An Investigation into Network Emulation as a Testing Platform

Robert Game

Abstract

In recent years technology for large enterprise has advanced significantly due to the introduction of virtualisation of infrastructure. This has allowed even the smallest of enterprise to adopt low-cost, virtual copies of their Production servers as a test bed and facilitates the segregation of Production and Development by holding a copy of the Production server for testing in Pre-Production. This means that when planning changes to a server it can first be tested in order to ensure that no unanticipated issues occur. This practise has been well adopted on the server infrastructure, however, when looking at network infrastructure there has not been significant uptake in the use of virtualisation to facilitate testing in this way. Many Network Operations teams rely on the use of a physical lab for the testing of changes, however these labs take considerable time to set up both from a cabling perspective and from configuring each device. This project aims to investigate more efficient methods of testing network changes through the use of router virtualisation. In order to do this I will investigate the Cisco's new product VIRL and how it can be used to create a tool which will provide a testing platform for network changes. In addition to simulating the network infrastructure this project will also investigate meaningful methods of creating testing evidence for use in large infrastructure. When implementing network changes in large infrastructure this is particularly useful evidence to have as Change Management teams continuously seek to avoid risk.

Contents

Chapter 1

Introduction

In recent years the use of a second, entirely separate infrastructure has been used to develop tools, test changes and simulate outages of IT Systems. In a large scale infrastructure these environments are usually created for testing changes on Server or Application environments, however the need for real traffic and the cost of dedicated hardware makes this difficult in the Network Infrastructure. The use of a physical lab takes time to set up and teams usually do not use them unless they are planning for large scale infrastructure changes.

This project will aim to create a tool intended for use in a Network Operations team in large scale enterprise. This tool will aim to pull together different aspects of Network Management tools and use information available to create a network model inside of the Cisco VIRL emulated environment. Using this model the Network Operations team can test environment changes on a model of their own network. Using data gathered from this simulation and the changes observed during any configuration edits the tool will then aim to create a piece of testing evidence which can be used to support any Change Management required on the infrastructure.

Large organisations, which are the primary target of the proposed tool, will often have Configuration Repositories and Configuration Management Databases (CMDB) that store relevant configurations and server locations. Alternatively, a networks team may want to generate a emulated version of an unknown network in order to understand its topology and configuration. The proposed tool will combine network emulation technology with these data sources to allow an Administrator to mimic changes to a production network and observe the impact that they have on data flow and connectivity. In particular the tool will generate testing evidence as proof of any impact (or lack of) when raising a network change.

The proposed tool provides value to a Network Engineer as it allows them to be more flexible in their network testing. With the proposed solution an Engineer will no longer need to create a physical clone of the network in a lab environment, nor will the need rely on outdated documentation. A emulated copy of the network can be produced from a node-edge representation of the topology along with the configurations for devices. This will save the Engineer time in the planning as well as allowing them to implement the changes in a non-production environment, in addition to the time savings an Engineer will also be able to produce testing evidence which can be used for Change Management purposes.

From initial findings it is understood that there are currently solutions for creating a simulated network, however, these tools rely on an Engineer to rebuild a network from scratch. Imple-

menting the topology and interconnecting devices requires a well-documented network and in addition to the physical layout, the simulated devices also require a configuration which matches the real network. This project will aim to create a tool which will provide use to a Network Engineer regardless of the documentation they have for their infrastructure, allowing the Engineer to produce an emulated copy of their physical network without the need to reproduce the infrastructure on a device-by-device basis. The tool will produce an emulation of the network as a whole rather than focusing on devices.

1.1 Scope

This solution is targeted at medium to large enterprise which have a considerable routing set-up For example a global or regional backbone which connects multiple branch offices to a Headquarters. My tool will support both Interior and Exterior Routing protocols. In particular I am using my experience in the finance industry where Network Operations teams also work in an ISP role connecting through EBGP as well as traditional interior Distance Vector or Link State routing. This will create a tool that is scalable to the users needs, allowing a user to create a emulated copy of their local network through dynamic device discovery or a larger scale Engineer to produce an emulated version of a section of their network using Configuration Repositories and Network Diagrams.

1.2 Objectives

- Create a tool which can take information from existing resources and create a working model of the Network Infrastructure within Cisco VIRL.
- Extend the tool so that it can dynamically discover new network devices.
- Facilitate the creation of a simulation within five minutes of starting the tool.
- Produce a visual representation of the network for the user to view
- Allow the user to interact with the emulation through traditional methods such as SSH or telnet
- Investigate new methods for an Engineer to interact with a network
- The production of detailed testing evidence during the simulation, such as neighbour tables, routing entries or link statuses.

Chapter 2

Literature Review

This chapter will review current publications in the field of network simulation and testing, I will use this literature first to gain inspiration for this project but also as a gap-analysis of the proposed tool. It will first focus on the use of Virtualisation in computer infrastructure, before looking at the use of network management techniques to gather information about a the infrastructure before finally focusing on 'What-if?' scenarios on a network and which tests are of most beneficial to a Network Engineer.

2.1 Virtualisation of Infrastructure

Virtualisation of infrastructure components has grown rapidly in the previous decade with the implementation of Cloud Computing being "one of the most explosively expanding technologies in the computing industry today" (Younge et al., 2011).

The technology relies on creating a sandbox within a physical host in which "multiple operating systems can safely coexist on one physical machine" (Mergen et al., 2006) these virtual operating systems are governed by a tool known as a Hypervisor or Virtual Machine Monitor which "safely multiplexes the hardware resources of the physical machine but leaves the specific hardware resource allocation to the operating system in the virtual machine" (Mergen et al., 2006) creating the illusion to these virtual operating systems that they have full control of their hardware when in fact they are running in a sandbox environment cut-off from the 'real' hardware.

The benefits of server virtualisation come in both a technical and economic form. (Crosby and Brown, 2006) says "the artifacts of current operating-system and system-software architecture result in most servers today running at under 10 percent utilization." from a technical perspective the implementation of a virtual server for multiple operating system instances versus a separate physical host each allows for much better utilisation of a single server and thus provides gains in capacity management for infrastructure. Economic benefits come from the amount of physical hardware required in a Data Centre, (Daniels, 2009) says "Data center floor space and rack space are prime real estate in computing environments. Cooling and electricity costs have risen in recent years" and the adoption of virtual infrastructure can help keep costs down which allows enterprises to implement practises such as High Availability services.

At first glance the adoption of virtual infrastructure may seem to only bring benefits to enterprise, however (Kotsovinos, 2010) highlights some issues that arise with implementing virtualisation technology in large infrastructure. In this article the author talks about issues such as System Sprawl where 'developers forget to return the VMs they do not use to the pool after the end of a project' creating problems in accountability and server ownership. In addition to this there is also a breakdown in the conventional responsibility of teams (silos) as issues can arise from multiple areas of infrastructure, the author says 'cross-silo collaboration and communication are of paramount importance, requiring a true mentality shift in the way enterprise infrastructure organizations operate' which may be difficult to implement. A third issue that arises is the added complexity of changes as 'Sharing the infrastructure comes with centralization and, therefore, with potential bottlenecks that are not as well understood' this creates additional complexity in Change Management as it is no longer as clear how a decision can affect the infrastructure as a whole.

From the literature it is clear that virtualisation is a technology which is gaining momentum in large enterprise. The use of which allows engineers to make use of their infrastructure efficiently in both capacity and cost. It is clear that virtualisation is the correct method of implementing a network testing tool due to its scalability and ease of implementation when compared to physical equipment.

2.2 Virtualisation of Network Components

From a testing perspective, virtualisation in the network infrastructure comes in the form of device emulation. (Galán et al., 2004) discusses the use of virtualisation tools within a network lab, in this paper the author describes a scenario in which they have used a virtual network as a platform for learning. Features such as the ability to quickly create a simulation and the ease of restoring defaults when problems arise are highlighted as a significant benefit of simulation over real devices, particularly in a learning environment where misconfigurations are likely to occur. However, the author also highlights opportunities that are missed with simulated devices, such as having no physical contact with the routers means that hands-on experience is lost.

(Knight et al., 2012) says "Emulated networks, which run a real router operating systems inside virtual machines, offer realistic yet inexpensive network experimentation. However they are time-consuming to configure, which limits their use in network research". In this paper there is a proposal of a tool to simplify the creation of emulated networks, Autonetkit, which provides the ability to generate thousands of lines of configuration code across multiple devices from a high-level description of the topology as a whole. In particular this paper highlights the difficulty faced when creating an emulated network from scratch. When dealing with networks of hundreds or possibly thousands of nodes the time taken to configure each can outstrip the benefits of an emulated network.

This section of the Literature Review has given views on both the benefits and shortfalls of using virtualisation in network infrastructure. A significant benefit to network emulation is cost, with physical devices there is a significant investment in a single device with most requiring a support contract. For a large organisation this may be seen as an unnecessary expenditure, whereas the licensing cost for a VM may be considerably more cost effective. One shortfall of emulation that has been identified in the literature is the difficulty of defining a network from scratch, this will be taken into consideration when designing the proposed solution.

2.3 Network Management and What-if? Scenarios

Configuration and Change Management is a particularly important aspect in modern infrastructure systems, in particular improved vigilance on Change Management to Networks can improve reliability and uptime in for systems as a whole. (Bellovin and Bush, 2009, p.270-273) discusses these aspects both from a network and whole infrastructure perspective. In the paper's case study it talks of the importance of router configuration management to a large Internet Service Provider, one particular use of configuration management is understanding a malicious attack by viewing the current configuration as 'the differences between the authorized configuration and the one installed by the intruder can give valuable clues as to goals and motives'

'What-if' scenarios are a section of Network Testing that allows an Engineer to understand how their topology will cope should an event occur. (Lad et al., 2006) discusses a potential solution to the uncertainty of a network change on the Internet, in this paper the author proposes a tool which allows the impact of a BGP update to be understood by providing a visual representation of changes to an Autonomous System's forwarding path. This tool allows an Engineer to understand where traffic is flowing in the event of a change to a network. However, this tool falls short as it only allows an Engineer to view routing changes in real-time instead of gaining an understanding of what could happen.

This section of the Literature Review was particularly difficult as there are very few tools available that allow both emulation of a real network and feedback on changes to the emulated network. This indicates a gap in current solutions which the proposed solution may be able to target.

Chapter 3

Current Solutions and Cisco VIRL

3.1 Current Solutions

3.1.1 Hardware

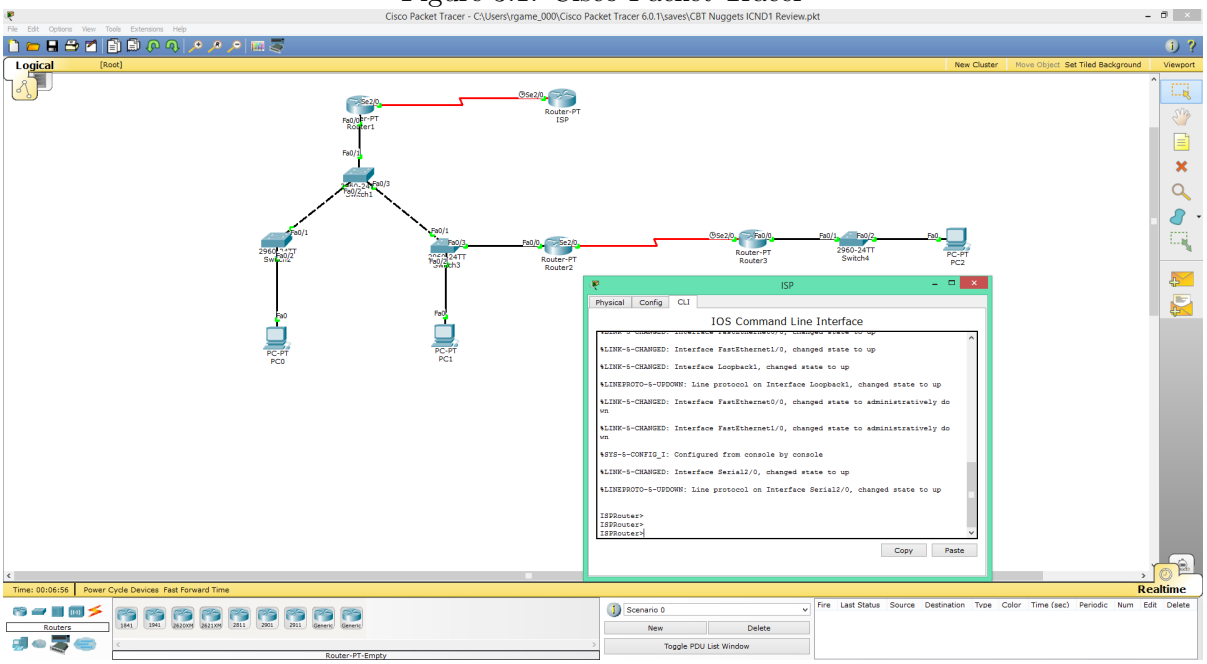
Traditional approaches to Network Testing would rely on hardware to be a 'true' copy of the Production Network. However as infrastructure grows and becomes more complex, the flexibility of this method diminishes. The use of hardware for testing relies on an organisation having a complete network lab on-site which an Engineer can use to build smaller sections of the Production network, replicating the topology by using identical device models to the production counterpart along with the same software version and running configuration. This creates an additional burden on any proposed changes to a Production network as every change requires a physical build of a clone of the network before testing can occur. In addition to the time cost associated with this testing, there is an additional monetary cost to an organisation as they need to hold additional hardware, usually with a large support contract attached, in order to be able to carry out testing. For some smaller enterprise this additional cost may not be viable and some risks may be taken by implementing changes without prior testing.

3.1.2 Simulation

Network Simulation extends the concept of testing by allowing a user to create a simulated network, however, this is not full emulation and is not as reliable as a testing tool when compared to other methods. In recent years simulation of networks has been used as a tool for vendor-specific educational courses such as Cisco's CCNA or Juniper's JNCIA certification programs.

Cisco's offering on the simulation market is an educational tool for basic learning of the Cisco Certified Network Associate (CCNA) certification. Packet Tracer (Cisco Systems, 2013) takes form as a topology view which allows a user to drag and drop routers and switches onto the topology. The devices run a simplified version of the Cisco IOS and allow the user to connect to the devices serial port in order to configure them. An example of Packet Tracer is shown in Figure 3.1.

Figure 3.1: Cisco Packet Tracer



These simulations tend to focus on creating a learning environment for their tools and as such do not perform adequately as simulated 'real world' devices. These devices tend to work as a simulator in that they mimic the behaviour of a network, and not the devices. For example, the software will decide to build an OSPF neighbouring should the required fields match on the devices. In contrast, a simulator (and real device) will build the relationship through a series of negotiations and packet transfers. As a result these simulators cannot be used reliably as network testing tool as the behaviour is predefined and may not be a true reflection of real hardware.

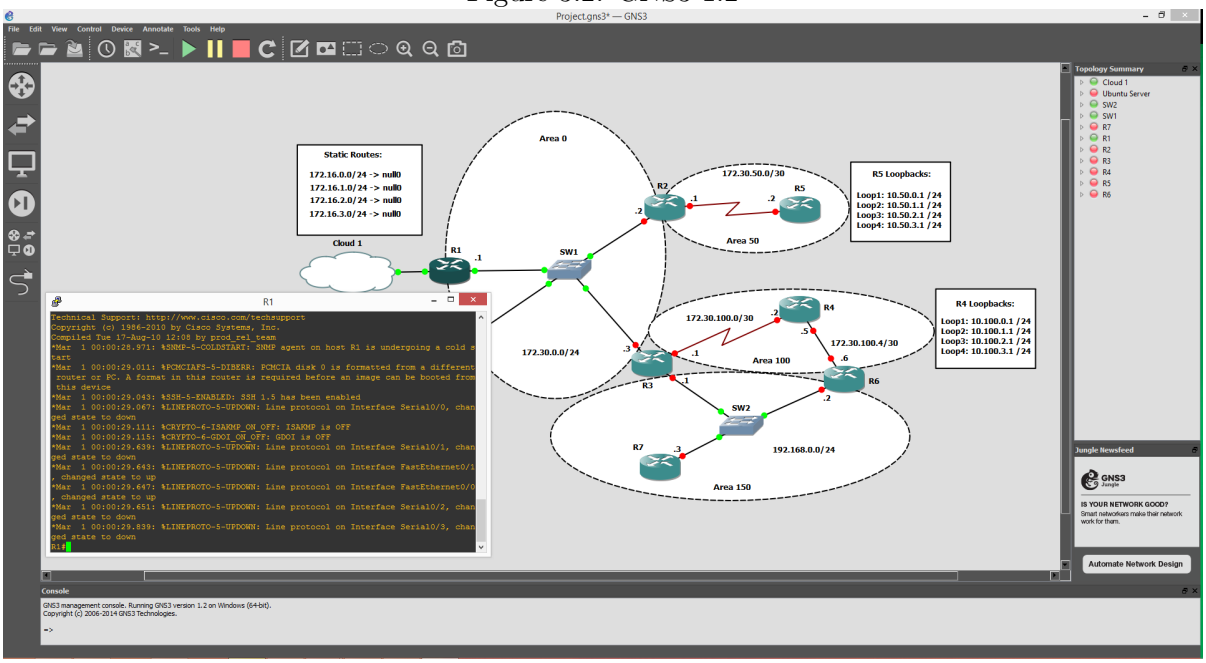
Other network device vendors such as Juniper, HP and Huawei have offered simulated versions of their network equipment which work on similar principals, however these tools have not been a focus of this section as they do not emulate Cisco software.

3.1.3 Emulation

The Open Source community has also played a role in this virtual networks field, a project named GNS3 (Grossmann, 2014) was started in 2006 which aimed to create unified tool to pull together hardware simulation and topology design into one UI. From this they created GNS3, a front end which allowed users to create a topology in a drag and drop style and then start equipment as required. GNS3 is intended to be a multi-vendor tool but is particularly used for Cisco simulations. Using a hypervisor called Dynamips the tool can create a VM of a Cisco router which can be interconnected to create a real topology. In addition to this, GNS3 can be adapted as a network emulator, which enables a user to connect their simulated topology to their physical network.

GNS3 works well for Cisco router simulation, however it falls short when looking at Layer 2 simulation. Features such as Spanning Tree Protocol rely heavily on the fast processing speed

Figure 3.2: GNS3 1.2



of Application Specific Integrated Circuitry (ASICs) which are controllers within the switches that can quickly process frames. Cisco IOS combined with Dynamips cannot process frames fast enough in order to successfully implement these protocols. Therefore when testing redundancy scenarios such as outages which force a Layer 2 topology change cannot be tested. In addition to the issues outlined above there is no support for network discovery and the building of topologies for network testing.

In addition to the shortfalls in some protocols, GNS3 lacks the scalability of some of its competitors. On a conventional PC it can emulated 10-15 devices, however may use up to 90% of the host's CPU if the IOS version is not calibrated for the host PC, because of this lack in scalability it may not be of any use to large enterprise.

From the review of current solutions there has not been a tool which can generate simulations of existing hardware. The only solutions at current are bare-bones building tool for network designs. The proposed tool is different in that its purpose is to be used in existing infrastructure to test disaster recovery or planned configuration changes.

