TOWARDS LARGE SCALE SOFTWARE BASED NETWORK ROUTING SIMULATION

Submitted in fulfilment
of the requirements of the degree of

MASTER OF SCIENCE

of Rhodes University

Alan Herbert

Grahamstown, South Africa
4 November 2014

Abstract

Software based routing simulators suffer from large simulation host requirements and are prone to slow downs because of resource limitations, as well as context switching due to user space to kernel space requests. Furthermore, hardware based simulations do not scale with the passing of time as their available resources are set at the time of manufacture.

This research aims to provide a software based, scalable solution to network simulation. It aims to achieve this by a Linux kernel-based solution, through insertion of a custom kernel module. This will reduce the number of context switches by eliminating the user space context requirement, and serve to be highly compatible with any host that can run the Linux kernel.

Through careful consideration in data structure choice and software component design, this routing simulator achieved results of over 7 Gbps of throughput over multiple simulated node hops on consumer hardware. Alongside this throughput, this routing simulator also brings to light scalability and the ability to instantiate and simulate networks in excess of 1 million routing nodes within 1 GB of system memory.

Acknowledgements

I would like to acknowledge and extend thanks to a number of individuals who gave me support and guidance through the duration of this research. Firstly I would like to give thanks to my supervisor, Prof Barry Irwin. His guidance and support has been key to my successes in this research.

I would like to also extend thanks to my family for their continuous support and care for my physical and mental well-being during the course of this year. Without my upbringing and freedom to chase my interests, this research would never have come to fruition.

I pass my gratitude on to Sindisiwe McDonald who took the time in her busy work week to proof read this paper and point out parts that needed further explanation due to assumed prior knowledge. In all, this helped me to produce a well rounded document that feels more complete.

I would like to thank the Computer Science Department at Rhodes University for providing me access to the equipment and space required to carry out this research.

Finally I wish to acknowledge the support of the Council for Scientific and Industrial Research (CSIR) and the Armaments Corporation of South Africa (Armscor) for the financial support for this research.

Contents

List of Tables

List of Figures

Abbreviations

The following list describes the various abbreviations and acronyms used throughout this document. Citations of research related to these areas listed below can be found in the text to better retain the context upon which the citation relies.

ALU	Arithmetic Logic Unit
API	Application Programming Interface
AS	Autonomous System
ASIC	Application-Specific Integrated Circuit
BGP	Border Gateway Protocol
eBGP	external Border Gateway Protocol
iBGP	interior Border Gateway Protocol
BSD	Berkely Software Distribution
CAIDA	The Cooperative Association for Internet Data Analysis
CD	Compact Disk
CORE	Common Open Research Emulator
DCCP	Datagram Congestion Control Protocol
DoS	Denial of Service
EiB	Exbibyte
EGP	Exterior Gateway Protocol
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
Gbps	Gigabits per second
GCC	GNU Compiler Collection
GIMP	GNU Image Manipulation Program
GPIO	General Purpose Input Output
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IC	Integrated Circuit
ICMP	Internet Control Message Protocol
IGP	Internal Gateway Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISEAGE	Internet-Scale Event and Attack Generation Environment
MAC	Media Access Control
MB	Megabyte

Mbps	Megabits per second
NAT	Network Address Translation
NIC	Network Interface Controller
NRL	Naval Research Laboratory
NTP	Network Time Protocol
OS	Operating System
OSPF	Open Shortest Path First
RIP	Routing Information Protocol
RSVP	Resource Reservation Protocol
SCTP	Stream Control Transmission Protocol
SIMD	Single Instruction Multiple Data
SSH	Secure Shell
TCP	Transport Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VHDL	Verilog Hardware Descriptor Language
VM	Virtual Machine
VOD	Video On Demand
WAN	Wide Area Network

1

Introduction

As the world comes to rely ever further on the availability of well-founded and reliable networks, the number of available services that reside on these networks grow. An amalgamation of these networks and services has led to what is known today as the Internet (Locke, 1995). This research seeks to find a solution for a testing and simulation environment that will allow a better understanding of these large scale networks, as well as a readily available tool for testing of services and systems in the isolation of a pseudo-live environment. The need for network simulation arises because currently business and educational organizations are driven to train users to use these physical networks, as well as develop systems for them (Cisco, 2009), with few well featured options to help in training and development such as an isolated network environments.

Few organizations have access to the hardware required for a large scale (non-production) routing infrastructure, or the funds to gain access to the routing simulator which meets their requirements. With this setback, organizations are forced into getting the basics of a new system right and fixing the errors by means of patches once the system is exposed to the Internet; these patches can be in the form of security and reliability and this can result in loss of service during patching (Potter and Nieh, 2005). Mitigation of these errors is further set back by staff that are inexperienced in the problem at hand due to

the fact that most study takes a theoretical rather than a hands-on, practical approach. Development of tools in this area can help mitigate production and educational costs producing better services and stronger candidates into the work force.

1.1 Problem Statement

With the growth of the Internet in recent years (this referring to the development into improved security and performance of web based applications and the growing availability of Internet enabled devices) most companies have invested into the Internet through means of advertisement or web based services (Internet World Stats, 2014). The role of the information technology division has developed from a strictly day-to-day office maintenance role, into a role that requires an understanding of the impact that a single line of code on a server side application can have on a client's experience of the service.

From a research point of view, protocol development to support the growing demands of the general user base of the Internet is a must. This development comes in with two sides as one needs to develop hardware support for these protocols along with the protocols themselves. The logic behind this hardware usually undertakes multiple iterations in its development cycles in order to achieve support of this protocol (Burns and Dennis, 1985). This means that to reduce the errors that propagates from the protocol design, one can benefit from extensive software testing before moving onto an in-hardware implementation. One should also test that the interaction between other protocols that exist on the network in consideration does not have an effect on the protocol in development, and vice versa.

This knowledge is usually learned through practice and not through theoretical studies of a topic where the finer details are often omitted or don't apply to the situations in which system administrators or developers may find themselves in. Some universities and higher education institutes do accommodate for these theoretical shortcomings through allocated time for practical hands on experience with pseudo-live systems. However, this usually falls short due to the sheer expense and/or difficulty of recreating pseudo-live environments in which a student can take part and study these finer details (James, 2011). This fact, coupled with the rate at which the Internet is growing and the variety of online services one can be expected to be familiar with, leads to further complexity in creating these pseudo-systems.

The need to facilitate major protocols used in the Internet is a crucial one in any form of network training. Even with the introduction of a larger address space made through the use of IPv6 (Deering, 1998), transport protocols such as UDP (Postel, 1980) and TCP (Postel, 1981c) stay relevant. These protocols are what most services really on for any network-based communications and as such, an understanding of how to control and manipulate these protocols is fundamental to any low-level network control as well as high level APIs and web based languages.

1.2 Research Goals

This research aimed to investigate and achieve the following goals:

1. An investigation into the viability of supporting simulation in software of IPv4 (Postel, 1981a) routing and interfacing to live physical hosts externally connected to the simulation. This feature creates the basis from which all simulation will be built up. Reliably supporting packet routing at a network layer level will allow for the addition of higher layer protocols to be successfully supported.

2. Further investigation into support within the routing simulator of TCP, UDP and ICMP networking transport protocols to allow for externally connected hosts to communicate via well-defined software applications and service mechanisms. With layer 4 protocol support in place, one is now able to successfully use applications that make use of layer 5 protocols such as FTP (Postel, 1985), HTTP (Berners-Lee *et al.*, 1996) and SSH (Ylonen and Lonvick, 2006).

3. Addition of features that allow delay and network jitter to simulate wireless link behaviour and congestion as well as packet loss and packet mangling in order to allow corruption of a packet in real time to further add to the realism of network routing. This will allow for real world use case simulation such as a lossy connection due to a wireless link interaction or misbehaviour within the physical layer of a network.

4. Finally, the ability for this software based network routing simulator to achieve 1 Gbps of throughput on commodity hardware and further investigation into projections of performance based on the development of technology and growth of the Internet. This will allow for planning in use case testing and application of this routing simulation software.

1.3 Scope

This research aims to remain within the scope of an easily accessible routing simulator, this means that it should not only be easy to acquire the hardware necessary for its execution, but also allow for easy configuration in a well supported software environment. For this reason this research looks towards laying its foundation within commonly available hardware as found in retail outlets.

Because of this limitation imposed on the research, the ability of application-specific logic units, super computers and micro-controller units to provide a foundation for this research's implementation on are discussed; however testing results do not consider the results that these systems yield in practice or theory.

As this research relies primarily on the simulation of stateless rule-based nodes within a network, ASs were removed from the scope of this implementation. ASs that make use of protocols such as BGP, RIP, OSPF and variants are therefore not included in the functionality of this routing simulator.

1.4 Document Structure

The remainder of this document is structured as follows. This document will begin with a literature review in Chapter 2. This will provide an overview of what simulation is and how it can be applied to models and systems that exist or are in production. This will then be followed by what routing is and what routing simulators exist for one's use. This chapter is also accompanied by tools and other means of applying network simulation to other areas of research and security.

Chapter 3, the design of this routing simulator, gives a detailed outline of what features this simulator will aim to provide and how these are achieved. Major choices are outlined through overviews of possible tools and methods to solve the problems that arise. Best practice and ease of use is considered while host system requirements are kept in focus throughout these design choices.

Some points on the finer details in the implementation of this routing simulator are brought to light in Chapter 4. These points include even distribution when considering randomly generated numbers, how to configure this implementation and the environment in which synchronization during testing was constructed.

The results of testing this routing simulator's implementation are documented in Chapter 5. These outline the testing environment used for this routing simulator and aim to show relevant results for system requirements and throughput acquisition of this routing simulator. The results collected from Chapter 5 allow for discussion on the possible use cases for this routing simulator in Chapter 6. These use cases include examples from testing of basic network infrastructures, such as a power grid or cellphone network, to using this routing simulator to allow for enhanced education and product testing.

This research document concludes in Chapter 7 with an overview of the results acquired in Chapter 5 and discusses comparisons between this routing simulator's ability to represent existing physical networks and the goals set out for this research. The chapter ends with a conclusion as to the success of this research and finally future extensions that can be made to this research.

2

Literature Review

This chapter provides background into what simulation is and what its applications are in a networking context. This is approached in Section 2.1, which firstly outlines what simulation is and why one should consider including simulation in different types, areas and stages of development of systems. A basic understanding of network routing is given and protocols used to ensure reliability within a network are detailed. Then these ideas are brought forward into a simulation aspect. The application of emulation is introduced in Section 2.2 and an example of its use is given as contrast to simulation. After this is clarified, a clear definition detailing the differences between simulation and emulation is then in Section 2.3.

Section 2.4 summarizes what network routing is and outlines commonly used protocols that are in place within network routing in order to ensure reliable and robust data traversal of these networks. After reaching an understanding of what routing is, this chapter continues into the idea of simulation of these routing protocols, where they can be applied, what existing platforms one can use to implement this type of simulation and also the use of existing implementations with a short review of each. This is covered in Sections 2.5 through to 2.7.

In drawing this chapter to a conclusion, Section 2.8 reviews means of data collection and analysis that can be used to better create these simulated environments. Section 2.9 takes a view on the use of network simulation in real life simulation models. This is followed by tools that can be used for network traffic generation in Section 2.10 and lastly the implementation of honey pots is reviewed in Section2.11, as network simulation can play a role in improving these defensive mechanisms.

This chapter is supported by illustrations where necessary to enhance explanation of a topic or discussion. Each section within this chapter presents itself with background into the topic at hand. This is then followed by how the simulation is applied to the topic and finally a discussion into what results or data can be yielded from these techniques is drawn.

2.1 Why Simulation

Simulation is a method in which one represents a model or an implemented system to discover key underlying mechanisms that create the behavioural patterns and results of that system or model (Maria, 1997). These patterns and results can lead to further understanding of a proposed model, or help to solve an existing problem in an implemented model.

As models and systems grow more complex, the ability to successfully predict an outcome from an overview, be it a graphical or mathematical model, diminishes. Simulation can help to further increase the accuracy of a predicted outcome for the relevant system in specific situations (TMN Simulation, 2011). This can increase the chance of the system's success, and thus the success of a company or research project.

To further explain the inability of an accurate prediction through means of graphical or mathematical models, one could consider this example. A company is required to serve VOD requests (Youtube is a good example of this[1]). Said company wishes to save on costs by cutting down on their excess servers that are often left idle due to lack of client requests. The data collected over a fixed period of time shows that the average client watches a video of six minutes in length, and that ten clients are serviced per hour.

With this data one can see that being able to service one client's request at a time over an hour would hold to constraints of the data. This is because ten clients in an hour

[1] http://www.youtube.com

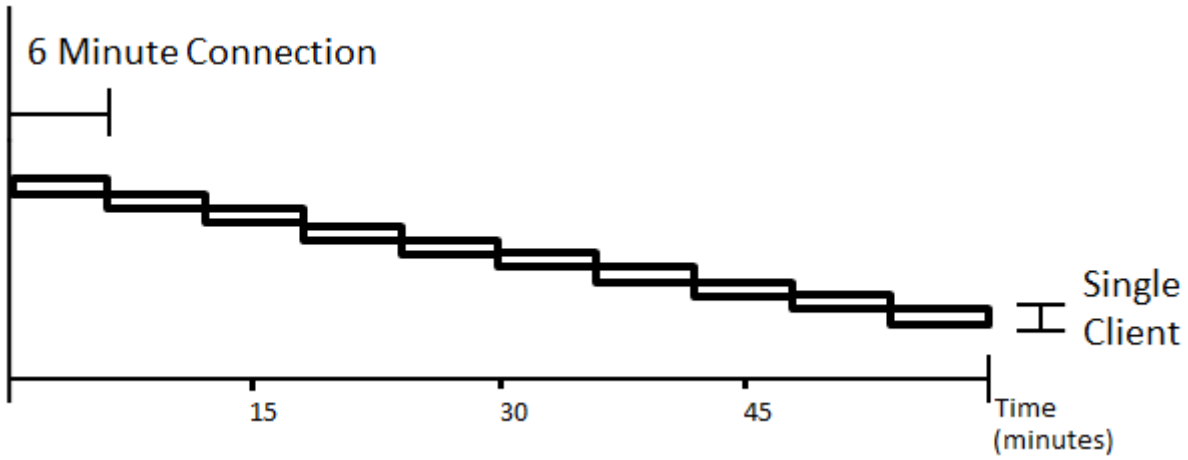


Figure 2.1: Expected Client Connection Projection using Average Case of VOD Service

averages a client every six minutes, if each client watches a six minute video, then only one client needs to be serviced at any point in the hour. This proposal, although it omits many details, holds on paper in its simplest form and is logically sound. However in a live implementation this proposal would fail.

A reason for this failure that one can immediately come to is the event that all ten requests are made in the first six minutes, or even the event that a request is made a few seconds before another client is finished viewing their video. There is also the fact that the average total sum of videos watched may be greater than six minutes, or that there may be more than ten clients to service. These, and many more, problems arise in a live implementation of a system that is no longer represented in a simple mathematical model or graphical representation of the system or model in question, as shown in Figure 2.1.

These oversights in terms of a VOD service based company can lead to a serious decrease in quality of service between server and client communications, the medium in which the videos are served. This can lead to further problems arising through average connections extending as a client waits for the service to be fulfilled and eventual client loss and failure of a company through loss of income.

Simulation should be considered when a system has a high volume of production, high investment, planned change in the existing system or when a high level of entropy exists within the system (TMN Simulation, 2011; Jelsim Partnership, 2009; Robinson, 2004). An example of some of these points was shown in the above example.

A high volume of production refers to a product that is usually mass produced, or a

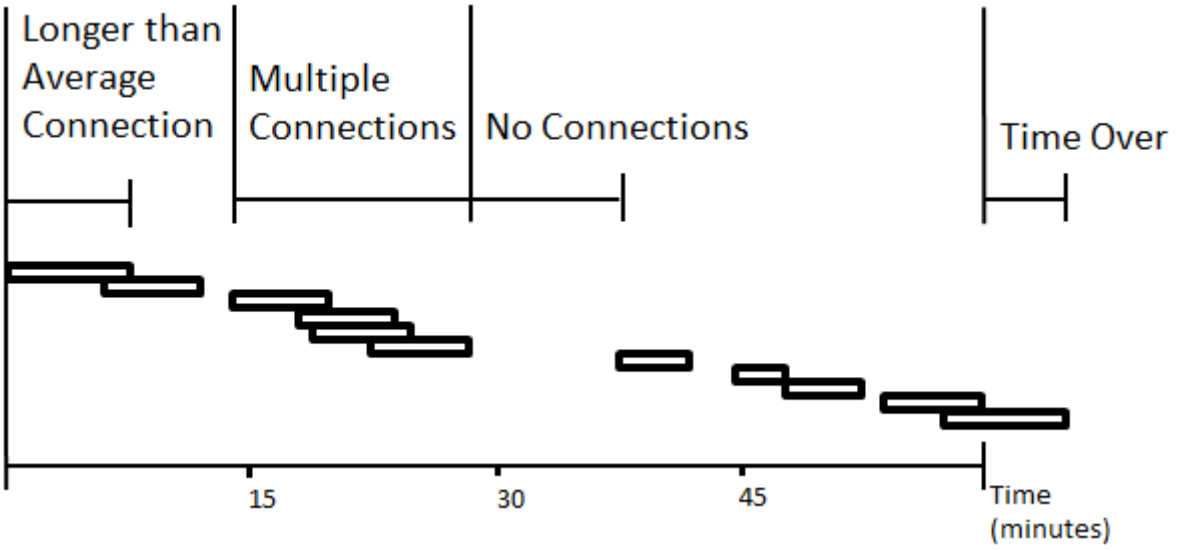


Figure 2.2: Real Client Connection Data from Implemented VOD Service

product that is wide spread, in terms of location, over the course of its lifetime. This product usually involves a large investment, in both funding and time, by the company that produces said product. If a problem arose within the product itself, the ability to recall or go to each client and service the product to mitigate the risen issue would require a large investment of man power, funding and time by the company. This can have dire effects on the client base's trust in the product and company.

With high cost products, a project's budget can usually only afford one development cycle. In such situations, simulation can increase the chance of a successful project. Failure with respect to this type of product or research can lead to companies or researchers never reaching their goals, and possibly never reaching their potential.

A change in a system or product should also cause developers consider simulation. This is due to the original development or production team not being present in the following iteration of a system or product. As such, finer details put in place by the developers in the past iteration may be overlooked, or parts of the system may be added that are redundant. These oversights can lead to the updated product having inferior performance or not fulfilling its purpose completely.

