of this complex network is shown in Figure 4.2.

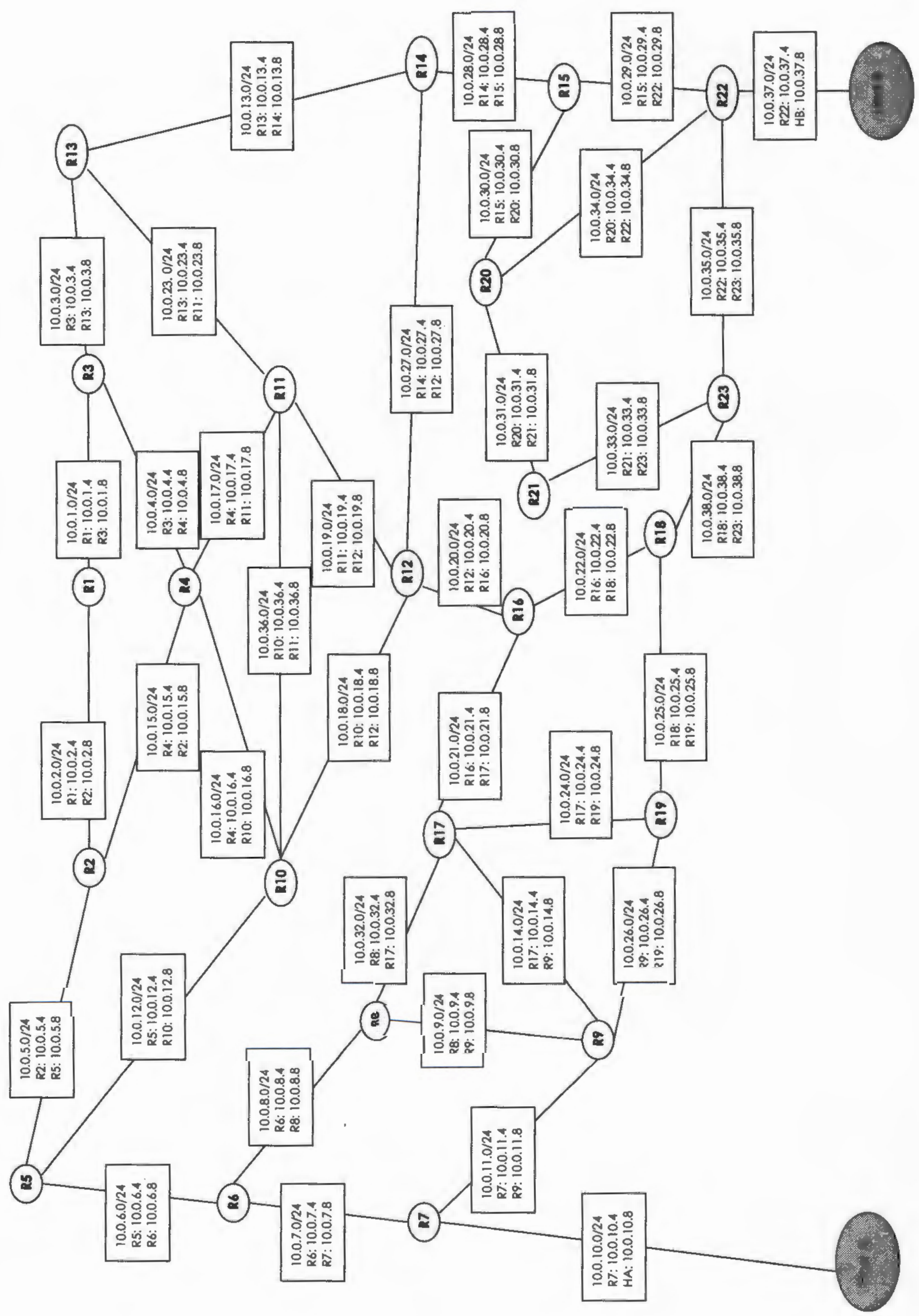


Figure 4.2: A twenty-five node network topology configured with OSPF

4.2.2 Implementation and configuration with OSPF

In this section, we implemented the network topology described in Section 4.2.1 using VNUML on an UBUNTU Linux machine. The methodologies for design and implementation of this virtual network testbed are similar to the approach used in Section 3.3.2. The major difference between them is the size of the GÉANT network; we needed to patch the UBUNTU host machine with SKAS3 features [1] in order to enhance performance to meet the larger resource requirements of the GÉANT network emulation. Below is a set of basic steps for the implementation and configuration of the virtual GÉANT network:

1. We patched the default Linux kernel of the UBUNTU host machine [12] with SKAS3 obtained from [1]. We downloaded these patches and compiled a Linux kernel on UBUNTU systems to include SKAS3 features.

2. We then installed VNUML tool in the Linux environment of the host machine. This tool and the installation procedures can be downloaded from [8]. The VNUML tool is designed to easily create simple and complex network emulation scenarios.

3. Next, we installed Quagga in the system-wide /etc/ directory of the host machine. Quagga is a routing software package that provides TCP/IP-based routing services and protocol dæmons. A machine installed with Quagga serves as a dedicated router.

4. We wrote implementation code for the network topology specified in Figure 4.2 in an XML file. The purpose of this file was to include specifications for creating the virtual GÉANT network. We ensured that the XML file specifications conformed to the VNUML DTD [8] that came with the VNUML tool. Details of the XML specifications are included in Appendix C.

5. We then created the VNUML session and individual machines by running the commands in Figure 4.3. When we were finished with the networking scenario, we killed the scenario processes by running the commands specified in Figure 4.4. A screen shot of the virtual GÉANT network testbed is shown in Figure 4.5. Each of the windows or machines in Figure 3.4 represents a node on the virtual network testbed.

6. The network created in step five had strictly local connectivity, but this network ignored the global network topology. This type of connectivity means that only adjacent routers could communicate with each other. To enable network connectivity, we then configured each router in the network with OSPF by creating these files: **zebra.conf**, **ospfd.conf** and **vtysh.conf** in **/etc/quagga** directory. These three configuration files were created and designated for each router. A sample of each configuration file for the router-R1 is included in Appendix D. See Appendix D for more details.

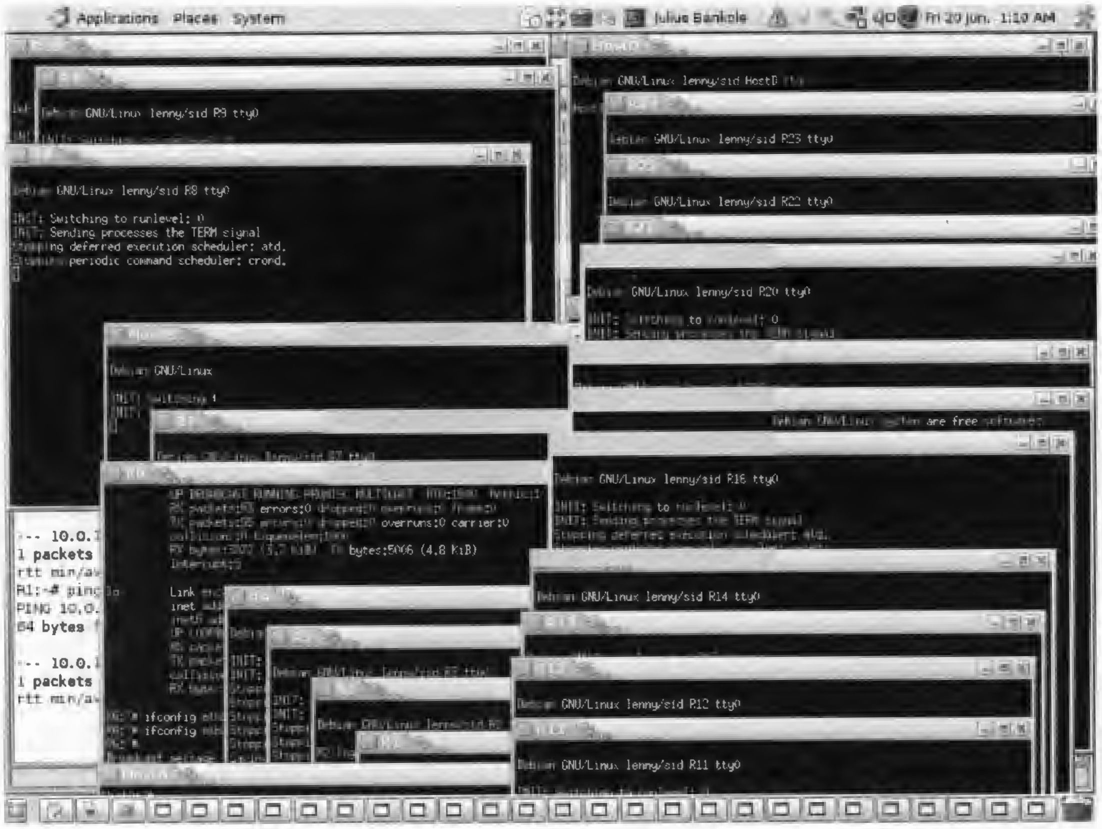
7. We then started and stopped OSPF dæmon by running commands as shown in Figure 4.6. We included a piece of code from XML specifications for starting and stopping **ospfd** dæmon as shown in Figure 4.7. See the XML file in Appendix C for more details.

```
vnumlparser.pl -t /usr/share/vnuml/NIYIospf.xml -v -u root
```

Figure 4.3: Commands for creating virtual GÉANT network

```
vnumlparser.pl -d /usr/share/vnuml/NIYIospf.xml -v
```