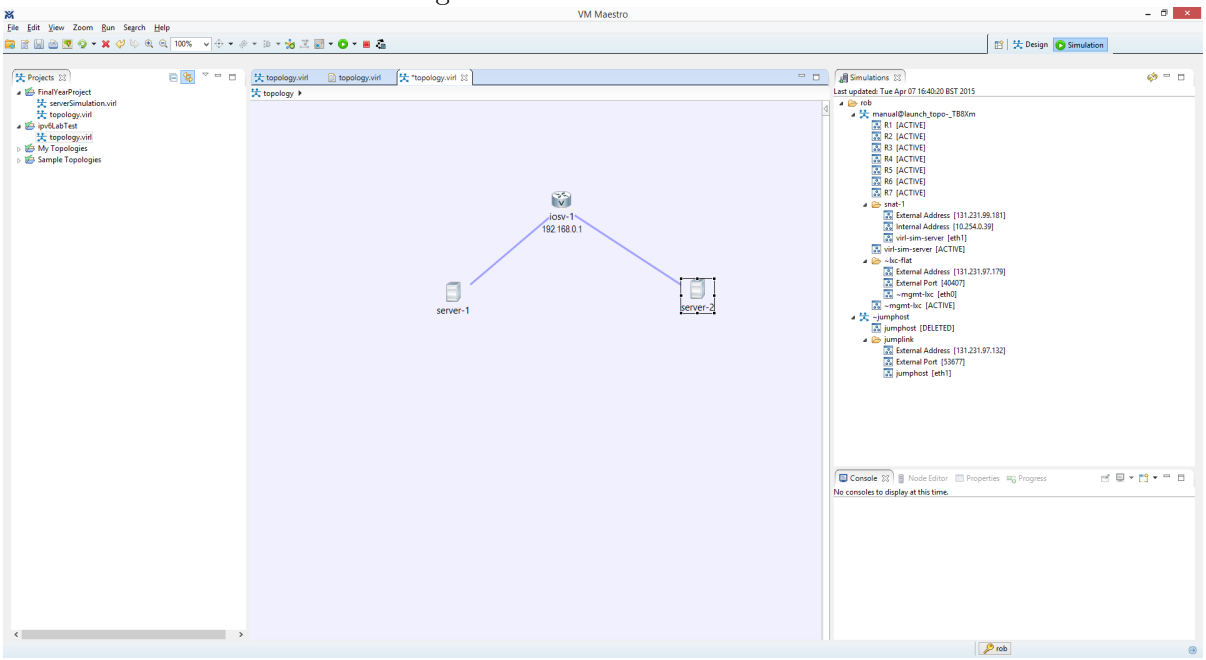
3.2 Cisco Virtual Internet Routing Lab (VIRL)

Cisco VIRL is a relatively new offering on the Virtualisation market, VIRL is pre-release software which forms the back end for a future Cisco product called Cisco Modelling Labs. This tool provides emulation using a new version of Ciscos IOS and NexusOS, tailor made for virtualisation and fast deployment.

VIRL is a relatively new offering from Cisco, and is their first software specifically created to support virtual copies of their routers. It was first released to the public in November 2014 as a stand-alone client installed on a local machine. However for the purposes of this Project I have

been fortunate enough to have access to the University's own server-based version of VIRL. This provides support for up to 200 instances in comparison to the Personal Edition which is capped at 15 instances. This has enabled me to take advantage of extra features such as Servers which can be attached to a network topology, providing added features such as network analysis and traffic creation.

Figure 3.3: Cisco VM Maestro



Connection to the VIRL Server is provided though an Eclipse-based front-end called VM Maestro. This interface provides an sandbox to build network topologies using the provided images of Cisco hardware within VIRL. For this project I will be concentrating on the use of Cisco's IOSv image, an image of the traditional IOS but adapted for use with a hypervisor. This allows the image to be run on a server without any dedicated ASICs. VIRL can then interconnect these devices provide connectivity and traffic. Using these connections the Routers can use routing protocols such as OSPF, EIGRP or BGP in order to create routing tables - this enables the user to simulate the Layer 3 functionality of their topology.

VIRL has also been designed with flexibility in mind for its users, one specific feature of note is the support of a tool called Autonetkit. This integration allows VIRL to create a fully functioning topology in a matter of seconds as configurations such as ip addressing and routing protocols can be passed off to Autonetkit which will auto-generate a working topology.

In addition to the features provided for functionality within VM Maestro, there are also a variety of configurations available to connect VIRL to the outside world. The first available connection is FLAT Networking - this provides a layer 2 interface which can bridge VIRL with physical hardware. This is particularly useful as it allows a user to integrate Layer 2 functionality with their simulation, an area in which VIRL is currently lacking. The second method of connectivity is through a SNAT connection on the VIRL simulation, this allows the simulation to assign external addresses to simulated objects such as Routers and Servers.

Server support within VIRL is provided through pre-defined image varying in system require-

ments. The default images are Ubuntu but other images can be imported if required. These servers can be connected to the internet in order to customise them for to the users needs, an example would be installing an SNMP Server in order to poll the routers for information.

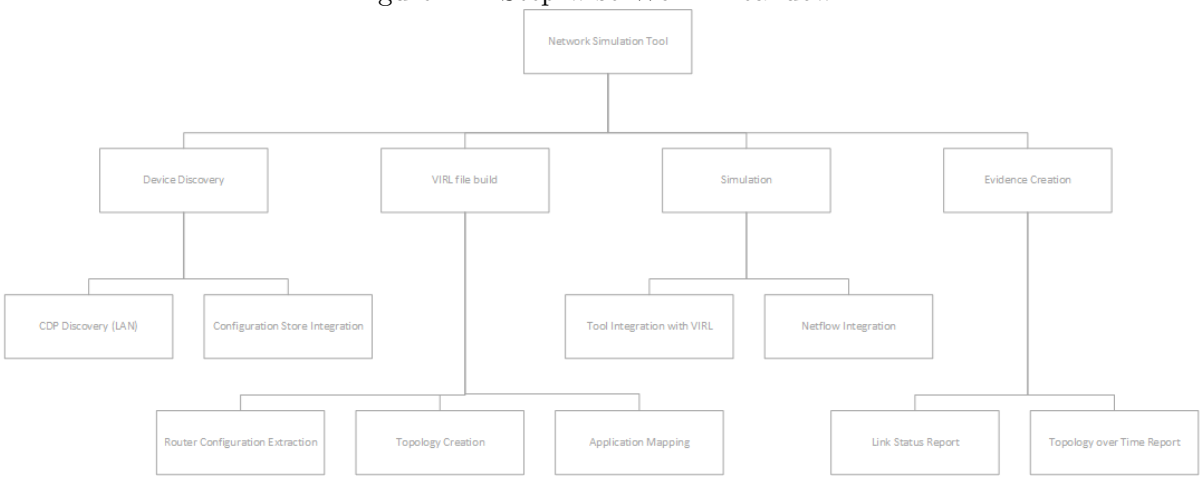
Chapter 4

Proposed Tool

4.1 High-Level Design

At the highest level the proposed tool will provide an Engineer with an emulated version of their physical network for use as a test bed for changes or disaster recovery simulations. At a more granular level the tool is split into four distinct sections: Network Discovery, Network Representation, Emulation and Network Interaction. This chapter will focus on these sections from a design perspective, investigating how they can be implemented and which technologies can support their creation. Figure 4.1 shows these sections further broken down into separate sub-tasks.

Figure 4.1: Step-wise Work Breakdown



The tool will be of a linear format, current expectations will be that the tool will generate a simulation from given inputs, at which point the user will be able to interact with the simulation and feedback of their changes will be available. Due to this linearity, the tool's functionality is best shown as the activity diagram in Figure 4.2

4.2 Network Creation

An emulated network can be created using two distinct inputs, the first is a copy of the configuration files for all devices in the network which will determine how the components interact with one another. For example, a configuration file can provide the settings for a routing protocol such as OSPF, specifying information required to exchange routes with other devices and choosing the networks they wish to advertise to their peers. The second input is a form of detailing the physical layout of the network, representing the devices as nodes and their inter-device links as edges. Representing this data in a format that is easily readable by a program allows the proposed tool to create emulation with all the interfaces and connections of the physical network, an accurate reproduction of these components ensures that the emulation will behave in the same way as its physical counterpart. An example of a Cisco configuration file is in Appendix E.

The proposed tool will provide two options for creating an emulated network. First allowing a user to create a version of a well-known network by allowing the input of already existing configuration backups and topology definitions from tools such as Rancid or Opsware. Alternatively, the user will be able to create an emulated version of their local network by using the tool to discover, map and backup their local devices before passing the information on to the simulation element.

4.2.1 Explicit Input:

A well-maintained network will have two components of management, the first is a detailed collection of network diagrams detailing the positions of routers and switches, the interfaces used for these links and any IP addressing these connections have. Tools used for the creation of these documents are programs such as Microsoft Visio or yED, these tools enable a Network Engineer to draw their diagrams and save them in a user-friendly visual format. A further extension of these tools allows the saving of a topology in a programmer-friendly format such as json or xml, the proposed tool will investigate using this data format for representing the emulated network. Once the topology of the network has been created it has no defined behaviour.

The second management process is a backup tool of network device configurations, large enterprise keep a backup of all router configurations as a precaution in case of unintended changes or hardware failure. If given access to these backups the proposed tool will be able to create an emulated network whose behaviour is defined by these files.

4.2.2 Network Discovery

For networks that do not have the management techniques defined above, the tool will also be able to discover devices and create an emulated version dynamically. As before, the network will require both the topology definition and the device configurations to function. In order to gather this information it will utilise some existing network technologies and standards in order to gather the information required in the most efficient method possible.

In order to map the network topology dynamically, the proposed tool will use a combination of SNMP and CDP/LLDP in order to discover the devices in the local network. These technologies,

when combined will allow for the tool to create a representation of the network in an edge-and-node format which can be parsed in the emulation software. CDP is a protocol created by Cisco which details a devices local neighbours giving information such as the neighbour's hostname, device type, IP Address, the connections local port and destination port. This information can be gathered from a device by using a standard called SNMP (Simple Network Management Protocol)

Network configuration files can be extracted from running devices in multiple ways, the most common is to log on to a device and copy its configuration to a network store such as a tftp (Trivial File Transfer Protocol) server where they are stored for later use. An alternative method is through further use of SNMP to poll a device for its configuration which is returned over a terminal. The tool will use one of these methods to connect to and store a devices configuration. Once both the topology and the configuration of a network is obtained, it can then be passed to Emulation.

4.3 Network Representation

Although the VIRL system provides a topology view for running simulations, I am not intending on using this as part of the proposed tool. Instead, I will investigate methods of displaying a network using the information gathered in the Network Discovery section. I aim to investigate tools such as yED which allows the use of extensions to interact with network drawings. For example, when a topology has been drawn and displayed, it may carry additional attributes such as its IP Address, using this information the graphing software can be extended to implement features such as a terminal emulator for direct access to a device, or an API for sending defined commands to an emulated network. In addition to the interaction with the emulation, I will also investigate methods of representing feedback from devices. This information can be displayed directly on the visual topology with labels such as the status of a network connection and its current throughput. This idea can be extended and the use of colour on a network link can show the forwarding path for a destination IP address from its source. Any changes made to the routing configuration during this time will be reflected by the colour of the links and any impact of the forwarding tables for a destination can be seen.

In addition to the feedback required from a visual representation of a network, there also needs to be a level of interactivity with the network view. A topology can be viewed as its physical layout but it may have other configurations underneath this layer such as Autonomous Systems, Routing Areas or Redundancy Pairs. This information should be viewable by changing a filter on the network representation as it gives a more granular view on how the network components interact with one another. Adding these extra filters will enable a Network Engineer to view the impact of planned changes on additional aspects of the network which may not have been considered previously.

4.4 Network Emulation

The proposed tool will use Cisco's Virtual Internet Routing Lab (VIRL) as its platform for simulation, this was chosen as it provides a full Cisco IOSv image for network components which can scale more efficiently that other simulation platforms. In the proposed tool VIRL

will be used as a back-end technology and the proposed tool will not make use of the Eclipse-based VM Maestro interface. The User can interact with devices by using SSH to connect to a jumphost which will provide access to the components using the management network which VIRL creates.

In order to utilise VIRL to its fullest potential the proposed tool will connect to VIRL using its defined REST API. From initial research it is understood that a simulation can be created by creating a request to the VIRL server and including a defined .virl file. Upon receiving this file the VIRL host will create a simulation based on the on the topology defined in the file. Information about a simulation can also be obtained by making a separate request to the server for a list of running simulations. The proposed tool will make use of both of these features and allow the server to create an emulation, gather information about the emulated network and interact with the network.

4.5 Interaction

The tool will allow a user to interact with their emulated network in a variety of ways. A traditional terminal access will be provided through the web simulation as this is the primary way that an Engineer will interact with and make changes to a real network. In addition to terminal access, the tool will also investigate non-traditional methods of interaction, one benefit of producing a network diagram is the possibility of directly interacting with components in the topology, making changes such as shutting interfaces down or rebooting hardware.

Interaction with network components will provide feedback to the user through the graphical interface provided in the 'Network Representation' section. This section of the tool will display the network dynamically, using information gained from devices such as the forwarding table or bandwidth usage on a link. This information will be interpreted by the graphing software and will provide visual feedback of a network change, such as using coloured links to show a forwarding path through a network.

Connection to the emulation will be provided through a server connected to both a management network within the emulation and a public facing interface for connections from the internet. This server will facilitate the traditional command-line based connections by acting as a jumphost into the network. In addition to command-line access this server will also host a python-based web API which will allow the more unconventional methods of interaction through using networking APIs such as Cisco's OnePK tool set.

4.6 Requirements

Requirement ID:	1
Description:	The system must be able to discover a physical network layout
Rationale:	A user needs to be able to discover and emulate an undocumented network
Type:	Functional
Design Constraint:	No
Fit Criteria:	A user can create an accurate model of their local network

Requirement ID:	2
Description:	The system must accept existing network information
Rationale:	A user needs to be able to create a emulation based on a well-documented network
Type:	Functional
Design Constraint:	No
Fit Criteria:	A user can create an accurate model based on existing network information

Requirement ID:	3
Description:	The system will discover Cisco Devices
Rationale:	The tool needs to discover unknown Cisco devices on a network
Type:	Functional
Design Constraint:	No
Fit Criteria:	All devices on the local topology will be discovered

Requirement ID:	4
Description:	The system will extract configurations from Cisco devices
Rationale:	Configuration files are required in order to emulate the behaviour of devices
Type:	Functional
Design Constraint:	No
Fit Criteria:	The configuration of each discovered device is extracted

Requirement ID:	5
Description:	The system will produce a document detailing the physical layout of the network
Rationale:	A topology document will allow the tool to visually represent a network
Type:	Functional
Design Constraint:	No
Fit Criteria:	A document is produced which details the topology

Requirement ID:	6
Description:	A topology map must be produced in less than 1 minute
Rationale:	Allows the discovery of a network to be efficient so it is of use to a network engineer
Type:	Non-functional
Design Constraint:	No
Fit Criteria:	Device discovery takes one minute or less

Requirement ID:	7
Description:	Router configurations must be extracted in an efficient method
Rationale:	Lessens the time taken to produce a working emulation
Type:	Non-functional
Design Constraint:	No
Fit Criteria:	Configuration Extraction takes three minutes or less

Requirement ID:	8
Description:	Connections to Production devices must not compromise security
Rationale:	Production devices can be left vulnerable if security is lessened
Type:	Non-functional
Design Constraint:	Yes
Fit Criteria:	The tool requires no additional access to systems when compared to an industry standard tool

Requirement ID:	9
Description:	Any Cisco device can be represented within a emulation
Rationale:	A device must be able to be represented in an emulation regardless of model
Type:	Functional
Design Constraint:	No
Fit Criteria:	Any standard Cisco router will be represented in an emulation

Requirement ID:	10
Description:	Network Discover will be able to cope with non-router devices
Rationale:	Not every network will be a collection solely of interconnected routers
Type:	Functional
Design Constraint:	No
Fit Criteria:	The discover will cope with Layer-2 devices such as switches and will deal with them effectively

Requirement ID:	11
Description:	All interface types will be represented in an emulation
Rationale:	Cisco devices have multiple types of interfaces that need to be included
Type:	Functional
Design Constraint:	No
Fit Criteria:	Any standard Cisco interface will be represented in a emulation

Requirement ID:	12
Description:	Physical interfaces will be converted into an equivalent weighting in the emulation
Rationale:	Different interface types will have different characteristics which will influence routing
Type:	Functional
Design Constraint:	No
Fit Criteria:	An interface will influence routing in the same way both in real hardware and in a emulation

Requirement ID:	13
Description:	Configurations will be converted into an equivalent emulation configuration
Rationale:	The emulation must include exactly the same features and characteristics in order to be of use as a testing platform
Type:	Functional
Design Constraint:	No
Fit Criteria:	There will be no configuration loss in the emulation when compared to the real hardware

Requirement ID:	14
Description:	The tool will produce a Cisco VIRL file
Rationale:	The emulation engine, Cisco VIRL represents a topology in a VIRL file
Type:	Functional
Design Constraint:	No
Fit Criteria:	A VIRL file defining the topology is produced

Requirement ID:	15
Description:	The produced emulation file will be a valid VIRL configuration file
Rationale:	The produced file will be passed to an external process for emulation and therefore must be complete and valid
Type:	Non-Functional
Design Constraint:	Yes
Fit Criteria:	All produced VIRL files will be accepted as valid by the VIRL engine

Requirement ID:	16
Description:	The tool will automatically produce a simulation when a valid VIRL file is created
Rationale:	An emulation needs to be produced without any input by the user or any third-part GUI
Type:	Functional
Design Constraint:	No
Fit Criteria:	An emulation is created as part of the tool running

Requirement ID:	17
Description:	The emulated network will be accessible through the internet
Rationale:	The emulation needs a method of access to the network
Type:	Functional
Design Constraint:	No
Fit Criteria:	The emulation is available online

Requirement ID:	18
Description:	The tool will provide a single server as a point of entry to the emulation
Rationale:	Having multiple publicly available devices on the internet poses a security risk
Type:	Functional
Design Constraint:	No
Fit Criteria:	The tool provides one single publicly routable ip address for access

Requirement ID:	19
Description:	The user will be able to view a diagram of the discovered network before emulation
Rationale:	A user will want to check the discovered network is correct before creating an emulation
Type:	Functional
Design Constraint:	No
Fit Criteria:	A diagram is produced before the network is emulated

Requirement ID:	20
Description:	The emulated network will contain a separate management network for the external server to communicate on
Rationale:	A management network will ensure that any management tools will not interfere with the emulation routing
Type:	Functional
Design Constraint:	No
Fit Criteria:	The management of the network is on an entirely separate network to the emulation

Requirement ID:	21
Description:	The user will be able to interact with an emulation in a conventional way
Rationale:	A user will want to interact with the emulation in a familiar way
Type:	Functional
Design Constraint:	No
Fit Criteria:	The user will have command-line access to any router produced in the emulation

Requirement ID:	22
Description:	The user will be able to influence the network in non conventional ways
Rationale:	The tool will provide other methods of influencing the emulated network
Type:	Functional
Design Constraint:	No
Fit Criteria:	The user will influence the network in methods other than CLI

Requirement ID:	23
Description:	The network diagram will produce visual feedback representing the current state of the network
Rationale:	The user will want to see methods other than the CLI for understanding the behaviour of the network
Type:	Functional
Design Constraint:	No
Fit Criteria:	Visual feedback is produced on the CLI

4.7 Risks

The project has the following risks identified which could impact development:

- Constraints on access to University infrastructure
  - External to the VIRL Server there may be a requirement to have an external server which is able to access the VIRL network - as this is in the domain of IT Services, this may not be possible
- VIRL may not provide the functionality this project requires
  - As the current implementation of the Cisco VIRL service is pre-release, there is some aspect of missing functionality or suffers from pre-release bugs
- Added complexity through lack of documentation
  - As Cisco VIRL is still pre-release and limited availability there is a lack of documentation and community support for the product
- Requirement of Python in Cisco Development
  - Cisco products (In particular onePK) require a knowledge of Python. This is a new language that is not well known to the developer. Therefore there may be added difficulty in understanding its function.
- Risk of hardware failure
  - Project may be delayed in the event of hardware failure, particularly the VIRL Server in place which requires a significant time investment to install

4.8 Equipment Used

4.8.1 Servers

For the production of this tool it is understood that two distinct servers are required, one local server which can handle the Network Discovery aspect of the tool and a secondary server which is contained within the VIRL emulation which will handle feedback based on changes to the emulated network. These two servers will be in contact with one-another throughout the course of a tools use. There are two solutions available for these servers, the first being a standard Linux server distribution and the second being a Windows Server installation.