Lastly, a high chance of entropy is a good indication of a need to simulate. This is because one cannot say for sure what will enter the system. This entropy will often produce a wide range of errors in live implementation leading to multiple patches. Simulation can bring to light these errors before the product is finished, and more thought out solutions can be generated from these results that does not already have clients using it.

Although the above points cover many products, research projects, systems and models in existence, not all errors can be picked up through simulation. Unfortunately, there are errors that only arise once a system goes live and so one must not hold onto any false pretences that a system is safe because it has undergone simulation (Fishwick, 1995). However, a system that has undergone simulation is certainly more stable than one that has not.

2.2 Why Emulation

Like simulation, emulation can help in the creation of predictors for the outcomes of a product, system or model one wishes to design and test before physical implementation. It can reduce costs, development time and help in the training of users of a system much like simulation. However, as emulation does not attempt to model the in workings of a system, but rather only deal with producing the same output to a given input as the emulated system would, emulation tends more towards an "is this possible" rather than a "this is how it is created" tool (Stevenson and Lindberg, 2010).

If one omits the underlying mechanisms of a system one can start to adapt those systems to be understood and produce like results on other systems that do not contain the same underlying mechanisms. A simple example of this would be the Java Virtual Machine running on an AMD[2] or Intel[3] architecture (Lindholm *et al.*, 2014). The instructions fetched for execution within a Java application are operational codes native to the Java Virtual Machines. These codes are then emulated on the AMD or Intel architecture to take the same inputs and produce the same results as if the instructions were running on an architecture that natively understood Java instruction. The way in which these instructions are processed are of little importance: the only thing that matters is that they take and produce the same result, for this reason emulation is perfect in this situation.

2.3 Simulation Versus Emulation

To simulate means to mimic the underlying processes of an existing system. To emulate means to mimic the outcome of the entire system. Within a simulation one should be able

[2]http://www.amd.com/en-gb
[3]http://www.intel.co.za/content/www/za/en/homepage.html

to observe results and specifics about a the simulation that one would also observe in the real system. Usually the goal of emulation is to replace a mechanism within a system, or the system entirely, with the emulated system. Essentially, emulation can be thought of as a box that takes the target system's inputs in the one side and spits the same results out on the other side, where as simulation focuses more on what actually happens inside the box to produce the same result as the target system (Stevenson and Lindberg, 2010).

It is also worthy to note that simulation does not always result in emulation. The point on which a simulation fails to also emulate a system is the fact that a simulation can be slower than the system one is simulating. If this is the case, then the simulation cannot be said to emulate the target system.

Within the context of this paper's research topic and goals, simulation in this case would refer to a system that is able to produce the results and behaviours of a real network. This includes allowing for editing of underlying features within the simulation and then observation of how these modifications change the internal mechanisms of the simulated network. Producing a usable result, that being an actual packet being routed from one endpoint to another endpoint, is a result of all the smaller mechanisms producing results that fall within the expected range of a real network and its behaviour.

In the case of network emulation, one would not expect to have access to these underlying features. Instead of a system that is comprised of all the components that represent the physical network that would be simulated, an emulation seeks to replace these components with different parts that return the same results as what would be expected from the physical equivalent of the network.

In contrasting a simulated network against an emulated one, one will notice key differences in the way key objects are represented for replication of the physical network in question. A simulation would go about representing each individual router, link and any other significant entities as individual objects. In an emulation one would rather replace the bulk of components into more general pieces that act as the many parts that the simulation represents should.

Emulation does allow for greater potential to reach higher performance levels than that of a simulation; the reasons for this are simply a question of what the underlying hardware was designed to achieve. In simulation the hardware or underlying physical entity one wishes to represent in the simulation has to act the same and so the hardware that runs the simulation is expected to do tasks of that of the physical entity; a task that

the simulation's host hardware was never intended to perform. As a result simulation performance can be worse.

In the case of emulation, one only expects the result to be equivalent to that of the system being replicated. This creates the opportunity to take advantage of the simulation's host's hardware to achieve this task. One is no longer bound to the mannerisms and semantics of the system one wishes to replicate and so one can use the tools best fitted for producing the equivalent results that can be found within the simulation's host's hardware.

2.4 IP Routing

In its simplest form, to route a packet means to move a packet from a source host to a destination host through a network (Leighton *et al.*, 1994). What is required to route a packet through a network may very depending on the network. There may be a direct connection to the destined host from the source host, or the packet may need to undergo a series of transfers between routers, hubs and even load balancing systems[4].

There are three primary methods of routing. The first method is interior gateway routing through link states. This method keeps connection and session state to determine the link in which a packet is forwarded within an AS. The second method is interior gateway routing through use of path vectors or distance vectors, which is based on weights. These weights are defined by the volume of traffic on a link compared to the traffic it can handle and the distance between source and destination hosts. This method also serves to route packets within an AS. The last method is named the exterior gateway protocol and makes use of tree-like topologies to determine the route which a packet takes between ASs (Mills, 1984).

BGP makes use of path vectors to best determine the path of a packet. This protocol works within an AS between nodes referred to as peers. The paths these peers use to transfer packets within the AS is determined by predefined routes as manually configured by the system administrator. If a BGP runs between two peers within the AS, this is referred to as an iBGP. On the other hand, if the BGP connection is between two ASs it is known as an eBGP . These peers can also learn new routes, as well as share these routes between their peers (Rekhter *et al.*, 2006). BGP is the successor of EGP and is the most widely used exterior routing protocol on the Internet.

[4]System in place to distribute client requests among multiple servers.

Commonly, IGP is used for exchanging routes and other routing data between gateways inside an AS. This protocol can be broken into two major categories, these being distance-vector routing and link-state routing. RIP falls into the former category of exchanging routing information between gateways and makes use of distance values in order to determine the best link for a packet to be forwarded on. This works in such a way that each node within the AS, usually a router, advertises the distance values that it calculates to other nodes and receives their calculated distance values. From here each node updates its routing tables and then reiterates the initial step. This continues until distance values between nodes settle and converge (Malkin, 1998).

To touch on the latter IGP category, one can look to the OSPF protocol. This protocol relies on a core node within the network which holds all routing paths within the entire AS. In larger subdivided systems it holds routing paths common to that subdivision. This protocol also makes use of link costs which are used in decision making when it comes to forwarding a packet on a link. These costs can be drawn from a number of factors, from the distance of the link, to the reliability and throughput of the considered link (Moy, 1998; Coltun *et al.*, 2008).

Although these protocols were put in place to route packets, this is not their only purpose. These protocols serve to prevent loops from forming within the routing tables of a network. These protocols also attempt to optimize routes within a network to reduce the time which a packet takes to route from source host to destination host. This also reduces the number of packets within a network at any given time and thus allows more effective packets into a network for routing (Mills, 1984).

Design and configuration of ASs that implement BGP is made easier by BGP modelling tools such as C-BGP (Quoitin and Uhlig, 2005). C-BGP aims to reduce the difficulty of investigation of the implications due to changes within large ISP networks, be it internal or external. This tool helps to find bottle necks, identify under saturated components within a network and create a more effective AS. Benefits in this are lower network latencies within the AS and cost reductions in both equipment required and running costs of the AS.

2.5 Applications of Routing Simulation

As these networks grow in node count, so does the complexity of these networks; the Internet is an example of this. This complexity is also prevalent within internal networks

of the companies. These companies often run multiple systems and services on their networks. If a system were to be connected to these internal networks that had flaws in its networking code, it could lead to lower throughput, higher latencies and even complete denial of service within the internal network (Mahajan *et al.*, 2003). In this case, a simulated environment would allow for isolated testing of the effects of the new system or service on a replica network as designed to the internal network.

Network simulation is also used in both education and research sectors. This is often due to the large funding and space requirements for acquiring and housing the infrastructure required to produce a large physical network. A simulated network in this case would help reduce the costs and the floor space required for a network of this scale. A simulated network in this instance also provides easier configuration, higher control and a greater security than that of a physical one. These simulated networks can also be reconfigured and replayed far more easily than a physical one. This type of simulation is provided in software by the NS-3 project. NS-3[5] allows for scalable network creation and configuration, as well as connection to existing physical networks (National Science Foundation, 2013).

Simulation is also applicable to wireless network infrastructures. This is mentioned due to the increase in popularity of wireless protocols in recent years. This increase in popularity has led to the development of a variety of wireless protocols, such as the IEEE 802.11 and 802.16 families of specifications, chosen depending on one's requirements (Adachi, 2001). Although wired protocols are more set in their communication protocol standards, development in both wireless and wired mediums has led to an increase in bandwidth, and more reliable connections (Popov, 2009).

Security can also benefit from simulation. This can be done through simulating resources that are considered high value targets for an attacker. More and more companies are directing their resources to network services and opening their resources up for access so employees can work remotely. In this case access to these resources, or destruction of these services, can lead to a reduction in profits of a company (Stanescu, 2011).

2.6 Hardware, Software and Hybrid Simulation

Network simulation comes in three main types, each targeted to fulfil the needs of the user or client. These three types are hardware-, software- and hybrid-based simulation.

[5] http://www.nsnam.org/

In the following section these types are each considered through application example and then discussion on the advantages and disadvantages of each. Key specifics of each type of simulation platform are then detailed, as are specifics on how they function and what this adds or removes from the type of simulation platform.

2.6.1 Hardware

This type of approach is implemented by the companies Apposite Technologies (Apposite Technologies, 2013) and Packet Storm (Communications Inc, 2014). This hardware-specific design has a major benefit in processing power over the other two implementation types. This advantage rises from the fact that the network simulation, or packet generation, is processed on hardware specific for that given task. This hardware generally has an FPGA to mimic the hardware of that of a simulation device, or an ASIC that is a manufactured chip with the logic to perform the task at hand.

Although this option provides high performance, it also comes at a higher price. Network simulation is not for everyone, and so the demand for the development of the product is relatively small in comparison to that of regularly used devices such as computers or cell phones. This means that parts cannot be mass produced as the likelihood of using all the components in a time that is profitable is minimal. This has a direct effect on the manufacturing cost of this hardware (Devin and Warren, 1983).

There is a further detail that must be noted when undertaking an investment into a hardware-based system, which is that the hardware does not change in its computational power. This means that if it can do workload X today, it will still only do workload X in ten years time when networks will likely be faster and larger and the network it is intended to simulate will almost certainly have been upgraded.

The Field Programmable Gate Array

An FPGA is a chip designed to simulate hardware logic, giving rise to the ability to easily prototype or replace existing hardware. FPGAs also allow for replacement of obsolete hardware. This is one of the objectives of the GOP-XC3S200 produced by OHO-Electronik (Randelzhofer, 2009) and the C-Mod developed by Digilent (Digilent, 2004).

The FPGA's ability to simulate hardware before a design goes into full production (through the chip's reprogrammable capability) allows the designer to mitigate prototyping costs.

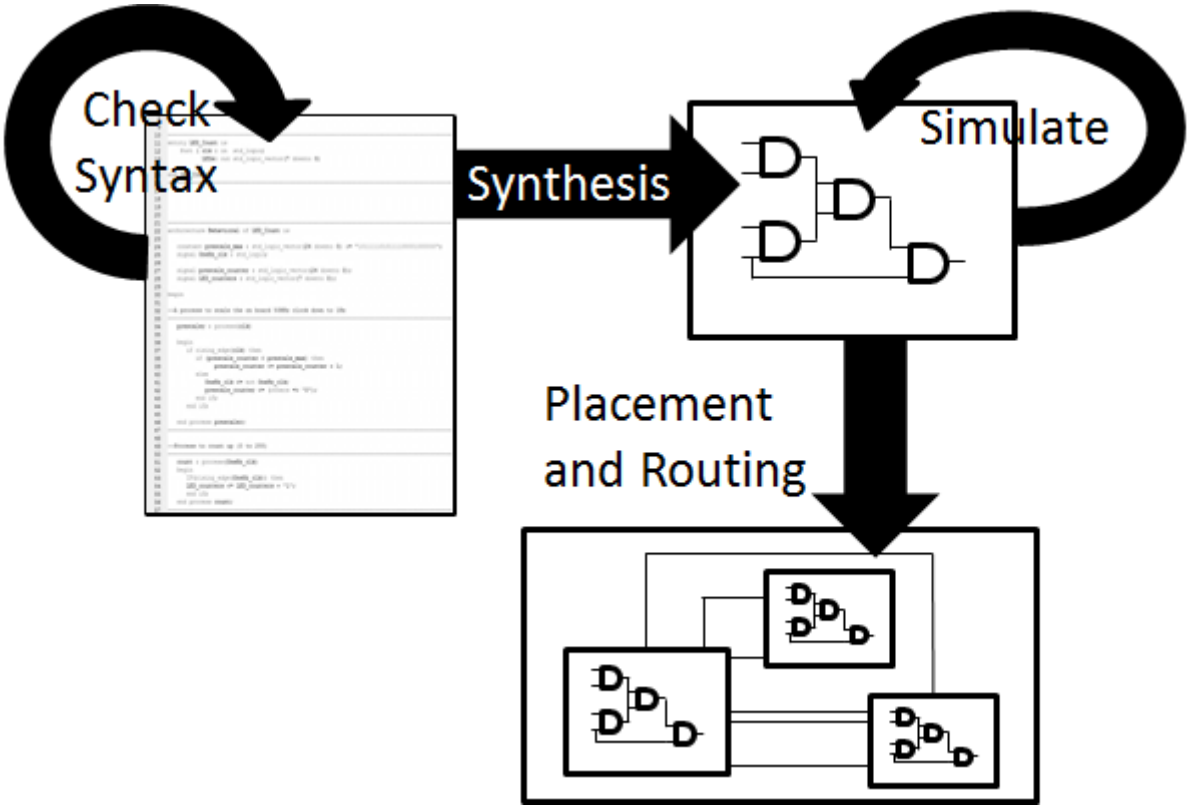


Figure 2.3: FPGA Synthesis Process

This also allows for better prototyping and so reduces the chance of a hardware failure once the device reaches the consumer. These FPGAs are also increasing in simulation speed to a point that they are often faster than the hardware they are intended to replace (Xilinx Inc., 2014).

These chips are programmed through Verilog or VHDL. These languages seek to achieve a task different from the conventional sequential languages (such as C/C++ or Java) that micro-controllers are typically programmed in. Rather than the language providing instructions for hardware to follow, like C/C++ or any other language intended for CPUs or GPUs, Verliog and VHDL describe what the hardware should be like in order to complete the given task.

Another difference is in compilation of these languages. Conventional languages such as C/C++ or Java would parse the written code into instructions that can be processed by the intended platform. Verliog and VHDL utilise synthesis rather than compilation. Figure 2.3 illustrates this synthesis which goes about translating the Verilog and VHDL code into the logic gates and how they should be connected in order to achieve the task of the hardware (Std, 1988). Further steps in this process include simulation to ensure there

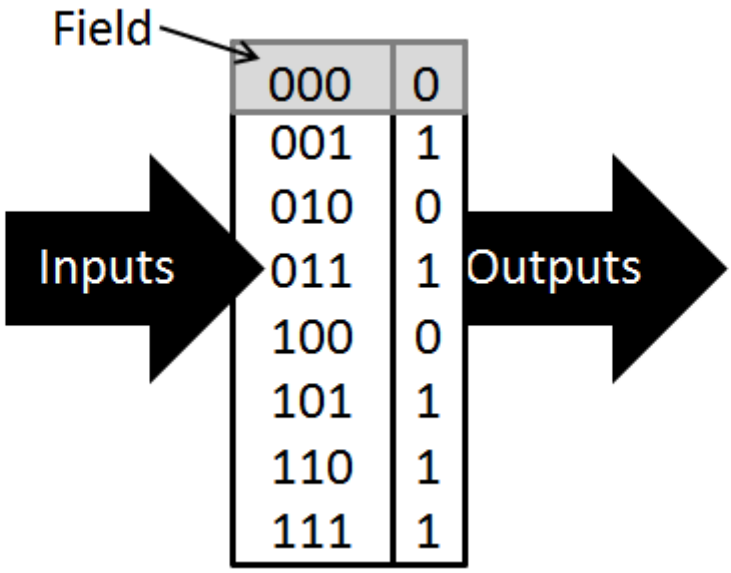


Figure 2.4: Simulated Logic Gate

are no short circuits, floating pins and other such bad circuitry. This is then followed by placement and routing, which maps the simulated hardware to slices in the FPGA. The final compiled form is called the bitstream and is loaded onto the FPGA in the boot up sequence of the device.

The logic gates that make up the simulated hardware are constructed through population of fields within tables inside the FPGA. These fields simulate the actual logic gates of the hardware through a process called a flow. Figure 2.4 is a simplified explanation of this process. Essentially, a logic gate is a lookup table that, as shown in Figure 2.4, takes inputs in on one side and returns the corresponding result out of the other. The output can be an input to other lookup tables, or can be an output to a buffer that drives a pin for GPIO.

FPGAs have been used to great effect in the areas of packet generation and packet analysis. The former can be found in a product such as BreakingPoint by Ixia (Ixia, 2014). This product aims to generate traffic according to applied filters that aim to stress test systems and networks. This allows for analysis of the problems within the security of these systems and networks. These problems can then be dealt with by the development and maintenance teams.

NetFPGA, a project by Stanford University, focuses on packet analysis and modification (Stanford University Engineering Computer Science, 2013); these two features are disjoint and do not depend on each other for operation. Every packet introduced into

or out of a network interface that is attached to the NetFPGA[6] is passed through to the on-board FPGA. This FPGA can perform analysis, modification or both analysis and modification to the packets passing through the interface. This allows for one to mitigate the expense of packet analysis and modification's overhead from software execution by porting its operations to the FPGA on the NetFPGA hardware.

Application-Specific Integrated Circuit

This is not a simulation device like the FPGA; instead this chip is an IC designed for a specific use. What ASICs and FPGAs do have in common is the language they use. Like FPGAs, ASICs use Verilog and VHDL to describe the logic flow in order to perform the specified task. ASICs sharing a language with a logic simulation devices like the FPGA has its benefits. Because of this one can effectively simulate hardware intended for an ASIC on an FPGA before fabricating the design into ASIC. Use of an FPGA in this situation allows for simulation to be effectively applied to an IC such as an ASIC to mitigate any design flaws before production of the circuit.

Such applications of this logic level implementation can be found in Netronome's product line (Netronome, 2014). ASIC implementation can be found on the NFP-6xxx Flow Processor series[7]. This solution allows for connection link speeds of up to 100 Gbps through each of the four interfaces located on the card.