Figure 4.4: Commands for killing virtual GÉANT network



```
Debian GNU/Linux lenny/sid R9 tty0
```

```
Debian GNU/Linux lenny/sid R8 tty0

INIT: Switching to runlevel: 0
INIT: Sending processes the TERM signal
Stopping deferred execution scheduler: atd.
Stopping periodic command scheduler: crond.
```

```
Debian GNU/Linux lenny/sid HostB tty0
```

```
Debian GNU/Linux lenny/sid R23 tty0
```

```
Debian GNU/Linux lenny/sid R22 tty0
```

```
Debian GNU/Linux lenny/sid R20 tty0

INIT: Switching to runlevel: 0
```

```
Debian GNU/Linux
```

```
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:[illegible] errors:0 dropped:0 overruns:0 frame:0
          TX packets:[illegible] errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:[illegible] (3.2 KiB)  TX bytes:5006 (4.8 KiB)
          Interrupt:5

          Link [illegible]
          inet [illegible]
R6:~# ifconfig [illegible]
R6:~# ifconfig [illegible]
R6:~#
```

```
--- 10.0.[illegible]
1 packets
rtt min/a[illegible]
R1:~# ping [illegible]
PING 10.0.[illegible]
64 bytes [illegible]

--- 10.0.[illegible]
1 packets
rtt min/a[illegible]
```

```
Debian GNU/Linux lenny/sid R16 tty0

INIT: Switching to runlevel: 0
INIT: Sending processes the TERM signal
Stopping deferred execution scheduler: atd.
```

```
Debian GNU/Linux lenny/sid R14 tty0
```

```
Debian GNU/Linux lenny/sid R12 tty0
```

```
Debian GNU/Linux lenny/sid R11 tty0
```

Figure 4.5: A screen shot of the virtual GÉANT network testbed

```
        sudo vnumlparser.pl -x start@NIYIospf.xml # Starting the daemons
```

```
        sudo vnumlparser.pl -x stop@NIYIospf.xml # Stopping the daemons
```

Figure 4.6: Commands for starting and stopping the OSPF dæmons

```
    <filetree root="/etc/quagga" seq="start">R1</filetree>
    .....
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
```

Figure 4.7: XML code for starting and stopping the *zebra* and *ospfd* dæmon

4.2.3 Validating a model of the GÉANT network

In this section, we performed validation tests on the virtual GÉANT network. The results of these validation tests from our emulated network confirmed the expected behavior as observed on the physical topology of the **GÉANT** network. The validation tests were carried out in three different ways: testing the network for reachability, computing routing tables for each router, and tracing the route of packets in the network. These three tests assured us that our virtual representation of the **GÉANT** network was functional and reliable for our case studies in Section 4.3.

4.2.3.1 Testing the network reachability

The first validation test was to check for connectivity in this complex virtual network. When there is no routing protocol in the network, routers have local connectivity, that is, they are only connected to immediate neighbors. When there is a routing protocol such as OSPF in the network, OSPF routers flood the network with link state information. All routers will receive updates and re-compute their routing tables.

We used the `ping` command for this test. For instance, we tried from console R5 to reach Host A on the virtual network. We obtained the result: "Network is unreachable"; this is due to lack of a routing protocol in the network. After we had successfully configured the network with the OSPF dæmon, the connectivity confirmed the protocol was working correctly. The result of a `ping` command on the router-R5 console to reach Host A is shown in Figure 4.8. This testing confirmed that the OSPF protocol was working correctly, and we could proceed to the next test.

4.2.3.2 Managing the routing information with OSPF

The second validation test was to compute the routing tables for each router. We wanted to display summary information about all routes for the OSPF protocol.

We ran the `route` command directly from the console of each router. After executing the command, we obtained results — routing tables — that indicated routing entries:

```
pinging
R5:~# ping 10.0.10.8 -c2 #Host A without OSPF daemon
connect: Network is unreachable

R5:~# ping 10.0.10.8 -c2 #Host A with OSPF daemon
PING 10.0.10.8 (10.0.10.8) 56(84) bytes of data.
64 bytes from 10.0.10.8: icmp_seq=1 ttl=62 time=60.7 ms
64 bytes from 10.0.10.8: icmp_seq=2 ttl=62 time=0.647 ms

--- 10.0.10.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 0.647/30.719/60.791/30.072 ms
```

Figure 4.8: **Pinging** from R5 to Host B

destinations, gateway or path to different destinations, metric — cost, interfaces and flags. These results showed the routing information of the virtual network and they are shown in Figure 4.9 for the router-R5 console. This test also confirmed that the OSPF protocol was working correctly and we could proceed to the third test.

4.2.3.3 Tracing packets in the virtual GÉANT network

The third validation was to ascertain how packets travel in our virtual network. This validation test showed a list of routes traversed, and allowed us to identify the path taken to reach a particular destination in the network.

We used the `traceroute` command to investigate the route taken by packets across the virtual GÉANT network. The result showed paths that were taken by the packets and the corresponding time spent in milliseconds. Figure 4.10 shows a session through the router-R5 console. The result showed how a packet would travel on the router and the respective times in milliseconds. This test also confirmed that both the network and OSPF protocol were functioning properly. From the topology in Figure 4.2, we could use a physical examination of the network to obtain the computation of shorthest paths/hops for tracing the packets from R5 to Host B. Both the physical

R5 console
```
R5:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
192.168.0.16    *               255.255.255.252 U     0      0      0 eth0
10.0.20.0       10.0.12.8       255.255.255.0   UG    30     0      0 eth3
10.0.21.0       10.0.6.8        255.255.255.0   UG    40     0      0 eth2
10.0.22.0       10.0.12.8       255.255.255.0   UG    40     0      0 eth3
10.0.23.0       10.0.12.8       255.255.255.0   UG    30     0      0 eth3
10.0.16.0       10.0.12.8       255.255.255.0   UG    20     0      0 eth3
10.0.17.0       10.0.5.4        255.255.255.0   UG    30     0      0 eth1
10.0.18.0       10.0.12.8       255.255.255.0   UG    20     0      0 eth3
10.0.19.0       10.0.12.8       255.255.255.0   UG    30     0      0 eth3
10.0.28.0       10.0.12.8       255.255.255.0   UG    40     0      0 eth3
10.0.29.0       10.0.12.8       255.255.255.0   UG    50     0      0 eth3
10.0.30.0       10.0.12.8       255.255.255.0   UG    50     0      0 eth3
10.0.31.0       10.0.12.8       255.255.255.0   UG    60     0      0 eth3
10.0.24.0       10.0.6.8        255.255.255.0   UG    40     0      0 eth2
10.0.25.0       10.0.12.8       255.255.255.0   UG    50     0      0 eth3
10.0.26.0       10.0.6.8        255.255.255.0   UG    40     0      0 eth2
10.0.27.0       10.0.12.8       255.255.255.0   UG    30     0      0 eth3
10.0.4.0        10.0.5.4        255.255.255.0   UG    30     0      0 eth1
10.0.5.0        *               255.255.255.0   U     0      0      0 eth1
10.0.6.0        *               255.255.255.0   U     0      0      0 eth2
10.0.7.0        10.0.6.8        255.255.255.0   UG    20     0      0 eth2
10.0.1.0        10.0.5.4        255.255.255.0   UG    30     0      0 eth1
10.0.2.0        10.0.5.4        255.255.255.0   UG    20     0      0 eth1
10.0.3.0        10.0.5.4        255.255.255.0   UG    40     0      0 eth1
10.0.12.0       *               255.255.255.0   U     0      0      0 eth3
10.0.13.0       10.0.12.8       255.255.255.0   UG    40     0      0 eth3
10.0.14.0       10.0.6.8        255.255.255.0   UG    40     0      0 eth2
10.0.15.0       10.0.5.4        255.255.255.0   UG    20     0      0 eth1
10.0.8.0        10.0.6.8        255.255.255.0   UG    20     0      0 eth2
10.0.9.0        10.0.6.8        255.255.255.0   UG    30     0      0 eth2
10.0.10.0       10.0.6.8        255.255.255.0   UG    30     0      0 eth2
10.0.11.0       10.0.6.8        255.255.255.0   UG    30     0      0 eth2
10.0.37.0       10.0.12.8       255.255.255.0   UG    60     0      0 eth3
10.0.36.0       10.0.12.8       255.255.255.0   UG    20     0      0 eth3
10.0.38.0       10.0.12.8       255.255.255.0   UG    50     0      0 eth3
10.0.33.0       10.0.12.8       255.255.255.0   UG    60     0      0 eth3
10.0.32.0       10.0.6.8        255.255.255.0   UG    30     0      0 eth2
10.0.35.0       10.0.12.8       255.255.255.0   UG    60     0      0 eth3
10.0.34.0       10.0.12.8       255.255.255.0   UG    60     0      0 eth3
```

Figure 4.9: An example of OSPF routing table for R5 console

examination and emulation results produced the same number of shortest hops/paths, i.e., six hops to the final destination.

We could confirm by using the above three validation tests that our emulated GÉANT network was functional and the OSPF protocol was running accordingly in the network. Because the network topology has been validated, next we proceed with our networking experiments.

```
traceroute
R5:~# traceroute -n 10.0.37.8 # Host B
traceroute to 10.0.37.8 (10.0.37.8), 30 hops max, 40 byte packets
 1  10.0.12.8  33.176 ms  0.366 ms  0.255 ms
 2  10.0.18.8  47.148 ms  0.728 ms  0.587 ms
 3  10.0.27.4  50.409 ms  0.995 ms  0.914 ms
 4  10.0.28.8  41.210 ms  0.676 ms  0.484 ms
 5  10.0.29.8  46.643 ms  0.651 ms  0.555 ms
 6  10.0.37.8  46.573 ms  0.952 ms  0.659 ms
```

Figure 4.10: `Traceroute` from R5 to Host B

4.3 Case studies in the virtual GÉANT network

In this section, we present two case studies investigated in the virtual GÉANT network. The experimental set-up is similar to that detailed in Section 3.2, but slightly modified as explained in Section 4.2.2 to conform with the requirements necessary for a large scale networking scenario.

In the first case study, we examined the impact of removing links as detected in the total routing cost. In the second case study, we investigated the effects of the routers' removal and the corresponding total routing cost. The two case studies helped us to understand the impact of link and router failures in the virtual GÉANT network and their resulting total costs.

In [20, 21], Quoitin *et al.* partitioned the routing changes into four different classes: *Peer change*, *Egress change*, *Intra cost change* and *Intra path change*. Each of these changes was described in their work. In our emulated GÉANT network, we focus on *Intra cost change* as the routing changes, because other changes cannot be measured in this network. This particular change occurs when there is no egress change except for the change in the IGP cost of the ingress-egress path in the network. Our emulated GÉANT network assumed that the link weights were constant — ten units, and link weights reflected the cost of using a link. To minimize the overall cost, OSPF routing protocol runs Dijkstra's algorithm to determine the shortest path — least-cost path — in our case studies.