One requirement of the proposed tool is the ability to display a network topology for the user to understand the layout of an emulated network. This will take form in a web browser, therefore there needs to be some aspect of a GUI on the server itself. Because of this, a standard Linux server CLI has been ruled out in favour of a more desktop-like server experience.

A second requirement of the proposed tool is the ability to interact with protocols such as CDP and SNMP. From initial research this level of interaction with packet-level networking is considerably easier and more flexible on a Unix-based operating system as existing tools can be run in a standard shell to interact with CDP. In addition to this, there are additional python libraries which can create and/or intercept packets which require a Unix operating system to run.

Due to the two requirements outlined above, it is proposed that a Linux desktop or server with display server should be used for this work. In particular, Ubuntu will be used for its user-friendliness in comparison to other distributions.

4.8.2 Networking Equipment

The proposed tool is intended to be deployed in a real-world environment, however due to the constraints I have on physical equipment for this project I have opted to create this topology in GNS3 (See Existing Solutions). GNS3 provides full simulation of IOS images and therefore is the closest representation of physical equipment. These simulations were created and run on a Windows PC using the GNS3 client. In addition to router simulation, GNS3 also has support for Virtualbox integration within the tool, this allows Virtual Servers to be connected to a topology and function correctly on a network. I will use this functionality in order to connect the Network Discovery server (as discussed in the previous section) to a separate emulated network. The use of GNS3 rather than real equipment enables a more efficient development process as work does not need to be done on-site with the physical equipment. In addition to this, the use of an emulated network allows a more thorough testing process as a multitude of testing scenarios can be created quickly, enabling more in-depth test procedures.

4.8.3 Python

In addition to physical equipment, the tool will also use Python as its primary language. This has been selected due to its flexibility and existing position in the networking development

world. Due to this foothold Python has an extensive set of libraries which facilitate easier access and processing of network information.

From initial research, Python can be used to create REST network connections which allow it to interface directly with VIRL to create, edit and destroy existing simulations. Selecting Python will enable the tool to seamlessly discover networks, collate network information, create a simulation and interact with a simulation.

4.8.4 CDP - Cisco Discovery Protocol

A second software that has been identified is CDP, a proprietary protocol created by Cisco which facilitates mapping of a network through local neighbour advertisements. This protocol is enabled on all Cisco devices by default, sending an advertisement to all local neighbours every 60 seconds. In this advertisement the device specifies its hostname, whether the device is a router or a switch, the model number of the device and the interface on its local side on which it is sending the advertisement.

From an advertisement the receiving device can piece together all the received information in order to build a picture of its local network. An example of a CDP table for a device is shown below. From this table it can be seen that there are two local routers (R2 and R3) on the local interface FastEthernet0/0. In addition to this information, it can also be seen that R2 and R3 can see R1 on their FastEthernet0/0 interfaces, with further analysis, because R1 can see R2 and R3 on the same local interface it can be inferred that there is a switch connecting all three devices.

```
R1# show cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
S - Switch, H - Host, I - IGMP, r - Repeater

Device ID       Local Intrfce   Holdtme   Capability Platform Port ID
R2              Fas 0/0          157        R S I    3725     Fas 0/0
R3              Fas 0/0          156        R S I    3725     Fas 0/0
```

4.8.5 SNMP - Simple Network Management Protocol

An additional tool that will be used is SNMP, Simple Network Management Protocol (Case et al., 1989) is a protocol designed to allow access to sensors on a networked device. The protocol relies on a Management Information Base (MIB) which defined on a Vendor basis. Inside these MIBs there are defined Object Identifiers (OIDs) which relate to a specific sensor on the device. Access to this OIDs are managed on the device, requiring a community string which a plaintext password in SNMPv2 as a security device to prevent unauthorised access. Additional security is provided by only allowing Read-only access to OIDs which prevent malicious attacks to a system. For the purpose of this project there will only ever be a need for Read-only access, this is due to the fact that Read-Write is disabled on devices due to security concerns.

When making a call to an SNMP Agent, specifying the OID and community string the device returns a plaintext response with the requested information. This can be parsed for and adapted for use in the Network Discovery portion, or alternatively can be used in the Network

Feedback sections in order to understand forwarding information. The following output shows the returned information for a call to OID '.1.3.6.1.2.1.2.2.1.2', the OID which holds information relating to the interfaces on a device.

```
rob@netman-server:~$ snmpwalk -v 2c -c public 172.30.0.1 .1.3.6.1.2.1.2.2.1.2
IF-MIB::ifDescr.1 = STRING: FastEthernet0/0
IF-MIB::ifDescr.2 = STRING: Serial0/0
IF-MIB::ifDescr.3 = STRING: FastEthernet0/1
IF-MIB::ifDescr.4 = STRING: Serial0/1
IF-MIB::ifDescr.5 = STRING: Serial0/2
IF-MIB::ifDescr.6 = STRING: Serial0/3
IF-MIB::ifDescr.8 = STRING: Null0
IF-MIB::ifDescr.13 = STRING: Loopback0
IF-MIB::ifDescr.14 = STRING: NVI0
```

4.8.6 SDN Sources using OnePK

OnePK is Cisco's current offering on the Software Defined Networking marketplace and allows a programmatic approach to controlling and extracting information from Cisco devices.

Software Defined Networking (SDN) is a relatively new concept in the network space and although has not fully been adopted in real-world standards it can be used to extract information from routers with a relatively small footprint in production. Where traditionally a routing table of a Router would be extracted by using a tool such as a screen-scraper or SNMP, the routing table from a OnePK enable device can be requested using an API-like interface. This approach scales more efficiently than using CLI-scraping as the data can be requested through an constant connection rather than opening a telnet/SSH session. Data returned from a OnePK call can either return data in a structured format or as a copy of the text displayed on the CLI. In the case of the structured data the appropriate information can be accessed directly, in the case of screen data it can be extracted using regular expressions.

With regards to this project, OnePK will be used in the Network Feedback section in order to process information such as forwarding tables. In addition to this, it can also be used to control the network as a whole, allowing the tool to make changes to interface states or costs in order to influence routing decisions.

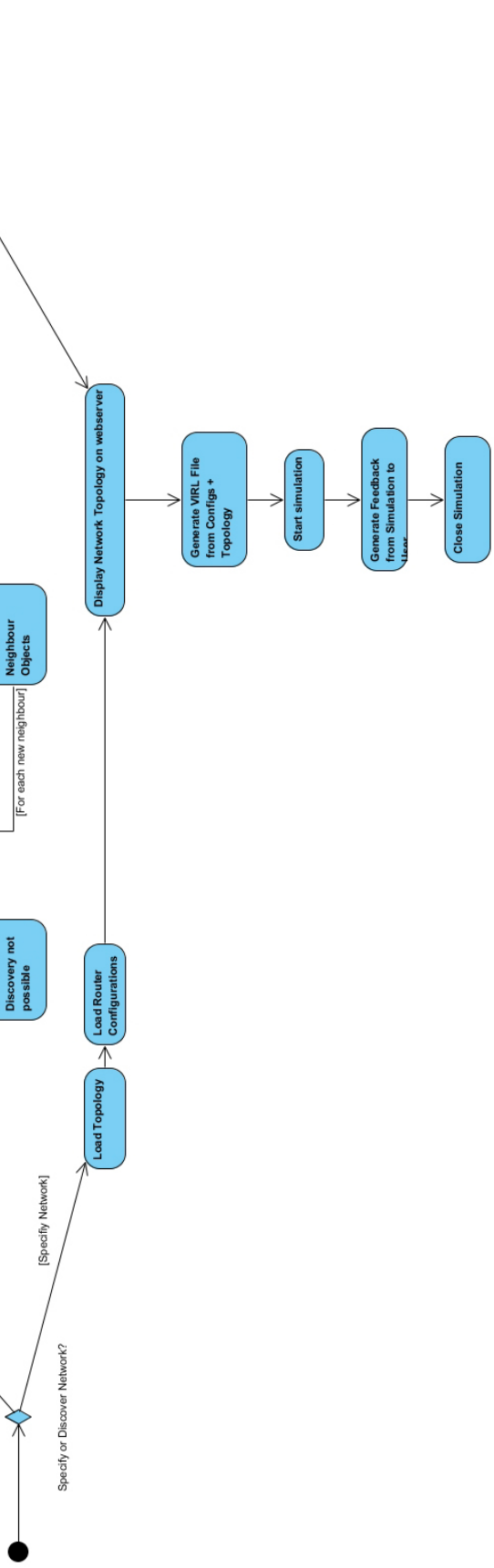


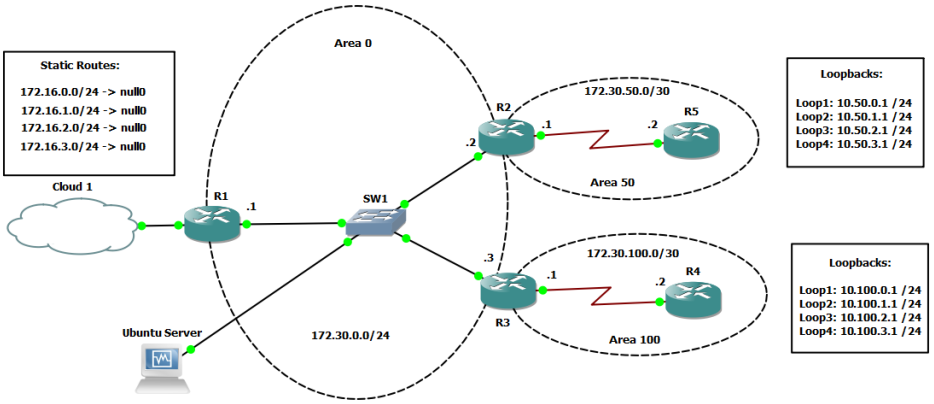
Figure 4.2: Activity Diagram

Chapter 5

Implementation

5.1 Lab Design:

Figure 5.1: GNS3 Topology



A key feature of the simulation tool is the ability to replicate a 'Real-world' network in emulated space. In order to gather the information required for this feature an extra tool set for Network Discovery has been created. This allows a user to discover devices on the network and extract interface information and router configurations in order to build an emulated instance of the network.

For the development of this Router Discovery tool the above topology was created and used. As explained earlier this is implemented within GNS3 to utilise its simulation of real IOS images. The above shows a Topology I have created as a use-case for emulation testing, an Ubuntu 14.04 server is also attached to the topology will which be used for the Network Discovery portion of the tool. In addition to this server, a 'cloud' object is also connected, this provides internet access to the Lab through a bridged interface on the local machine.

The Ubuntu server is implemented using VirtualBox integration within GNS3 and allows direct connection to the 172.30.0.0/24 network. This Ubuntu server will hold the tool used for Router Discovery and Configuration Extraction. As this server is sitting on the same private network

as all of the routers this allows me to find all the routers using defined ip addresses or through using CDP.

The connection to the Internet is provided through the cloud in the diagram above. This is inbuilt functionality within GNS3 which allows the topology to interface with the local hosts connection. This is implemented by adding a loopback to the local PC and bridging the internet connection to this loopback. GNS3 then connects to this loopback address and provides an interface to the Internet in the form of the Cloud object. Once this connectivity is provided the topology has a defined router which is used for internet connections, in this scenario R1 provides the connection. This router is then configured to provide dhcp to any servers on the local subnet, providing a default gateway and DNS Servers for the connected hosts. This router is also configured to provide NAT services in order to allow these private addresses to communicate to the outside world. Once internet connectivity is established the system can deploy.

The base lab used for development has been designed to contain different elements of networks that may be faced in a real-world scenario. The routing of the network is provided via OPSF, a Distance Vector protocol which is used for interior routing in large enterprise. The network is broken into four OSPF areas, these segment the network and prevent routing tables from becoming too large. A second characteristic of this lab is its inclusion of switches, these are 'dumb' switches provided by GNS3 and are not running the Cisco IOS, however they introduce additional complexity to the network which the tool will need to deal with efficiently. The final characteristic of the network is the types of interfaces. As seen in Figure 5.1 the topology includes two distinct types, the first is the expected FastEthernet interfaces which is used for the majority of modern connections, the second is a more traditional Serial interface which would be used for long distance leased-line connections. The variety of interfaces will need to be included in the emulation and their behaviour will also need to be reproduced as it will influence routing decisions.

5.2 Network Documentation

One key feature of the documentation of a network is the data structure that will define it. As discussed in the design sections, a network can be defined with two documents, the device configurations and the physical layout of the connections. The tool provides the latter through a python object referred to in the code as 'topology' is of class routerList. At its highest level this object is a list of the devices that are on the network, this code for the class is shown below:

```
1   class routerList(object):
2       def __init__(self, routerList):
3           self.routerList = []
4
5       def addRouter(self, routerEntry):
6           self.routerList.append(routerEntry)
7
8   class router(object):
9       def __init__(self, cdp_entries):
10          self.cdp_entries = []
11          self.hostname = ''
```

```
            self.ipAddress = ''
            self.capabilities = ''

        def addCdpEntry(self, cdp_entry):
            self.cdp_entries.append(cdp_entry)

        def addHostname(self, hostname):
            self.hostname = hostname

        def addIpAddress(self, ipAddress):
            self.ipAddress = ipAddress

        def addCapability(self, capability):
            self.capabilities = capability

        def removeCDPEntry(self, interfaceName):
            #Updates cdp_entries to include only those that DO NOT match
                interfaceName
            self.cdp_entries = [cdpEntry for cdpEntry in self.cdp_entries if
                cdpEntry.srcPort != interfaceName]
```

As seen above, routerList is a list of all devices on the network. At a lower level, a router is any device found on the topology, it is comprised of a Hostname, IP Address, capabilities and a list of CDP entries for the device. CDP Entries are used to define the physical connection on the topology and when fully processed each interface will have a single CDP entry per interface. A more detailed view of a CDP entry is as follows:

```
        class cdpEntry(object):
            def __init__(self, hostname):
                self.hostname = ''
                self.srcPort = ''
                self.dstPort = ''
                self.ipAddress = ''

            def addHostname(self, hostname):
                self.hostname = hostname
            def addIpAddress(self, ipAddress):
                self.ipAddress = ipAddress
            def addSrcPort(self, srcPort):
                self.srcPort = srcPort
            def addDstPort(self, dstPort):
                self.dstPort = dstPort
```

As shown above, a CDP entry comprises of a Hostname, IP Address and source and destination ports. This information maps the links in the topology. In this instance, the Hostname and IP Address relate to the remote device on the link. This information allows the routerList object to hold the full physical layout of the topology.

5.2.1 Network Discovery

Network Discovery allows the user to dynamically understand the topology of local network. The user has additional options available when taking this route to simulation. If the user specifies the –specify option when running the script, the tool will present them with an interactive prompt where the user can specify the network ranges to be added to the simulation. This is shown below:

```
root@netman-server:~/NetworkSimulation/python-scripts# python NetworkDiscovery.py
    --specify
Enter the network address at the first prompt, followed by the subnet at the second.
    Type "end" to finish
Enter the network address:
172.30.0.0
Enter the subnet mask in CIDR notation (Example: /24)
/24
Enter the network address:
192.168.0.0
Enter the subnet mask in CIDR notation (Example: /24)
/16
Enter the network address:
end
```

This functionality uses a module called ipaddr (Google, 2014) which creates an object bound to the IPv4 specification, this enables the tool to check user input for erroneous entries such as entering an IP address rather than a network. Setting the tool to run with specified networks sets a flag which is used throughout the tool's operation to ensure that discovered devices are within the bounds of the networks identified.

As this tool focuses on the emulation of Cisco devices it allows the tool to utilise Cisco standard technologies as a method of interacting with devices. One such technology is Cisco Discovery Protocol (see 4.8.4) which is a protocol enabled on all Cisco devices allowing them to understand the local network. This protocol is sent as a broadcast on the device's local network, the tool takes advantage of this fact as it can 'sniff' for a CDP packet in order to discover the closest Cisco Router. To do this, the tool uses an external program called Cisco Discovery Protocol Reporter (CDPR) (MonkeyMental, 2013) to listen on a local interface for a CDP Packet. Once a packet is seen, CDPR returns hostname and IP Address of the device that created the packet.

This function is defined in Appendix A, it uses a python library called os, this allows the code to create a subprocess on the server in order to run external programs. This subprocess runs CDPR on the eth0 interface and then returns the output from the subprocesses pipe, at which point python saves this to a variable and processes this variable until only the Hostname, IP Address and capabilities of the device remain. Once this information is extracted the tool creates a base router, this is used as a starting point of the Network Discovery process.

Error handling occurs in this function through the use of a timeout in CDPR. The default send time for a CDP process on a Cisco device is 60 seconds so it can be assumed that if not packet is received after 61 seconds there is no local Cisco device on the network, when this error is hit the tool notifies the user and then closes the script entirely as no further processing can be done.

One caveat of using CDPR as a tool is its requirement of root privileges in order to look at network traffic, because of this the this portion of the tool requires superuser access. on the host. One alternative to this was to write the code with an expect module, which would allow the current account to request root privileges. However this would require the script to know the local user's password, this is insecure and was deemed too much of a security risk.

Once a baseRouter is created, this object is used as a point of entry into the network. The tool then runs a series of SNMP calls to the device looking for CDP information such as hostnames, IP Addresses and interfaces. These functions make calls to the device specifying the OID the information it requires. The OIDs used in the code are defined in Figure 5.2.