2.6.2 Software

A software approach to network simulation is slower than that of a hardware approach discussed above. This is primarily due to the fact that the software implementation is being processed on hardware not specifically designed for the implementation. This can also lead to tedious software design due to the work-arounds necessary in order to run the implementation on hardware not intended for the desired purpose.

Software implementations do also have their advantages, these being lower monetary cost and freedom of design. One can find free implementations like NS-3 (National Science Foundation, 2013) that help to mitigate costs for smaller network or low throughput simulations that one may require but not have the funds for a hardware implementation,

[6] Of which there are four separate interfaces.
[7] http://www.netronome.com/product/development-platforms/

or low throughput simulations that one may require despite lacking the funds for a hardware implementation. These software implementations also provide modularity, which allows one to easily pull out one software module and replace it with another without requiring extra hardware to provide that module's functionality. Furthermore, software modules do not require the specialised skills or knowledge that may be needed to install a hardware module.

Multi-core Processors versus Hyper-Threading

This class of processor can process more than one application in parallel at a time. This is achieved through multiple physical cores within a single processor (Ramanathan, 2006). Single-core processors simulate parallel processing through the speed at which they perform a piece of a task, switch to another task, and repeat that process throughout all tasks on the process queue.

Hyper-threading is a form of multi-threading technology that simulates multiple cores through duplication of certain hardware within a processor. The hardware that is duplicated within a processor is the parts that hold the logical state of the program; the actual parts that do the work of the program are not duplicated. This allows for a reduction of the overhead required to switch between processes within the process queue, and is done by loading the set of registers that aren't being worked on (Marr *et al.*, 2002).

To simplify this concept, one can describe a conventional processor as having a set of registers that store the values to do the work, and then ALUs to do the work. If one were using this design to process a task, one would load the registers and then perform the required operations on the ALUs. When it is the turn of the next process to use the processor, there would be an overhead: writing out the current process's register values and then loading in the values of the next process in order to perform the required operations of that process. This leaves a chunk of time in which the ALUs are not being used as the old process is being written out and the new one is being written into the registers in the processor.

Instead, a hyper-threaded processor has two sets of registers and one set of ALUs. This means that one set of registers can interact with the ALUs while the other set is being read out, and while new data is being written in. This means that no time is lost through idling the ALUs while a process is being switched into registers to access the ALUs.

2.6.3 Hybrid

Hybrid simulation is derived from the simulation system being comprised of both software and hardware parts. This type of simulation also comes with advantages and disadvantages. Hybrid systems allow for the advantage of using readily available commodity hardware to process the software parts of the simulation; this reduces the funding required for such a system. Furthermore, the intensive sections of the simulation's process can be transferred onto a hardware-specific module that can be attached to the generic hardware that processes the software. This hardware can then be given tasks by the software side of the implementation to handle the process intensive segments of the simulation. This design means that menial tasks can be performed by the software and so one does not have to include this support in hardware; this reduces the cost for this type of simulation.

The disadvantage in this approach comes in the communication between the software and the specific hardware. Each component that is introduced into a system like this is subject to communication over a form of a bus[8]. This bus is shared by all components of the simulation system and so a slow-down can occur in bus contention (Null, 2006, Pages 33, 179-181). Slow-downs can also occur in the bandwidth limit of the system's bus: if the bus is not fast, or wide enough[9] this will slow down the data transfer rates between the hardware device and software application.

Referring to the NetFPGA mentioned in Section 2.6.1, it is important to note that although this device can act as a stand alone device within a system, it can also access the memory on the system that hosts it. As the NetFPGA has to access this memory through the host systems bus, large data throughput might lead to bus contention and ultimately a loss in expected performance by the NetFPGA.

2.7 Network-Based Simulation and Emulation Solutions

This section briefly evaluates existing simulation and emulation solutions and provides an overview of the services provided by, as well as the features and configuration of these existing implementations. These existing implementations are not limited to network simulation alone, but also allow for traffic generation. Both software and hardware implementations will be considered and contrasts will be drawn where possible.

[8] Derived from the Latin word omnibus, which means 'for all.'
[9] In terms of how much data can be moved in one clock cycle.

2.7.1 NS-3

NS-3 is a Linux-based routing simulator that is available for free download from the NS-3 website[10] and stems from NS-2[11]. NS-3 is developed with education in mind and is licensed under the GNU GPLv2 license as open source software. As such, the National Science Foundation encourages a large community of users and developers and supports open validation of its releases by this community (National Science Foundation, 2013).

NS-3 can be configured from file and configuration is built through a C++ environment. The configuration files are transformed from this into the XML mark up language and from here can be loaded into the NS-3 environment. These XML configuration files can be manually created if one would rather approach the configuration in that fashion; this is relevant as it allows for one to manually fine tune configuration scripts. It is also interesting to note that NS-3 also supports configuration from raw text files using specific directives within the text.

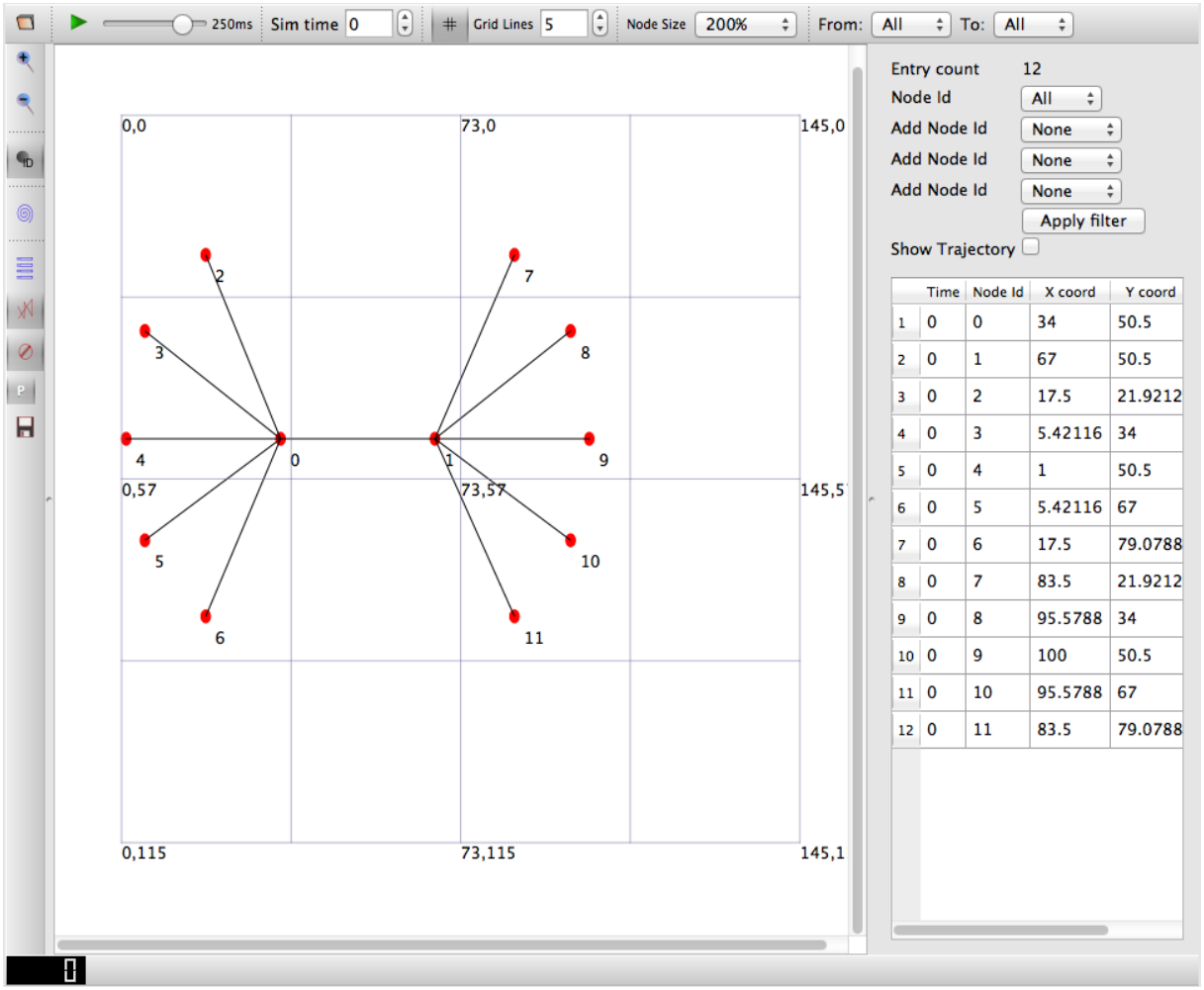
NS-3 takes the approach of full node simulation. This means that an endpoint within the simulation is treated as a simulated host. This allows for a reduction in additional hardware needed to source and sink traffic into and out of NS-3. These endpoints can be individually configured as to which is a client and which is a host, or each can be configured as both if one wishes. From here one can further add a range of communications and traffic interactions between these endpoints within the simulation.

It is notable at this point that NS-3 also boasts a GUI, and an example is depicted by Figure 2.5. This graphical interface can be used to better observe the simulated network's topology and communication through animations that occur on screen that symbolize the various network events and traffic. The simulation speed can also be controlled from this interface, along with various other options.

2.7.2 Linktropy and Netropy

Linktropy and Netropy[12] are hardware implementations of traffic generators developed by Apposite Technologies. The traffic generated is aimed at the form of WAN and includes

[10] http://www.nsnam.org/
[11] http://www.isi.edu/nsnam/ns/
[12] http://www.apposite-tech.com/index.html



	Time	Node Id	X coord	Y coord
1	0	0	34	50.5
2	0	1	67	50.5
3	0	2	17.5	21.9212
4	0	3	5.42116	34
5	0	4	1	50.5
6	0	5	5.42116	67
7	0	6	17.5	79.0788
8	0	7	83.5	21.9212
9	0	8	95.5788	34
10	0	9	100	50.5
11	0	10	95.5788	67
12	0	11	83.5	79.0788

Figure 2.5: NS-3 GUI

both IPv4 and non-IPv4 packet generation. This hardware is designed to aid in the testing of systems and links that spread over a large topology. An example of this would be a company's main offices and their off-site servers (Apposite Technologies, 2013).

Both Linktropy and Netropy generate traffic of the same form factor. However, their key difference arises in the bandwidth they provide. The Linktropy range provides single- or quad-port gigabit Ethernet WAN traffic generation, whereas the Netropy range provides up to 40 Gbps of WAN traffic distributed through multiple disjoint WAN traffic generators.

There exists a third range from Apposite Technologies namely the Linktropy Mini. This is approximately the size of a CD and comes in 2 separate grades of WAN traffic generation, these being 100 Mbps and 1 Gbps bandwidth generation. These Linktropy Minis are marketed with the mindset of ease of use for development environments to increase quality of the current revision of an implementation.

These implementations are configured through a GUI that is served as a website. To access this interface one simply needs to connect to the address of the hardware using a web browser. From here all settings are available to the user to allow a test iteration to be as granular as one needs. This interface can be viewed in Figure 2.6.

2.7.3 Packet Storm Range

Packet Storm, Inc.[14] produces a range of network simulation hardware, and like the aforementioned Linktropy and Netropy, the range focuses on WAN traffic generation. These products vary in name and thus there is no specific range of device - their products can better be referred to by a class. These classes are: network traffic generators, WAN specific traffic generators, packet stream duplicators, replay systems and packet capture systems (Communications Inc, 2014).

Similar in implementation to the Linktropy and Netropy, Packet Storm's range of devices also make use of configuration through a web-based GUI. Some of their devices, like the PacketStorm 200E, also include a display on the front panel of their housing. This display is accompanied by buttons, as seen in Figure 2.7, that allow for configuration and monitoring through direct means rather than through a web browser interface.

[13] http://i.ytimg.com/vi/CnObOoZ4wNs/hqdefault.jpg
[14] http://packetstorm.com/

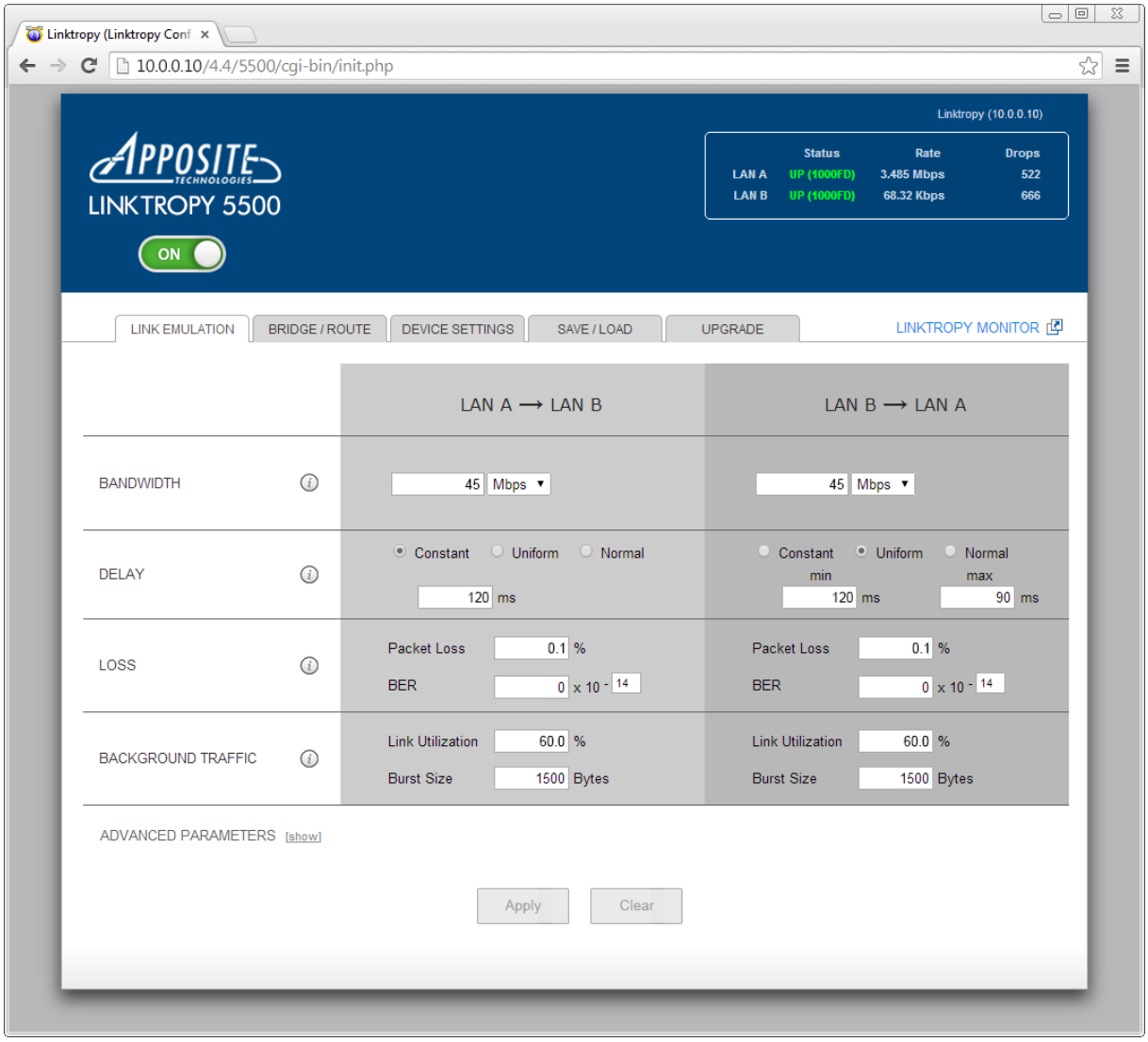


Figure 2.6: Linktropy 5500 by Apposite Technologies Web Based GUI[13]

Figure 2.7: PacketStorm 200E WAN Emulation and Network Simulation Device[15]

These devices' bandwidth generation ranges from 1 Gbps, at their entry level class, to 20 Gbps via their flagship model. This 20 Gbps bandwidth is generated through two disjoint 10 Gbps NICs. This hardware allows for impairment change and other configuration revisions at runtime, allowing one to change the context of a test iteration without having to stop the device and begin a new run; this is good for simulated traffic change as based on time of day and region.

2.7.4 BreakingPoint

BreakingPoint is developed by IXIA[16] and is a hardware implementation of a traffic generator. The system is designed as a host system that gives space for expansion through daughter boards named Blades. This means that an upgrade to the BreakingPoint system's cost is mitigated through the re-usability of the host system hardware. Furthermore, multiple Blades can be run in parallel on a single host system, which allows for upgrades to provide extended functionality rather than replacement of an implemented system (Ixia, 2014).

As the base host system does not change, one should look towards these Blades in order to get an idea of what a user can be provided with IXIA's systems. Their Blades generate

[15]http://www.comworth.com.sg/products/packetstorm-ip-network-emulators/
[16]http://www.ixiacom.com/

Figure 2.8: BreakingPoint Web Based GUI[17]

traffic ranging from 1 Gbps of unique traffic through each of eight separate interfaces totalling 8 Gbps of bandwidth generation, to 40 Gbps of unique traffic through each of two ports to total 80 Gbps of bandwidth generation. The traffic generated by these Blades is configured through the host system through the means of a replay of a known session, or through the means of a pattern or template of what the traffic should be formed as.

Like Apposite Technologies and Packet Storm's implementation of a GUI, BreakingPoint also makes use of configuration through a web-based interface. To access this, one simply needs to have a compatible browser with which to access the address of the Breaking-Point device in question. Figure 2.8 shows the BreakingPoint landing page showing the features that the system makes available for use by a user.

Besides BreakingPoint, IXIA offers a range of services and solutions that range from security systems, hardware based firewalls and testing to load distribution servers.