We also used the case studies to identify the links whose loss produces higher routing costs and the routers whose loss yields the largest routing costs. Our goal in these case studies was to demonstrate that the emulation method produces useful results for understanding intra-domain routing. In the next section, Section 4.4, we use this information to make a comparison of emulation and simulation techniques.

4.3.1 Single-link failures with OSPF

Most network operators do not have a sufficient understanding of the effect of link failures in the network. Evaluating and determining which link failures will change the outcome of the route selection in a large network configured with OSPF is a difficult problem. Understanding and evaluating effects of link failures are important because routing changes often lead to traffic shifts and traffic congestion. For a network operator, it is important to determine whether the network will be able to accommodate the traffic load when single link failures occur. In addition, it is also necessary to identify the links in the network whose loss would cause increases in the total routing cost (i.e., sum of *Intra cost change*) and protect such links by the addition of parallel links to mitigate the impact of link failures in the network.

In this case study, we experimented with the removal of links in the emulated GÉANT network. The first objective was to compute the head node routing costs of each link for a fully functional network and compare it with that of a missing-link networking scenario. We aimed to identify the links most affected by the single link failures in the network. The second objective was to relate our results from the emulated GÉANT network with those of simulations carried out by Quoitin *et al.* in [20, 21].

We used our virtual GÉANT network that was validated in the last section for these experiments. Firstly, we conducted the experiments as described in steps five to seven of Section 4.2.2 for separate single link failures for each router. When the network was functioning, we recorded the routing tables for each router. Secondly, we removed each link from the virtual network, one at a time, and recorded the corresponding routing tables for the router at the beginning of the link. This removal of link was done for all the links in the network, and each time we computed the total cost of routings and re-routings of these links. In this experiment, we selected cost as an index for measuring routing changes. Data collection was done by running the `route` command directly from the console of the beginning router of the removed link. The beginning of a link is the starting router and the ending of the link is the ending router for any link in the network. An example of data collection for a removed link R1-R2 is as follows. The sum of total cost for routings and re-routings was collected from the console of the router-R1 before and after link R1-R2 was removed.

The resulting changes in the routing tables for each link removed are shown graphically in Figure 4.11. In this figure, we show results for both conditions, that is, when the network was fully functional and when there was removal of individual links. In Figure 4.11, the links of the virtual network are shown on the x-axis while the head node routing costs are shown on the y-axis.

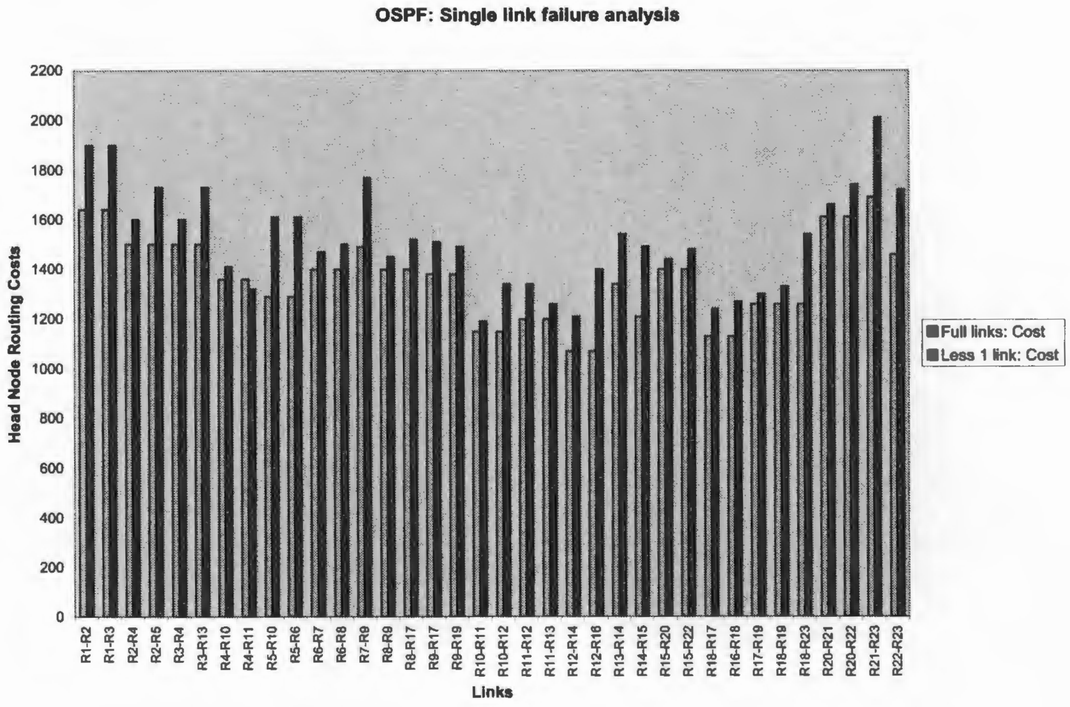


Figure 4.11: Single link failure analysis for OSPF

For each link removed, we observed the total routing cost at the head nodes of the links — the sum of *Intra cost changes.* There were remarkable changes in the number of routers that increased their cost (metric) as a result of single-link failures in the virtual GÉANT network. We observed about 12% variation in the total routing cost for all the links, and an average of 13%, with a range of 5% to 20% increases in the cost of re-routings of all the links in the emulated GÉANT network. We also observed remarkable increases in the total routing cost at the link head nodes of an average of 18%, with a range of 16% to 20% in the following links: R1-R2, R1-R3, R2-R5, R3-R13, R5-R10, R5-R6, R7-R9, R9-R17, R10-R12, R12-R16, R13-R14, R18-R23, R21-R23, and R22-R23. From Figure 4.2, these links show that their failures would increase the total cost of re-routings in the network. In this project, we selected any link from 10% increases to constitute a potential major change in the total cost of routing. The increase in the total cost accounted for the fact that those links were important for routing in the network; removal of such links would have moderate effects in the network.

In our experiments, we also observed small increases in the total routing cost at the head nodes of these links: R2-R4, R4-R10, R15-R22, R17-R19, and R18-R19. These routing costs, the sum of *Intra cost change*, are of an average of 6%, with a range of 5% to 7% for their re-routings in the network. The slight increases in the total routing cost at the head nodes of these links demonstrate that their effect in the network is of lesser importance. The links with higher routing costs in the network are more important, and their removal or breakage moderately affected the routing costs.

Without missing links, the network functioned smoothly; and we recorded the head node routing costs from the routing tables. However, we observed differences in the the head node routing costs when we conducted experiments with link failures in the same network. Link failures clearly resulted in moderate changes in the total routing cost at the head nodes of these links in the network, and such identified links need to be protected to prevent traffic congestion.

In these experiments, we observed a change in the total routing cost at the node at the head of a link that was removed. This result is qualitatively consistent with the description of `OSPF` given in Section 2.5.2.

4.3.2 Single-router failures with `OSPF`

Most network operators and ISPs desire to understand the impact of router failures in the network. Evaluating and determining which router failures will change the outcome of the route selection is a difficult problem in a large network configured with OSPF. Understanding and evaluating effects of router failures are important, because routing changes often lead to traffic shifts and traffic congestion. For a network operator, it is often important to predict whether the network will be able to accommodate the traffic load when single-router failures occur. In addition, it is necessary to identify which of the routers in the network should be protected against router failures by the addition of parallel routers.

In this case study, we experimented with the removal of routers in the emulated GÉANT network. In a large AS, it is often difficult to predict which router failures will most affect the total routing cost — sum of *Intra cost change*. Our first objective was to collect the routing information for the fully functional network and compare the results with routing information for a missing-routers networking scenario. We sought to identify which routers are most affected by the single router failures in the network. Our second objective is to relate our results from the emulated GÉANT network with the simulation results reported by Quoitin *et al.* in [20, 21]. We discuss the comparison in Section 4.4.

As explained in Section 2.5.2, an OSPF router typically runs Dijkstra's shortest path algorithm to determine a shortest path tree to all subnets, with itself as the root node. In this network, we assumed that each link has a cost of ten, and consequently the least-cost path is the same as the shortest path.

For these experiments, we used the virtual GÉANT network that had already been validated as described in the last section. Firstly, we ran the experiments as described in steps five to seven of Section 4.2.2 for a total of twenty-three times, that is, each time for each router. When the network was fully functional, we recorded the total routing cost from the routing tables for each router. Secondly, we removed one router for each experiment using our XML specifications in Appendix C. Emulations of the network were performed sequentially until the effects of a single router failure for each router in the network were recorded. For each router disabled, the data were collected by running the `route` command directly from the console of each remaining router, and summing the costs. These experiments consumed a lot of time and resources because we had to collate the resulting data for each of the twenty-three routers that was disabled and their remaining routers on each occasion.

Normally, a router broadcasts link state information whenever there is a change in the network. The OSPF routers periodically flood the network with their advertisements, thereby adjusting their routing tables and letting other routers know that they are still functional. The resulting changes in the number of routing entries for each router removal are shown graphically in Figure 4.12. The results for both conditions are displayed: when the network was fully functional and after the removal of each router. Each router in the virtual network is shown on the x-axis, and the total routing cost in the network is shown on the y-axis.

When each router was removed, we observed moderate increases in the total routing cost for some routers. These variations were of an average of 7.5%, with a range of 5% to 10% in the total routing cost (i.e., the sum of *Intra cost change*) for these routers: R5, R6, R10, R12, and R14. In this project, any router with more than 5% increases was considered to have a potential major change in the total cost of routing. These results confirmed that the removal of any of these routers in the network would lead to increases in the total routing cost. However, failure of such routers could lead to increasing cost of reaching some destinations in the network.