Figure 5.2: OID Definitions

Object Identifier	**Description**
.1.3.6.1.4.1.9.9.23.1.2.1.1.6	Returns a list of Hostnames which are sourced from CDP
.1.3.6.1.2.1.4.1	Returns the value of the device's forwarding (Layer 3 routing) setting, this is used to determine whether a device is a switch or not
.1.3.6.1.4.1.9.9.23.1.2.1.1.4	Returns a list of IP Addresses which are sourced from CDP
.1.3.6.1.4.1.9.9.23.1.2.1.1.7	Returns a list of remote interfaces which are sourced from CDP
.1.3.6.1.2.1.2.2.1.2	Returns a list of local interfaces on the device

The SNMP interaction consists of one function per SNMP call, in this function the call is made, the data is processed and a list of information related to the call is returned. a pseudo code version of this function is shown below:

```
function getCDPHostnames(router):
   Make an SNMP call for hostnames seen in device's CDP table
   Store the returned data into a variable
   Parse the output until only the hostnames remain
   Return a list of hostnames
```

This information is added to the baseRouter object, at this point in the code only the base router and its links are known. At this point a recursive function updateRouters is called. This recursive funtion discovers all remaining devices in the topology. The control for this function is based on whether routers that are currently seen already exist in the topology object, if already known, the routers are ignored. Alternatively, if the routers are unknown an object is created for them and the function is recursively called with the new routers. The below pseudo code gives shows the logic used in this recursive discovery, the full implementation can be seen in Appendix B.

```
function updateRouters(updateRouter, topology)
   Get all hosts connected to updateRouter
   Get all IP Addresses connected to updateRouter
   FOR each host identified
      IF a router does not exist for this host
         Create a new router for this hostname
         Get hostnames connected to the new device
         Get IP Addresses connected to the new device
         FOR each device connected to the new device
            Create CDP Object for each connection on the new device
            Add CDP Object to the new device
         Add the new device to the Topology
         Call updateRouters with the new device
```

The function has two distinct paths, it can either add all devices it discovers to the topology. In this case the code following line 52 is executed, a device is added to the topology if it is unknown. An alternative path is the on line 13, the checkNetworks flag is set to true, all devices are checked against the list of required networks and if any do not match they are ignored. This method is effective for removing unwanted devices from the topology, however the connection from other devices to the device in question will still remain. Any links that do not have a defined destination will not create a valid JSON or VIRL file and the emulation engine will fail. In order to avoid this situation, when an unwanted router is encountered it is logged to a global variable, unresponsiveDevices after device discovery is complete a function removeUnresponsive() deletes any links with reference to these devices from the topology. Pseudo code for this function is shown below:

```
fuction removeUnresponsive(topology):
   Create a set of unresponsive devices
   FOR every unresponsive device:
      FOR every device in the topology
         FOR every cdp entry in this device (links)
            If the link connects to the unresponsve device, remove it from the topology
```

Once the unresponsive devices are removed from the topology the first stage of the emulation layout is complete, however in its current form the topology will be rejected by VIRL. This is because VIRL only supports GigabitEthernet interfaces in an emulation, the topology needs to be reformatted to remove any FastEthernet or Serial connections and replace them. This is handled by formatInterfaces(). The function looks at each device in the topology, inspecting each interface on the device and mapping the interface to a GigabitEthernet connection on the emulation. It maintains a list (intChanges) of each host and the interfaces that were changed on the host, saving them to a file for use in the configFormatter section of the tool.

When the function is nearing its end it passes this list on to the replaceInterfaces() function. This function iterates intChanges, looking at every device in the topology and each CDP entry for that device. When the function meets the current device in intChanges it looks for any CDP entries that have a source interface of the interface to be changed, once it is found the interface name is replaced with its emulated counterpart. At this stage in the discovery process there can be multiple CDP entries with the same source interface, this occurs when there is a switch in the network. When designing this function this caused issues as there can be an

expectation that CDP entries map to interfaces in a one-to-one fashion, in the original version of this function a flag was set when a source interface was changed. However, this was naive as when this code was deployed to topologies that were not just point-to-point links the code would fail as pre-emulation interfaces would remain after the function ended.

```
function replaceInterfaces(topology, intChanges):
   FOR each change entry
      FOR every device on the topology
         FOR each device's CDP entries
            IF the device is the one to be changed and this is the port to be changed
               Update the port
            IF the device is not the one to be changed but has a connection to the
                device in question
               Update the remote port in the CDP entry
```

One important aspect of this function is ensuring that CDP entries of devices that connect to the device in intChanges are also updated, the second statement in the function allows for this and checks the CDP entry on every device, if the device has a CDP entry connecting to the current device to be updated the destination port is updated. Doing this ensures that all links in the emulation connect to the correct location.

At this point in the Network Discovery process the physical layout of the topology is known, however there is additional information that can be extracted from the topology, one example is the identification of Layer-2 devices such as switches. One indication of such device is a router including two CDP entries on one interface, this shows that there are two local devices connected through one physical cable which is not possible without some layer-2 device in the middle. To accommodate switches the tool has a function findSwitches() which is documented in Appendix C. This function again iterates over all the devices in the topology, for each device it notes the occurrence of each local interface in the CDP entries. If any encountered interface already exists in the seen set it is noted as a duplicate and added to the dupes list.

The function then inspects this dupes list (if any duplicates exist) for each duplicate a switch is created for the LAN segment connected to the interface. This switch is given a hostname and CDP entries are created both for the local interface and for the remote interfaces that are connected to the local link. This effectively splits the connection in half, inserting a switch into the link which connects to both routers. Once the switch is created, another function updateSwitchLinks() is called which will update the CDP entries on the local and destination interfaces.

The function updateSwitchLinks() iterates through each update, looking at each host's CDP entries until the defined in the update is seen. At this point the function replaces the remote devices CDP information with the new entries generated for the connections to the switch. At this the topology is complete and ready to be passed to the next stage.

The second stage of the discovery process is configuration extraction from the devices defined in the topology. For reasons explained at the end of this section, an external tool called Rancid (ShrubberyNetworks, 2014) was used for this section of discovery. The configuration extraction is defined in the file rancidSetup.py, in this script the router dictionary produced by networkDiscovery is used to create a group of devices in rancid's configuration. Once a group is created and defined the script logs in as the rancid user on the host. Once logged in the rancid

user runs the script rancid-cvs, this script creates a CVS repository for the group that has been created. The user then adds the configuration for the routers, first adding each hostname to router.db within the group which adds the router to the extraction process. After adding this it then adds the login information for each router to rancid's .cloginrc file. This file defines the method of connection, which is ssh with 3DES encryption for this script. It also adds the login credentials which must be configured on the router. Once this information is saved the script rancid-run is called, this starts the configuration extraction process.

While creating this section of the code it was difficult to produce a method that did not breach Requirement 8 in the Requirements section (4.8). This requirement relates to security methods of interacting with a Router on a Production network and requiring this tool to not introduce security flaws in these networks. In initial designs the tool would extract configurations itself using SNMP to poll each router, however when reading around this subject it became clear that this method of interaction with the routers would require Read/Write access on SNMP. This access was defined on a device level and not to a single OID, and thus would leave any devices exposed to exploitation if this access was mismanaged. Due to this the decision was taken to offload this section to a standard tool in the Networks world. Rancid was chosen due to its lightweight design and licensing which allows its use without any additional purchases or fees.

One issues faced when implementing RANCID on my lab network was that by default Cisco routers are set to SSH Version 2, this requires additional configuration on the device such as a domain and generation of an RSA key for access. For the design and testing phase of this development this expectation was removed through the use of 'ip ssh version 1' on each device. However in normal operation SSH will be set up on each device and this error will not occur.

5.2.2 Network Entry

Network Entry works in a similar fashion to Network Discovery, however the devices on the emulation are created by reading a json file documenting the physical layout of the topology and also adding the configuration files for each device in the 'config-files' directory in the script's location. The script first loads the topology json file, creating a new router object for each node defined and adding it to the overall topology. The second operation loads the link information from the json, however this needs to mapped to CDP entries in order to follow the networkDiscovery convention. To do this the sourceRouter for each link is read and then mapped to the correct device in the topology. This creates a full network.

At this point the topology file is ready to have its interfaces mapped to emulated as was the case in networkDiscovery. The topology is passed through the same formatInterfaces function as before and interfaces are mapped to their emulated counterparts. The same routerDictionary and interfaceChanges lists are created and saved to a file. From this point forward both the discovered and input networks are treated in the same way.

5.3 Configuration Editing

Before creating a simulation the Cisco configuration files need to be edited in order to match the new layout defined in the topology files. Specifically the interfaces need to be replaced with their emulated versions and an additional interface needs to be added for the management network when the emulation is live. Due to the fact that a Cisco configuration file is based on a hierarchy to edit the file directly as changing information under an interface could break the hierarchical link between the interface and its children. In order to overcome this issue, the tool uses an additional python library, ciscoconfparse (Pennington, 2015). This library allows python to read a configuration file and break it down into the relevant parent-child relationships that are required to make a clean edit to the interface.

The extraction of the parent-child relationships in a configuration file allows the tool to make changes to interfaces without any loss in configuration. For example, an interface that has been set to passive in a routing protocol would need to be reflected in an emulation. Removal of a passive interface could introduce errors into an emulation as this interface can now negotiate routing with the devices it connects to, this could have impact of the routing decisions made on the topology. Maintaining this mirrored configuration ensures that the emulation generated for a network is a as true to real hardware as possible.

An additional feature that was considered was the type of interface before it is converted to an emulation Gigabit interface. When representing a FastEthernet interface on an emulated network it replacing the port in name-only would be representing a link with a maximum bandwidth of 100mb/s with one that is 1000mb/s. This direct replacement would break the routing decisions as all links on the topology would become an equal weighting. In order to prevent this the tool appends a child to each interface relating to the original bandwidth of the link, for an interface that was originally FastEthernet a 'bandwidth 100000' child is added to the interface. If an interface was originally a serial connection, a 'bandwidth 2000' child is added, this is the speed of an E1 connection. If any such configuration exists under the interface it remains and no action is taken.

After the relevant changes are made to interfaces on the configuration, a new interface is added. Within VIRL there the ability to connect the topology through a management network, this as called a Private Simulation Network. In order to allow this network to function an additional interface, GigabitEthernet0/0 is added to each configuration file.

Finally any information pertaining to security on the device is removed from the configuration. This is to allow access to the device when it is on the emulation. Information such as the the login credentials and enable password are removed, in addition to this the vty lines on the device or opened with the 'no login' command.

5.4 VIRL File Creation

As discussed in Section 4.4 an emulation in Cisco VIRL is defined through an xml-based file known as a virl file. In order the data gathered up to this point to be represented in a valid virl format the tool uses an external library called lxml (Behnel, 2015) to create xml tags based on the topology. This allows the tool to quickly generate correct xml that the VIRL host recognises as valid.

This section of the tool is outlined in the file virlCreation.py. In this file the tags are first created to hold the xml and virl specifications which are then added to the xml object. After this, the script adds information to the topology as a whole, such as specifying that CDP is enabled and that the devices are running the onePK API. One key feature that is added at this point is is setting for a management network on the emulation, setting this to 'exclusive' specifies that the management network is local to the emulation and does not interact with other simulations.

Once the settings are defined the script then loads the routerDictionary which was defined earlier in the tool. It then begins to iterate over each device, creating a node tag for each and setting the device to an IOSv Router for any routers seen or to a SEGMENT for any switches seen.

One issue encountered when designing this section of the tool was finding a way to efficiently represent a switch on the network. Losing the connectivity provided by a switch would introduce extra links to the topology that will in turn create discrepancies between the emulated routing and the real hardware routing. Due to this fact, the tool uses the device SEGMENT. This device produces a layer-2 broadcast domain which acts in a similar fashion to the switch. In later versions of the VIRL software these segments have been replaced with 'dumb' switches. These act in a similar fashion and do not make real switching decisions, however a full IOSvL2 image is due to be added in the next version of Cisco VIRL.

Once the node is created for a device the configuration is then added, to do this the script loads the router's '.edited' configuration file that has been produced in the local 'config-files' directory. This configuration was produced in the previous section. The tool appends the contents of this file to the nodes 'config' attribute. Once the configuration has been added the tool then iterates of the routers CDP values, adding an interface for each one. At this point the CDP Values give unique interfaces as switches have been added to the topology. This interface does not dictate the physical links in the topology.

An additional entry at this point is made to the emulated topology. This is an ubuntu server which will act as a gateway into the network. This implementation uses two distinct objects in the virl file. The first is the server, this is created using a cloud-init script defined in Appendix F. This sets up basic access on the server such as user accounts and home directories. An interface is added to the server for its connection to the internet. The second object is a SNAT Object, this provides Network Address Translation between the emulation and the outside world, when configured this object allows inbound connections to the server.

The final part of this file defines the connections between the devices in the topology. This section proved difficult as the design for a VIRL file does not match the design that has been used for the storage of information up to this point in the design. The links in a VIRL file are mapped using the relative position of the devices in the file rather than the identifier that has been assigned, in addition to this there is only one definition of a link regardless of the source and destination device. This introduced additional complexity to generation of the virl file. This was overcome by adding an additional check against a list of existing interfaces on the topology file.

Once all devices and links have been added to the xml object the file is generated and saved locally as the file 'topology.virl'

5.5 Simulation

At this stage of the implementation an additional barrier to the design was introduced. The VIRL host that is used for all simulations is only contactable if the host machine is connected to the Loughborough University VPN. This added a requirement that the tool must be run on the VPN in order to be granted the appropriate access. A second requirement was that the local host machine must have a shared interface in order for the Lab design in Section 5.1 to connect to the Internet. It is now understood that a security requirement on the Loughborough VPN and Eduroam network dictates that any devices with a shared network interface is blocked from connecting, this created a conflict in the design.

In order to overcome this issue, the design now uses two identical servers with differing network connections in order to create a simulation. In this design the server on the lab network gathers and collates the information about the network, produces a virl file and then uploads this file to an intermediate SFTP server. Once this has been uploaded the lab is powered down and the same server is loaded, however this instance uses a traditional NAT connection to the internet. This allows it access to Loughborough University infrastructure and the internet. This server downloads the VIRL file from the SFTP server and pushes it to VIRL to create an emulation.

The simulation aspect is handled in the file uploadTopology.py. This file connects to the Cisco VIRL REST API in order to create and retrieve information about simulations. The script begins by loading the virl file into a variable, it then creates a POST request to the API's '/simengine/rest/launch' rule. In this request it attaches the virl file and a defined naming scheme to be included in the simulation ID. The server then makes a GET request to the API's '/roster/rest/' rule, this returns information for every node on every simulation for the user that is authenticated. The script iterates over this information, for every IOSv need it sees for the simulation made it keeps a dictionary of the IP Address and hostname. When the function meets the 'virl-sim-server' for the current simulation it saves information such as it's internal address, external address and the port number for which a user can access the server's console line.

Listing 5.1: Extract of /roster/rest information

```
"rob.My_Topologies@topology-gymofc.virl.~lxc-flat.External Address": {
"Annotation": "131.231.97.151",
"SimulationHost": "158.125.102.75",
"managementIP": "131.231.97.151",
"simExpires": null,
"simID": "My_Topologies@topology-gymofc",
"simLaunch": "2015-01-06T16:06:59.534563",
"simStatus": "ACTIVE"
},
"rob.My_Topologies@topology-gymofc.virl.~lxc-flat.External Port": {
"Annotation": "59313",
"SimulationHost": "158.125.102.75",
"managementIP": "59313",
"simExpires": null,
"simID": "My_Topologies@topology-gymofc",
"simLaunch": "2015-01-06T16:06:59.534563",
```

```
"simStatus": "ACTIVE"
},
```

The information gathered about the IOSv hosts and the virl-sim-server are used in the function setupServer(). At this stage in the simulation the server does not have a valid network configuration. This function uses a telnet library, telnetlib (Developers, 2015). Telnet was used for this section of development as the only internet-facing access into virl-sim-server is a console port provided on VIRL. The function telnets into this console port, gaining access to the console line. The tool then configures the eth1 IP address to match the internal SNAT address that was received earlier in the function. At this point the server has a valid IP address and can communicate with the outside world, however it does not have a default gateway so cannot respond to traffic. The function adds this default gateway to the server's routing table. The server can now contact with the outside world, the tool returns the IP address of the public-facing interface and the user can SSH into the virl-sim-server, using it as a jumphost into the simulation.

Figure 5.3: Management Network Design

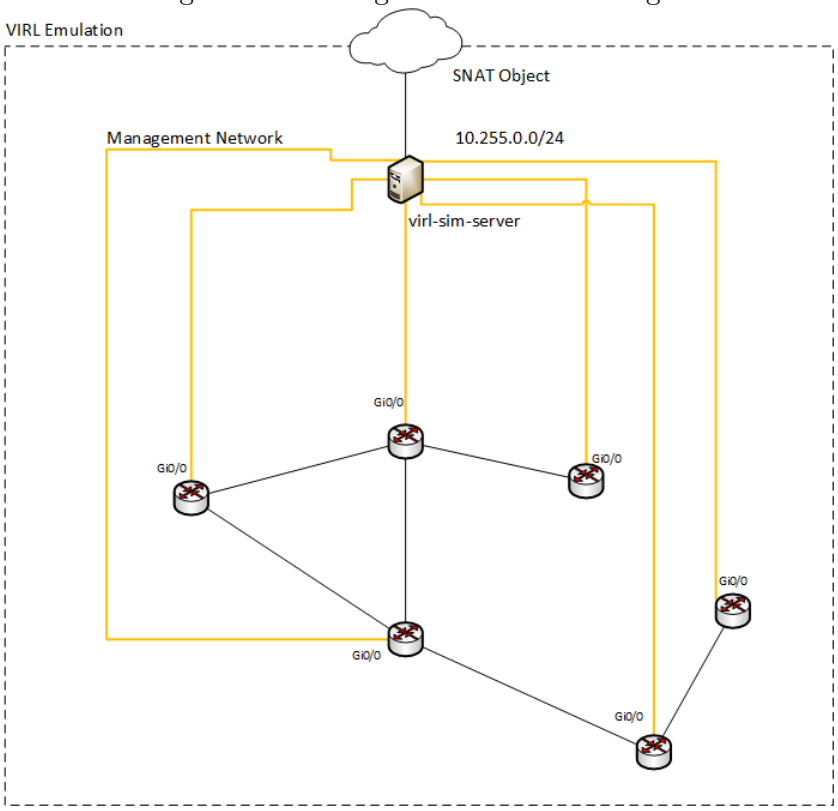


Figure 5.3 shows a simplified view of the network produce in VIRL. In this diagram the emulated routers are represented by the router icons and 'real-world' cables are represented by black links. In addition to these links the emulation also has a management network, denoted by the yellow links, which interconnects every device on the network. For simplicity, Figure 5.3 only shows the management connections through to the virl-sim-server, this network operates on the 10.255.0.0/24 subnet. All IOSv on the topology connect via the GigabitEthernet0/0 interface created in Section 5.3. This connectivity will be used to allow the virl-sim-server to connect

and gather information from the IOSv devices on the topology.

5.6 Interaction

The produced tool allows a Network Engineer to SSH to the virl-sim-server, from here the user can interact with the emulation through a traditional method, telnet. This provides unrestricted access to the device's VTY lines from which an engineer can implement any changes to the network, gathering feedback via methods such as viewing routing tables or using traceroutes through the network. In addition to the VTY access an engineer can also deploy other tools on to the virl-sim-server which can connect to the emulated routers. Examples of these tools are outlined in the following subsections.

5.6.1 SNMP Monitoring

IOSv instances support SNMP just as a physical router would. Using this access the Engineer can set up an SNMP Monitor on a server within the simulation which could provide detailed statistics such as interface counters. In addition to this, the IOSv instances can utilise SNMP traps in order to be notified of any significant changes on the network.

5.6.2 Netflow

Netflow is supported by Cisco IOSv and gives a more detailed view of traffic across router interfaces. Configuring each IOSv instance as a Flow Exporter and adding a collector such as Cacti will provide a detailed breakdown of traffic based on its Layer 3 and 4 information. This adds further granularity to the testing evidence that is created and allows an Engineer to understand how traffic is reacting to the changes made on the network.

5.6.3 Packet Capture

Using a Cisco SPAN session an Engineer can configure an IOSv device to mirror the traffic seen on one interface to another defined interface. This mirroring can be used for analysis by an Engineer in order to understand the impact of a change to the network at a packet-level. An example could be examining the impact of an interface being taken down on a TCP session, looking at the packet information will enable to see the exact time that any impact to traffic was made, along with the impact on the session.

5.6.4 Additional Interaction

Although the technologies outlined so far in this section add value to the tool produced, it unfortunately does not meet the full requirements outlined in Section 4.6. In this section there are requirements regarding investigation of new methods of interaction with a network emulation. Initial designs of this section were to include a webserver installed and dynamically configured on virl-sim-server which would host a page containing a further tool which allows the user to interact with the emulation through a web interface.