[17] http://blogs.ixiacom.com/default/assets/Image/Blog%20Images/Release%202.1%20Blog%20Post%206.png

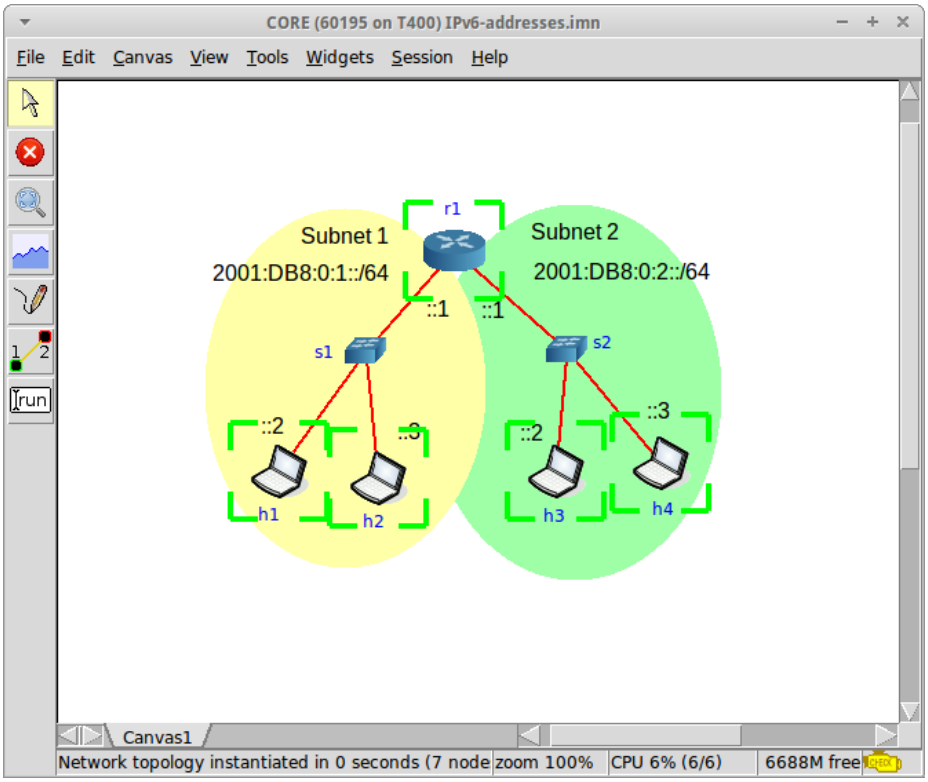


Figure 2.9: CORE GUI[20]

2.7.5 CORE

CORE[18] is a Linux-based software network simulator developed by the NRL. This network emulator makes use of Linux containers to achieve the task of node simulation (Naval Research Laboratory, 2014). This style of network simulation allows for full host endpoint simulation as well as full host simulation of each node.

The links within this network emulator are handled through software such as Quagga or Zebra[19] to simulate routers and switches as commonly found within a network's topology. As these nodes acting as routers and switches run a full Linux stack, they can be accessed through SSH and modified at the runtime of the emulated environment. This allows for more detailed emulation to mimic physical world events that may occur on a network.

This implementation runs through a GUI that provides drag and drop tools to create a network topology as one needs, which can be viewed in Figure 2.9. The green squares around the separate devices represent that their containers have been initialized and

[18] http://www.nrl.navy.mil/itd/ncs/products/core
[19] http://www.nongnu.org/quagga/
[20] http://www.brianlinkletter.com/wp-content/uploads/2014/05/IPv6-simple-LAN-120.png

that they are online. Loading and saving a network configuration from and to file is also supported by CORE which allows for re-usability of known configurations. It should be noted that even though one can create and destroy links and new nodes within the GUI at runtime, these modifications only take place when a new session is started.

This emulator allows for wired or wireless connections to be created within the environment over both IPv4 and IPv6 protocols. Furthermore, one can set the wireless devices to move at runtime and set connection ranges of these devices to dynamically connect and disconnect from the network as the emulation progresses; this makes mesh network emulation possible.

2.7.6 ISEAGE

ISEAGE is laboratory designed to hold a testbed specifically to create a virtual Internet environment (Jacobson, 2007). This testbed provides a controlled environment in which to allow for research, design and testing of cyber defence mechanisms. The ISEAGE laboratory contains equipment specific for recreating these attacks on point-to-point and distributed configurations against any configuration one my set up within the laboratory.

This testbed is based on cluster computing and as such configuration can be done on a system to system basis, or through grouping sets of systems into larger systems and configuring them through the relevant means. As this is hardware based one can also introduce in specific ICs into the testing environment in order to perform an attack. As this testbed uses current hardware and software, it is expected that results produced by these systems are extremely accurate.

2.8 Collecting Routing Data

Enumerations of a network can be performed through the use of the program traceroute. The windows equivalent of this program is tracert and comes standard with the Windows OS. One can use these programs with their default settings by calling the program name and an IP to which one wants to trace a route. One can also specify one or more of the options made available by these programs in order to acquire more accurate data. Both of these programs allow for a range of protocols to be used in an attempt to penetrate into networks that would usually drop the packets due to firewall rules blocking

them. These protocols include TCP (Postel, 1981c), UDP (Postel, 1980) and ICMP (Postel, 1981b). Tracert allows for the use of a crafted IP packet as an alternative method of route resolution (Malkin, 1993).

These programs work through examination of the packets that are returned to the host that emitted them. This is achieved through sending packets at the target IP, starting with a TTL of one, and then incrementing this value for each packet sent up to the maximum hop number specified. The first packet sent would expire after the first hop; the node that the packet expired at would then create a ICMP packet signalling that the TTL expired on the received packet. This new ICMP packet would be sent back to the source IP of the packet that expired. This new ICMP packet would contain the source IP of the host in which the original packet expired. This packet would be retrieved by traceroute on the host that created the original packet, which would now retrieve the source IP from the ICMP packet that was created by the node in which the original packet expired. At this point the IP of the first hop in the route to the target IP is now known. From this point the packet with the TTL of two will expire at the second hop and this process will be repeated until the destination is either found, or the maximum hop limit is reached (Cisco, 2005).

Traceroute is destined at an ephemeral port number that is most likely closed. This means that when the traceroute generated packet reaches the destination, the port it is destined to on the target host would most likely be closed. This would then cause the target host to notify the source host that the port is closed by creating an ICMP packet to denote this fact. This packet, being different to the other ICMP packets that notify that the TTL of the original packet has expired, would signal that the host that has been encountered is in fact the destination host. At this point the trace is complete and so traceroute terminates and returns the results (Cisco, 2005).

A route can be resolved using traceroute in the following format: traceroute [options] “target IP”. These options include setting the number of hops to resolve to and protocol to use, and whether or not to resolve the host name. These options can be found on the traceroute man page provided by Linux (Jacobsen, 2001). An example of a resolved route can be seen in Figure 2.10. The options applied to this run of traceroute are, “-I” to signal the use of the ICMP protocol, and “-m 5” to set the limit of hop resolution to five hops maximum. It is notable that the default protocol used by traceroute for tracing a route is UDP and that the default maximum hop count is set to thirty hops.

CAIDA is an organisation that specializes in the collection of network topology data and the analysis of network data and packets (CAIDA, 2014). These analyses includes IP ge-

```
example@ubuntu:~\$ sudo traceroute -I -m 5 www.google.co.za
traceroute to www.google.co.za (74.125.233.31), 5 hops max, 60 byte packets
 1  ict.gw.ru.ac.za (146.231.120.1)  1.420 ms  1.496 ms  1.497 ms
 2  maincampus-1.net.ru.ac.za (146.231.2.25)  2.283 ms  2.290 ms  2.345 ms
 3  strubencore.net.ru.ac.za (146.231.2.10)  0.880 ms  0.889 ms  1.363 ms
 4  border-struben.net.ru.ac.za (146.231.0.2)  0.226 ms  0.237 ms  0.238 ms
 5  tenet.net.ru.ac.za (192.42.99.1)  0.753 ms  0.908 ms  1.026 ms
```

Figure 2.10: Use of traceroute with ICMP and Hop Count Set to 5

ographic locations, packet type analysis from regions and distribution of IPv4 and IPv6 hosts. CAIDA also provides tools that allow for one's personal analysis of networking data. These tools comprise of five main categories of tool: geographic, library, performance, plotting and data curation, and topology tools.

In terms of the data that CAIDA collects, the range is extensive. The data collected ranges from simple anonymous traces throughout the Internet (Hyun and Broido, 2003), to collection and analysis of malicious attacks (Zdrnja *et al.*, 2007). This data can be used to fuel research in areas of defence, or to build up a topology so as to simulate an attack environment.

2.9 Network to Real Life Mapping

The execution of malware that replicates itself, through the process of copying itself into other data[21] located within a computers storage devices, is known as a computer virus (Aycock, 2006). These computer viruses often have adverse effects upon the general operation of a host. These effects can include excessive use of storage space, use of CPU time, accessing private information and even data corruption. These computer viruses can even lead to the computer becoming completely defunct.

A virus in the biological sense is an infectious agent that replicates within the living cells of the host organism; these organisms include all types of life forms (Koonin *et al.*, 2006). The effects of these viruses can range from discomfort (caused by contracting a virus like the common cold (Heikkinen and Järvinen, 2003)), to death (caused by a more serious virus like Influenza C (Yuanji *et al.*, 1983) or Hantavirus (Martinez *et al.*, 2005)). Such effects would depend on the acquired virus and the administered treatment. The

[21] This being both applications, data files and the boot sector.

described interaction of biological viruses with organic living organisms can be compared to the interactions of the technological environment.

This idea can be extended to include interactions between hosts, both biological and technological. If one were to observe biological interaction, one would notice that there is often contact, prolonged times of communication and time spent within the infected host's environment. If an uninfected host was to interact with multiple infected hosts, the chance of infection would increase within the uninfected host.

This is also common in the computing environment. Communication and interaction with other hosts within the computing environment is based upon networks[22] and data transfer via flash and optical media. The more hosts a computer host comes in contact with, the greater the chance of an infected host, service or application infecting that host system.

With these comparisons one can conclude that the simulation of biological viruses in a computing environment is possible. One has to question the adaptation and evolution of biological viruses. A biological virus adapts to an environment and gains resistance to medicines that seek to eradicate it (Emerman and Malik, 2010). In the case of a computer virus, this adaptation and evolution can be compared to to be the development of programs that exploit new flaws in existing programs. In this way, computer viruses also evolve, mimicking their biological counterpart (Zhang *et al.*, 2008).

This similarity in characteristics between computer and biological viruses allows one to describe the characteristics and behaviour of the two types of virus in the same terms. If a virus in either type can be found or developed that mimics the other, the results produced from one can be used to determine an outcome for the other. This is useful when approaching a computer simulation of a biological virus, as it is easier for one to develop a computer-based virus than a biological one. This allows for simulation of biological viruses, which is an opportunity for better planning and thus better reactions to biological circumstances that may be foreseen through simulation (Zhang *et al.*, 2008).

2.10 Traffic Generation

Traffic generation refers to a system or process that generates packets. These generated packets can be a replay of real traffic, random or built from a template. Common practice

[22]The Internet and all Internet based services being the major inclusion.

in replaying of traffic is to use a standard pcap (Garcia, 2014) file to define the data and timestamp of each packet. This allows one to replay the recorded session as if the traffic was being produced in real time. Applications like tcpreplay tool (Turner and Bing, 2012) allow for one to replay these pcap files and modify fields of individual packets within the pcap file before replay.

Random packet generation has little purpose and serves to do little more than use up network bandwidth. These packets can be made more useful through application of a template. A template allows for more useful directed traffic through spoofing of known source IPs and known destination IPs (Sommers and Barford, 2004). This alone allows for traffic to be routed to a destination; however this is usually not enough to warrant a response from the owner of the destined IP. In order to receive a response, one usually requires a known open port and knowledge of a payload that is meaningful to the destined host. This is usually application specific and is subject to the test being performed. Ostinato (Ostinato Open Source Project Group, 2013) is an application which serves the above purpose. It allows for generation of traffic either randomly, or through a series of patterns and templates.

Using traffic generated in a specific way and targeted at a known system or host in a controlled environment can help to develop DoS mitigation systems. A network based DoS attack is one in which excess meaningful network traffic is purposefully directed at a host or system. This consumes the responding system's resources, as it now allocates resources for the connection, resulting in the service it provides becoming less available to legitimate clients. Using a traffic generator, one can develop mechanisms to better handle these DoS attacks (Douligeris and Mitrokotsa, 2004).

Traffic generation is particularly useful as it can feed network packets into a live network as if it were many source hosts. This is observed in Figure 2.11, where one can observe packets generated by the packet generator destined to different hosts. Depending on the source IP of the packet and other fields, such as ports and what is in the payload, the destined hosts may reply to the source IP of the generated packet. This packet would most likely be dropped from the network at some point unless the source IP of the generated packet belongs to an existing host within the network, or a system is in place to manage these packet responses.

This makes traffic generation tools particularly useful for system and network testing. One can effectively generate meaningful traffic for a system to use and respond to, or generate packets for a network infrastructure in order to test the stability of the network.

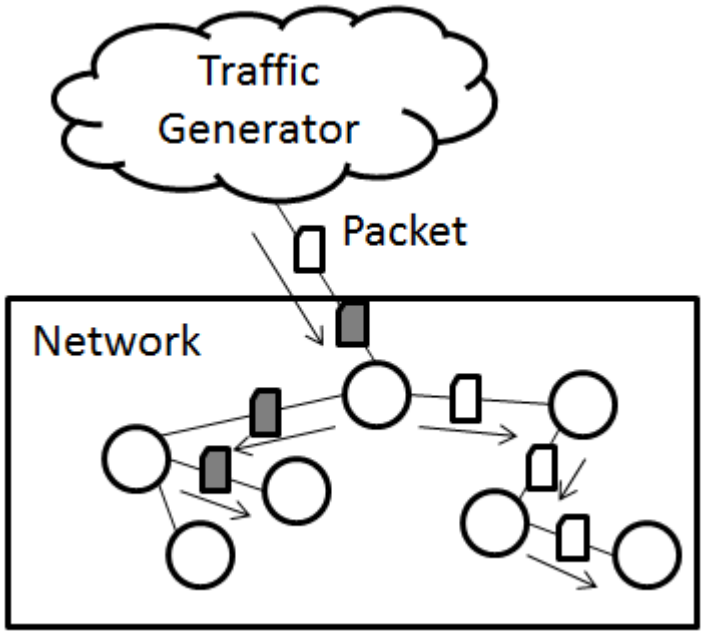


Figure 2.11: External Traffic Generator Injecting Traffic into Live Network

2.11 Honeypots - Service Level Emulation

A honey pot is best described as a trap set up to catch third-party systems or people who aim to break into one's system (Spitzner, 2003, Pages 68-70). A honey pot mimics valuable data, a host or a network that represents a form of resource that an attacker would find valuable. Once an attacker has been detected and has been given access to the honey pot, instead of the system or resource which the honey pot aims to defend, the attacker's actions can then be monitored by this security system. Any further actions taken from this point are at the discretion of the system's owner.

Honey pots can be classified into two main types. These are production- and research-focused honey pots. A production-classified honey pot is directed at companies and organizations that wish to bolster the security of their infrastructure and products, but are not interested in logging data for analysis of what an attacker may be after, or where the attack originated. Research honey pots, on the other hand, are more focused on this data that is not logged by the production class of honey pot. There are further classifications that can be applied to these research-based honey pots, but they are out of the scope of this research. The main difference between a research class honey pot and a production-based one is that in the former, the data and actions of an attacker are logged and analysed in order to create better and more focused defensive mechanisms (Spitzner,

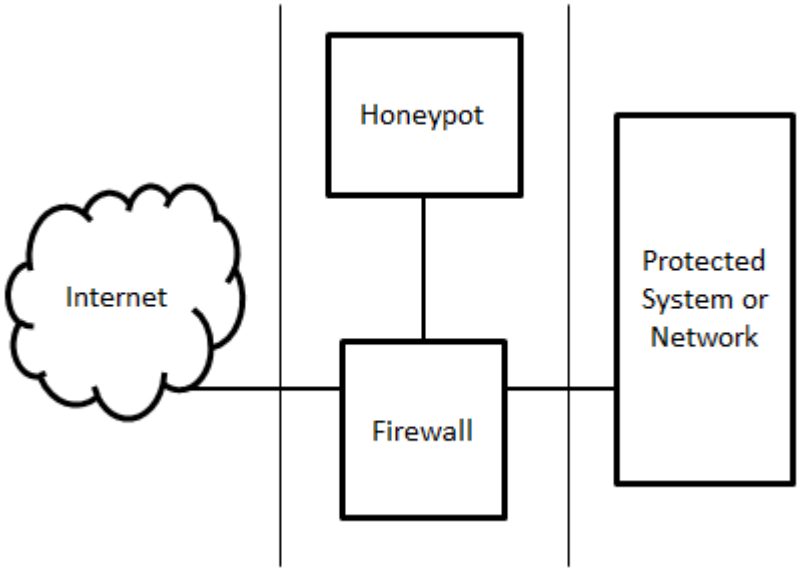


Figure 2.12: Example Protected Network

2002).

Figure 2.12 shows a basic setup of a honey pot in a network which is connected to the firewall. Honey pots can also be found within the protected system or network, as well as before the firewall is even reached. In this example, the firewall that accepts connections into the protected system also has the ability to detect known attacks on the system. If an attack is detected, the attacker is then given access to the honey pot. At this point the honey pot simulates the data and systems that an attacker may want to gain access to. The attacker then believes he has gained access through the firewall and into the protected system, but in reality he is being monitored in a controlled, simulated environment.

After the attacker has left the honey pot, in what the attacker assumes to be a successful attack, the honey pot then resets itself and waits for the next attacker. This in itself is a defensive mechanism as if a repeated attack is performed, the credentials used to access the original honey pot may be denied by the new honey pot, thus deterring another attack carried out under the assumption that the system owner knows of the intrusion.

Some existing honey pot implementations and the emulation each provides are listed below:

- Honeyd[23] - Common Host Services

[23] http://www.honeyd.org/

- Nepenthes[24] - Common Server Side Application
- Kippo[25] - SSH

2.12 Summary

The logical flow of this chapter followed in the outline provided by the goals set in Section 1.2. This chapter began by through defining the purpose for which simulation and emulation were created and then brought forth the idea of what is required to route a packet and why one would want to route a packet. Simulation and emulation of packet routing and previous steps that other research in the field had taken were then tied directly into the question of the possibilities of simulating the routing of a packet.

Other ideas and applications of routing simulation and emulation were pointed out and an understanding of the underlying mechanisms of these implementations was drawn. From this point, this research can now take the knowledge brought to light in Chapter 2 and look towards design and implementation in Chapter 3. Chapter 3 will lay the groundwork onto which this research's implementation of a routing simulator will be built by defining first what is required of a routing simulator and then what the host platform will be. From this grounding, the manner in which each functionality will be designed and the form they will take will be discussed and selected.

[24] https://www.shadowserver.org/wiki/pmwiki.php/Involve/BuildAHoneypot
[25] http://code.google.com/p/kippo/

3

System Design

This chapter presents a theoretical approach to the design of the implemented system. The aspect of feature objectives and resource management is considered first in Sections 3.1 and 3.2, as the need to define and understand the interactions of the underlying structure of this implementation is important. Without a concrete foundation and clear goal, the errors created through misdirection would propagate into features built on top of this foundation. Each memory structure this implementation can use to represent its network structure is detailed next in Section 3.3.