We also observed that routers: R7, R17, R19, R20, R21, and R22 have least effects because their removal would lead to a shortfall of an average of 4.5%, with a range of 2% to 7% in the total routing cost (i.e., the sum of *Intra cost change*), not the individual cost for a each router in the network. This shortfall means that their

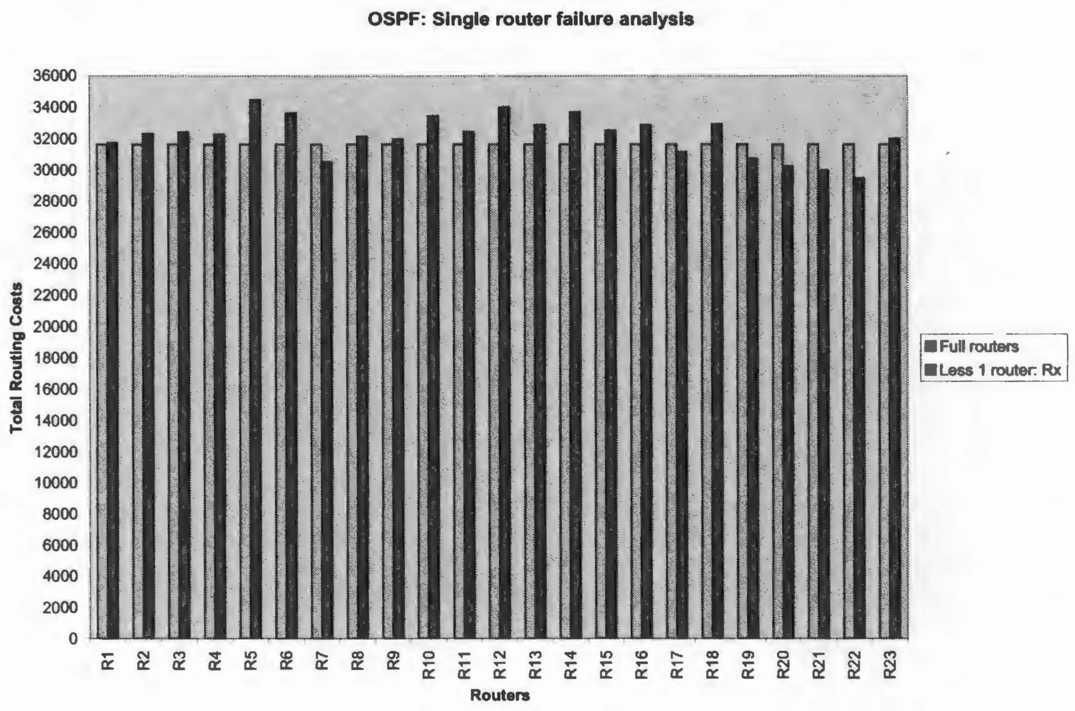


Figure 4.12: Single router failure analysis for OSPF

removal could reduce the total routing cost (i.e., the sum of *Intra cost change*) in the network. It would lower the total cost for traversing the whole network. The fluctuation in the total routing cost gave us the hint on the possible changes in the cost of maintaining traffic flows in the network.

Essentially from the data collected, we observed that there is a fluctuation in the total routing cost for the missing routers networking scenario when compared with the original, fully functioning network. These results confirm the concepts in Section 2.5.2 that there are changes in the routing tables when each router is removed from the network.

4.4 Comparison of emulation and simulation results for the GÉANT network

In this section, we compare the results obtained from our emulations studies of the GÉANT network with the simulation studies for this same network carried out by Bruno Quoitin in [20, 21]. We briefly discuss the main difference of these two experimental techniques.

The emulation and simulation techniques take complementary approaches toward computing routing. Typically, the goal of emulation techniques is to closely reproduce features and behaviors of real world devices while the goal of simulation techniques is to predict outcomes of running a set of network devices in a network based on the internal model of the specified simulator. In our emulation studies, we obtained results that show similar pattern to that of Quoitin *et al.* [20, 21] regarding the change in network conditions as caused by link and router failures in the network.

From our emulation studies of single link failures, we observed that a single link failure often leads to noticeable changes in the total routing cost at the head nodes of the links in the network. All of the links in our virtual GÉANT network indicate variation in the head node routing costs as shown graphically in Figure 4.11. For instance on Figure 4.2, links: R1-R2, R1-R3, R2-R5, R3-R13, R5-R10, R5-R6, R7-R9, R9-R17, R10-R12, R12-R16, R13-R14, R18-R23, R21-R23, and R22-R23 represent some increase in the cost while links: R2-R4, R4-R10, R4-R11, R6-R7, R8-R9, R10-R11, R15-R20, R17-R19, and R18-R19 show slight increases for re-routings when a link removal occurred in the network. This is similar to the results obtained in [20, 21]; the changes in the routing updates for simulation work can be obtained from page 87 of [20]. From our investigations, we observed that all of our virtual GÉANT links caused nearly 20% fluctuations in the head node routing costs when they failed for our emulated network which was primarily intra-AS. On the other hand, Quoitin *et al.* observed that about 60% of the **GÉANT** links caused more than 100,000 routing changes when they failed in their simulated network. The differences in percentages obtained for our emulated network and that of the simulated can be explained by the fact that our emulated network percentage is only for the total head node routing costs while the simulated network percentage is for all the four classes of routing changes as described in their work. It can be seen clearly that both experimental techniques identify important links that are most affected by single link failures in their model of the **GÉANT** network.

We also agree with Quoitin *et al.* that the number of intra domain re-routings is few. We recorded a relatively low number of routing changes in our virtual network because our network design mainly focused on an intra-AS, that is, we concentrate on purely intra domain re-routing. In [20, 21], Quoitin *et al.* also remarked that there are few routing changes in the intra-cost change (that is, change in IGP cost without egress change) and intra-path change (that is, same IGP cost for an ingress-egress) classes. This is because models of the **GÉANT** would not capture all the changes in the routing updates of single link failures for a transit network like the **GÉANT**.

For our emulation studies of single router failures, we observed that failures of **GÉANT** routers often lead to changes in the total routing cost. The failure of a single router is also equivalent to the failure of all links that are attached to this router. In [20, 21], it was observed that failure of some routers could lead to the unreachability of some destinations. These routers: R5, R6, R10, R12, and R14 accounted for moderate increases in the total routing cost; as their failures also affected the most critical links that were connected to these routers. These links and routers are also identified in Figures 4.11 and 4.12. The changes in the routing changes for simulation work can be obtained from page 88 of [20]. Our emulation result is consistent with that of simulation work in [20, 21]. It can be seen clearly that both experimental techniques identify important routers that are most affected by single-router failures in their model of the GÉANT network.

There is a noticeable difference in the number of routing changes (i.e., the routing cost) in our emulation studies when compare to that of simulation work. This difference occurred because Quoitin *et al.* included BGP routes in his simulation studies while our emulation studies focused purely on intra-AS routes. This explains the huge number of routing changes recorded in his experiments. However, the pattern of the routing costs reflects similar behavior for cases with link and router failures in the network.

4.5 Conclusions

In this section, we provide discussions and experimental conclusions. We used emulation technique to model the GÉANT network, carried out validation tests and conducted two experimental case studies for intra-domain routing. We used the two emulation case studies do a comparison with the simulation work and infer the following conclusions.

In the first case study, we used emulation techniques to examine the impact of link failures on the head node routing costs in the network using the OSPF routing protocol. From our results, we inferred that single-link failures in the virtual network account for less than 20% difference in the total head node routing costs. Our emulation result shows similar patterns with that of simulation work in [20, 21]. We inferred that both emulation and simulation studies identified important links that were important in the models of GÉANT network. Such links were not exactly the same because of different routing data used in the two different experimental techniques.

In the second case study, we used emulation techniques to understand the impact of router failures on total routing cost using the OSPF routing protocol. We used the single-router failure analysis to identify heavily-used routers in the network and also to understand their behavior using the OSPF routing protocol. Such routers include R5, R6, R10, R12, and R14, and they accounted for moderate increases in the total routing cost. Our results are similar to the simulation work in [20, 21] which also observed that failures of single routers, that is, the GÉANT routers often cause different classes of routing changes and lead to traffic congestion. Finally, we were able to use the emulation technique and OSPF routing protocol to understand changes in the routing costs of our emulated GÉANT network as caused by link and router failures.

Chapter 5

Conclusions and Future Work

This chapter concludes the project report. Section 5.1 is a summary of the work done, and the main conclusions are presented in Section 5.2. Lastly, we discuss future directions of this research in Section 5.3.

5.1 Project Summary

We used emulation techniques to design and implement two virtual network testbeds in this project. In this work, we emulated simple and complex networks; these networks were validated and used for our investigations. We implemented a fifteen-node virtual network testbed and configured it with `RIP` routing protocol. We also modelled a complex network, the `GÉANT` and configured it with a more powerful routing protocol, `OSPF`. These testbeds were used to explore experiments on routing changes caused by link or router failures in the two networks.

With our simple network testbed, we were able to use emulation techniques to produce meaningful results that are comparable to the expected results for a small network —

a fifteen node network. This testbed provided us with an environment for testing RIP protocol while it works dynamically to re-configure the network against fluctuations or changes of conditions in the network. In our experiments, RIP protocol was useful for identifying missing links and critical links in the network: this result confirmed our theoretical expectations of RIP. In addition, we observed that when there was a single-router failure in the RIP configured network, it was equivalent to the simultaneous failure of all the connecting links. During the single-router failure experiment the network, there was no need to wait for network stabilization; the next available router immediately became the next hop and adjusted the routes accordingly.

We carried out two case studies in the virtual GÉANT network testbed to investigate routing changes. The first case study, an evaluation of the impact of missing links on a complex network, we observed the network behavior for missing-link scenarios. These scenarios revealed critical links that were important for operations of the network. For network operations, link failures accounted for changes in the total routing costs on the routing tables. The second case study, an evaluation of the impact of missing routers in the GÉANT network testbed, we observed that the failures of certain routers could lead to increase in the cost of reaching some destinations. The testbed enabled us to study the behavior of the network when it was used for an intra-domain routing. From the results collected, we observed that OSPF protocol was efficient at re-computing the routing tables in the case of missing routers. The protocol flooded the network with routing information updates and adjusted quickly to new conditions. Our results are consistent with those obtained when similar experiments were performed on the simulation of the GÉANT network for the missing routers.