The visual aspect of this section of the tool is implemented using a javascript library called d3.js (Bostock, 2015). This library implements a data-driven implementation of visual components using html5 SVG elements. D3.js interprets the json representation of the topology used in Section 5.2 to create a visual representation of the network. The topology is created using a model known as a Force-directed graph, a model based on nodes and edges, which this tool replaces with devices and interconnecting cables. This graph uses a physics simulation on the back-end in order to display the graph in a meaningful format. As the Discovery section of this tool only discovers the devices and links, it does not have any information regarding the physical positioning of devices on a topology view. The tool uses the physics simulation in order to create a view which does not have any overlapping nodes or edges and allows a viewer to understand the layout of a network at a glance. The force-directed graph was used as it is a tried and tested method for node-and-edge display and allows a quick rendering of the topology when compared to a standalone tool for graphing tool. In addition to this, running this section of the tool on a webserver allows the User to view the topology discovered at any given time by viewing the webpage. This enables the user to check that the discovery process has found the information the required, before creating an emulation session.

Figure 5.4: Web Topology View

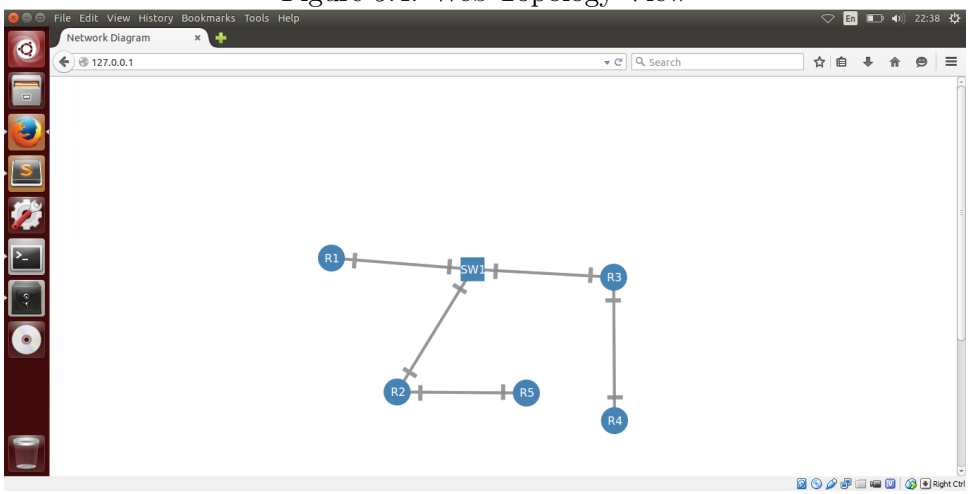


Figure 5.4 shows the topology view of this webserver. When first displayed the Force-directed graph moves to find equilibrium, the forces of each node moving until the charges cancel one-another out and any movement stops. At this point the simulation is stopped and the topology view is fixed, not moving unless the user interacts with the topology. Nodes can be dragged around the canvas in order to create the desired layout.

In addition to the view created from the information from json, additional information about the hosts in the simulation is stored in the DOM section of the SVG elements on the screen. This information can be used to identify IP addressing, node types and interfaces. This information is also encoded in to the links on the screen, each one holding a source device, source interface, destination device and destination interface. This information is displayed in a tool-tip on links in order to understand interface names on the topology in order to compare to the real devices when implementing changes.

Further information is stored under the elements in the topology, information such as the IP address of a link allows information such as 'Next hop' IP addresses to be used in order to

Figure 5.5: Web Topology View Tooltip

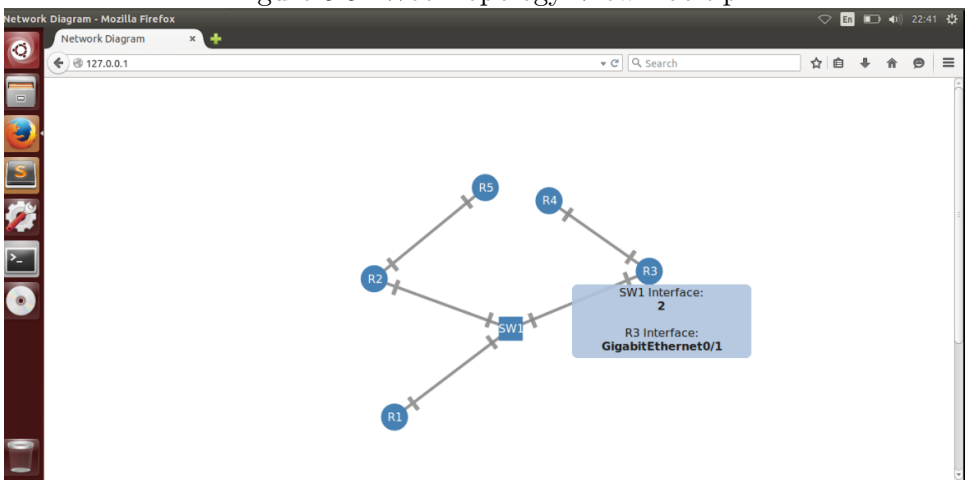
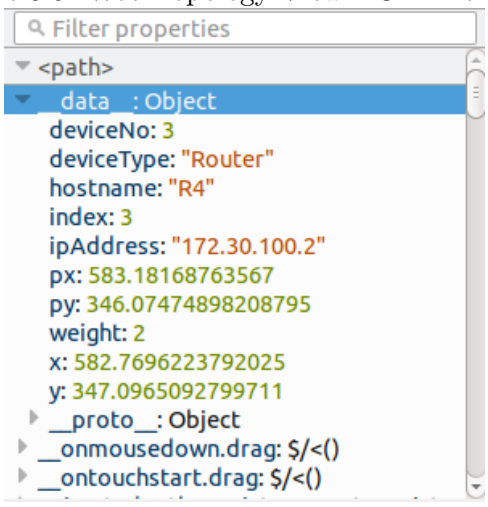


Figure 5.6: Web Topology View DOM Elements



display routing paths on the topology. Uses for this information will be outlined in Section 5.6.5

5.6.5 Webserver Design

Due to an issue arising on the Loughborough University firewall further work on this section of the webserver portion of the tool was halted. It was identified that Port 80 and 443 were not permitted to the 131.231.99.0/24 network, the range used for SNAT network connections. It was not possible to have this access permitted in order to produce a working prototype of this section. Further restraints were also prevalent, such as the ability to interact with devices using Cisco OnePK in order to gather information from emulated hosts. This tool-set was not available in the IOS versions that were used in development of the tool and interaction could only occur on the IOSv emulated hosts.

Due to the reasons outlined above, rather than outlining the implementation of this part of the tool it will instead outline the design of this functionality.

The design of this section was based around the implementation of a webserver and Python API on virl-sim-server. The combination of these two tools provide a webservice, available on the public SNAT IP Address that the User can connect to in order to interact with the emulation. On the front-end this webserver would host the same topology view which was outlined in Section 5.6.4, however it would have additional functionality provided through the API and Cisco OnePK.

One feature of this tool would be the visualisation of routing through the links on the topology, this would be implemented by changing the colour of the links used in a forwarding path in order to display to the user which decisions were made by each router. This information would be updated constantly, providing a view of the current state of routing between two locations. In particular this would allow an Engineer to see how a change within the topology will affect routing between two locations. Allowing them to better understand the full impact of their changes and whether any latency or impact to users will be involved.

The REST API used for this section of the tool would be implemented using a python-based Web API such as Bottle (**?**) or Flask (**?**). These tools enable the interaction of web-based languages such as Javascript with traditional server based ones such as Python. This allows the webserver portion of the tool to interact with python, and thus any python based code on the server. Creating a library of resources using this API will enable the webserver to interact directly with the emulation on VIRL. Allowing the user to interact with an emulation without connecting to a device via a console session.

The tool would also use the Cisco OnePK (**?**) platform to programmatically extract information from emulated hosts such as the hosts currently in the emulation, their power status, link status and bandwidth on links. The use of the OnePK Software Development Toolkit enables a more dynamic method of extracting information, it removes the need for extraction of required details from SNMP calls and also creates a more streamlined model of interaction when compared to traditional 'screen-scraping' tools which rely on telnet or ssh to gather information. Using OnePK allows the tool to poll any emulated device for the required information.

A RESTful API works on a collection of paths in the URL known as routes, these routes defined both the behaviour required by the call and the objects to perform that behaviour on. The

design would use the an emulation at it's root in the URL, allowing calls to be made for data retrieval or data addition on the emulation. Some examples of this behaviour is defined in Figure 5.7.

Figure 5.7: Example REST Routes

HTTP Method	**Action**	**Example URI**	**Payload**
GET	Get a list of all devices on Emulation	/api/topology	null
GET	Get device information by ID	/api/topology/1	null
GET	Get a traceroute list of hops from X to Y	/api/traceroute	from: X to: Y
GET	Get forwarding table from device	/api/forwarding/1	null
POST	Create a new device on the topology	/api/topology/1	hostname: x IP Address: X.X.X.X Interfaces: {Gi0/1, Gi0/2}
PUT	Update hostname of a device on the topology (by ID)	/api/topology/1	hostname: x
PUT	Update link status of a device on the topology (by ID)	/api/topology/1/interface/1	status: DOWN

The routes defined in Figure 5.7 will execute server-side python scripts that will use OnePK to interact with emulated devices and either return the requested information or update the emulated devices by using the information specified in the payload of the request. As an example, if the tool sent the request /api/traceroute with the payload *from: R1, to: R7* the tool would execute a OnePK script that would extract this traceroute information from R1, returning a list of the hops made on it's routed path. This list can be processed by the webserver, which can indicate the routing path by colouring the corresponding links on the topology to indicate their involvement.

Chapter 6

Testing

For the produced tool testing has been done in two distinct methods. The first round of testing will focus on the code used in this tool, particularly error handling and parsing of incorrect files. The second round will focus on the created emulations, as the tool is focused on creating a counterpart to a real network, the testing will focus on ensuring that created emulations exhibit the same behaviour as real topologies. This will be performed by using multiple topologies running separate routing protocols in order to create variety in the test cases. The correctness of produced emulations can be compared using the visual topologies, routing tables and neighbour tables.

6.1 General Testing

The tests defined in Figure 6.1 define a set of tests that are general, the network being used for discover will have no affect on these tests and the outcomes will be standard across all inputs.

Figure 6.1: General Test Results

Test No.	Target File	Purpose of Test	Expected Result	Actual Result
1	networkDiscovery	Check that CDP is recognised	CDP Packet of local device is seen and a topology device is created	CDP Packet seen and device created
2	networkDiscovery	Check SNMP can communicate	SNMP communicates with Device	SNMP call returns all CDP hostnames
3	networkDiscovery	Check that discovery finds all devices	Recursive function will find all devices in search space	All devices on topology are discovered and have corresponding device objects
4	networkDiscovery	Check that all interfaces are mapped correctly	The number of interfaces used on the hardware match the number of interfaces in a device objects cdp_entries	Numbers match as expected
5	networkDiscovery	Check JSON output is correct	JSON file created is accepted by the webserver and displayed	As expected,
6	networkDiscovery	Check that unresponsive devices are dealt with correctly	SNMP call will timeout, tool will remove all reference to device from topology	As expected, device is omitted from topology. Message is displayed on Command Line
7	networkDiscovery	Check that script times out if nothing found	Script will time out after 61 seconds	As expected, time out after 61 seconds if no CDP is seen
8	networkDiscovery	IP Address input works as expected	Only devices within the specified IP Address ranges are added to the topology	As expected, in testing case only R1, R2 and R3 were added to topology when 172.30.0.0/24 was specified on the OSPF topology
9	networkDiscovery	Check that switches are correctly added	A switch is created and added in any location where a device can see two other devices on one link	As expected, two switches were created in the entire OSPF topology
10	networkDiscovery	User specifies network which is not local	networkDiscovery fails and user is informed	As expected

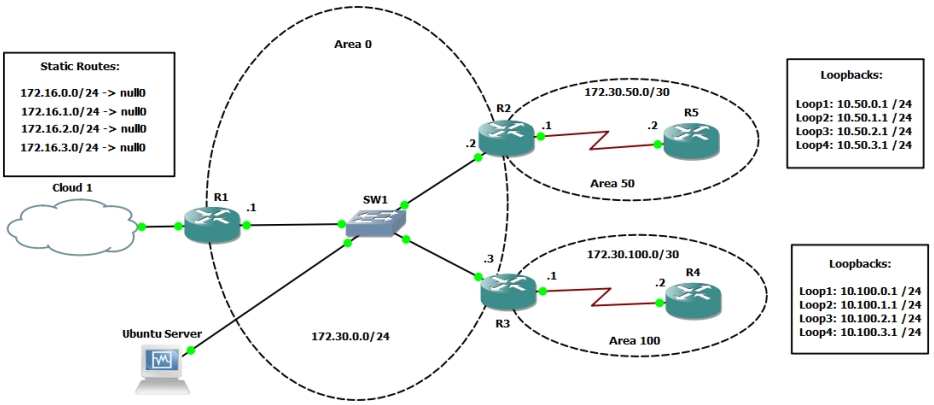
Test No.	Target File	Purpose of Test	Expected Result	Actual Result
11	networkDiscovery	Switches are correctly identified and created	Where two devices are seen on a single interface, a switch is made	As expected
12	rancidSetup	Check script is run as root	If script is not run as root and IOException is thrown, this will be handled	Exception handled and user notified
13	rancidSetup	Device does not respond to rancid	User is not notified - limitation of using rancid	As expected, user is not notified
14	configFormatter	User specifies the –noCopy parameter	Script uses the files already in configfiles	As expected
15	configFormatter	Config file is missing from location	User is notified and script is halted	As expected, user is notified
16	configFormatter	Interface settings are not deleted in formatting	A Gigabit interface which represents a serial interface will have the bandwidth of a serial connection set	As expected, by default this is set to 2000 if not specified
17	configFormatter	Management interface is added in correct location	The management interface, GigabitEthernet0/0 is added before the first interface on the device	As expected
18	configFormatter	Login information is stripped from device	All usernames and passwords are removed, 'no login' added to vty lines	As expected
19	configFormatter	Invalid configuration files are rejected	Adding a non-cisco configuration file throws an exception when passing with ciscoconfparse. This is handled and processing is stopped	As expected
20	virlCreation	Check tool produces a valid VIRL file	Tool creates a .virl file which is created from xml	Valid xml is created

Test No.	Target File	Purpose of Test	Expected Result	Actual Result
21	virlCreation	Routers and their links are reproduced correctly in VIRL	An IOSv node is created for every router device, each link is specified only once	As expected, the topology is correctly represented
22	virlCreation	Check switches are correctly added as LAN segments	Switches are represented as SEGMENT devices and links are produced to connect them to Routers	Switches added as expected
23	virlCreation	Tool adds the virl-sim-server and correctly connects it to the topology	virl-sim-server is added to topology, connection to topology is provided through eth0 to Private Simulation Network. Connection to internet is through eth1 which connects to SNAT	As expected, server is added and connections are correct
24	virlCreation	Check SNAT device is correctly added	SNAT device is added and link is connected to virl-sim-server eth1 interface	Created and added as expected
25	uploadTopology	Check tool copes with server being unavailable	Connection will timeout after 5 seconds	Timeout was hit - server was not connected to VPN and therefore could not contact VIRL
26	uploadTopology	Check script retrieves emulation information	Tool makes call to /roster/rest/ on API and extracts information from returned JSON	Call was made successfully and information was retrieved
27	uploadTopology	Check script can telnet to virl-sim-server and configure it	Tool telnets to virl-sim-server configuring it's interfaces, default gateway and host information	Updates were made successfully
28	uploadTopology	Check server is available on the public SNAT address	Server can be accessed using default username and password	Logged into server successfully

Test No.	Target File	Purpose of Test	Expected Result	Actual Result
29	uploadTopology	Check that routers can be accessed from virl-sim-server	A telnet session to the router hostname will give the router prompt at privileged mode	Prompt was given as expected

6.2 Topology 1: Multi-area OSPF

Figure 6.2: OSPF 'Real-World' Topology



I have designed this Topology to represent a simple network with a complex OSPF routing set-up, which may be a possible scenario that a network engineer may wish to experiment with before implementing changes in Production.

The topology represents the OSPF backbone of a large enterprise network, it includes three OSPF areas; area 0 (Backbone), area 50 and area 100. In this scenario R2 and R3 are taking the role of Area Border Routers (ABR) and providing route summarisation between the areas. For both area 50 and area 100 there are also addition Loopbacks created on the end routers which provide additional routes for this summary.

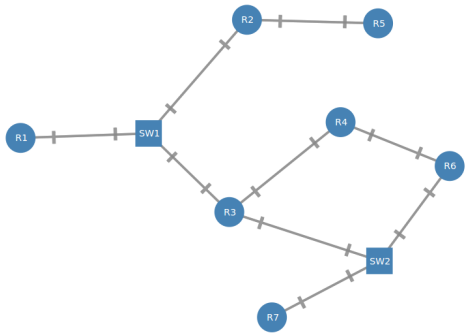
In addition to the routes being introduced via the loopbacks on R4 and R5 the topology also injects external routes into OSPF on R1. These are static routes defined on R1 which represent routes from either the Internet or routes from another routing protocol such as EIGRP, IS-IS or BGP. In this form the routes are sent to null 0 (blackhole) as there is no need for connectivity to them.

6.2.1 Topology Test Plan

The Test Plan in Figure 6.2 define a set of scenarios for which the tool should perform in an expected way. This testing assumes that all the pre-requisites defined in the User Guide are met. This subsection reviews the testing of the tool at a high-level, reviewing the emulation of this topology at three sections of the tool. The first is the initial running of networkDiscovery.py,

at this stage the tool has produced a topology object and a list of interface mappings for the emulation.

Figure 6.3: OSPF - Produced JSON Topology



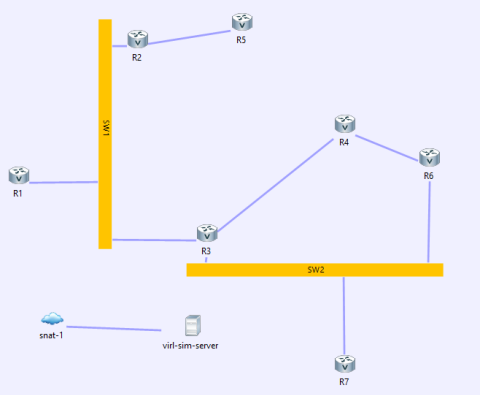
As seen in Figure 6.3, the visual representation of the JSON created in networkDiscovery matches the real topology. Each node is present and each interconnecting link is in the right location when compared to the interface changes, these interface changes are shown in Listing 6.1

Listing 6.1: Interface mappings for OSPF topology

```
R1 was: FastEthernet0/0 now: GigabitEthernet0/1
R2 was: FastEthernet0/0 now: GigabitEthernet0/1
R2 was: Serial0/0 now: GigabitEthernet0/2
R3 was: FastEthernet0/0 now: GigabitEthernet0/1
R3 was: FastEthernet0/1 now: GigabitEthernet0/2
R3 was: Serial0/0 now: GigabitEthernet0/3
R6 was: FastEthernet0/0 now: GigabitEthernet0/1
R6 was: FastEthernet0/1 now: GigabitEthernet0/2
R4 was: FastEthernet0/0 now: GigabitEthernet0/1
R4 was: Serial0/0 now: GigabitEthernet0/2
R7 was: FastEthernet0/1 now: GigabitEthernet0/1
R5 was: Serial0/0 now: GigabitEthernet0/1
```

In addition to the JSON displayed in Figure 6.3, later on in to the execution of the tool the topology is represented as a .VIRL file, by loading this file into VM Maestro the view in Figure 6.4 was produced.