The following text in Section 3.4 brings up the manner in which the routing process takes is handled and the OS is chosen after this in Section 3.5. Section 3.6 and 3.7 address details around the overheads context switches cause in a user space application and how one can make use of threads within this implementation. Next, Section 3.8, this research fleshes out the details of how interfacing to a physical network is performed and how one would go about configuring this implementation is explained in Section 3.9. Finally, Section 3.11 recaps this chapter by listing the final design choices made for this routing simulator's implementation.

Each element considered takes into account the many solutions that could perform the task at hand. For this reason the general form of this text is that each solution is detailed

and discussed before a decision is made as to the best solution for this implementation. The selection for the decision is also discussed so as to keep the reader in line with the authors thought process.

3.1 Functional Requirements

The process of network routing, which is responsible for the way in which network packets are transferred from host to host in the Internet (Savage *et al.*, 1999), has some specific hardware to enable this functionality. This hardware ranges from routers, switches and endpoint hosts, to the medium which carries the bits of one's packet. The hardware involved in routing, as well as the mediums used, will be focussed on to simulate the general functionality of network routing.

Packet routing, as opposed to circuit switching (Marques *et al.*, 2006), is the main focus of this implementation; this is performed on a node to node basis. Throughputs of greater than 7 Gbps were achieved through this design (see Section 5.3) and the means by which this performance was obtained is further detailed in Section 3.4. There are some behaviours that arise in packet routing that occur naturally due to errors in links and within the processing logic of nodes (Bolot and Jean-Chrysotome, 1993). These behaviours, which most commonly occur in large scale networks, are:

- Delay - The time between packet emission from source host and arrival at destination host.
- Jitter - A common occurrence within wireless networks leading to packet retries, and thus added delay.
- Packet Loss - The event of an unrecoverable failure in a packet's transmission.
- Packet Mangling - The event of a packet's data on arrival not representing the data it was emitted with.

Some further functional additions for this research's implementation is introduced to allow for offloading workloads as well as network monitoring. These functions are:

- Worm Hole Routing - Hop aggregation to allow process offloading for improved performance.
- Packet Monitoring - A mechanism to view packet details at simulation time.

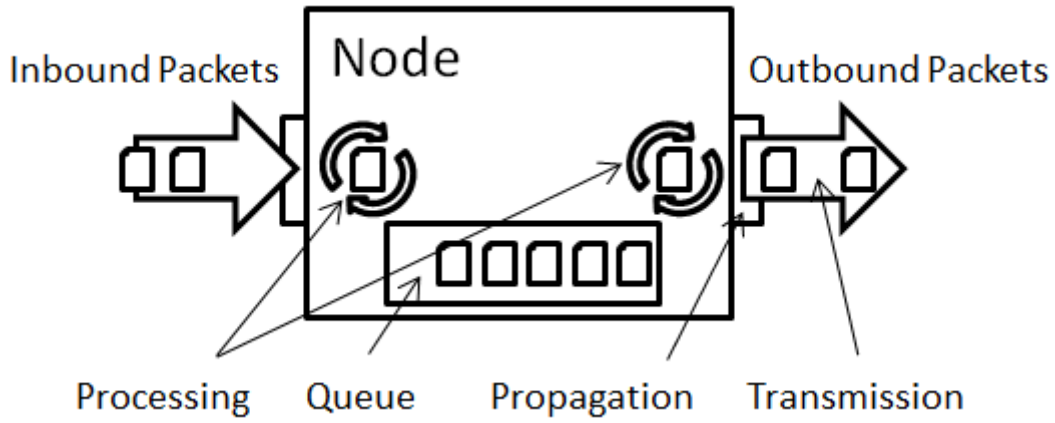


Figure 3.1: Delays in Typical Node

3.1.1 Network and Propagation Delay

Figure 3.1 describes delay as a combination of four factors. These factors are process, propagation, queue and transmission delays (Jim Kurose, 2010). To better describe each, processing delay is the delay taken to process a packet in node, be it host or router. Propagation delay is the time it takes for a node to put a packet onto the wire; this should not be confused with transmission delay which is the time taken for a packet to travel across the wire from source to destination. Last is queue delay; this is the time taken for a packet to reach the head of the internal queue for emission onto the physical network medium within a node.

Delay causes a natural physical restraint on large networks and is a point that was taken into consideration in the design of this implementation. The reason for this is that delay has potential adverse effects on continuous connections, such as file transfers or streaming (Zheng *et al.*, 2001), as protocols now should know to adapt to longer connections, or periods of no response while a packet is in transit.

3.1.2 Packet Loss

Packet loss is the non-arrival of packets within a network. Causes for such loss can be medium signal fade, network congestion or TTL expiry (Jim Kurose, 2010). The fact that this is a common occurrence in current-day networks is reason enough to include packet loss as a key feature within this network simulator. Such packet loss can lead to failed file transfers, disconnection in streaming sessions, or failures in other connections that

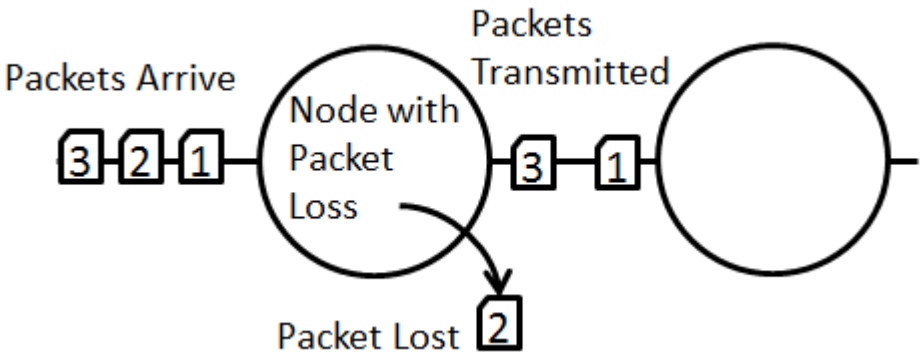


Figure 3.2: Packet Lost in Transit

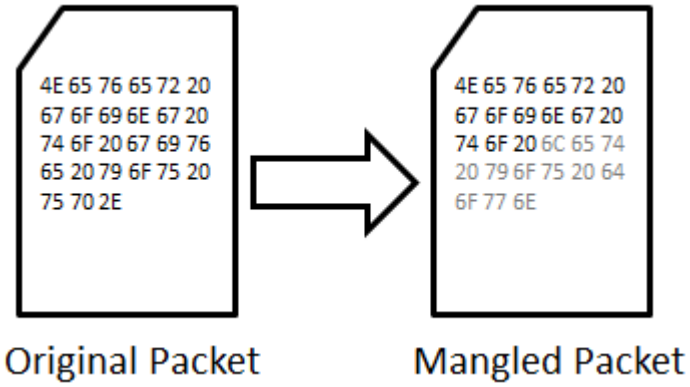


Figure 3.3: Example of a Packet Being Mangled

require a lossless data transfer protocol. A visual example of packet loss is given by Figure 3.2. Protocols that help to mitigate this problem, such as TCP, are discussed in Section 3.4.3.

3.1.3 Packet Mangling

Like packet loss, packet mangling is another behaviour of large scale networks, although less common, that can lead to sub-optimum connection link states. In short, a mangled packet is one that has lost context due to the fact that parts of the packet have been modified either prior to, or during transit (Bautts *et al.*, 2005). Figure 3.3 depicts this through an example of the original packet on the left being modified into a mangled state on the right.

In most modern routers, the common behavioural approach to receiving a mangled packet is simply to drop it, as it has lost context (Jim Kurose, 2010); this is typically checked via the packet's checksum. This means that the handling of a mangled packet in this manner would lead to the same result as a lost packet. Detection of this packet loss is usually through observation of the packet's original checksum not equalling what the now modified packet's checksum should be (Wireshark Foundation, 2013).

There is also intentional packet mangling where a packet is specifically formed or modified in such a way as to disrupt expected functionality of systems. This is achieved through creating packets that look like they are from different sources to achieve anonymity or resource allocation. These are usually created with malicious intent and classify under denial of service attacks (McDowell, 2009). These packets are usually targeted at a service or network to use all resources that the service or network provides. With no more resources left for allocation, the service or network can no longer provide a optimal experience for legitimate users; this gives rise to the idea of attack.

As packet mangling, in the sense of packet corruption, does occur in real world physical application, it was also identified as a feature that should be included in this implementation. This will allow for testing of rigidity of protocols and applications set to be tested on this routing simulator in future application.

3.1.4 Jitter

Also known as packet delay variation (Demichelis and Chimento, 2002), jitter in networking terms is the variation in latency between two nodes within a network (Comer, 2008). This is most prominent in wireless connections as packets transmitted over a wireless network have a greater chance of needing to be retransmitted than in a wired connection due to signal strength entropy.

Jitter can lead to problems regarding quality of service as a wide variation of packet arrival can lead to unpredictable transfer speeds, as well as bursts in transmission that could overload specific links or nodes (Demichelis and Chimento, 2002). This can also lead to disruption in streaming of live communications on applications such as Skype[1], Youtube, audio streams, video streams or the combination of each found on video on demand sites.

[1] http://www.skype.com/en/

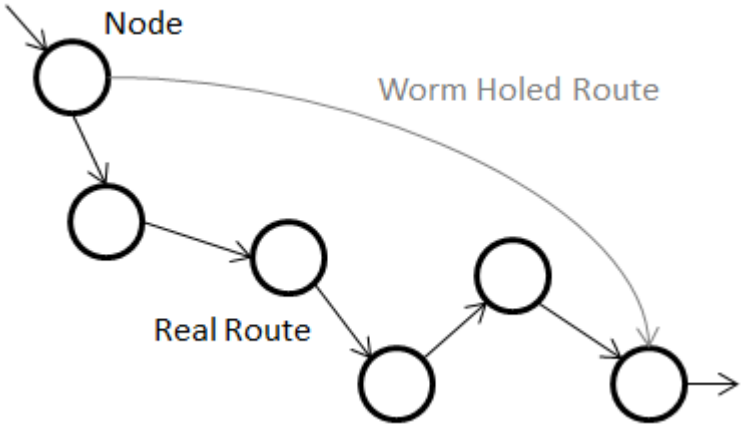


Figure 3.4: Example of Worm Holing in a Route

As wireless links do exist in real world application, the inclusion of such a feature within this implementation was considered important. If a new protocol were to be developed, one should have the ability to test it over a simulated wireless link to ensure robustness.

A more subtle function that this feature provides is out of order packet routing. This occurs because of the delay of a hop no longer being constant, but rather varying due to the retransmissions caused by this jitter. This can cause packets transmitted subsequently to a packet suffering from this jitter to arrive before the initial packet.

3.1.5 Worm Hole Routing

Worm hole routing is the ability for a packet in simulation to compress a series of hops into a single hop. This takes into account the sum of all delays of all hops to the destined node. The simulation proceeds to process the transmission from the source to arrival at the worm hole destination as a single hop with a delay of the sum of all hops in the route.

In implementation, if criteria are matched while routing a packet, the packet is then scheduled once to hop directly to the node to which it is destined with the preconfigured sum of delays as the delay incurred on the transmission delay of the hop. Figure 3.4 depicts this logic. This allows for lower CPU usage as a single schedule is made to the work queue and this removes the need to manually process all the hops in between.

This ability to source packets from a node and sink them at a node regardless of hops in between allows for the ability to inject packets at a node in the simulated network.

This can be achieved by sourcing incoming traffic from a host generator and sinking the traffic at a node within the network through the use of this worm holing function. This will allow more time for the processing of other routes within the simulation.

3.1.6 Packet Monitoring

This is simply the ability to capture packets from a node within the simulated network. This is done by transmitting packets that meet set criteria out of a pseudo or real interface. This will then allow for existing applications, such as wireshark[2] or tcpdump[3], to capture on these interfaces and log any data that is deemed important or significant. This also allows for opening up a pcap interface for injection and sniffing of packets of the node that the interface is bound to, and allows integration with existing tools.

Other more customizable applications can be created through the use of Libpcap in Python, C, C++ or any other programming language that supports this library. One can go further and write Linux kernel modules using the TUN/TAP framework[4]. TUN/TAP allows for high speed layer 2 and layer 3 interface handles within the Linux kernel. As these access a network interface in a standard manner, the speed of these handles is still measured by standard network conventions, this being in Mbps.

The speed at and accuracy with which data is managed through these handles can only be determined by the capability of the hardware within the host that is requesting use of these handles. One should also take into account other processes requesting use of the resources that TUN/TAP requires at time of access, as this can add extra overheads to the service.

3.2 Packet Handling

This section explains what packet handling means in the context of this implementation, why it is required and how it is done. Simply put, packet handling refers to the method in which packets are received and transmitted in this implementation.

As this is a routing simulator, the actual nodes that generate traffic are not found within the simulation. This is because the focus of this implementation is on routing the packets

[2] http://www.wireshark.org/
[3] http://www.tcpdump.org/
[4] https://www.kernel.org/doc/Documentation/networking/tuntap.txt

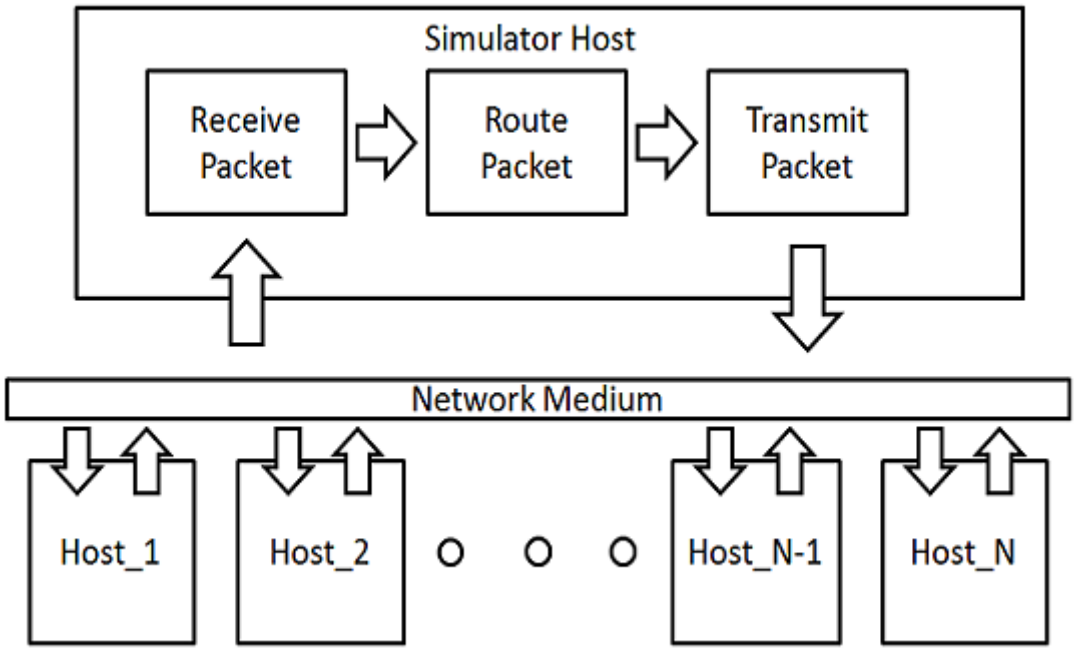


Figure 3.5: Overview of Packet Handling

introduced into the simulation, and not on creating these packets. Internal packet generation would create a need to use up host resources in order to instantiate new network stacks on VMs, or use up CPU time in the event of packet generation specific applications. Instead packets are brought in from external sources. In order for this to work, one needs to provide sufficient handles to both receive and transmit packets. This is explained in more detail in Section 3.2.1 and Section 3.2.2 respectively.

The simulated nodes and physical hosts undergo a binding process. This is used to direct where traffic from external nodes is placed when received into the simulated network, and where it is sent when a destination in the simulation is reached. This process takes a real IP and MAC address and couples it with a simulated node's IP and metadata. This then allows for a simple interface between the routing simulator and the physical hosts used in this simulation.

3.2.1 Application of Netfilter and Low Level Sockets

Netfilter is a hook-based implementation within the Linux kernel that allows for external function call backs to occur when a packet traversing the network stack reaches the function's hook in the network stack (Ayuso, 2010). In practice, one is required to create

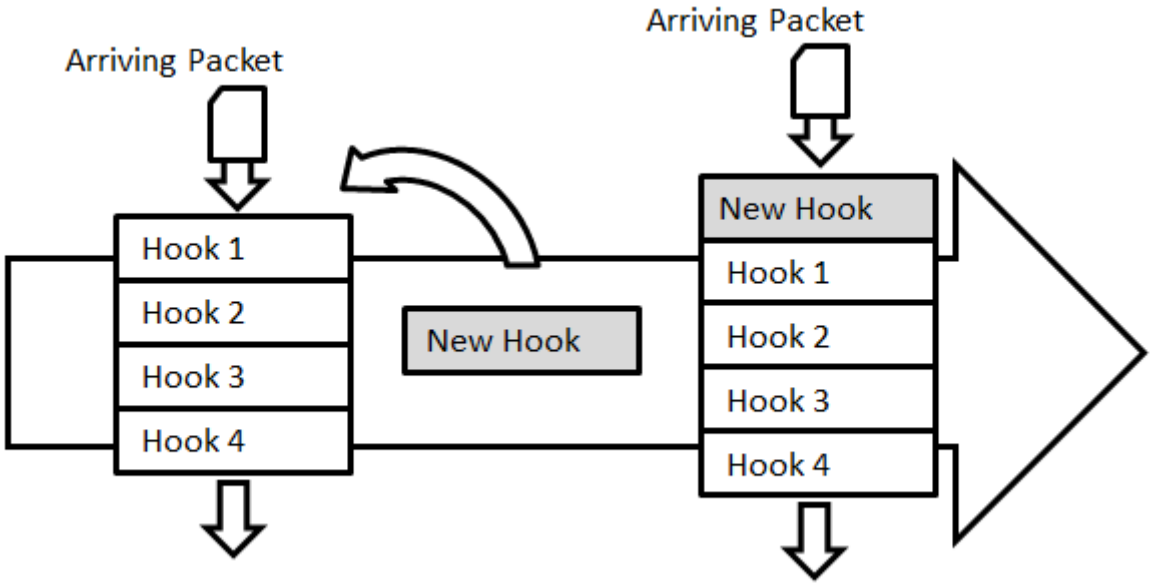


Figure 3.6: Adding a Hook into Netfilter

the netfilter hook, create a pointer to the function, and then set a priority as to where the hook will be placed within the hook stack. Figure 3.6 shows the new hook function being placed in first priority in the function stack. This means that one can set it to be the first hook in the stack and so reduce delays of idling while waiting for every other hook function to complete, a simple performance booster.