Lastly, we used emulation techniques to gain invaluable experience creating, configuring, and managing virtual networks that are similar to live networks. This practical knowledge and understanding provided us insights for the deployment of physical networks when needs arise. We gained much-needed knowledge and hands-on experience to building and maintaining small and large scale networks.

In this project, there were two major limitations to conducting our experiments. The first limitation was our inability to obtain the network graph for the simulated **GÉANT** network of Quoitin *et al.* This limitation prevented us from having the exact representation of links and routers in the design of our emulated networking scenarios. Though, the **GÉANT** network had ceased to exist, but it would be nice if **GÉANT2** operators can produce a network graph of their new network for future research purposes.

The second limitation was our inability to obtain and use the same routing data that Quoitin *et al.* collected on November 24th, 2004 [20, 21]. In our experiments, we had to generate our routing data from our emulation of the **GÉANT** network. This lack of routing data accounted for non-replicate references to individual links and routers in the graphical presentation of our results. However, we observed a similar pattern of link and router behaviors as recorded in the simulation experiments.

5.2 Conclusions

There are five main conclusions from this project:

1. Our experiments in Chapters 3 and 4 enable us to develop virtual network testbeds that are re-usable and re-configurable by users. These testbeds will enhance learning and testing of network applications and services by students and network administrators. Network configurations and training can be provided to students without requiring a real network. Our testbeds can be used as templates for practising and learning network configurations.

2. Our experiments in Chapters 3 and 4 show that emulation-based experiments demonstrate typical behaviors of both `RIP` and `OSPF` protocols in any network. Both dynamic routing protocols are able to quickly re-compute routing tables when there are missing-links, and the `OSPF` protocol effectively handles missing-routers scenarios by flooding the network with new routing information.

3. Our experiments in Chapters 3 and 4 confirm that emulation-based experiments can help ISP operators to understand routing changes and assess the total routing costs of traversing the network respectively. Virtual network testbeds can be used to study missing routers and links in simple and complex networks. This information is useful because emulation environments closely reproduce features and behaviors of real world devices. Emulated networks undergo the same packet exchanges and state changes that occur in real world.

4. Our experiments in Chapters 3 and 4 confirm that emulation techniques produce reasonable results that are consistent with simulation techniques. Our emulation results are consistent with simulation results in identifying critical links and routers that can influence routing changes and traffic distributions in the network. The experimental work in Chapters 3 and 4 evaluated the impact of link and router failures in the network, achieved comparable patterns of network behavior when there were link and router failures in the network, and identified network links and routers that needed to be protected.

5. Our experiments in Chapters 3 and 4 affirm the cheap and fast ways to model complex networks. To conduct emulation of networks, we simply obtain free download of the `VNUML` tool and install it on a Linux machine for easy creation of networks. Network analysts and ISP operators can easily use this fast approach to investigate their desired networks.

5.3 Future work

In this project, our focus was to investigate routing changes and total routing costs for intra-AS. It would be interesting to expand these techniques and explore network behaviors for inter-AS routing changes and traffic distributions. For future work, we recommend the use of `VNUML` tool to study inter-domain routing changes. This work will involve using exterior gateway protocols (`EGP`) for interconnecting different autonomous systems `ASs`. The study of `EGP` will help to understand the operations of the Internet and the collection of `ASs` that make up the Internet.

The `GÉANT` network had ceased to exist, and has since been replaced with the `GÉANT2` network. Another avenue of investigations is to conduct similar experiments in the new network and compare the effects of missing links and routers on routing changes in the network. The results obtained will be more relevant and provide useful suggestions for ISP operators based in the new network.

It is also worth studying the use of combined experimental techniques for studying routing changes and total routing costs. Applying these techniques: simulation, emulation and live testing will allow researchers to determine which combination of two or three techniques will improve network tests and experiments.

Bibliography

[1] Paolo Giarrusso a.k.a. BlaisorBlade. Skas patches:. `http://www.user-mode-linux.org/~blaisorblade/uml-utilities`.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *19th ACM Symposium on Operating Systems principles*, Bolton Landing, NY, USA, 2003.

[3] James H. Cowie. Scalable simulation framework api reference manual. `http://www.ssfnet.org`.

[4] DANTE. An overview map of GÉANT network. `http://www.geant.net/upload/pdf/Topology_Oct_2004.pdf/`.

[5] Jeff Dike. *User Mode Linux*. Pearson Education, Inc, 1st edition, 2006.

[6] Kevin Fall and Kannan Varadhan. The Network Simulator Manual. `http://www.isi.edu/nsnam/ns/`.

[7] Tony Dongliang Feng, Rob Ballantyne, and Ljiljana Trajković. Implementation of BGP in a network simulator. Technical report, Simon Fraser University, 2004.

[8] Fermín Galán and David Fernández. VNUML tutorial. `http://www.dit.upm.es/vnumlwiki/index.php/Tutorial`.

[9] Fermín Galán, E. García, C. Chávarri, D. Fernández, and M. Gómez. Design and implementation of an IP multimedia subsystem emulator using virtualization techniques. Technical report, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) Av. Canal Olimpic, s/n Castelldefels, Spain, May 2006.

[10] Christian Huitema. *Routing in the Internet*. Prentice Hall, 2nd edition, 2000.

[11] Kunihiro Ishiguro. Quagga Routing Software Suite. `http://www.quagga.net`.

[12] Linux Kernel. The vanilla kernel:. `http://www.kernel.org`.

[13] Bruce Kneale and Ilona Box. A virtual learning environment for real-world networking. *Informing Science Journal,*, June 2003.

[14] Bruce Kneale, Ain Y. De Horta, and Ilona Box. VELNET: Virtual environment for learning networking. In *6th Australiasian Computing Education Conference (ACE2004)*, Dunedin, New Zealand, 2004. Australian Computer Science Society.

[15] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Pearson-Addison Wesley, 3rd edition, 2005.

[16] Steve Liu, Willis Marti, and Wei Zhao. Virtual Networking Lab (VNL): its concepts and implementation. In *Proceedings of the 2001 American Society for Engineering Annual Conference & Exposition*, 2001.

[17] Kuthonuzo Luruo and Shashank Khanvikar. Virtual networking with user-mode linux. *Linux For You Magazine*, January 2003.

[18] Luca Mottola. *Overlay Management for Publish-Subscribe in mobile Environments.* PhD thesis, Politecnico Di Milano, 2003-2004.

[19] Wendell Odom. *CCNA Intro: Exam Certification Guide.* Cisco Systems, 2004.

[20] Bruno Quoitin. *BGP-based Interdomain Traffic Engineering.* PhD thesis, Université Catholique de Louvain, Belgium, August 2006.

[21] Bruno Quoitin and Steve Uhlig. Modeling the Routing of an Autonomous System with C-BGP. In *IEEE Network Magazine.* IEEE, November/December 2005.

[22] Massimo Rimondini. Emulation of computer network with Netkit. Technical report, Università Degli Studi Di Roma Tre, 2007.

[23] Massimo Rimondini. *Interdomain Routing Policies in the Internet: Inference and Analysis.* Computer science and Engineering, Roma Tre University, 2007.

[24] Andreas Steffen, Eric Marchionni, and Parik Rayo. Advanced network simulation under user-mode linux. *Gesellschaft für Informatik*, 2005.

[25] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, Upper Saddle River, NJ, 3rd edition, 1996.

[26] Yuichiro Tateiwa, Takami Yasuda, and Shigeki Yokoi. Virtual Environment Based Learning for Network Administration Beginner. In *ABR & TLC Conference Proceedings*, Hawaii USA, 2007.

[27] OPNET Technologies. OPNET. `http://www.opnet.com`.

[28] Jeroen van der Ham and Gert Jan Verhoog. Virtual environments for networking experiments. Technical report, University of Amsterdam, the Netherlands, 2004.

[29] David Watson, Farnam Jahanian, and Craig Labovitz. Experiences with monitoring ospf on a regional service provider network. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, Ann Arbor, Michigan, US, 2003. Computer Science Society.

[30] Klaus Wehrle, Frank Pahlke, Hartmut Ritter, Daniel Muller, and Marc Bechler. *The LINUX Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel.* Pearson-Prentice Hall, 1st edition, 2005.

[31] Wikipedia. An overview of GÉANT network. `http://en.wikipedia.org/wiki/GEANT`.

[32] Chris Wright. Virtually Linux: Virtualization techniques in linux. In *Proceedings of the Linux Symposium, Volume Two*, Ottawa, Ontario, Canada, 2004.

Appendix A

The XML file for a testbed with RIP

A.1 A fifteen-node virtual network testbed

The following XML file describes a sample scenario of fifteen nodes to be used with UML and VNUML parser to set up a virtual network testbed. This testbed is configured with RIP to verify whether or not this intra-domain routing protocol is functioning correctly. We also use the testbed to study routing instability in the network.