Figure 6.4: OSPF - Topology visualised in VIRL



As the produced .virl file was accepted by the VIRL Engine, the output of the tool was correct. However this does not indicate that the behaviour of the emulation will match that of its hardware counterpart. In order to compare the behaviour of the two some testing was also done to compare the outputs of the OSPF neighbour table, the forwarding table and traceroutes within the emulation. These are shown in Appendix G.

From the information in Appendix G it can be seen that the OSPF behaviour has been well represented on the emulation. The neighbouring used with OSPF has been reflected exactly, however there is some slight variations in the Metric for OSPF routes. This may be due to the bandwidths added to links in order to have them behaviour in a similar manner to their hardware counterparts, if these numbers do not match exactly there is some discrepancy across the metrics. However, the routing choices made were not affected by these changes.

6.3 Topology 2: EIGRP Based Routing

The second topology uses another routing protocol, EIGRP, to handle communication through the network. This is a Distance Vector protocol which uses updates from a routers neighbours in order understand the network as a whole. This is a Cisco proprietary protocol and is used primarily is enterprises which have a full Cisco network. The benefits of EIGRP are that it can provide fast routing changes when outages arise due to it's use of Successor (Primary) and Feasible Successor (Secondary) routes.

The primary metric of EIGRP is the Feasible Distance of the of the route in the topology table. EIGRP takes the route with the lowest Feasible Distance and adds it to the routing table. The topology also includes summary routes as part of EIGRPs capabilities and these should be reflected in the emulated topology.

6.3.1 Topology Test Plan

Figure 6.6 shows a visual implementation of the topology when stored in a JSON format. From a visual perspective all devices and links are correct. Listing 6.2 shows the mappings of the 'real' links to their emulated counterparts, all links that were part of the discovery process are displayed.

Figure 6.5: EIGRP 'Real-World' Topology

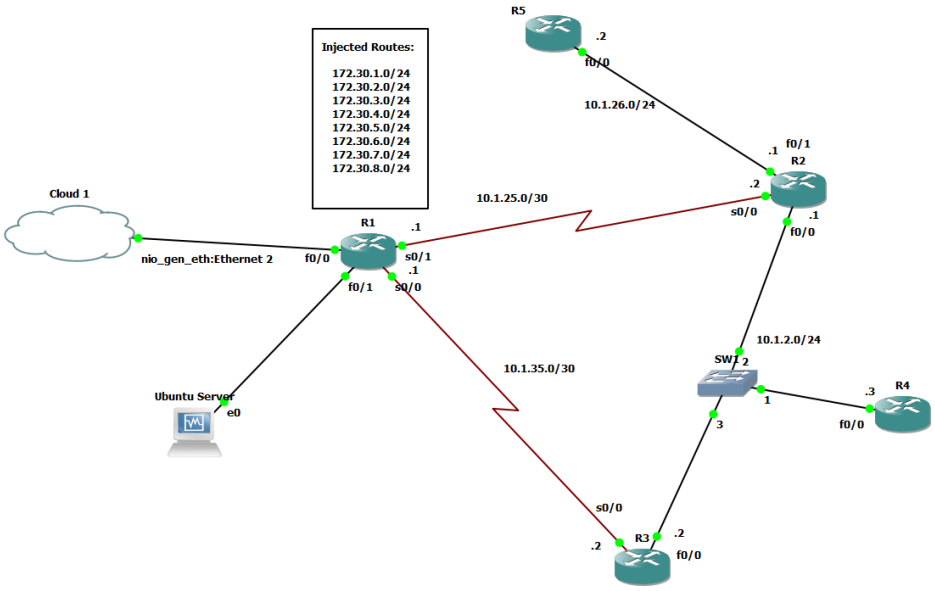
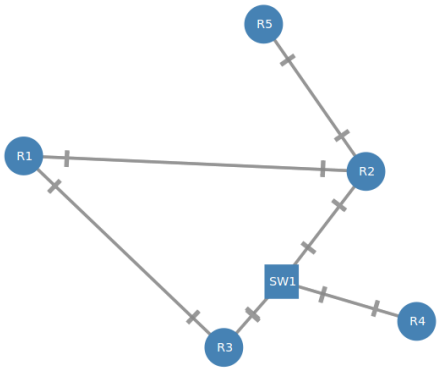


Figure 6.6: EIGRP - Produced JSON Topology



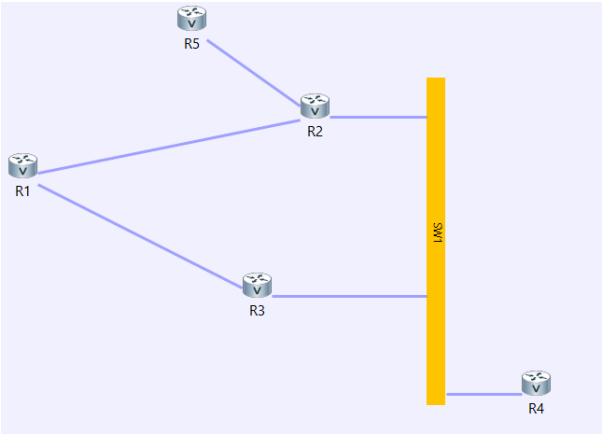
Listing 6.2: Interface mappings for EIGRP topology

```
R1 was: Serial0/0 now: GigabitEthernet0/1
R1 was: Serial0/1 now: GigabitEthernet0/2
R3 was: FastEthernet0/0 now: GigabitEthernet0/1
R3 was: Serial0/0 now: GigabitEthernet0/2
R2 was: FastEthernet0/0 now: GigabitEthernet0/1
R2 was: FastEthernet0/1 now: GigabitEthernet0/2
R2 was: Serial0/0 now: GigabitEthernet0/3
R4 was: FastEthernet0/0 now: GigabitEthernet0/1
R5 was: FastEthernet0/0 now: GigabitEthernet0/1
```

A second method of testing whether the topology is correct is to compare the original topology to the visual representation of a .virl file. This is shown in Figure 6.8, it can be seen that the general structure is the same as the topology and all links remain in their correct locations. One obvious change is the replacement of switches with SEGMENT objects, however this is

purely cosmetic.

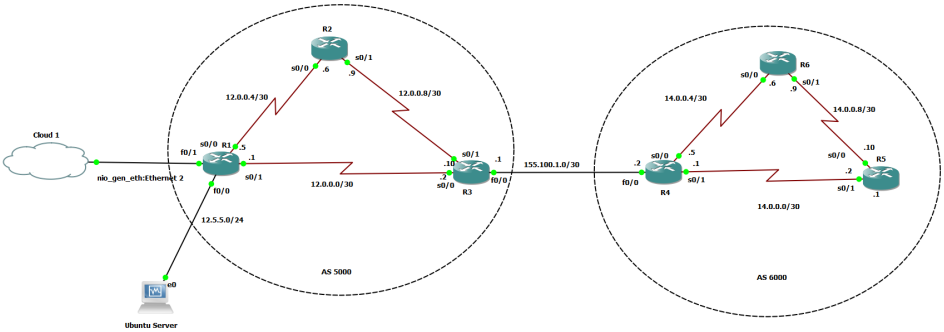
Figure 6.7: EIGRP - Topology visualised in VIRL



The produced .virl file was accepted by the VIRL engine and a virl-sim-server was created, from this information was extracted from the emulated devices which was compared to the hardware routers. This information is viewable in Appendix H. A review of this information shows that the emulated representation of the network behaved in a similar way, with routing tables being reflected across the two systems. One particular difference that was noted was the changes in Feasible Distance for routes across the two systems, this has been identified as being caused by a lapse of information on interfaces and the difference between Serial and Gigabit links. As can be seen when comparing the two routing tables however, this made no impact on the routing choices made by R1. Another change that is of not is two additional routes to 10.255.0.0/16, this is the connection to the Management Network provided via GigabitEthernet0/0.

6.4 Topology 3: Service Provider BGP Topology

Figure 6.8: BGP 'Real-World' Topology



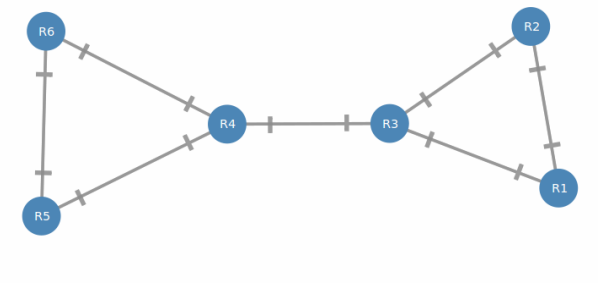
The final topology used for testing is using BGP, the protocol used for routing across the internet. This lab was inspired by the coursework used in the Computer Science module COC190, it represents two Internet Service Providers with unique Autonomous System Numbers (ASNs) which are peering and exchanging routes. The leftmost ISP is using ASN 5000 and the right uses ASN 6000, the two maintain a single BGP peering on the central link using the 155.100.0.0/30

network. Interior routing of the two Autonomous Systems are provided by a combination of iBGP and IS-IS. iBGP provides the interior network with information about external routes, in AS5000 these routes are provided by R3 and in AS6000 they are provided by R4. The interior protocol IS-IS provides a fast and lightweight method of propagating local routes throughout the network. This test will check whether the combination of IS-IS, iBGP and eBGP work correctly when ported to an emulated network.

6.4.1 Topology Test Plan

The BGP peering in this topology is maintained through the use of Loopbacks as a source for the connection, this was used as an added test to ensure that Loopbacks were correctly represented in emulation networks. For the purposes of the tools deployment however it was found that all links in each Autonomous System were required to be in BGP, otherwise the tool could not contact the routers. Once this change was made to the routing the topology was correctly emulated in VIRL. Figure 6.5 shows the topology defined by the discovered network, Listing 6.3 shows the mapping of interfaces from the real device to the emulated version.

Figure 6.9: BGP - Produced JSON Topology

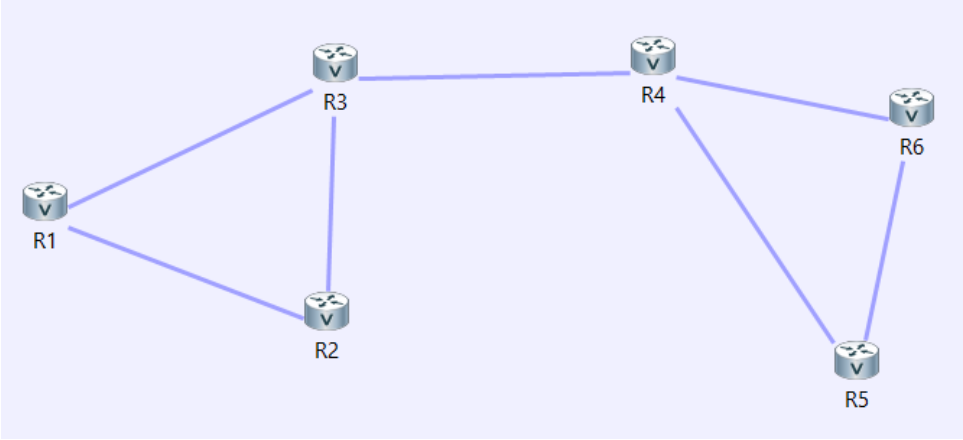


Listing 6.3: Interface mappings for BGP topology

```
R1 was: Serial0/0 now: GigabitEthernet0/1
R1 was: Serial0/1 now: GigabitEthernet0/2
R2 was: Serial0/0 now: GigabitEthernet0/1
R2 was: Serial0/1 now: GigabitEthernet0/2
R3 was: FastEthernet0/0 now: GigabitEthernet0/1
R3 was: Serial0/0 now: GigabitEthernet0/2
R3 was: Serial0/1 now: GigabitEthernet0/3
R4 was: FastEthernet0/0 now: GigabitEthernet0/1
R4 was: Serial0/0 now: GigabitEthernet0/2
R4 was: Serial0/1 now: GigabitEthernet0/3
R6 was: Serial0/0 now: GigabitEthernet0/1
R6 was: Serial0/1 now: GigabitEthernet0/2
R5 was: Serial0/0 now: GigabitEthernet0/1
R5 was: Serial0/1 now: GigabitEthernet0/2
```

This case also passed the VIRL Engine checks and produced a valid .virl file, this was parsed in VM Maestro and produced the topology shown in Figure 6.7.

Figure 6.10: BGP - Topology visualised in VIRL



Once this .virl file was passed to emulation engine the result was a fully working BGP network. BGP peers defined in the real topology were able to start as intended and routes available and correct. In addition to the eBGP routing the interior IS-IS and iBGP routing works as intended and there is full network connectivity. Appendix I has some examples of the tests run to compare the emulated network to its hardware counterpart. As can be seen in when comparing BGP Peering for R1 there is one additional prefix for the real device, this is the route to the LAN segment which holds the discovery server and is omitted from the emulation.

Chapter 7

Evaluation

7.1 Summary of Tasks

The aim of this project was to provide an investigation into the potential uses of Network Emulation as a testing platform for medium to large enterprise. On the whole the tool produced enables an Engineer of such organisation to choose an area of their network to emulate, the tool pushes this from the discovery phase through to giving the user a jumphost in order to interact with their emulation. This is the method used to implement changes on any Cisco IOS device in large enterprise and would provide an Engineer with all the tools needed to implement a change. In addition to this CLI access to the IOSv devices, the tool also produces a jumphost into the management network of the emulation. Using this network an Engineer can make use of further network management tools to provide an in-depth view of the network through technologies such as Netflow, SNMP or Software Defined Networking.

Later parts of the Design phase included the implementation of a system which would provide a new type of interaction with an emulated network. Taking form as a webserver on the virl host it would allow the user to interact with the emulation through interaction with elements such as routers and links, these interactions would produce visual feedback to the user about the impact. This could be in the form of coloured links indicating current traffic volumes or routing decisions. Unfortunately due to some technical problems defined in Section 5.6.5 and restrictions on time this implementation was not possible and the project was de-scoped.

7.2 Difficulties Encountered

Throughout the project a few small difficulties were encountered, the one with the largest impact on development being the unavailability of HTTP access to the SNAT Network. However there were further constraints on the development of this tool, one notable constraint was the access that was allowed to the University infrastructure. As discussed in Section 5.1 the development was based on another emulation tool called GNS3, this was chosen for development as it is a lightweight method of emulating real Cisco IOS versions. In addition to this is provides connection methods such as virtualbox integration and the ability to connect the topology to the internet. The connection to the internet was provided via a Loopback interface on

the Windows desktop on which development was done, using Windows' in-built network tools the primary internet connection could be shared to this loopback, thus connecting the GNS3 emulation to the internet. Security restrictions on both the Loughborough VPN and Eduroam wireless network prevent computers with shared internet connections from connecting to the network. As the VIRL section of this tool requires a connection to the VPN this made direct connections from the GNS3 simulation through to VIRL. The solution to this problem was to use an intermediate server running SFTP as a storage area whilst the shared connection was disabled and the VPN connection was established.

A second area of difficulty was the interaction with the VIRL server. As VIRL is a relatively young platform there is only Vendor documentation to support it's use. In the development of this tool it was found that this documentation was not necessarily detailed enough for use outside of Cisco, this added difficulty in the later stages of development when interacting with the REST API on VIRL. One specific issue found was the time taken for a simulation to go live, this could change under seemingly random circumstances. This was exacerbated by another problem in that there was no way to know when an emulation was ready for use, a state flag was available for a device but for Ubuntu servers this would be flagged as Active 10-15 seconds before the server could accept network traffic. In order to overcome this uncertainty some parts of the tool use features such as timeouts on network requests and 'wait' statements in code in order to overcome changes in boot time, however this sometimes does not work and the VIRL server cannot respond causing exceptions in the code.

7.3 Future Work

As the development of this tool fell short to technical reasons, future work would first include the webserver outlined in Section 5.6.5. This webserver would provide a more user friendly experience whereas the tool in it's current state is primarily a set of command-line scripts. In addition to this additional user-experience it will also add higher-level methods of understanding how traffic is moving around a network, this will improve the tools use for those with less understanding of networks in Change Management scenarios.

Other future work would be to implement Layer 2 capabilities at it's full extent, Cisco are providing a Layer-2 IOSv image in later versions of VIRL which would accommodate this functionality. When implemented a User could emulate an entire network down to the access-layer of their topology, providing testing at granular levels such as ensuring Spanning Tree fail overs occur as designed and the use of VLANs for segregating network traffic in security scenarios.

A third improvement that could be made is to expand the functionality of this tool to include real-world servers. Virl allows the creation of pre-set Ubuntu images that can hold the full contents of a real server. The addition of these servers will allow the User to investigate the impact of network changes on a server's traffic and will enable the testing of the servers tolerance to network interruption and even emulate Disaster Recovery situations, understanding how servers fail-over in order to maximise uptime.

7.4 Thoughts on the project

I feel that this project was a success in its original concept of producing virtual networks for a Network Engineer to use for both testing and as a learning environment. However there is still further functionality that can be added to the tool in order to produce the abilities that were discussed in earlier sections. I feel that what has been produced gives a firm base for future work in this area as the ability to quickly produce a sandboxed, emulated network allows for in-depth testing such as the What-if? scenarios or granular traffic analysis without the need for expensive hardware. In addition to the uses for testing, as a student I can currently see uses for this tool in education. Allowing a lecturer to quickly produce emulated networks for students to test concepts or to implement their own topologies with. As discussed in Section 2.3, currently there is no industry standard use for virtual network environments. However I believe that in the coming years there will be uptake in these tools both as a testing and learning platform. In a high-stakes sector such as Finance any changes to infrastructure must be fully understood and documented before implementation can occur, tools such as this can provide an Engineer with this knowledge.

Bibliography

Stefan Behnel. *lxml 3.4.2*, 2015. URL `http://lxml.de/`.

Steven Michael Bellovin and Randy Bush. Configuration management and security. *IEEE Journal on Selected Areas in Communications*, 27(3):268–274, 2009.

Mike Bostock. *D3.js*, 2015. URL `https://github.com/mbostock/d3`.

Jeffery Case, Mark Fedor, Martin Schoffstall, and C Davin. A simple network management protocol (snmp), 1989.

Inc. Cisco Systems. *Packet Tracer, Version 6.0.1*, 2013. URL `https://www.netacad.com/web/about-us/cisco-packet-tracer`.

Simon Crosby and David Brown. The virtualization reality. *Queue*, 4(10):34–41, 2006.

Jeff Daniels. Server virtualization architecture and implementation. *Crossroads*, 16(1):8–12, 2009.

Developers. *telnetlib*, 2015. URL `https://docs.python.org/2/library/telnetlib.html`.

Fermın Galán, David Fernández, Javier Ruiz, Omar Walid, and Tomás de Miguel. Use of virtualization tools in computer network laboratories. In *Proc. 5th International Conference on Information Technology Based Higher Education and Training (ITHET 2004)*, pages 209–214, 2004.

Google. *ipaddr 2.1.11*, 2014. URL `https://code.google.com/p/ipaddr-py/`.

Jeremy Grossmann. *GNS3 1.2*, 2014. URL `http://www.gns3.com/index.php`.

Simon Knight, Askar Jaboldinov, Olaf Maennel, Iain Phillips, and Matthew Roughan. Autonetkit: Simplifying large scale, open-source network experimentation. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 97–98, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1419-0. doi: 10.1145/2342356.2342378. URL `http://doi.acm.org/10.1145/2342356.2342378`.