All interfaces are watched by netfilter on the host in which it resides. The ability to determine the interface in which a packet was received is provided through the metadata within the struct that the packet is buffered in after it is received on a interface. There are also other useful fields such as packet length, protocol and a whole set of pointers to each header in the packet residing in the buffer (Santa Clara University, 2004). The determining part of the buffer is the interface in which the packet was received. This can be used to separate out multiple networks or simply increase the host's bandwidth through use of multiple interfaces.

A drawback of netfilter is that it works at layer 3 of the IPv4 stack (Ayuso, 2010). This means that the MAC address at which a packet is received is not stored in the packet buffer as a MAC, and protocol type is layer 2 (Postel, 1981a). This means that when binding between a physical host and simulated node is created, the MAC address for the physical host has to be stored as one cannot obtain this from the packet received through netfilter. This causes a need for knowledge of specifics about a physical host and thus adds to the time taken for set up.

3.2.2 Transmission of Packets

Transmitted packets within this implementation follow a far simpler method than that of packet handling does not require any of the hooks necessary in netfilter. Instead one creates a packet buffer with all required fields, or uses an existing buffer such as one obtained through netfilter. If one were to use a netfilter packet buffer, one must remember to attach a layer 2 header that is not obtained through receiving a packet with netfilter, as mentioned in Section 3.2.1.

The `dev_queue_xmit()` function is used to send out the packet buffer, byte for byte, through a raw socket that is handled by the kernel. The interface out of which it is sent is equivalent to the one found in the interface field of the packet buffer. This is as simple as packet transmission gets and allows the user to hand craft packets as one pleases.

3.3 Node Memory Structure and Access

In many applications, the manner in which data is represented can have a drastic toll on performance if implemented or designed incorrectly (Kopp, 2011). As this is a time constrained implementation, some design considerations should be given with regards to how nodes and packets are accessed in memory and how long functions take to execute. Known enumerable data structure implementations are detailed in this section. Following an overview of of each data structure, which is detailed in Subsections 3.3.1 to 3.3.5, a final conclusion on the best suited data structure for this routing simulator is reached in Subsection 3.3.6.

3.3.1 Arrays

Arrays allow for indexed lookups with a one-to-one association of index to data. This means that an array has a time complexity of *O(1)*. This is further improved by the fact that arrays have an *O(1)* worst case access time (Black, 2004). An array is simply an allocated memory block of a set data type into which one indexes using the index provided and the data type's byte length to generate the offset in which the indexed data resides.

This does come with some drawbacks. The entirety of the search space of an array has to be allocated at the instantiation time of the array; this has a severe impact in the memory

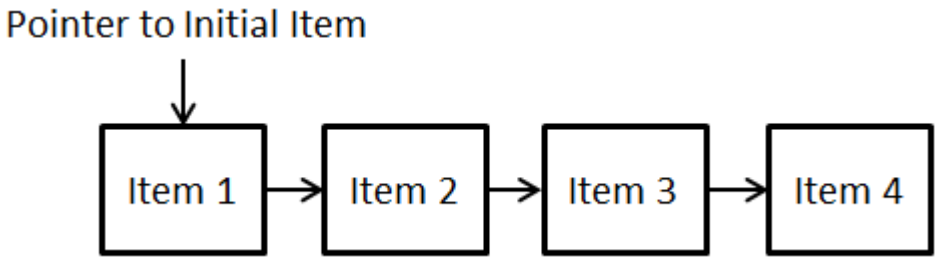


Figure 3.7: A Simple Linked List

requirements of this implementation. To further stress this point one should consider the size of IPv4, that being a 32-bit address space (Postel, 1981a). This means that one has to allocate more than four billion memory address at creation of this memory structure. In application, keeping to 32-bit integers to adhere to IPv4's address size, one would require 16 GB of memory just to address each IP in the simulated network.

However, one is unlikely to use the entirety of IPv4 addressable space in testing. The reason for this simply comes down to the fact that not every IP in IPv4 has been used yet, although many \8 blocks[5] have been allocated. (Huston, 2014). It is also notable that even if a \8 block has been allocated it does not mean that all IPs have been allocated within the \8 block, as these blocks are pre-emptively assigned rather than assigned on demand. This means that a real world application of IPv4 would not saturate the entirety of IPv4's address space.

3.3.2 Linked List

Linked lists represent a simple chain structure, each item in it is a link in the chain. This data structure is primarily used to form queue and deque structures within memory as it has properties that make it inherently simple to do so. This is because the simple linked list only keeps track of what is at the head of the memory chain (as shown in Figure 3.7). There are, however, types of linked lists that keep track of the tail or indexes into the chain to speed up search time, or to allow for further functionality such as use of the chain as a deque (Knuth, 1997, Page 294).

However, this data structure is not viable for use in fast lookups as it is inherently slow and has lookup time of *O(n)*. This is because it only keeps the location of the head of the linked list, and in some implementations the tail. A lookup starts at the either side of

[5]Where the number after the slash represents the number of leading set bits in the prefix mask.

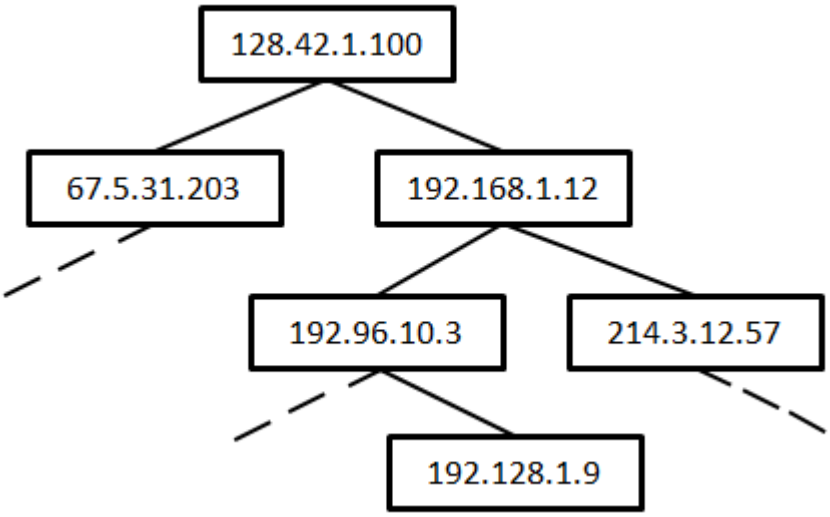


Figure 3.8: A Simple Binary Tree Using IPv4 Addressing as a Key

the linked list and progresses through item by item until the item is found. This can be sped up through indexing which allows one to start searching at a pre-indexed point in the list, rather than at the beginning or end (Knuth, 1997, Page 298).

There is not much practical use for this data structure in the context of this routing simulator as other than its cost effective memory usage, it has no advantages over other, faster data structure implementations that are available for use.

3.3.3 Binary Trees

A binary tree is a data structure that ensures that each node has one, two or no branches leading to other nodes in the data structure (Louck, 2008). In this form the binary tree implementation has a best case time complexity of *O(log(n))* and a worst case of *O(n)*. However the worst case is mitigated through further refinement of this data structure's implementation. If one were to work with the entire IPv4 address space, this data structure's time complexity would require 22 accesses on average to get to the requested item in the data structure.

To understand the worst case scenario, one must consider the traversal of the whole binary tree in order to lookup or insert data into it. This happens when every node in the tree has a single branch that results in an elongated linear structure. In this form the binary tree represents a glorified link list (Ford *et al.*, 2002).

Methods to prevent this from happening are implemented in specialized binary trees, namely: full, perfect, complete, infinite complete and balanced binary trees (Zou and Black, 2008). These use methods from ensuring that each node has two branches or none, to ensuring that branches descend to a common depth count (Black, 2008).

In the event that the worst case does arise, the access count will be significantly larger from the average case when compared to the advantage of the best case. To further drive this point, there is only ever one best case, where one can have multiple sub-optimum cases. This is unfortunately one of the characteristics that arises when dealing with any tree based structure.

3.3.4 Hash Tables

A hash table is a data structure that uses a calculated hash as an index into an array in order to locate the requested information. This calculated hash is produced through use of a suitable hash function that is designed to lower the chance of collisions within the hash table[6]. This data structure is particularly useful in quick insertion and retrieval of data and all data stored takes up a constant space in memory (Leiserson *et al.*, 2001).

In theory hash tables yield a *O(1)* best case, and *O(n)* worst case time complexity. This is due to the nature of the hashing function combined with the number of indices available in the data structure. If there are too few indexes, or the hashing function does not ensure a unique index for each item in the data structure, the time complexity of the algorithm increases.

With this in mind we look towards the time complexity for this data structure in practice. In most implementations the hashing function used evenly distributes the indexes of each inserted item into an unused location. This means that the only real limitation is the number of unique indexes available. Because of this, in practice it has been found that the average time complexity of a hash table is in fact $O(\frac{1}{1-\frac{N}{K}}))$, where N is the number of items to be inserted and K is the number of unique indexes available (Dunham, 2009). Furthermore, one can note that it is in fact impossible to achieve a time complexity of *O(1)* in practice by this equation.

When considering this implementation in the context of the IPv4 address space, to ensure an acceptable lookup time one has to take into account the time taken for index retrieval.

[6] A collision being two items that evaluate to the same value when the hashing function is applied to them.

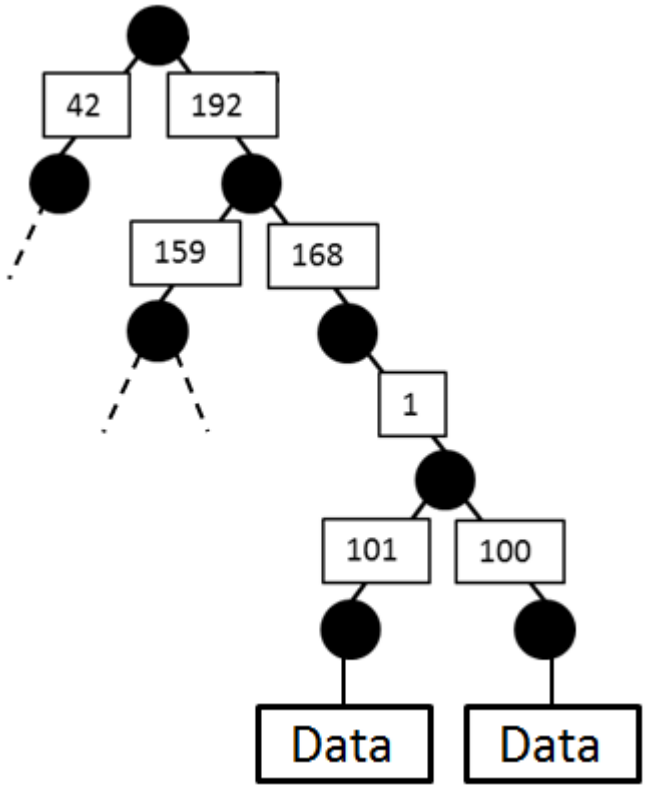


Figure 3.9: Example of IP Addresses in Radix Trie

This means a reduction in the clashes the hashing function produces, which means one needs a larger number of unique search spaces in order to achieve the fast lookup that hash tables try to ensure. This means that the actual memory required for an item in this data structure is 1 to $\frac{K}{N}$ to ensure the fast lookup speed this algorithm provides.

Within a small dataset this is not a large overhead, especially in modern computing, but to provide this kind of overhead in a dataset of one to two million entries makes it difficult to justify. This is re-enforced by the limitations on memory to begin with as this implementation is aimed at commodity hardware as discussed in Section 1.2.

3.3.5 Radix Tries

A radix trie, or binary patricia tree (Edelkamp, 2010), is a tree-representing data structure that bases its search on a prefix-based algorithm. This algorithm is designed to condense trees into more granular searches based on a prefix of the inserted key. A search within this tree structure becomes more refined as more keys of a similar nature

are added into the data structure. This, however, means that more nodes are added to ensure that the refined keys are placed within this structure.

In application, this functionality is useful when considering the structure of an IP address. It is simply 4 bytes which is equivalent to a **int** type[7] in languages such as C and C++. With this structure, using an IP as a key, one can apply a prefix based search to the data inserted into this data structure as shown in Figure 3.9. If one were to use each byte of an IP address as a prefix, one could effectively build up a tree structure that in worst case would prefix on each bit of the IP address resulting in a requirement of 32 accesses to access the required data (Knizhnik, 2008).

Unlike the binary tree, the radix trie implementation's memory overhead requirements are only one-to-one in the case that the new entry cannot be prefixed by the entries already existing the radix trie. This means that for every item inserted into this data structure, the amount of extra memory required to ensure functionality is most likely less than the memory required to store the entry by itself due to the key being prefixed.

In practice a radix trie shows an *O(n)* complexity. Comparing this to a standard binary tree would show little merit due to a binary tree being *O(log(n))* with a worst case of *O(n)*. The radix trie's time complexity is due to the nature of how the prefix-based search function looks up data. As it is a prefix-based search, every item in the key has to be considered, hence the n look ups for a key of length n. This is mitigated by the fact that, due to its prefixed nature, a radix trie is smaller than a binary tree and there are therefore fewer hops to be traversed. Moreover, when considering the data in question, that being a 32-bit IP address, there is only ever a key of length 4 bytes. This means that any lookup will happen in at most 32 hops from the top of the tree, as the length of any key is 32 bits.

3.3.6 Data Structure Selection

After the initial set up phase of this routing simulator the bulk of memory accesses will be only reads. The only time writes would occur is in the event that the user wishes to change the simulated network's structure during runtime; this is likely to represent a very small proportion of operations. With this in mind, one can largely ignore the insertion time complexity within factors that affect the overall performance of this routing

[7]This is based on a 32-bit architecture, otherwise one could use **uint32_t** or **int32_t** from the types.h library in a kernel context.

Table 3.1: Access Time Complexity vs. Memory Usage of Data Structures

Data Structure	**Best Case**	**Worst Case**	**Memory Usage**
Array	*O(1)*	*O(1)*	1 to $\frac{K}{N}$
Link List	*O(N/2)*	*O(N)*	1 to 1
Binary Tree	*O(log(N))*	*O(N)*	1 to 1
Hash Table	$O(\frac{1}{1-\frac{N}{K}})$	*O(N)*	1 to $\frac{K}{N}$
Radix Trie	*O(N)*	*O(N)*	1 to 1 if item is unique, else less.

where K = Search Space, N = Item Count

simulator during runtime. With this fact in mind, the table represented in Table 3.1 removes insertion time from the time complexity calculations. Instead it looks towards access time, that being time to look up data, and memory usage within the host in which this routing simulator shall be executed.

Considering the contents of Table 3.1, if one were strictly concerned about time, then an array would be the best data structure for this implementation. This is, as mentioned in Section 3.3.1, not viable in terms of memory requirement scalability. Instead this implementation makes use of a radix trie to store the simulated network.

Radix tries, when compared to the other memory structures available, do not weigh up as a fast data structure. Other than having a one to one memory usage, radix tries obtain a best case time complexity that is equivalent to the other data structure's worst case. Further more, in the case of comparing a radix trie to an array, the radix trie's best case is worse than the array's worst case.

Even though radix tries have little method of removal during a power cycle of a host, the advantage of using a radix trie over the other data structures discussed in Sections 3.3.1 to 3.3.4 is the fact that the key for the data we are using to store the data is IP addresses. As mentioned in Section 3.3.5, one only ever requires a maximum of 32 hops to lookup a key and retrieve the associated data. One can go further to say that this means the actual time complexity for a radix trie in this implementation's context is *O(32)*. This simplifies down to order *O(1)*, which is in the same range of fast lookups that arrays provide.

Another advantage is that radix tries are provided as a standard component within the Linux 3.2 kernel (Velikov *et al.*, 2014). This means that it is a well supported data structure within the kernel. With this speed, and also the low overhead in memory use per item inserted into this data structure, a radix trie justifies its own use as the primary data store in this routing simulator.

3.4 Internal Routing

This section provides information on how the routing of packets is handled initially within the routing simulation environment. Routing, as found in real world application, can be defined in its simplest form as a rule-based lookup table. This is achieved through the destination IP being used as a key to define the interface through which a packet should be forwarded (Doyle and Carroll, 2005).

This concept was brought forward into the design of this implementation. Simply, a packet should be received into the routing simulator; it should then be identified as to whether it should be routed or not. If it should be routed, then placement into the simulated network should be defined. From there routing should continue until the packet is dropped due to the packet loss feature, modified (as though even corrupted there is a chance that the packet is still deemed legitimate and thus won't be dropped), or passed to its destination.

3.4.1 Use of Forwarding Tables

Each simulated node contains its own forwarding table. The motivation for this was based on memory usage and stateless routing. The other options available were to preprocess every route and store it in a source-to-destination based table, or to have a global routing table that is accessed each time a node needs to forward a packet.

As one may wish to reconfigure this implementation at runtime, the use of pre-processing every route within the simulated network would prove problematic. This approach would lead to major pauses at runtime whenever a network is reconfigured, due to the program having to reprocess large parts of the simulated network, if not all of the routes in the simulated network, as would be necessary when core nodes in the simulated network are modified.

A global routing table, although algorithmically easier to implement, has its drawbacks in memory usage. In short, for every node added to the routing table, every other node has to be updated as to whether it can or cannot directly route to it, and if so, metadata then has to be added to this. It is also notable that at runtime this method has a higher CPU usage than the preprocessed method, however it saves in memory and real time recalculation is faster.

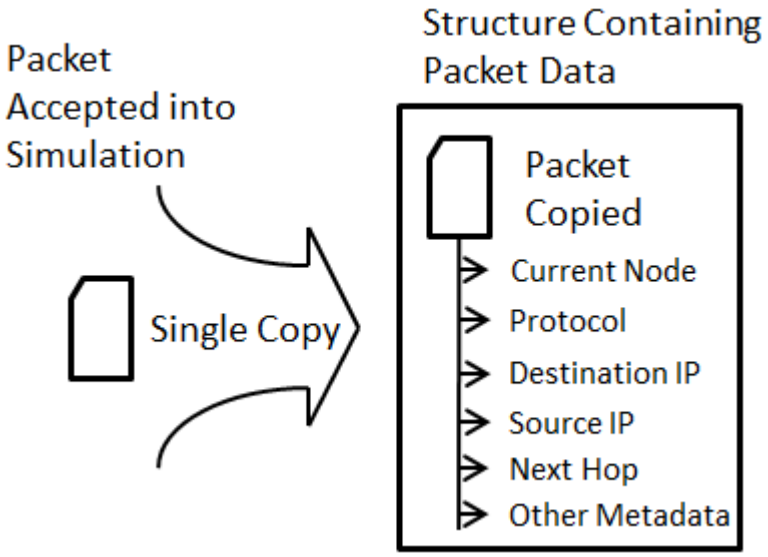


Figure 3.10: Single Copy of Accepted Packet into Structure

The third method, which was ultimately used in this implementation, is a stateless approach where each node has its own forwarding table. This means that no node needs to interact with any other node when an update occurs. The only interaction that happens between the nodes in the simulated network is the forwarding of packets to each.