The XML file is stored in /usr/share/vnuml/RIP15nodes.xml directory of a host machine, and a copy of this XML specification is included in the report as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">

<vnuml>
   <global>
      <version>1.8</version>
      <simulation_name>RIP15nodes</simulation_name>
      <automac/>
    <vm_defaults exec_mode="mconsole">
     <filesystem type="cow">/usr/share/vnuml/filesystems
     /root_fs_tutorial</filesystem>
    <kernel>/usr/share/vnuml/kernels/linux</kernel>
    <console id="0">xterm</console>
    </vm_defaults>
  </global>
  <net name="Net0"  mode="uml_switch" />
  <net name="Net1"  mode="uml_switch" />
  <net name="Net2"  mode="uml_switch" />
  <net name="Net3"  mode="uml_switch" />
  <net name="Net4"  mode="uml_switch" />
  <net name="Net5"  mode="uml_switch" />
  <net name="Net6"  mode="uml_switch" />
  <net name="Net7"  mode="uml_switch" />
  <net name="Net8"  mode="uml_switch" />
  <net name="Net9"  mode="uml_switch" />
  <net name="Net10"  mode="uml_switch"/>
  <net name="Net11"  mode="uml_switch"/>
  <net name="Net12"  mode="uml_switch" />
  <net name="Net13"  mode="uml_switch" />
  <net name="Net14"  mode="uml_switch" />
  <net name="Net15"  mode="uml_switch"/>
  <net name="Net16"  mode="uml_switch"/>
  <net name="Net17"  mode="uml_switch" />

  <vm name="HostA">
   <if id="1" net="Net0">
     <ipv4 mask="255.255.255.0">10.0.0.3</ipv4>
   </if>
   <route type="ipv4" gw="10.0.0.1">default</route>
  </vm>

  <vm name="R1">
    <if id="1" net="Net1">
    <ipv4 mask="255.255.255.0">10.0.1.3</ipv4>
   </if>
   <if id="2" net="Net0">
    <ipv4 mask="255.255.255.0">10.0.0.1</ipv4>
   </if>
   <if id="3" net="Net2">
    <ipv4 mask="255.255.255.0">10.0.2.3</ipv4>
   </if>
   <forwarding/>
     <filetree root="/etc/quagga" seq="start">r1</filetree>
     <exec seq="start" type="verbatim">hostname</exec>
```

```
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/zebra -d</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/ripd -d</exec>
      <exec seq="stop" type="verbatim">hostname</exec>
      <exec seq="stop" type="verbatim">killall zebra</exec>
      <exec seq="stop" type="verbatim">killall ripd</exec>
   </vm>

   <vm name="R2">
    <if id="1" net="Net2">
     <ipv4 mask="255.255.255.0">10.0.2.5</ipv4>
    </if>
    <if id="2" net="Net4">
     <ipv4 mask="255.255.255.0">10.0.4.3</ipv4>
    </if>
    <if id="3" net="Net5">
     <ipv4 mask="255.255.255.0">10.0.5.3</ipv4>
    </if>
    <if id="4" net="Net7">
     <ipv4 mask="255.255.255.0">10.0.7.5</ipv4>
    </if>
    <forwarding/>
    <filetree root="/etc/quagga" seq="start">r2</filetree>
      <exec seq="start" type="verbatim">hostname</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/zebra -d</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/ripd -d</exec>
      <exec seq="stop" type="verbatim">hostname</exec>
      <exec seq="stop" type="verbatim">killall zebra</exec>
      <exec seq="stop" type="verbatim">killall ripd</exec>
   </vm>

   <vm name="HostB">
    <if id="1" net="Net4">
      <ipv4 mask="255.255.255.0">10.0.4.5</ipv4>
    </if>
    <route type="ipv4" gw="10.0.4.3">default</route>
   </vm>

   <vm name="R3">
    <if id="1" net="Net3">
   <ipv4 mask="255.255.255.0">10.0.3.3</ipv4>
  </if>
  <if id="2" net="Net1">
   <ipv4 mask="255.255.255.0">10.0.1.5</ipv4>
  </if>
  <if id="3" net="Net7">
   <ipv4 mask="255.255.255.0">10.0.7.3</ipv4>
  </if>
  <forwarding/>
  <filetree root="/etc/quagga" seq="start">r3</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/ripd -d</exec>
    <exec seq="stop" type="verbatim">hostname</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
 </vm>

 <vm name="R4">
  <if id="1" net="Net5">
   <ipv4 mask="255.255.255.0">10.0.5.5</ipv4>
  </if>
   <if id="2" net="Net8">
   <ipv4 mask="255.255.255.0">10.0.8.3</ipv4>
  </if>
  <if id="3" net="Net9">
   <ipv4 mask="255.255.255.0">10.0.9.3</ipv4>
  </if>
  <forwarding/>
  <filetree root="/etc/quagga" seq="start">r4</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/ripd -d</exec>
    <exec seq="stop" type="verbatim">hostname</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
 </vm>

 <vm name="R5">
```

```
    <if id="1" net="Net3">
     <ipv4 mask="255.255.255.0">10.0.3.5</ipv4>
    </if>
    <if id="2" net="Net6">
     <ipv4 mask="255.255.255.0">10.0.6.5</ipv4>
    </if>
    <if id="3" net="Net11">
     <ipv4 mask="255.255.255.0">10.0.11.3</ipv4>
    </if>
    <if id="4" net="Net12">
     <ipv4 mask="255.255.255.0">10.0.12.3</ipv4>
    </if>
    <forwarding/>
    <filetree root="/etc/quagga" seq="start">r5</filetree>
      <exec seq="start" type="verbatim">hostname</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/zebra -d</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/ripd -d</exec>
      <exec seq="stop" type="verbatim">hostname</exec>
      <exec seq="stop" type="verbatim">killall zebra</exec>
      <exec seq="stop" type="verbatim">killall ripd</exec>
   </vm>

   <vm name="HostC">
    <if id="1" net="Net6">
      <ipv4 mask="255.255.255.0">10.0.6.3</ipv4>
    </if>
    <route type="ipv4" gw="10.0.6.5">default</route>
   </vm>

  <vm name="R6">
    <if id="1" net="Net8">
      <ipv4 mask="255.255.255.0">10.0.8.5</ipv4>
    </if>
    <if id="2" net="Net10">
      <ipv4 mask="255.255.255.0">10.0.10.3</ipv4>
    </if>
    <if id="3" net="Net17">
     <ipv4 mask="255.255.255.0">10.0.17.5</ipv4>
    </if>
    <forwarding type="ip"/>
      <filetree root="/etc/quagga" seq="start">r6</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/ripd -d</exec>
    <exec seq="stop" type="verbatim">hostname</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>

 <vm name="HostD">
  <if id="1" net="Net10">
    <ipv4 mask="255.255.255.0">10.0.0.5</ipv4>
  </if>
  <route type="ipv4" gw="10.0.0.3">default</route>
 </vm>

<vm name="R7">
   <if id="1" net="Net9">
    <ipv4 mask="255.255.255.0">10.0.9.5</ipv4>
  </if>
  <if id="2" net="Net16">
    <ipv4 mask="255.255.255.0">10.0.16.5</ipv4>
  </if>
  <if id="3" net="Net17">
    <ipv4 mask="255.255.255.0">10.0.17.3</ipv4>
  </if>

  <forwarding type="ip"/>
   <filetree root="/etc/quagga" seq="start">r7</filetree>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/
    quagga/ripd -d</exec>
    <exec seq="stop" type="verbatim">hostname</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ripd</exec>
</vm>

<vm name="R8">
  <if id="1" net="Net12">
    <ipv4 mask="255.255.255.0">10.0.12.5</ipv4>
    </if>
    <if id="2" net="Net13">
      <ipv4 mask="255.255.255.0">10.0.13.5</ipv4>
    </if>
    <if id="3" net="Net15">
      <ipv4 mask="255.255.255.0">10.0.15.3</ipv4>
    </if>
    <if id="4" net="Net16">
     <ipv4 mask="255.255.255.0">10.0.16.3</ipv4>
    </if>

    <forwarding type="ip"/>
     <filetree root="/etc/quagga" seq="start">r8</filetree>
      <exec seq="start" type="verbatim">hostname</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/zebra -d</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/ripd -d</exec>
      <exec seq="stop" type="verbatim">hostname</exec>
      <exec seq="stop" type="verbatim">killall zebra</exec>
      <exec seq="stop" type="verbatim">killall ripd</exec>
  </vm>

   <vm name="HostE">
    <if id="1" net="Net15">
      <ipv4 mask="255.255.255.0">10.0.15.5</ipv4>
    </if>
    <route type="ipv4" gw="10.0.15.3">default</route>
   </vm>

  <vm name="R9">
    <if id="1" net="Net11">
      <ipv4 mask="255.255.255.0">10.0.11.5</ipv4>
    </if>
     <if id="2" net="Net13">
      <ipv4 mask="255.255.255.0">10.0.13.3</ipv4>
    </if>
     <if id="3" net="Net14">
      <ipv4 mask="255.255.255.0">10.0.14.3</ipv4>
    </if>
    <forwarding type="ip"/>
     <filetree root="/etc/quagga" seq="start">r9</filetree>
      <exec seq="start" type="verbatim">hostname</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/zebra -d</exec>
      <exec seq="start" type="verbatim">/usr/lib/
      quagga/ripd -d</exec>
      <exec seq="stop" type="verbatim">hostname</exec>
      <exec seq="stop" type="verbatim">killall zebra</exec>
      <exec seq="stop" type="verbatim">killall ripd</exec>

  </vm>

   <vm name="HostF">
    <if id="1" net="Net14">
      <ipv4 mask="255.255.255.0">10.0.14.5</ipv4>
    </if>
    <route type="ipv4" gw="10.0.14.3">default</route>
   </vm>
</vnuml>
\end{Verbatim}
```

Appendix B

Configuration files for Zebra, RIP and vtysh

In these configuration files, you can specify the debugging options, a vty's password, the `RIP` routing dæmon configurations, a log file name, and so forth.

We wrote three configuration files for each router configured with `RIP`. These files are `zebra.conf`, `ripd.conf` and `vtysh.conf`, and are described below. These brief descriptions are as follows:

- The first file is a default configuration file, and it is called `zebra.conf`. This file, `zebra`, is an IP routing manager and is used to provide kernel routing updates, interface lookups, and the redistribution of routes between different routing protocols [11].
- The second file is a default configuration file, and it is called `ripd.conf`. This configuration file contains a `ripd` dæmon that implements the `RIP` protocol. This `RIP` protocol requires interface information maintained by `zebra` dæmon. It is mandatory to run `zebra` before running `ripd` dæmon, and `zebra` must be

invoked before we use an **ripd** dæmon.

- The third file is **vtysh.conf** file and it configures the virtual terminal — (**vty**). The **vty** is a command line interface (CLI) for user interaction with the routing daemon. Users can connect to the dæmons via the telnet protocol. To enable a **vty** interface, users have to setup a **vty** password.

These files are usually kept in **/etc/quagga** directory of a computer machine. For ease of reference, we will upload all the configuration files for this project to this website: **http://web.unbc.ca/~bankole/** after the project defense.

Below is a set of sample files for router, R1, regarding the configuration files explained above.

B.1 zebra.conf

```
! -*- zebra -*-
!
! zebra sample configuration file
!
! $Id: zebra.conf.sample,17:26:38 developer Exp $
!
hostname R1
 password xxxx
! enable password zebra
!
! Interface's description.
!
!interface lo
! description test of desc.
!
!interface sit0
! multicast

!
```

```
! Static default route sample.
!
!ip route 0.0.0.0/0 203.181.89.241
!