Evangelos Kotsovinos. Virtualization: blessing or curse? *Queue*, 8(11):40, 2010.

Mohit Lad, Daniel Massey, and Lixia Zhang. Visualizing internet routing changes. *Visualization and Computer Graphics, IEEE Transactions on*, 12(6):1450–1460, 2006.

Mark F Mergen, Volkmar Uhlig, Orran Krieger, and Jimi Xenidis. Virtualization for high-performance computing. *ACM SIGOPS Operating Systems Review*, 40(2):8–11, 2006.

MonkeyMental. *CDPR 2.4*, 2013. URL `http://www.monkeymental.com/`.

Mike Pennington. *ciscoconfparse 1.2.18*, 2015. URL `https://github.com/mpenning/ciscoconfparse`.

ShrubberyNetworks. *rancid 3.2*, 2014. URL `http://www.shrubbery.net/rancid/`.

Andrew J Younge, Robert Henschel, James T Brown, Gregor Von Laszewski, Judy Qiu, and Geoffrey C Fox. Analysis of virtualization technologies for high performance computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 9–16. IEEE, 2011.

Appendices

Appendix A

Function getLocalCDPInfo()

```
1  def getLocalCDPInfo(baseRouter):
2      #Function listens on interface for a CDP packet and creates a router if
           successful. Script closes if no packet found
3      try: #Catches exception generated when CDPR returns null (CDP packet not seen)
4          cdpdiscovery = subprocess.check_output(["cdpr", "-d", "eth0", "-t", "61"])
5          #Starts subprocess for CDP, timeout is set to 61 as default CDP timer is 60
6          #CDP returns the information in lines, below code breaks in into a list and
               processes the information
7          cdpList = string.split(cdpdiscovery,'\n')
8          cdpValues = [s for s in cdpList if re.search('value', s)]
9          for idx, val in enumerate(cdpValues):
10             cdpValues[idx] = val.replace(" ", "")
11         for idx, val in enumerate(cdpValues):
12             cdpValues[idx] = val.partition(':')[2]
13         #The below code creates a router object for the baserouter and returns it
               to main
14         if checkNetworks == True:
15             #Networks ranges have been specified for discovery
16             notInNetwork = True
17             ipAddress = str(cdpValues[1]) + '/32'
18             #Create string to hold device IP address
19             for idx, val in enumerate(specifiedNetworks):
20                 #Check each network that has been specified
21                 if ipaddr.IPv4Network(ipAddress) in val:
22                     #Check the ip address of the device is inside the network
23                     #If it is, create a device for it
24                     notInNetwork = False
25                     baseRouter.addHostname(cdpValues[0])
26                     baseRouter.addIpAddress(cdpValues[1])
27                     deviceCapability = getCapabilities(baseRouter)
28                     baseRouter.addCapability(deviceCapability)
29                     return baseRouter
30             if notInNetwork == True:
31                 #If the device is not in the network, throw an error
32                 print 'Local device is not in specied networks. Aborting Network
                       Discovery'
33                 sys.exit()
```

```
34          else:
35          #Networks have not been specified, create an object for the device
36          baseRouter.addHostname(cdpValues[0])
37          baseRouter.addIpAddress(cdpValues[1])
38          deviceCapability = getCapabilities(baseRouter)
39          baseRouter.addCapability(deviceCapability)
40          return baseRouter
41      except:
42          #Exception thrown, this will occur when no CDP packet is seen within the
                timeout of CDPR
43          print 'No CDP packet discovered. Aborting Network Discovery'
44          sys.exit()
```

Appendix B

Function updateRouters()

```
1  def updateRouters(updateRouter, topology):
2      routerHosts = getCDPHostnames(updateRouter)
3      routerIPs = getCDPIPs(updateRouter)
4      #For each connected device to the update Router
5      for idx, val in enumerate(routerHosts):
6          routerExists = False
7          for routerIDX, routerVal in enumerate(topology.routerList):
8              #Check whether the device already exists
9              if val == routerVal.hostname:
10                 routerExists = True
11         if routerExists == False:
12             #If the device is not already known
13             if checkNetworks == True:
14                 #If there is a specified network set
15                 ipAddress = routerIPs[idx] + '/32'
16                 deviceAdded = False
17                 #Check if device is within the specified subnets
18                 for specIDX, specVal in enumerate(specifiedNetworks):
19                     if ipaddr.IPv4Network(ipAddress) in specVal:
20                         if deviceAdded == False:
21                             #Create the device
22                             newRouter = router([])
23                             newRouter.addHostname(routerHosts[idx])
24                             newRouter.addIpAddress(routerIPs[idx])
25                             try:
26                                 deviceCapability = getCapabilities(newRouter)
27                                 newRouter.addCapability(deviceCapability)
28                                 hostnameList = getCDPHostnames(newRouter)
29                                 IPAddressList = getCDPIPs(newRouter)
30                                 dstInterfaceList = getCDPDstInterfaces(newRouter)
31                                 srcInterfaceList = getCdpSrcInterfaces(newRouter)
32                                 for newIDX, newVal in enumerate(hostnameList):
33                                     #Checks that the connected interfaces to the
                                           device fall within the defined subnets, if
                                           they do not, ignore them
34                                     newIPAddress = IPAddressList[newIDX] + '/32'
35                                     deviceAdded = False
```

```
                                for newSpecIDX, newSpecVal in
                                    enumerate(specifiedNetworks):
                                    if ipaddr.IPv4Network(newIPAddress) in
                                        newSpecVal:
                                        if deviceAdded == False:
                                            newCDP = cdpEntry([])
                                            newCDP.addHostname(hostnameList[newIDX])
                                            newCDP.addIpAddress(IPAddressList[newIDX])
                                            newCDP.addSrcPort(srcInterfaceList[newIDX])
                                            newCDP.addDstPort(dstInterfaceList[newIDX])
                                            newRouter.addCdpEntry(newCDP)
                                            deviceAdded = True
                            print 'Adding ' + routerHosts[idx]
                            topology.addRouter(newRouter)
                            updateRouters(newRouter, topology)
                        except:
                            #Exception thrown - device is unresponsive to SNMP
                            unresponsiveDevices.append(routerHosts[idx])
            else:
                #All networks are to be included. Add the device
                try:
                    newRouter = router([])
                    newRouter.addHostname(routerHosts[idx])
                    newRouter.addIpAddress(routerIPs[idx])
                    deviceCapability = getCapabilities(newRouter)
                    newRouter.addCapability(deviceCapability)
                    hostnameList = getCDPHostnames(newRouter)
                    IPAddressList = getCDPIPs(newRouter)
                    dstInterfaceList = getCDPDstInterfaces(newRouter)
                    srcInterfaceList = getCdpSrcInterfaces(newRouter)
                    for newIDX, newVal in enumerate(hostnameList):
                        #Add each interface to the device
                        newCDP = cdpEntry([])
                        newCDP.addHostname(hostnameList[newIDX])
                        newCDP.addIpAddress(IPAddressList[newIDX])
                        newCDP.addSrcPort(srcInterfaceList[newIDX])
                        newCDP.addDstPort(dstInterfaceList[newIDX])
                        newRouter.addCdpEntry(newCDP)
                    #add the new device to the topology
                    print 'Adding ' + routerHosts[idx]
                    topology.addRouter(newRouter)
                    updateRouters(newRouter, topology)
                except:
                    #Exception thrown - device is unresponsive to SNMP
                    unresponsiveDevices.append(routerHosts[idx])
```

Appendix C

Function findSwitches()

```
1  def findSwitches(topology):
2      #Function iterates through all devices, looking for switches on the topology
           (Two local CDP entries on the same interface)
3      print 'Looking for switches'
4      switchList = []
5      switchID = 1
6      #For all devices
7      for routerIDX, routerVal in enumerate(topology.routerList):
8          #If device is a router
9          if routerVal.capabilities == 'Router':
10             #Create a set to hold local interfaces on the router
11             seen = set()
12             uniq = []
13             dupes = []
14             for cdpIDX, cdpVal in enumerate(routerVal.cdp_entries):
15                 if cdpVal.srcPort not in seen:
16                     #If we haven't encountered this interface before add it to seen
17                     uniq.append(cdpVal.srcPort)
18                     seen.add(cdpVal.srcPort)
19                 else:
20                     #We have seen it before, therefore it is a duplicate
21                     dupes.append(cdpVal.srcPort)
22             #Remove any duplicates from dupes
23             set(dupes)
24
25             #If there are duplicate interfaces on this device
26             if len(dupes) > 0:
27                 #Iterate over the duplicate interfaces
28                 for dupeIDX, dupeInterface in enumerate(dupes):
29                     #List to hold local updates for both the router and switch
30                     switchUpdates = []
31                     #Create a device for the switch
32                     newSwitch = router([])
33                     #Hostname increments per device
34                     switchHostname = 'SW' + str(switchID)
35                     newSwitch.addHostname(switchHostname)
36                     #Set its capability
```

```
                    newSwitch.addCapability('switch')
                    #First local update holds the current device and switch updates
                    localUpdate = switchUpdate([])
                    localUpdate.addswitchHostname(switchHostname)
                    localUpdate.adddstHostname(routerVal.hostname)
                    localUpdate.addSwitchport(dupeInterface)
                    switchUpdates.append(localUpdate)
                    #Iterate over the CDP entries for the local device
                    for cdpIDX, cdpVal in enumerate(routerVal.cdp_entries):
                        #If the source port is the current duplicate
                        if cdpVal.srcPort == dupeInterface:
                            #Create an update for the DESTINATION interface and device
                            endUpdate = switchUpdate([])
                            endUpdate.addswitchHostname(switchHostname)
                            endUpdate.adddstHostname(cdpVal.hostname)
                            endUpdate.addSwitchport(cdpVal.dstPort)
                            switchUpdates.append(copy.copy(endUpdate))
                            endUpdate.clear()
                    switchExists = False
                    #Checks whether the defined switch has already been added
                    for deviceIDX, deviceVal in enumerate(topology.routerList):
                        if deviceVal.hostname == switchHostname:
                            switchExists = True
                    #If the switch does not already exist
                    if switchExists == False:
                        #Add it to the topology
                        topology.addRouter(newSwitch)
                        switchList.append(newSwitch.hostname)
                        switchID+=1
                    #For every update defined, create a CDP entry for the switch
                    for updateIDX, updateVal in enumerate(switchUpdates):
                        switchCDP = cdpEntry([])
                        switchCDP.addHostname(updateVal.dstHostname)
                        switchCDP.addSrcPort(str(updateIDX))
                        switchCDP.addDstPort(updateVal.switchport)
                        #Append the CDP entries to the switch device
                        newSwitch.addCdpEntry(copy.copy(switchCDP))
                    #Function call to update the CDP information on the routers
                    updateSwitchLinks(topology, switchUpdates, switchHostname)
```

Appendix D

topology.json Example


```
{
   "nodeList": [
      {
         "ipAddress": "172.30.0.1",
         "hostname": "R1",
         "deviceType": "Router",
         "deviceNo": 0
      },
      {
         "ipAddress": "172.30.0.2",
         "hostname": "R2",
         "deviceType": "Router",
         "deviceNo": 1
      },
      {
         "ipAddress": "172.30.0.3",
         "hostname": "R3",
         "deviceType": "Router",
         "deviceNo": 2
      },
      {
         "ipAddress": "",
         "hostname": "SW1",
         "deviceType": "switch",
         "deviceNo": 3
      }
],
   "linksList": [
      {
         "dstIpAddress": "",
         "dstRouter": "SW1",
         "srcRouter": "R1",
         "target": 3,
         "srcPort": "GigabitEthernet0/1",
         "value": 1,
         "source": 0,
         "dstPort": "1"
```

```
    },
    {
       "dstIpAddress": "",
       "dstRouter": "SW1",
       "srcRouter": "R2",
       "target": 3,
       "srcPort": "GigabitEthernet0/1",
       "value": 1,
       "source": 1,
       "dstPort": "3"
    },
    {
       "dstIpAddress": "",
       "dstRouter": "SW1",
       "srcRouter": "R3",
       "target": 3,
       "srcPort": "GigabitEthernet0/1",
       "value": 1,
       "source": 2,
       "dstPort": "5"
    },
    {
       "dstIpAddress": "",
       "dstRouter": "R1",
       "srcRouter": "SW1",
       "target": 0,
       "srcPort": "0",
       "value": 1,
       "source": 3,
       "dstPort": "GigabitEthernet0/1"
    },
    {
       "dstIpAddress": "",
       "dstRouter": "R2",
       "srcRouter": "SW1",
       "target": 1,
       "srcPort": "1",
       "value": 1,
       "source": 3,
       "dstPort": "GigabitEthernet0/1"
    },
    {
       "dstIpAddress": "",
       "dstRouter": "R3",
       "srcRouter": "SW1",
       "target": 2,
       "srcPort": "2",
       "value": 1,
       "source": 3,
       "dstPort": "GigabitEthernet0/1"
    }
  ]
}
```

Appendix E

Cisco Configuration File Example

```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname R3
!
boot-start-marker
boot-end-marker
!
!enable password <removed>
!
no aaa new-model
memory-size iomem 5
no ip icmp rate-limit unreachable
ip cef
!
no ip domain lookup
!
multilink bundle-name authenticated
!
!username rancid password <removed>
archive
log config
hidekeys
!
ip tcp synwait-time 5
ip ssh version 1
!
interface Loopback0
   ip address 3.3.3.3 255.255.255.255
!
interface FastEthernet0/0
   ip address 192.168.0.1 255.255.255.0
   duplex auto
   speed auto
!
```

```
interface Serial0/0
   description SER-R4
   bandwidth 500
   ip address 172.30.100.1 255.255.255.252
   clock rate 2000000
!
interface FastEthernet0/1
   ip address 172.30.0.3 255.255.255.0
   duplex auto
   speed auto
!
interface Serial0/1
   no ip address
   shutdown
   clock rate 2000000
!
   interface Serial0/2
   no ip address
   shutdown
   clock rate 2000000
!
interface Serial0/3
   no ip address
   shutdown
   clock rate 2000000
!
router ospf 1
   router-id 3.3.3.3
   log-adjacency-changes
   area 100 range 10.100.0.0 255.255.248.0
   network 3.3.3.3 0.0.0.0 area 0
   network 172.30.0.3 0.0.0.0 area 0
   network 172.30.100.0 0.0.0.3 area 100
   network 192.168.0.0 0.0.0.255 area 150
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
snmp-server community public RO
no cdp log mismatch duplex
!
control-plane
!
line con 0
   exec-timeout 0 0
   privilege level 15
   logging synchronous
line aux 0
   exec-timeout 0 0
   privilege level 15
   logging synchronous
line vty 0 4
```

```
   login local
!
end
```

Appendix F

Virl Server Cloud Init

```
#cloud-config
hostname: virl-sim-server
runcmd:
- start ttyS0
- systemctl start getty@ttyS0.service
- ifconfig eth1 up 10.254.0.36 netmask 255.255.255.0
- sed -i '1 i 127.0.0.1 virl-sim-server' /etc/hosts
- sed -i '/^\s*PasswordAuthentication\s\+no/d' /etc/ssh/sshd_config
- service ssh restart
- service sshd restart
users:
- default
- gecos: User configured by VIRL Configuration Engine 0.10.10
lock-passwd: false
name: cisco
plain-text-passwd: cisco
shell: /bin/bash
ssh-authorized-keys:
- VIRL-USER-SSH-PUBLIC-KEY
- ssh-rsa
    AAAAB3NzaC1yc2EAAAABJQAAAQEAkZj/7NygpIng1Sw/+Pd1yM6fjyfHNp9fnZ3UvJPATkFrm8+w5TynYuz/uNk4G0pR/ffT9S
rsa-key-20150211
sudo: ALL=(ALL) ALL
- gecos: User configured by VIRL Configuration Engine 0.10.10
lock-passwd: false
name: cisco
plain-text-passwd: cisco
shell: /bin/bash
ssh-authorized-keys:
- VIRL-USER-SSH-PUBLIC-KEY
- ssh-rsa
    AAAAB3NzaC1yc2EAAAABJQAAAQEAkZj/7NygpIng1Sw/+Pd1yM6fjyfHNp9fnZ3UvJPATkFrm8+w5TynYuz/uNk4G0pR/ffT9S
rsa-key-20150211
sudo: ALL=(ALL) ALL
write_files:
- path: /etc/init/ttyS0.conf
owner: root:root
```

```
content: |
# ttyS0 - getty
# This service maintains a getty on ttyS0 from the point the system is
# started until it is shut down again.
start on stopped rc or RUNLEVEL=[12345]
stop on runlevel [!12345]
respawn
exec /sbin/getty -L 115200 ttyS0 vt102
permissions: '0644'
- path: /etc/systemd/system/dhclient@.service
content: |
[Unit]
Description=Run dhclient on %i interface
After=network.target
[Service]
Type=oneshot
ExecStart=/sbin/dhclient %i -pf /var/run/dhclient.%i.pid -lf
    /var/lib/dhclient/dhclient.%i.lease
RemainAfterExit=yes
owner: root:root
permissions: '0644'
```

Appendix G

EIGRP Testing Evidence

This chapter compares the output from the hardware router and the emulated devices in VIRL.