Although a preprocessed network would be most ideal at runtime, as there would be little to no processing time required to route a packet, the memory footprint at runtime and CPU overhead when it comes to reconfiguration at runtime are far from ideal when compared to a global routing table approach. The third method, in which nodes are seen as stateless entities, can be seen as a refined implementation of the global routing method. Even though this stateless method does require more CPU resources at runtime, it is easily configurable, easy to re-use, and uses the least memory out of the three methods discussed.

As this implementation simulates large scale networks, it is more likely to be memory bound than CPU bound. This is due to the fact that comparing a destination address with a mask to a key in a table is a relatively low memory cost function. With this in mind, the focus in the design of this implementation was to cut down on memory usage so one could simulate as large scale a network as possible in software.

3.4.2 Single Copy Packet Buffer Routing

When a packet is introduced into this simulator via netfilter, as discussed in Section 3.2.1, the way in which the packet is stored in memory is important to determine the efficiency of the routing in this implementation. If one were to make a copy of the packet to each node in the route, this would prove resource heavy. It would also be necessary to include in this the wait times by the CPU for the copy in memory to occur.

Instead, a single data structure is used and when a packet is received into this implementation and a single copy occurs, copying the packet into this data structure as shown in Figure 3.10. Furthermore, all relevant routing information is extracted from the packet at time of initial copy. This allows for a simple pointer dereference to occur in order to get information such as TTL, source IP, destination IP and protocol used.

At time of routing, instead of this data structure, which includes the packet, being copied from node to node in the network, there is simply a field in which to store the current location of the packet in the network. This is then used as a lookup into the node's routing table (in which the packet is now located) to determine the next hop in routing. This means that in routing through this virtual network, there is only ever one copy made of the packet, metadata is extracted and routing occurs by simply updating the packet's location in the routing process within this data structure.

3.4.3 Protocols

This routing simulator was designed to support the routing of the three primary layer 4 protocols over the layer 3 IPv4 protocol, these are: ICMP (Postel, 1981b), TCP (Postel, 1981c), UDP (Postel, 1980). These protocols make up the basis upon which many higher level protocols base their transport[8]. For this reason, this implementation was aimed at these protocols as this would allow for testing to be aimed at well-known applications such as file transfer, web hosting and streaming connections.

One must also realise that the packet manipulation (to allow for external host interfacing) cannot be generalized for all protocols. Both UDP and TCP protocols require the inclusion of an IP pseudo-header; this contains the source and destination IP from the IPv4 header (Postel, 1980, 1981c). This brings up the need to recalculate both UDP

[8]At time of writing, all other layer 4 protocols are ignored and packets using layer 4 protocols other than the fore mentioned are not introduced into the simulated network at runtime.

and TCP checksums, as modifying either the source or destination IP (as happens in a simulation-to-real host translation) will require a recalculation of the UDP and TCP checksums. ICMP does not make use of any protocol in a layer above itself and only needs to know about its own header and the data it is transporting in order to calculate its checksum (Postel, 1981b).

These limitations are due to the manner in which a packet is handled. The packet arrives at the simulation host from a source, be it a physical host or a generated packet. This packet is destined to a node within the simulation. When this packet arrives at its destination after routing, if the destined node is bound to a physical host, then the packet needs to be ejected to that host. This means that the MAC address and destination IP have to be modified to be directed at the physical host. This causes the packet to be modified and so the checksum has to be recalculated within the packet for it to be legitimate. Without this checksum recalculation the packet would be rejected by the bound host, thus stopping any further communications from occurring.

Although large parts of the routing happen at layer 3, all protocols are different in the way that their individual checksums are calculated and in the way they store transport information, so they have to be handled separately. This means that for every new protocol supported by this routing simulator, individual attention to each protocol's behaviour has to be taken into account. This adds limitations to having a general function that handles all protocols in a blanket case. So in order to ensure the correct simulation of each protocol, individual protocol support has to be provided with specific handles for each.

3.5 CPU Architecture and Operating System

The ideas brought to light in this section follow through into Section 3.6. This is because the system architecture and OS go hand in hand with how a programming language is compiled and used in order to achieve its full effect.

In this section we outline the architecture used and OS selected. As this is a kernel level approach to a solution in software, Linux was chosen over the Microsoft's Windows[9], Apple's Mac OSX[10] and BSD[11]. This reasoning is partly due to cost requirements in that Windows and Mac OSX requires the purchase of the license to use the OS, and partly due

[9] http://windows.microsoft.com/en-us/windows/home
[10] http://www.apple.com/mac/
[11] http://www.bsd.org/

to the ease of introducing new modules into the BSD and Linux kernels, as these kernels are open source (Wheeler, 2005) and relatively easy to configure.

Having open source allows for easier access to documentation on the specifics of an OSs kernel and even access to the code from which the kernel is compiled. This allows one to more easily access features of an OS, or modify them to fit one's needs without breaching license agreements.

3.5.1 Architecture Specific Optimization Considerations

As this implementation is compiled for a specific architecture that is well documented, one can start to make use of some of the supported optimizations that compilers allow for, as well as optimization through knowledge of how the architecture works.

GCC allows for multiple levels of code optimization at compile time. This increases the amount of time necessary to compile the program, but can lead to speed ups at runtime of the application. One can achieve this through the use of the '-O#' flag, where '#' is a number that represents the level of optimization one wants. The higher the number, the better the optimization (GNU, 2013). An example of this command in use is shown in Listing 3.1.

Listing 3.1: GCC Compilation Command

```
1 user@generic:~$ gcc -O1 "source_file.c" -o "compiled_file"
```

This would optimize the compiled source file to level 1 of optimization. This level includes processing one's sources to allow auto increment and decrement of integers in loops, rather than explicit instructions, and to allow conversions of signed integers to unsigned ones where integers don't become negative. It is notable that if an optimization level is not supplied, then the optimization level defaults to 0. These aren't the only options though: one can select any specific optimization from the list provided in GNU's GCC compiler documentation (GNU, 2013).

One can also apply one's own knowledge of specifics within an architecture to optimize one's code. Such an application can be that of loop unrolling. This can be shown by the segment of code that follows in Listing 3.2.

In Loop Simple, one can see a standard for loop that simply iterates through an array, adding all the values from it to total. At run time this approach may suffer from a cache

miss when trying to access the data in the array. This means that Loop Simple can do nothing but wait until the miss is handled and the data requested from the array is brought up to the CPU for execution.

Listing 3.2: Loop Simple

```
1 // Standard for loop adding elements of array to total.
2 for(int i ; i < MAX ; i++){
3     total += array[i];
4 }
5 // end for
```

Listing 3.3: Loop Unrolled

```
1 // Loop unrolled for loop allowing for out of order execution.
2 for(int i ; i < MAX ; i+=4){
3     total1 += array[i+0];
4     total2 += array[i+1];
5     total3 += array[i+2];
6     total4 += array[i+3];
7 }
8 total = total1 + total2 + total3 + total4;
9 // end for
```

Listing 3.4: Loop Block Size

```
1 // Loop unrolled for loop allowing for out of order execution
2 // and correct cache fetch sizing.
3 for(int i ; i < (MAX/4) ; i++){ // where (MAX % 4 == 0)
4     total1 += array[i+0];
5     total2 += array[i+128];
6     total3 += array[i+256];
7     total4 += array[i+384];
8 }
9 total = total1 + total2 + total3 + total4;
10 // end for
```

The approach taken in Loop Unrolled (Listing 3.3) mitigates the occurrence of a cache miss, as even if one of the statements causes the aforementioned situation to arise, the other statements can still execute in a CPU that allows for out-of-order execution of instructions. This also reduces the number of comparisons that have to occur in the evaluation of the loop boolean statement.

Table 3.2: Comparison of 32-bit vs. 64-bit Compiled Binary Size

Architecture	Size in Bytes
32-bit	4929
64-bit	6765

Loop Block Size (Listing 3.4) makes yet another optimization in that it now takes into consideration the cache read and write size in the CPU's caches. If one's cache were to write in 128 byte blocks and an array was declared of type char or uint8_t, then we know that 128 indexes is of size 128 bytes in the cache. This means that we can ensure that a multiprocessor CPU can work on the same problem in memory without having to contest for the same block at a CPU cache level. This removes memory overheads in execution.

Further optimization can be achieved through allocation of variables on the stack rather than the heap, reduction of intermediate variables and optimization of frequently accessed code within an implementation by packing it together and rewriting it into an assembly level code to reduce instruction count rather than splitting it across the program.

3.5.2 64-bit vs 32-bit System

Notable difference between 32-bit and 64-bit architectures is the accessible address space made available in the 64-bit systems. This increases the 32-bit accessible memory of around 4 GB to 16 EiB[12]. The 4 GB of addressable memory space provided by a 32-bit OS is not necessarily available for use. This is because the host OS reserves some of this address space for its own kernel addressing (Microsoft, 2011) and device peripherals. This normally amounts to between 15% and 30% of the total memory address space.

The implication of this is that one can no longer store the whole of the 32-bit IPv4 address space in memory. This is still overshadowed by the fact that even if the full 32-bit of address space in memory was available, this would allow for only a single byte of data per node in IPv4 space.

Having 64 bits of address space essentially allows 4 GB of memory for each node in an IPv4 addressed network, as there are 2^{128} bytes of memory available and IPv4 has 2^{32} available addresses. This does not take into account the overhead required by the OS, but

[12] 1 EiB = 1152921504.6068 GB (i.e. 2^{128} bytes)

even when one factors in the worst-case overhead scenario, which would use 30% of the memory, each node would still have 2.867 GB of memory available. This is a significant allocation of addressable memory made available for each node's instantiation in memory.

Listing 3.5: Program 1

```
1  // This is compiled under two different architectures to compare
2  // size.
3  #define MAX 512
4
5  int main(void){
6
7      // declare variables for use in double loops
8      int array [MAX];
9      int i, total, total1, total2, total3, total4;
10     total = total1 = total2 = total3 = total4 = 0;
11
12     // simply populate array in inefficient manner
13     for(i = 0 ; i < MAX ; i++){
14         array[i] = i;
15     }
16
17     // learn from mistakes and do a better job
18     for(i = 0 ; i < (MAX/4) ; i++){
19         total1 += array[i+0];
20         total2 += array[i+128];
21         total3 += array[i+256];
22         total4 += array[i+384];
23     }
24
25     // add totals together
26     total = total1 + total2 + total3 + total4;
27
28     // end it all
29     return 0;
30 }
```

The downside of a 64-bit approach is the program's compilation size. Program 1, found in Listing 3.5, was compiled under gcc-4.7 to allow for compilation for 32-bit and 64-bit architectures. This yielded the results found in Table 3.2 and clearly shows that the

compilation size of a 64-bit application is larger than that of a 32-bit application.

This is due to the fact that address size are now 64-bits in size which increases them to 8 bytes over the conventional 4 byte address found in a 32-bit architecture. This is coupled with the fact that standard types such as integers are now redefined to be larger; this also increases the size of the application when compiled under a Unix based system (Smith, 2004).

A development within the Intel 64-bit architecture is the introduction of more registers for use in the CPU. This means that more variables can be brought up to the CPU for processing than that of the previous 32-bit architecture (Intel Corperation, 2001). This means less time is spent transferring variables from the CPU caches, or lower levels of the memory hierarchy, which wastes numerous clock cycles at runtime. In short, this allows more time for processing this simulation as higher register counts means that the ALU within the CPU can be better utilized.

3.6 System Context Switch

In this section we evaluate the benefits and disadvantages of using a kernel module over a standard user context implementation and a contrast between the standard C libraries available for use in each case. Support for advanced data structures such as hash tables and link lists are higher level concepts and these are omitted from the standard kernel module libraries. There are, however, benefits to kernel mode programming in that schedulers and kernel buffers become more readily available for one's use, thus mitigating the need for context switching. The benefits of this are discussed in Section 3.6.1.

Such libraries include the radix trie implementation, as detailed in Section 3.3.5, and low level calls to the kernel scheduler through work queues, which will be explained in detail in Section 3.7. User mode programs developed in C do not come with many supporting libraries either, however there are a lot of community written libraries that can simply be included into one's own code. The reason this can be done so easily in the user context is that there are no regulations that one has to follow strictly. Instead, the use of such libraries is decided by the one who is coding the implementation.

In a kernel context one cannot simply import libraries from third party providers. This is due to the fact that if the kernel is tainted[13] (Siewiorek and Swarz, 1992), the OS as a

[13]Code that causes kernel to crash

whole crashes, stopping all processes. In the worse case this can corrupt one's file systems and render the OS useless (The UNIX and Linux Forums Content, 2013a). Because of this, only libraries written for the general support and running of the kernel are usable when programming under a kernel context.

3.6.1 Context Switching

User context can be described simply as a sandbox that runs within the kernel context. In order for the user context application to acquire resources outside of its sandbox it has to request these resources from the kernel. This operation requires a context switch in order for the kernel to allocate the resource requested, if the resource is available to allocate.

A context switch is the process of loading out CPU registers used by a current application into memory, which is then this is followed by loading another process's values from memory into the now free CPU registers for execution to occur (The Linux Information Project, 2006). If a buffer is required or a cache miss occurs (mitigation of this is detailed in Section 3.5.1) a context switch occurs which essentially voids the time slice allocated to the current process on the CPU.

This brings up some serious overheads when considering a resource such as a network buffer. In order for a process in a user context to acquire such a resource, it has to make a request to the kernel. This means that a stall occurs on the current user context process requiring a context switch to occur, thus causing the process to go dormant until the request can be serviced.

From this point the kernel context process that can allocate the resources is required to handle the request from the user context process. This means that the kernel process has to be scheduled to execute on the CPU. The kernel process then has to join in at the back of the queue for the CPU. Only once the kernel process reaches the front of the scheduling queue can it then handle the request to allocate the resource to the user context process.

Following this, the kernel process gives up the rest of its time slice on the CPU as it is done processing the request made to it by the user context process. The user context process is then woken up[14] and scheduled for the CPU on the back of the scheduling queue. This chain of logic can occur multiple times on a single request as a resource may

[14] Able to be rescheduled as the stall is now handled.

Table 3.3: Compiled vs Interpreted Time to Count to 1,000,000 over 20 Iterations

Language	Language Category	Time Taken (s)
C	Compiled	5.867
Java	Interpreted	10.526
Python	Interpreted	8.266

not be ready for allocation, or the handling kernel process may be busy handling another request.

As this implementation was designed to mitigate this, it made use of running itself in the kernel context where it would have access to these buffers and reserved memory addresses. This mitigates the problem of relying on other processes to handle resource requests and so reduces the number of context switches at runtime. This means more time out of stalls and less time in the scheduling queue waiting for CPU resources.

3.6.2 Compiled vs Interpreted Languages

Most programming languages can be categorized into two major subgroups, that being an interpreted or a compiled language. The major difference between these two categories is how the programs function at runtime. Compiled languages are preprocessed and fully converted into byte codes specific for an architecture, whereas interpreted languages are translated to these byte codes at runtime.

Table 3.3 shows the results from an interpreted language against a compiled language in the time taken to count to 1,000,000 from 0; this was done through a while loop printing out each number. The interpreted languages used were Python and Java[15], where the compiled language used was C. One can see from these results that an interpreted language is in fact slower than a compiled one. This is due to the fact that an interpreted language needs to do something other than run just the program one wants to execute at runtime, that being interpreting the code to native byte codes. A compiled language on the other hand has no overheads at runtime and just runs the code as instructed byte code for byte code. For this reason this routing simulator was based on a compiled code base.

It is important to note that one can compile Python code and can in fact even write C code within Python functions and scripts (Python Software Foundation, 2013). This es-

[15] Although this is run under JIT which pre-compiles code to native byte codes at runtime

sentially raises the speed of the language to that of a compiled one. It is also notable that the use of language translators can convert one language to another in an attempt to bootstrap interpreted languages to compiled ones bringing their speed to near that of a compiled language; it is, however, a difficult task to achieve the full compiled speed as the language conventions and functionality that one can achieve through an interpreted environment may not be available within the native instruction set of the compiled architecture.

3.6.3 Application of Structs in C

C also comes with a handy language feature, namely the struct. This is essentially a template that one can use to define addressable complex data structures within an allocated memory blob, and thereby allow convenient access to them through the use of pointers. This tool allows one to easily allocate a block of memory, cast a struct defining where each member of the data is located and what size each is, and then access or assign them as one needs to.

Such use of a struct is easily shown in the relevant example of receiving a packet. A packet is essentially handed to the program as a blob of data, an allocated memory block without any details. Knowing that this blob of data in memory is in fact a packet, one can gain access to the packet's members through casting it to a struct of a packet. From here, one gains access to all the fields in the packet's headers and their types as the struct gives pointers to each field in the blob of data in memory. Ultimately this method was used to describe packets within this simulator as the use of pointers is simple and fast.

3.7 Thread Interfacing

Multi-core threading in this routing simulator is achieved through the use of work queues rather than conventionally spawning threads. This is because the Linux scheduler already implements what was needed in this implementation (Bovet and Cesati, 2000) and so recreating what is already there for use is a wasted effort. Having use of this work queue library means that it is well supported with the OS used, that being Linux version 3.2.0-4-amd64, and should cause the least debugging problems at time of development.

A work queue is implemented through hooking a function within one's process to the scheduler. From here one schedules work into the work queue created for the process.