!log file zebra.log
log file /var/log/zebra/zebra.log
```

B.2 ripd.conf

```
! -*- rip -*-
!
! RIPd sample configuration file
!
! $Id: ripd.conf.sample, 17:28:42 developer Exp $
!
hostname ripd
password zebra
!
! debug rip events
! debug rip packet
!
router rip
network 10.0.0.0/8
```

B.3 vtysh.conf

```
! vtysh sample configuration file
!
!username niyibank nopassword
log file /var/log/zebra/vtysh.log
```

Appendix C

The XML file for the virtual GÉANT network

In order to set up dynamic routing with the OSPF routing protocol, we configure the virtual network testbed with OSPF. This protocol is widely used in large networks such as enterprise networks and ISPs because it converges very quickly. By convergence, we refer to the time it takes to respond to changes in the network. These changes could occur due to link and router failures.

We need three separate configuration files - **zebra.conf**, **ospfd.conf** and **vtysh.conf** for each of the twenty-three routers. In the **ospfd.conf** file, each router defines the subnets and the OSPF areas that make up the network. Both **zebra.conf** and **vtysh.conf** resemble equivalent files that we already explained in Subsection 3.3.2. In the OSPF configuration file, we specify the debugging options, routing dæmon configurations and the name of the log file. We write three configuration files for each of the twenty-three routers. We use these configuration files in the XML specification files to create the virtual network. See sample of the three configuration files in Appendix E.

On the host machine, we locate the XML file and then start or stop these routing dæmons by specifying the necessary commands.

The XML file is stored in /usr/share/vnuml/NIYIospf.xml directory of a host machine, and a copy of this XML specification is included in the report as follows.

C.1 A twenty-three node virtual network testbed

The following XML file describes a scenario of twenty-three nodes to be used with UML and VNUML parser to set up a virtual network testbed. This testbed is configured with OSPF to verify whether or not this intra-domain routing protocol is functioning correctly. We also use the testbed to study routing instability in the network. Below is the script for the XML specifications.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">

<vnuml>
  <global>
    <version>1.8</version>
    <simulation_name>newGEANT</simulation_name>
   <automac/>
    <vm_defaults exec_mode="mconsole">
       <filesystem type="cow">/usr/share/vnuml/filesystems/
       root_fs_tutorial</filesystem>
       <kernel>/usr/share/vnuml/kernels/linux</kernel>
       <console id="0">xterm</console>
    </vm_defaults>
  </global>

   <net name="Net1"  mode="uml_switch" />
   <net name="Net2"  mode="uml_switch" />
   <net name="Net3"  mode="uml_switch" />
   <net name="Net4"  mode="uml_switch" />
   <net name="Net5"  mode="uml_switch" />
   <net name="Net6"  mode="uml_switch" />
  <net name="Net7"  mode="uml_switch" />
  <net name="Net8"  mode="uml_switch" />
  <net name="Net9"  mode="uml_switch" />
  <net name="Net10"  mode="uml_switch"/>
  <net name="Net11"  mode="uml_switch"/>
  <net name="Net12"  mode="uml_switch" />
  <net name="Net13"  mode="uml_switch" />
  <net name="Net14"  mode="uml_switch" />
  <net name="Net15"  mode="uml_switch"/>
  <net name="Net16"  mode="uml_switch"/>
  <net name="Net17"  mode="uml_switch" />
  <net name="Net18"  mode="uml_switch" />
  <net name="Net19"  mode="uml_switch" />
  <net name="Net20"  mode="uml_switch" />
  <net name="Net21"  mode="uml_switch" />
  <net name="Net22"  mode="uml_switch" />
  <net name="Net23"  mode="uml_switch" />
  <net name="Net24"  mode="uml_switch" />
  <net name="Net25"  mode="uml_switch" />
  <net name="Net26"  mode="uml_switch" />
  <net name="Net27"  mode="uml_switch" />
  <net name="Net28"  mode="uml_switch"/>
  <net name="Net29"  mode="uml_switch"/>
  <net name="Net30"  mode="uml_switch" />
  <net name="Net31"  mode="uml_switch" />
  <net name="Net32"  mode="uml_switch" />
  <net name="Net33"  mode="uml_switch"/>
  <net name="Net34"  mode="uml_switch"/>
  <net name="Net35"  mode="uml_switch" />
  <net name="Net36"  mode="uml_switch"/>
  <net name="Net37"  mode="uml_switch" />
  <net name="Net38"  mode="uml_switch" />

   <vm name="R1" order="">
   <if id="1" net="Net1">
     <ipv4 mask="255.255.255.0">10.0.1.4</ipv4>
   </if>
   <if id="2" net="Net2">
     <ipv4 mask="255.255.255.0">10.0.2.4</ipv4>
   </if>
   <forwarding type="ip"/>
   <filetree root="/etc/quagga" seq="start">R1</filetree>
   <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
```

```
     conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">hostname</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

  <vm name="R2">
    <if id="1" net="Net2">
      <ipv4 mask="255.255.255.0">10.0.2.8</ipv4>
    </if>
    <if id="2" net="Net5">
      <ipv4 mask="255.255.255.0">10.0.5.4</ipv4>
    </if>
    <if id="3" net="Net15">
      <ipv4 mask="255.255.255.0">10.0.15.8</ipv4>
    </if>

    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R2</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

  <vm name="R3">
    <if id="1" net="Net1">
      <ipv4 mask="255.255.255.0">10.0.1.8</ipv4>
    </if>
    <if id="2" net="Net4">
      <ipv4 mask="255.255.255.0">10.0.4.4</ipv4>
    </if>
    <if id="3" net="Net3">
      <ipv4 mask="255.255.255.0">10.0.3.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R3</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

  <vm name="R4">
    <if id="1" net="Net4">
      <ipv4 mask="255.255.255.0">10.0.4.8</ipv4>
    </if>
     <if id="2" net="Net15">
      <ipv4 mask="255.255.255.0">10.0.15.4</ipv4>
    </if>
     <if id="3" net="Net16">
      <ipv4 mask="255.255.255.0">10.0.16.4</ipv4>
    </if>
    <if id="4" net="Net17">
      <ipv4 mask="255.255.255.0">10.0.17.4</ipv4>
    </if>
    <forwarding type="ip"/>

    <filetree root="/etc/quagga" seq="start">R4</filetree>
   <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
   </vm>

  <vm name="R5">
    <if id="1" net="Net5">
      <ipv4 mask="255.255.255.0">10.0.5.8</ipv4>
    </if>
     <if id="2" net="Net6">
      <ipv4 mask="255.255.255.0">10.0.6.4</ipv4>
     </if>
     <if id="3" net="Net12">
      <ipv4 mask="255.255.255.0">10.0.12.4</ipv4>
     </if>
    <forwarding type="ip"/>

    <filetree root="/etc/quagga" seq="start">R5</filetree>
   <exec seq="start" type="verbatim">for f in /proc/sys/net/
   ipv4/conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
   <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
   <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
   <exec seq="stop" type="verbatim">killall zebra</exec>
   <exec seq="stop" type="verbatim">killall ospfd</exec>
 </vm>

 <vm name="R6">
    <if id="1" net="Net6">
     <ipv4 mask="255.255.255.0">10.0.6.8</ipv4>
   </if>
    <if id="2" net="Net7">
     <ipv4 mask="255.255.255.0">10.0.7.4</ipv4>
   </if>
    <if id="3" net="Net8">
     <ipv4 mask="255.255.255.0">10.0.8.4</ipv4>
   </if>
   <forwarding type="ip"/>
   <filetree root="/etc/quagga" seq="start">R6</filetree>
   <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
   <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
   <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
   <exec seq="stop" type="verbatim">killall zebra</exec>
   <exec seq="stop" type="verbatim">killall ospfd</exec>
 </vm>

 <vm name="R7">
   <if id="1" net="Net7">
     <ipv4 mask="255.255.255.0">10.0.7.8</ipv4>
   </if>
   <if id="2" net="Net10">
     <ipv4 mask="255.255.255.0">10.0.10.4</ipv4>
   </if>
   <if id="3" net="Net11">
     <ipv4 mask="255.255.255.0">10.0.11.4</ipv4>
   </if>
   <forwarding type="ip"/>
   <filetree root="/etc/quagga" seq="start">R7</filetree>
   <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
   conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
   <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
```

```
  <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ospfd</exec>
</vm>

<vm name="HostA">
  <if id="1" net="Net10">
    <ipv4 mask="255.255.255.0">10.0.10.8</ipv4>
  </if>
  <route type="ipv4" gw="10.0.10.4">default</route>
  <forwarding type="ip"/>
</vm>

<vm name="R8">
  <if id="1" net="Net8">
    <ipv4 mask="255.255.255.0">10.0.8.8</ipv4>
  </if>
   <if id="2" net="Net9">
    <ipv4 mask="255.255.255.0">10.0.9.4</ipv4>
  </if>
   <if id="3" net="Net32">
    <ipv4 mask="255.255.255.0">10.0.32.4</ipv4>
  </if>
  <forwarding type="ip"/>
  <filetree root="/etc/quagga" seq="start">R8</filetree>
  <exec seq="start" type="verbatim">for f in /proc/sys/net/
  ipv4/conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
  <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
  <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
  <exec seq="stop" type="verbatim">killall zebra</exec>
  <exec seq="stop" type="verbatim">killall ospfd</exec>

</vm>

<vm name="R9">
  <if id="1" net="Net9">
    <ipv4 mask="255.255.255.0">10.0.9.8</ipv4>
  </if>
   <if id="2" net="Net26">
    <ipv4 mask="255.255.255.0">10.0.26.4</ipv4>
  </if>
   <if id="3" net="Net14">
    <ipv4 mask="255.255.255.0">10.0.14.8</ipv4>
    </if>
    <if id="4" net="Net11">
      <ipv4 mask="255.255.255.0">10.0.11.8</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R9</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

<vm name="R10">
    <if id="1" net="Net16">
      <ipv4 mask="255.255.255.0">10.0.16.8</ipv4>
    </if>
     <if id="2" net="Net18">
      <ipv4 mask="255.255.255.0">10.0.18.4</ipv4>
    </if>
    <if id="3" net="Net36">
      <ipv4 mask="255.255.255.0">10.0.36.4</ipv4>
    </if>
    <if id="4" net="Net12">
      <ipv4 mask="255.255.255.0">10.0.12.8</ipv4>
     </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R10</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
 </vm>


<vm name="R17">
    <if id="1" net="Net21">
      <ipv4 mask="255.255.255.0">10.0.21.8</ipv4>
    </if>
    <if id="2" net="Net24">
```

```
      <ipv4 mask="255.255.255.0">10.0.24.4</ipv4>
    </if>
    <if id="3" net="Net14">
      <ipv4 mask="255.255.255.0">10.0.14.4</ipv4>
    </if>
    <if id="4" net="Net32">
      <ipv4 mask="255.255.255.0">10.0.32.8</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R17</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
     conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

 <vm name="R19">
    <if id="1" net="Net24">
      <ipv4 mask="255.255.255.0">10.0.24.8</ipv4>
    </if>
     <if id="2" net="Net25">
      <ipv4 mask="255.255.255.0">10.0.25.8</ipv4>
    </if>
     <if id="3" net="Net26">
      <ipv4 mask="255.255.255.0">10.0.26.8</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R19</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
     conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

 <vm name="R11">
    <if id="1" net="Net17">
      <ipv4 mask="255.255.255.0">10.0.17.8</ipv4>
    </if>
     <if id="2" net="Net19">
```

```
      <ipv4 mask="255.255.255.0">10.0.19.4</ipv4>
    </if>
    <if id="3" net="Net23">
      <ipv4 mask="255.255.255.0">10.0.23.8</ipv4>
    </if>
    <if id="4" net="Net36">
      <ipv4 mask="255.255.255.0">10.0.36.8</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R11</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
     conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

<vm name="R12">
    <if id="1" net="Net18">
      <ipv4 mask="255.255.255.0">10.0.18.8</ipv4>
    </if>
     <if id="2" net="Net19">
      <ipv4 mask="255.255.255.0">10.0.19.8</ipv4>
    </if>
     <if id="3" net="Net20">
      <ipv4 mask="255.255.255.0">10.0.20.4</ipv4>
    </if>
     <if id="4" net="Net27">
      <ipv4 mask="255.255.255.0">10.0.27.8</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R12</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

<vm name="R13">
    <if id="1" net="Net13">
```

```
      <ipv4 mask="255.255.255.0">10.0.3.8</ipv4>
    </if>
     <if id="2" net="Net13">
      <ipv4 mask="255.255.255.0">10.0.13.4</ipv4>
    </if>
    <if id="3" net="Net23">
      <ipv4 mask="255.255.255.0">10.0.23.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R13</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

<vm name="R14">
    <if id="1" net="Net13">
      <ipv4 mask="255.255.255.0">10.0.13.8</ipv4>
    </if>
     <if id="2" net="Net28">
      <ipv4 mask="255.255.255.0">10.0.28.4</ipv4>
    </if>
    <if id="3" net="Net27">
      <ipv4 mask="255.255.255.0">10.0.27.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R14</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

<vm name="R15">
    <if id="1" net="Net28">
      <ipv4 mask="255.255.255.0">10.0.28.8</ipv4>
    </if>
     <if id="2" net="Net29">
```

```
      <ipv4 mask="255.255.255.0">10.0.29.4</ipv4>
    </if>
     <if id="3" net="Net30">
      <ipv4 mask="255.255.255.0">10.0.30.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R15</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
     conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

<vm name="R16">
    <if id="1" net="Net20">
      <ipv4 mask="255.255.255.0">10.0.20.8</ipv4>
    </if>
     <if id="2" net="Net21">
      <ipv4 mask="255.255.255.0">10.0.21.4</ipv4>
    </if>
     <if id="3" net="Net22">
      <ipv4 mask="255.255.255.0">10.0.22.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R16</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

 <vm name="R18">
    <if id="1" net="Net22">
      <ipv4 mask="255.255.255.0">10.0.22.8</ipv4>
    </if>
     <if id="2" net="Net25">
      <ipv4 mask="255.255.255.0">10.0.25.4</ipv4>
    </if>
     <if id="3" net="Net38">
```

```
      <ipv4 mask="255.255.255.0">10.0.38.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R18</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
   conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>

  </vm>

 <vm name="R20">
    <if id="1" net="Net30">
      <ipv4 mask="255.255.255.0">10.0.30.8</ipv4>
    </if>
     <if id="2" net="Net31">
      <ipv4 mask="255.255.255.0">10.0.31.4</ipv4>
    </if>
     <if id="3" net="Net34">
      <ipv4 mask="255.255.255.0">10.0.34.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R20</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

 <vm name="R21">
    <if id="1" net="Net31">
      <ipv4 mask="255.255.255.0">10.0.31.8</ipv4>
    </if>
    <if id="2" net="Net33">
      <ipv4 mask="255.255.255.0">10.0.33.4</ipv4>
    </if>
    <forwarding type="ip"/>
    <filetree root="/etc/quagga" seq="start">R21</filetree>
    <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
```

```
      conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
     <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
     <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
     <exec seq="stop" type="verbatim">killall zebra</exec>
     <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

 <vm name="R22">
     <if id="1" net="Net34">
       <ipv4 mask="255.255.255.0">10.0.34.8</ipv4>
     </if>
      <if id="2" net="Net35">
       <ipv4 mask="255.255.255.0">10.0.35.4</ipv4>
     </if>
      <if id="3" net="Net37">
       <ipv4 mask="255.255.255.0">10.0.37.4</ipv4>
     </if>
     <if id="4" net="Net29">
       <ipv4 mask="255.255.255.0">10.0.29.8</ipv4>
     </if>
     <forwarding type="ip"/>
     <filetree root="/etc/quagga" seq="start">R22</filetree>
     <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
     conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
     <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
     <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
     <exec seq="stop" type="verbatim">killall zebra</exec>
     <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

 <vm name="R23">
     <if id="1" net="Net33">
       <ipv4 mask="255.255.255.0">10.0.33.8</ipv4>
     </if>
      <if id="2" net="Net35">
       <ipv4 mask="255.255.255.0">10.0.35.8</ipv4>
     </if>
      <if id="3" net="Net38">
       <ipv4 mask="255.255.255.0">10.0.38.8</ipv4>
     </if>
     <forwarding type="ip"/>
     <filetree root="/etc/quagga" seq="start">R23</filetree>
     <exec seq="start" type="verbatim">for f in /proc/sys/net/ipv4/
```

```
    conf/*/rp_filter; do echo 0 &gt; $f; done</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/zebra -d</exec>
    <exec seq="start" type="verbatim">/usr/lib/quagga/ospfd -d</exec>
    <exec seq="stop" type="verbatim">killall zebra</exec>
    <exec seq="stop" type="verbatim">killall ospfd</exec>
  </vm>

  <vm name="HostB">
    <if id="1" net="Net37">
      <ipv4 mask="255.255.255.0">10.0.37.8</ipv4>
    </if>
    <route type="ipv4" gw="10.0.37.4">default</route>
    <forwarding type="ip"/>
  </vm>

</vnuml>
```