G.1 OSPF Neighbour Table

G.1.1 Real Hardware

```
R1#sh ip ospf neighbor

Neighbor ID    Pri   State           Dead Time   Address         Interface
2.2.2.2          1   FULL/BDR        00:00:30    172.30.0.2      FastEthernet0/0
3.3.3.3          1   FULL/DR         00:00:32    172.30.0.3      FastEthernet0/0
```

G.1.2 Emulated Device

```
R1#sh ip ospf neighbor

Neighbor ID    Pri   State           Dead Time   Address         Interface
2.2.2.2          1   FULL/BDR        00:00:34    172.30.0.2      GigabitEthernet0/1
3.3.3.3          1   FULL/DR         00:00:35    172.30.0.3      GigabitEthernet0/1
```

G.2 OSPF Routing Tables

G.2.1 Real Hardware

```
R1#SH IP ROUTE
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
```

```
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route

Gateway of last resort is 192.168.137.1 to network 0.0.0.0

1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, Loopback0
2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/11] via 172.30.0.2, 00:37:44, FastEthernet0/0
3.0.0.0/32 is subnetted, 1 subnets
O       3.3.3.3 [110/11] via 172.30.0.3, 00:37:44, FastEthernet0/0
172.16.0.0/16 is variably subnetted, 5 subnets, 2 masks
S       172.16.0.0/24 is directly connected, Null0
O       172.16.0.0/16 is a summary, 00:38:20, Null0
S       172.16.1.0/24 is directly connected, Null0
S       172.16.2.0/24 is directly connected, Null0
S       172.16.3.0/24 is directly connected, Null0
172.30.0.0/16 is variably subnetted, 4 subnets, 2 masks
O IA    172.30.50.0/30 [110/210] via 172.30.0.2, 00:37:46, FastEthernet0/0
C       172.30.0.0/24 is directly connected, FastEthernet0/0
O IA    172.30.100.4/30 [110/220] via 172.30.0.3, 00:37:46, FastEthernet0/0
O IA    172.30.100.0/30 [110/210] via 172.30.0.3, 00:37:46, FastEthernet0/0
10.0.0.0/21 is subnetted, 2 subnets
O IA    10.50.0.0 [110/211] via 172.30.0.2, 00:37:46, FastEthernet0/0
O IA    10.100.0.0 [110/211] via 172.30.0.3, 00:37:46, FastEthernet0/0
O IA 192.168.0.0/24 [110/20] via 172.30.0.3, 00:37:46, FastEthernet0/0
C    192.168.137.0/24 is directly connected, FastEthernet0/1
S*   0.0.0.0/0 [1/0] via 192.168.137.1
```

G.2.2 Emulated Device

```
R1#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
a - application route
+ - replicated route, % - next hop override

Gateway of last resort is not set

1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, Loopback0
2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/2] via 172.30.0.2, 00:00:35, GigabitEthernet0/1
3.0.0.0/32 is subnetted, 1 subnets
```

```
O       3.3.3.3 [110/2] via 172.30.0.3, 00:00:45, GigabitEthernet0/1
10.0.0.0/8 is variably subnetted, 4 subnets, 3 masks
O IA    10.50.0.0/21 [110/202] via 172.30.0.2, 00:00:35, GigabitEthernet0/1
O IA    10.100.0.0/21 [110/202] via 172.30.0.3, 00:00:35, GigabitEthernet0/1
C       10.255.0.0/16 is directly connected, GigabitEthernet0/0
L       10.255.0.5/32 is directly connected, GigabitEthernet0/0
172.16.0.0/16 is variably subnetted, 5 subnets, 2 masks
O       172.16.0.0/16 is a summary, 00:01:30, Null0
S       172.16.0.0/24 is directly connected, Null0
S       172.16.1.0/24 is directly connected, Null0
S       172.16.2.0/24 is directly connected, Null0
S       172.16.3.0/24 is directly connected, Null0
172.30.0.0/16 is variably subnetted, 5 subnets, 3 masks
C       172.30.0.0/24 is directly connected, GigabitEthernet0/1
L       172.30.0.1/32 is directly connected, GigabitEthernet0/1
O IA    172.30.50.0/30 [110/201] via 172.30.0.2, 00:00:35, GigabitEthernet0/1
O IA    172.30.100.0/30
[110/201] via 172.30.0.3, 00:00:45, GigabitEthernet0/1
O IA    172.30.100.4/30
[110/202] via 172.30.0.3, 00:00:35, GigabitEthernet0/1
O IA  192.168.0.0/24 [110/2] via 172.30.0.3, 00:00:45, GigabitEthernet0/1
```

G.3 Example Traceroute

G.3.1 Real Hardware

```
R1#traceroute 172.30.100.6

Type escape sequence to abort.
Tracing the route to 172.30.100.6

1 172.30.0.3 40 msec 20 msec 8 msec
2 172.30.100.2 84 msec 40 msec 32 msec
3 172.30.100.6 44 msec 56 msec 44 msec
```

G.3.2 Emulated Device

```
R1#traceroute 172.30.100.6
Type escape sequence to abort.
Tracing the route to 172.30.100.6
VRF info: (vrf in name/id, vrf out name/id)
1 172.30.0.3 1 msec 5 msec 1 msec
2 172.30.100.2 8 msec 8 msec 14 msec
3 172.30.100.6 10 msec 10 msec 10 msec
```

Appendix H

EIGRP Testing Evidence

This chapter compares the output from the hardware router and the emulated devices in VIRL.

H.1 EIGRP Topology Table

H.1.1 Real Hardware

```
R1#sh ip eigrp topology
IP-EIGRP Topology Table for AS(1)/ID(172.30.8.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
r - reply Status, s - sia Status

P 10.1.2.0/24, 1 successors, FD is 10537472
via 10.1.35.2 (10537472/281600), Serial0/0
via 10.1.25.2 (20537600/281600), Serial0/1
P 10.1.26.0/24, 1 successors, FD is 10563072
via 10.1.35.2 (10563072/307200), Serial0/0
via 10.1.25.2 (20537600/281600), Serial0/1
P 10.1.25.0/30, 1 successors, FD is 20512000
via Connected, Serial0/1
P 10.1.35.0/30, 1 successors, FD is 10511872
via Connected, Serial0/0
P 172.30.3.1/32, 1 successors, FD is 128256
via Connected, Loopback2
P 172.30.2.0/24, 1 successors, FD is 128256
via Connected, Loopback1
P 172.30.0.0/21, 1 successors, FD is 128256
via Summary (128256/0), Null0
P 172.30.0.0/24, 1 successors, FD is 281600
via Connected, FastEthernet0/1

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
r - reply Status, s - sia Status
```

```
P 172.30.1.0/24, 1 successors, FD is 128256
via Connected, Loopback0
P 172.30.7.1/32, 1 successors, FD is 128256
via Connected, Loopback6
P 172.30.6.1/32, 1 successors, FD is 128256
via Connected, Loopback5
P 172.30.5.1/32, 1 successors, FD is 128256
via Connected, Loopback4
P 172.30.4.1/32, 1 successors, FD is 128256
via Connected, Loopback3
P 172.30.8.1/32, 1 successors, FD is 128256
via Connected, Loopback7
```

H.1.2 Emulated Device

```
R1#sh ip eigrp topology
EIGRP-IPv4 Topology Table for AS(1)/ID(172.30.8.1)
Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
r - reply Status, s - sia Status

P 10.1.26.0/24, 1 successors, FD is 10000640
via 10.1.35.2 (10000640/26112), GigabitEthernet0/1
via 10.1.25.2 (20000512/25856), GigabitEthernet0/2
P 172.30.2.0/24, 1 successors, FD is 128256
via Connected, Loopback1
P 10.1.35.0/30, 1 successors, FD is 10000128
via Connected, GigabitEthernet0/1
P 172.30.0.0/21, 1 successors, FD is 128256
via Summary (128256/0), Null0
P 10.1.2.0/24, 1 successors, FD is 10000384
via 10.1.35.2 (10000384/25856), GigabitEthernet0/1
via 10.1.25.2 (20000512/25856), GigabitEthernet0/2
P 172.30.1.0/24, 1 successors, FD is 128256
via Connected, Loopback0
P 172.30.3.1/32, 1 successors, FD is 128256
via Connected, Loopback2
P 172.30.6.1/32, 1 successors, FD is 128256
via Connected, Loopback5
P 10.1.25.0/30, 1 successors, FD is 20000256
via Connected, GigabitEthernet0/2
P 172.30.7.1/32, 1 successors, FD is 128256
via Connected, Loopback6
P 172.30.8.1/32, 1 successors, FD is 128256
via Connected, Loopback7
P 172.30.4.1/32, 1 successors, FD is 128256
via Connected, Loopback3
P 172.30.5.1/32, 1 successors, FD is 128256
via Connected, Loopback4
```

H.2 EIGRP Routing Table

H.2.1 Real Hardware

```
R1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route

Gateway of last resort is 192.168.137.1 to network 0.0.0.0

172.30.0.0/16 is variably subnetted, 10 subnets, 3 masks
C       172.30.3.1/32 is directly connected, Loopback2
C       172.30.2.0/24 is directly connected, Loopback1
C       172.30.0.0/24 is directly connected, FastEthernet0/1
D       172.30.0.0/21 is a summary, 00:01:18, Null0
C       172.30.1.0/24 is directly connected, Loopback0
C       172.30.7.1/32 is directly connected, Loopback6
C       172.30.6.1/32 is directly connected, Loopback5
C       172.30.5.1/32 is directly connected, Loopback4
C       172.30.4.1/32 is directly connected, Loopback3
C       172.30.8.1/32 is directly connected, Loopback7
10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
D       10.1.2.0/24 [90/10537472] via 10.1.35.2, 00:01:20, Serial0/0
D       10.1.26.0/24 [90/10563072] via 10.1.35.2, 00:01:21, Serial0/0
C       10.1.25.0/30 is directly connected, Serial0/1
C       10.1.35.0/30 is directly connected, Serial0/0
192.168.137.0/30 is subnetted, 1 subnets
C       192.168.137.0 is directly connected, FastEthernet0/0
S*   0.0.0.0/0 [1/0] via 192.168.137.1
```

H.2.2 Emnulated Device

```
R1#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
a - application route
+ - replicated route, % - next hop override

Gateway of last resort is not set
```

```
10.0.0.0/8 is variably subnetted, 8 subnets, 4 masks
D       10.1.2.0/24 [90/10000384] via 10.1.35.2, 00:01:40, GigabitEthernet0/1
C       10.1.25.0/30 is directly connected, GigabitEthernet0/2
L       10.1.25.1/32 is directly connected, GigabitEthernet0/2
D       10.1.26.0/24
[90/10000640] via 10.1.35.2, 00:01:38, GigabitEthernet0/1
C       10.1.35.0/30 is directly connected, GigabitEthernet0/1
L       10.1.35.1/32 is directly connected, GigabitEthernet0/1
C       10.255.0.0/16 is directly connected, GigabitEthernet0/0
L       10.255.0.3/32 is directly connected, GigabitEthernet0/0
172.30.0.0/16 is variably subnetted, 11 subnets, 3 masks
D       172.30.0.0/21 is a summary, 00:01:47, Null0
C       172.30.1.0/24 is directly connected, Loopback0
L       172.30.1.1/32 is directly connected, Loopback0
C       172.30.2.0/24 is directly connected, Loopback1
L       172.30.2.1/32 is directly connected, Loopback1
C       172.30.3.1/32 is directly connected, Loopback2
C       172.30.4.1/32 is directly connected, Loopback3
C       172.30.5.1/32 is directly connected, Loopback4
C       172.30.6.1/32 is directly connected, Loopback5
C       172.30.7.1/32 is directly connected, Loopback6
C       172.30.8.1/32 is directly connected, Loopback7
```

H.3 EIGRP Example Traceroute

H.3.1 Real Hardware

```
R1#traceroute 10.1.2.1

Type escape sequence to abort.
Tracing the route to 10.1.2.1

1 10.1.35.2 24 msec 8 msec 0 msec
2 10.1.2.1 88 msec 40 msec 40 msec
```

H.3.2 Emulated Device

```
R1#traceroute 10.1.2.1
Type escape sequence to abort.
Tracing the route to 10.1.2.1
VRF info: (vrf in name/id, vrf out name/id)
1 10.1.35.2 9 msec 1 msec 1 msec
2 10.1.2.1 6 msec 15 msec 9 msec
```

Appendix I

BGP Testing Evidence

This chapter compares the output from the hardware router and the emulated devices in VIRL.

I.1 BGP Summary

I.1.1 Real Hardware

```
R1#sh ip bgp summary
BGP router identifier 12.1.1.1, local AS number 5000
BGP table version is 188, main routing table version 188
19 network entries using 2280 bytes of memory
19 path entries using 988 bytes of memory
6/5 BGP path/bestpath attribute entries using 744 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 4036 total bytes of memory
BGP activity 21/2 prefixes, 21/2 paths, scan interval 60 secs

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
12.2.2.2        4  5000     137     137      188    0    0 02:13:07        0
12.3.3.3        4  5000     163     137      188    0    0 02:13:00       19
```

I.1.2 Emulated Device

```
R1#sh ip bgp summary
BGP router identifier 12.10.0.1, local AS number 5000
BGP table version is 19, main routing table version 19
18 network entries using 2520 bytes of memory
18 path entries using 1440 bytes of memory
5/5 BGP path/bestpath attribute entries using 760 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
```

```
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 4744 total bytes of memory
BGP activity 18/0 prefixes, 18/0 paths, scan interval 60 secs

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
12.2.2.2        4         5000      39      40       19    0    0 00:33:42        0
12.3.3.3        4         5000      46      39       19    0    0 00:33:40       18
```

I.2 IS-IS Topology Table

I.2.1 Real Hardware

```
R1#sh isis topology

IS-IS paths to level-1 routers
System Id           Metric    Next-Hop            Interface  SNPA
R1                  --
R2                  10        R2                  Se0/0      *HDLC*
R3                  10        R3                  Se0/1      *HDLC*
```

I.2.2 Emulated Device

```
R1#sh isis topology

Tag null:
IS-IS TID 0 paths to level-1 routers
System Id           Metric    Next-Hop            Interface  SNPA
R1                  --
R2                  10        R2                  Gi0/1      fa16.3e37.571b
R3                  10        R3                  Gi0/2      fa16.3eef.bcf0
```

I.3 Routing Table

I.3.1 Real Hardware

```
R1#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route
```

```
Gateway of last resort is 192.168.137.1 to network 0.0.0.0

155.100.0.0/30 is subnetted, 1 subnets
i L1   155.100.1.0 [115/20] via 12.0.0.2, Serial0/1
172.16.0.0/24 is subnetted, 4 subnets
S      172.16.0.0 is directly connected, Null0
S      172.16.1.0 is directly connected, Null0
S      172.16.2.0 is directly connected, Null0
S      172.16.3.0 is directly connected, Null0
12.0.0.0/8 is variably subnetted, 10 subnets, 3 masks
i L1   12.12.0.0/24 [115/20] via 12.0.0.2, Serial0/1
i L1   12.0.0.8/30 [115/20] via 12.0.0.6, Serial0/0
C      12.10.0.0/24 is directly connected, Loopback1
i L1   12.11.0.0/24 [115/20] via 12.0.0.6, Serial0/0
C      12.0.0.4/30 is directly connected, Serial0/0
C      12.5.5.0/24 is directly connected, FastEthernet0/0
C      12.0.0.0/30 is directly connected, Serial0/1
C      12.1.1.1/32 is directly connected, Loopback0
i L1   12.2.2.2/32 [115/20] via 12.0.0.6, Serial0/0
i L1   12.3.3.3/32 [115/20] via 12.0.0.2, Serial0/1
C   192.168.137.0/24 is directly connected, FastEthernet0/1
14.0.0.0/8 is variably subnetted, 9 subnets, 3 masks
B      14.12.0.0/24 [200/20] via 12.3.3.3, 01:16:44
B      14.10.0.0/24 [200/0] via 12.3.3.3, 01:16:44
B      14.11.0.0/24 [200/20] via 12.3.3.3, 01:16:44
B      14.0.0.8/30 [200/20] via 12.3.3.3, 01:16:44
B      14.6.6.6/32 [200/20] via 12.3.3.3, 01:13:46
B      14.4.4.4/32 [200/0] via 12.3.3.3, 01:16:44
B      14.0.0.4/30 [200/0] via 12.3.3.3, 01:16:44
B      14.5.5.5/32 [200/20] via 12.3.3.3, 01:13:46
B      14.0.0.0/30 [200/0] via 12.3.3.3, 01:16:44
S*  0.0.0.0/0 [1/0] via 192.168.137.1
```

I.3.2 Emulated Device

```
R1#sh ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
ia - IS-IS inter area, * - candidate default, U - per-user static route
o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
a - application route
+ - replicated route, % - next hop override

Gateway of last resort is not set

10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C       10.255.0.0/16 is directly connected, GigabitEthernet0/0
L       10.255.0.4/32 is directly connected, GigabitEthernet0/0
```

```
12.0.0.0/8 is variably subnetted, 12 subnets, 3 masks
C       12.0.0.0/30 is directly connected, GigabitEthernet0/2
L       12.0.0.1/32 is directly connected, GigabitEthernet0/2
C       12.0.0.4/30 is directly connected, GigabitEthernet0/1
L       12.0.0.5/32 is directly connected, GigabitEthernet0/1
i L1    12.0.0.8/30 [115/20] via 12.0.0.6, 00:34:27, GigabitEthernet0/1
C       12.1.1.1/32 is directly connected, Loopback0
i L1    12.2.2.2/32 [115/20] via 12.0.0.6, 00:34:27, GigabitEthernet0/1
i L1    12.3.3.3/32 [115/20] via 12.0.0.2, 00:34:27, GigabitEthernet0/2
C       12.10.0.0/24 is directly connected, Loopback1
L       12.10.0.1/32 is directly connected, Loopback1
i L1    12.11.0.0/24 [115/20] via 12.0.0.6, 00:34:27, GigabitEthernet0/1
i L1    12.12.0.0/24 [115/20] via 12.0.0.2, 00:34:27, GigabitEthernet0/2
14.0.0.0/8 is variably subnetted, 9 subnets, 3 masks
B       14.0.0.0/30 [200/0] via 12.3.3.3, 00:33:26
B       14.0.0.4/30 [200/0] via 12.3.3.3, 00:33:26
B       14.0.0.8/30 [200/20] via 12.3.3.3, 00:33:26
B       14.4.4.4/32 [200/0] via 12.3.3.3, 00:33:26
B       14.5.5.5/32 [200/20] via 12.3.3.3, 00:33:26
B       14.6.6.6/32 [200/20] via 12.3.3.3, 00:33:26
B       14.10.0.0/24 [200/0] via 12.3.3.3, 00:33:26
B       14.11.0.0/24 [200/20] via 12.3.3.3, 00:33:26
B       14.12.0.0/24 [200/20] via 12.3.3.3, 00:33:26
155.100.0.0/30 is subnetted, 1 subnets
i L1    155.100.1.0 [115/20] via 12.0.0.2, 00:34:27, GigabitEthernet0/2
172.16.0.0/24 is subnetted, 4 subnets
S       172.16.0.0 is directly connected, Null0
S       172.16.1.0 is directly connected, Null0
S       172.16.2.0 is directly connected, Null0
S       172.16.3.0 is directly connected, Null0
```

I.4 Example Traceroute

I.4.1 Real Hardware

```
R1#traceroute 14.0.0.9

Type escape sequence to abort.
Tracing the route to 14.0.0.9

1 12.0.0.2 8 msec 4 msec 4 msec
2 155.100.1.2 124 msec 48 msec 52 msec
3 14.0.0.6 [AS 6000] 36 msec 60 msec 40 msec
```

I.4.2 Emulated Device

```
R1#traceroute 14.0.0.9
Type escape sequence to abort.
```

```
Tracing the route to 14.0.0.9
VRF info: (vrf in name/id, vrf out name/id)
1 12.0.0.2 2 msec 2 msec 1 msec
2 155.100.1.2 8 msec 8 msec 16 msec
3 14.0.0.6 [AS 6000] 8 msec 16 msec 7 msec
```

I.5 BGP Peers on edge device

I.5.1 Real Hardware

```
R3#sh ip bgp summary
BGP router identifier 12.3.3.3, local AS number 5000
BGP table version is 27, main routing table version 27
19 network entries using 2280 bytes of memory
19 path entries using 988 bytes of memory
6/5 BGP path/bestpath attribute entries using 744 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
Bitfield cache entries: current 2 (at peak 3) using 64 bytes of memory
BGP using 4100 total bytes of memory
BGP activity 22/3 prefixes, 22/3 paths, scan interval 60 secs

Neighbor        V    AS MsgRcvd MsgSent TblVer  InQ OutQ Up/Down  State/PfxRcd
12.1.1.1        4  5000     138     164       27    0    0 02:14:40        0
12.2.2.2        4  5000     138     164       27    0    0 02:14:43        0
14.4.4.4        4  6000     125     126       27    0    0 01:50:13        9
```

I.5.2 Emulated Device

```
content...R3#sh ip bgp summary
BGP router identifier 12.12.0.1, local AS number 5000
BGP table version is 23, main routing table version 23
18 network entries using 2520 bytes of memory
18 path entries using 1440 bytes of memory
5/5 BGP path/bestpath attribute entries using 760 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 4744 total bytes of memory
BGP activity 18/0 prefixes, 18/0 paths, scan interval 60 secs

Neighbor        V          AS MsgRcvd MsgSent TblVer  InQ OutQ Up/Down  State/PfxRcd
12.1.1.1        4        5000     41      48       23    0    0 00:35:16        0
12.2.2.2        4        5000     42      48       23    0    0 00:35:18        0
14.4.4.4        4        6000     44      44       23    0    0 00:35:17        9
```