Appendix D

Configuration files for Zebra, Ospfd and Vtysh

In these configuration files, you can specify the debugging options, a vty's password, the `ospfd` routing dæmon configurations, a log file name, and so forth.

We describe the three configuration files: `zebra.conf`, `ospfd.conf` and `vtysh.conf`.

- The default configuration file name is `zebra.conf`. This file, `zebra`, is an IP routing manager and is used to provide kernel routing updates, interface lookups, and the redistribution of routes between different routing protocols [11].

- The default configuration file name is `ospfd.conf`. The `ospfd` dæmon implements the `OSPF` protocol which supports `OSPF version 2`. This `OSPF` protocol requires interface information maintained by `zebra` dæmon. Running `zebra` is mandatory before running `ospfd`, so `zebra` must be invoked before `ospfd`.

- The `vtysh.conf` file configures the virtual terminal — (`vty`). The `vty` is a command line interface (CLI) for user interaction with the routing daemon. Users can connect to the dæmons via the telnet protocol. To enable a **vty** interface, users have to setup a **vty** password.

These files are usually kept in **/etc/quagga** directory of a host machine. For ease of reference, we will upload all the configuration files for this project to this website: **http://web.unbc.ca/∼bankole/** after the project defense.

Below is a set of sample files for router, R1, regarding the configuration files explained above.

D.1 zebra.conf

```
! -*- zebra -*-
!
! zebra sample configuration file
!
hostname R1
password xxxx
!  enable password zebra
!
!  Interface's description.
!
!  interface lo
!    description test of desc.
!
!  interface sit0
!    multicast
!
!  Static default route sample.
!
!  ip route 0.0.0.0/0
!
!  log file zebra.log
log file /tmp/zebra.log
!
```

D.2 Ospfd.conf

```
! Config by Julius
! OSPF configuration
!
hostname R1
password xxxx
log file /tmp/ospfd.log
log stdout
!
debug ospf packet all send
!
!  interface dummy0
!
interface eth1
  ip ospf cost 10
!
interface eth2
  ip ospf cost 10
!
interface eth3
!
!  interface gre0
!
!  interface lo
!
!  interface sit0
!
!  interface teql0
!
!  interface tunl0
!
router ospf
!ospf router-id 10.0.0.255
!ospf rfc1583compatibility
!network 10.0.0.0/24 area 0.0.0.0
 network 10.0.1.0/24 area 0.0.0.0
 network 10.0.2.0/24 area 0.0.0.0
!
line vty
```

D.3 vtysh.conf

```

! vtysh sample configuration file
!
!username niyibank nopassword
log file /var/log/zebra/vtysh.log
```

Appendix E

Electronic version of my Project Report

I had promised in my project report, to make available to prospective users and students; my two virtual network testbeds that were used for experiments in my research. Students are allowed to copy, modify and re-configure both testbeds for their use and education. The configuration files will be available on my personal homepage.

For ease of reference, we will upload an electronic version of this project report to this website: `http://web.unbc.ca/~bankole/` after the project defense.