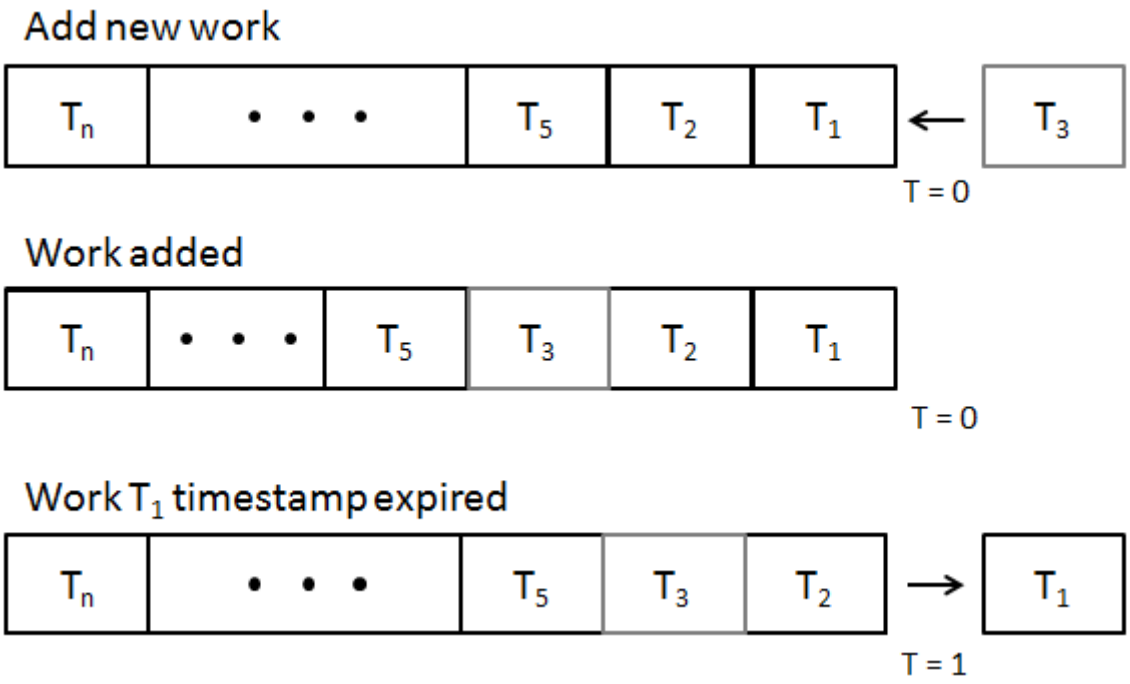


Figure 3.11: Work Queue Overview

Work in this context is made up of four primary fields, these being: the queue on which to schedule (as a process may have more than one queue), the work that needs to be done (which is represented by a pointer to the relevant data in memory), the core on which the work should be done, and finally the time stamp after which the work should begin.

Work in the work queue is then sorted by time stamp. Essentially, when the time stamp expires, a thread is spawned on the designated CPU core and is pointed to start at the work queue's bound function. The pointer to the memory location in which the work resides is passed to this function and processing occurs (Love, 2003, Page 9).

3.7.1 Delay Implementation

A desired feature of this routing simulator, as per Section 3.1.1, is delay. Using the time stamps of the work scheduler for the work queues, one can clearly see a relation between this type of scheduling and delay. Essentially the delay, as functionally described in Section 3.1.1, is implemented through using the knowledge of the delay of hop's length to schedule the packet's arrival at the destined node, and using the work queue to deliver the work at the expected known time. This expected known time is calculated by the following function:

$$WorkStartTime = (\sum_{i=0}^{k} Delay_i) + CurrentTime, where Delay_i \in HopDelayCauses$$

This means that the packet is scheduled to arrive at the destined node after a hop time which consists of the amalgamation of all causes of the delay added to the the time the packet was scheduled to leave the source node.

3.7.2 Resource Stability

Another problem that comes up in multi-threaded applications is resource contention. This is a problem that arises when multiple threads are reading and modifying a shared variable. This can result in the data being made inaccurate if, between one thread's read and write, a second thread were to modify a variable.

To further explain this, if a program was running two threads which were both reading a number, adding one to it and then writing that number back to the same variable, the following chain of logic could occur:

Listing 3.6: Threads Without Variable Locking

```
1  variable = 0
2
3  Thread 1: Read variable as 0
4  Thread 2: Read variable as 0
5  Thread 1: Add 1 to variable to make it 1
6  Thread 2: Add 1 to variable to make it 1
7  Thread 1: Write 1 to variable
8  Thread 2: Write 1 to variable
9
10 variable = 1
```

This is clearly incorrect, as if a variable that originated at 0 has 1 added to it twice, the value should be 2. This is due to both threads reading the variable while it is still in its 0 state, modifying it by adding 1, and then writing both their answers back.

This problem, however, does not occur in this routing implementation, as the bulk of routing operations are in fact reads. There is no data ever written back into the state of the simulated network or the routing tables within the network. Instead the data is written into the metadata of the packet buffer depicted in Figure 3.10. The packet is

handled by a single thread at a time as per the specification of the work queue and so no errors can occur via multiple threads having access to a single packet at any point in this simulation.

In the case of network modification, a packet that is scheduled to a node that does not exist will find itself arriving at a now down node. The routing simulator will handle this with an appropriate "destination not found" message within the ICMP specification (Postel, 1981b). This mitigates the errors caused by runtime network modifications by the user and serves as a response to a known problem found in large scale networks when nodes go down during transit of a packet to the node directly, or later on in the route that the packet was originally destined to follow. This ties directly into the packet loss feature specified in the features in Section 3.1.

3.8 Physical Interfacing

Deployment of this routing simulator is done through the isolation of all real host IP addresses that are bound into the network, as defined in Section 3.2.1 and 3.2.2. This allows for no external interference to occur between possible real hosts using the IP addresses that are being simulated within the routing simulator.

This is done by using the simulation host as a gateway through which all real hosts that cannot locate an IP address on the immediate network end up forwarding their traffic, via the implementation of a NAT. When the simulation begins, any traffic that is destined out of the local scope network, that is the hosts located immediately located through the attached switch/router, is then forwarded to the gateway. This gateway is the routing simulator's host and so the routing simulator can now determine whether the host's source IP address is within the simulation or not.

If the IP address is within the simulation, then the traffic is redirected into the routing simulator's simulated network. If the IP address is not used in the simulation, then the traffic is forwarded as per normal out into the network that lies beyond the gateway, if one exists. This also means that one cannot use the NAT's IP address range within the simulated network as these packets would be routed directly to the the host on the same network rather than through the gateway host.

It is important to note at this point that one can also start and stop the simulation in a seamless manner. The physical network also extends as far as wireless devices. This

would enable one to include hosts on a network that are using the simulation host as a gateway to change their network topology, or redirect them to servers set up for testing purposes without any break in communication whether the connection to the switch is wired or wireless.

3.9 System Configuration

This routing simulator was designed to allow for dynamic configuration without recompilation of the internal engine that drives this implementation. This brings forth issues that should be addressed when configuring an implementation within the kernel. The first problem is communication to the kernel module, the second being sane information passed to the module.

3.9.1 Communication Paths

Firstly, one requires a way to communicate with the kernel if one wishes to configure this implementation. This can be be done in one of three ways. The first, and the least convenient, is to open a configuration file from kernel space. This would require the user to be bound to a set file structure requirement as one cannot define where to find a file to a kernel module after compilation of the module.

The second method involves the use of ports. One could open a port to listen to any configuration data destined to it and use the port as if it were connected to a network as if one were receiving data. This is convenient, however it is not the best solution available to a developer.

The third method, which was ultimately chosen for use within this implementation, was that of procfs (The UNIX and Linux Forums Content, 2013b). This makes use of memory mapped buffers that are accessible through file descriptors that can be read and written to as if they were standard files. This allows for a simple interface to the kernel module without the need for setting up ports and handling a connection between the kernel module and the client that wishes to access it.

3.9.2 Anticipated Configuration

To touch on the second point, incorrect or malformed information can cause this implementation to crash or behave in an erroneous manner. In the best case the implementation will do nothing more than produce false results to the user. However, the worst case can lead to kernel panics and loss of data through file table corruption.

As this is a real possibility, it is required for all information to be checked for sanity, that it is fully intact and in the correct format before the configuration is passed to the kernel module. This is handled through an external library written in Python[16]. This library's job is to take in a configuration file, parse over it and finally return any errors that have been made. A successful parse results in the option to load the configuration into the routing simulator.

3.9.3 Remote Configuration

The use of remote configuration holds dual purposes; these purposes are security and a stand alone environment with no special software requirements by client hosts. The security perspective is purely based on the removal of the ability to directly access the file descriptors created by procfs on the simulation host. If one cannot directly run one's applications on the simulation host, the file descriptors created cannot be accessed in /proc/.

This is achieved through a web based interface using Pyramid[17] through Python. This allows one to connect to the simulation host's IP on a web browser, an application that comes standard with most GUI driven OSs, and load configuration files through the web page that is received. This means that the only way to load a configuration file into the kernel module routing simulator implementation is from an external system via the web based implementation, thus ensuring the aforementioned security aspect of this system. As this process is not resource intensive, one can use a language like Python to handle string manipulation as it is well suited to this even though it is an interpreted language.

The use of Pyramid in Python also allows for easy access to the Python library written to parse and process configuration files, as mentioned in Section 3.9.2 above. The difference

[16] Python was used as it easily manipulates strings and can be used to easily implement regular expressions to check the format of the provided data.

[17] http://www.pylonsproject.org/

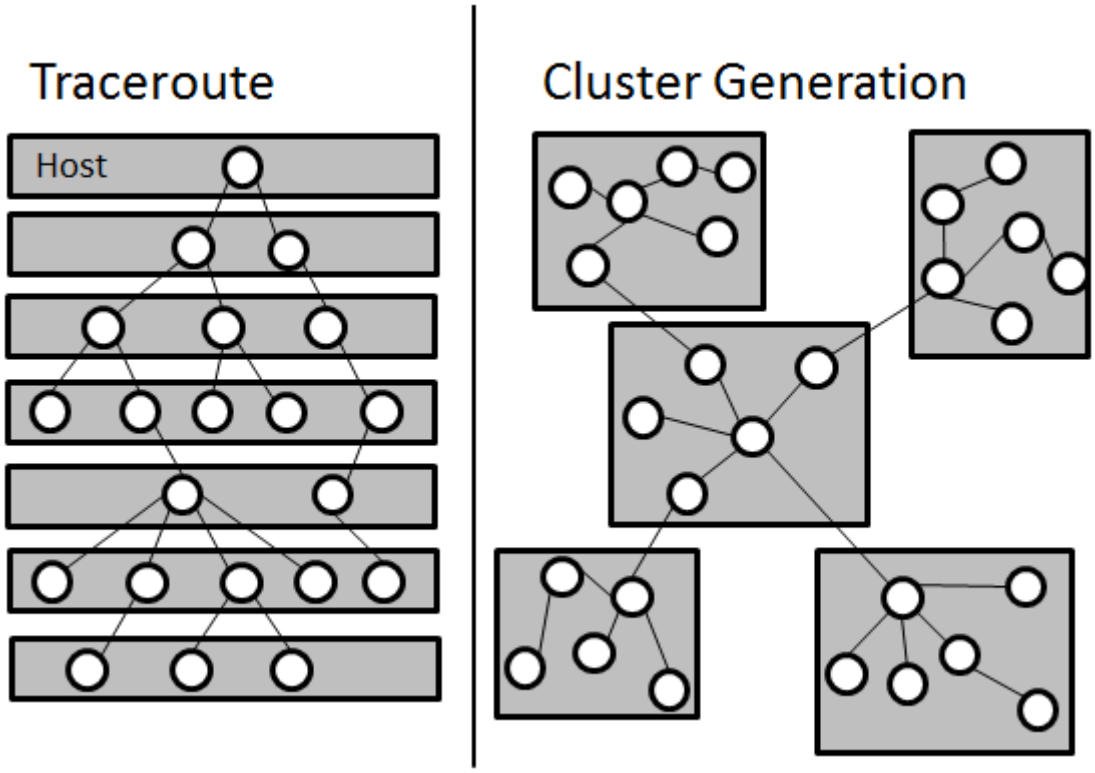


Figure 3.12: Network Generation Topology Overview

is that instead of being given a local file from which to read the configuration data as originally designed, the simulation host now reads the configuration file over the network from the connecting client that wishes to configure this routing simulator.

Essentially, this allows a client that is HTML compatible with a configuration file to safely submit a file for parsing and, if it is deemed sane, load this configuration into the routing simulation for use. This omits any need for specially-written OS specific software to use and configure this routing simulator.

3.10 Configuration Script Generation

This section touches on two tools that were created to support this network simulation implementation. The first tool simply generates a semi-random network. This first tool is given a set of parameters by which to model a network. These parameters are namely the number of nodes in the network, how many subnet clusters there are and how many hops exist on average between end points. These parameters are then used to create a new

random network with the specified topology. This can be shown by Cluster Generation, as seen in Figure 3.12. To elaborate on this figure, one can see nodes within grey boxes; each of these boxes is a cluster. Each cluster is generated separately and then a primary cluster is decided. The secondary clusters are then all connected to the primary cluster. A check for whether the primary cluster has enough endpoints to connect the secondary end points is performed prior to the primary cluster's selection.

The second tool is a traceroute[18] conversion tool that simply takes a set of traceroute generated data from a single host as the root point, and creates the network with all forwarding tables from it. This is done by simply regenerating the network and then traversing the tree to build up forwarding information. Although traceroute provides the user with real data from a real network, it does not create the best network topology as all the routes are derived from a single node, as seen in the Traceroute section of Figure 3.12. This shows that all collected nodes in the route to the node in question are derived from a single point. These routes are also subject to change and circular routes are a common problem with this method, as a single path trace performed twice may yield two different paths to the same destination.

Both of these tools return configuration scripts that can be directly loaded into the routing simulator, as well as a list of endpoint IPs to which real hosts can be bound at a later stage. This is useful, as configuration scripts logging each forwarding table entry, delay and other such metadata can be a laborious task for the user to create by themselves.

3.11 Final Specification

This section introduces the final specification of this routing simulator in a concise manner. Each choice will be followed by a recap on the reason why it was chosen and if applicable, what the specification aids in achieving.

3.11.1 Supported Features

This list contains the features that this simulater supports:

- Delay - This models delays found between hops in a network.

[18] http://linux.die.net/man/8/traceroute

- Packet Loss - This simulates bad links or overburdened nodes within a network.
- Packet Mangling - Packet corruption to further aid in simulating bad links and nodes within a network.
- Jitter - Introduced in the final design to replicate wireless connections through retrying lost packets.
- Worm Hole - Allows for process offloading through hop aggregation.

3.11.2 Final Engine Design Decisions

This list contains the design decision made to accomplish the above listed features:

- Architecture - 64-bit architecture allowing for larger addressable memory count as well as increased CPU register count.
- Operating System - Linux, chosen due to ease of configuration and development as it is open source and allowed inclusion of this implementation into the Linux kernel as a kernel module.
- Node Data Structure - Radix tries were decided on for this role as they allowed for dynamic allocation and accessing nodes within this data structure compliments the key used for each node's data store, this being IPv4 address.
- Packets Received - A hook onto netfilter allows for easy packet acquisition, further motivation for this is the ability to copy and then drop packets of hosts that are part of the simulation, thus allowing for network isolation.
- Packets Transmitted - Simple use of sockets allows for byte buffers representing spoofed packets to be generated by this routing simulation.
- Simulated Routing - This is accomplished through replication of rule based forwarding tables. This allows simulation of multiple interfaces as well as the ability to worm hole packets. This also means that each node can be represented as a stateless entity.
- Supported Protocols - Due to memory limitations, this routing simulator is restricted to the IPv4 address space.

- Threading - Use of work queues with individual core scheduling allows for parallel processing as the nodes within this routing simulator are stateless and so have no interaction; this removes the requirements to lock variables.
- Configuration - Web based configuration removes the need for requiring specific software to configure this routing simulator. This also allows for easier configuration checking through languages designed for completing such tasks.

3.12 Summary

This design chapter gave a broad overview of the implementation and what it strives to achieve through its design. The desired functionality was laid out and from there the best way to implement each part was selected to achieve each of the functionality goals in Section 3.1. In Sections 3.3 to 3.4 decisions were made, from what memory structure to use to the way in which routing would occur in this implementation in order to best utilize memory and CPU within the routing simulator's host.

The hardware architecture and choice in OS was chosen and justified in Section 3.5 and how one would go about using this choice in Section 3.6. This chapter wraps up with a look into how threads are interfaced to, what the physical network's requirements would be and how one would configure this research in Sections 3.7 to 3.10. Finally, tools were described that were developed to aid in the success of this implementation.

In Chapter 5 one can find the results to all the testing that was targeted at showing that this implementation achieves all functionality. This, as well as stress testing to show peak performances on set hardware. The use of netfilter and how it can be used to implement basic firewall ruling follows. Finally, an overview of how the physical network configuration should be performed and how one can better synchronize clocks for testing is detailed.

4

System Implementation Challenges

This chapter's primary focus is to bring to light specific issues encountered during system implementation and set up. These problems range from bad mathematical theory that throws off the statistical behaviour of this routing simulator, to poor documentation that causes trouble in use of libraries that form a core for this implementation. This chapter is intended as a completing text to anyone who wishes to re-implement this research and have it behave in the manner that the author intended.

The topics addressed in this chapter begin at Section 4.1 with an in-depth analysis of random number generation in the Linux kernel version for which this implementation is intended. Following this, a contrast to the successes of radix tries, as presented in Section 3.3.5, within this implementation is touched on and how one should use the work queues discussed in Section 3.7 to ensure multi-core support is detailed Sections 4.2 and 4.3.

The use of netfilter can also aid in configuration and isolation and this can be achieved through adding appropriate rules, as described in Section 4.4. A look into how physical network configuration should be set up is explained in Section 4.5. In closing, clock synchronization for long term testing is explained in Section 4.6 as well as how one can better synchronize clocks using a time keeper to allow for more accurate results as produced in Chapter 5.

4.1 Random Number Generation

Random number generation is used in this simulator for determining jitter, packet mangling rate and packet loss of nodes within the simulated network. For this one requires an accurate, and more importantly fair, random number generator. This is not possible with the current Linux 3.2 kernel[1]. Consider the Linux 3.2 kernel source code snippets in Listings 4.1 and 4.2.

To summarize this code, **get_random_int()** is a function that produces a random unsigned integer of 32-bits through a method that ensures its goal of minimal entropy pool depletion. This first method safe for cryptography; however, the resources required to generate this integer are reduced by using this implementation (Mackall and Ts'o, 2014).

The second method, shown in Listing 4.2, uses **get_random_int()** to ensure its functionality is **randomize_range()**. One can generate a random number within a range defined by the user with a range modifier. The range is simply taken as the number of integers between the start and end specified; the range modifier is also introduced at this point. From this point a 32-bit integer is requested from **get_random_int()** and then this integer has the modulus operator applied to it using the range that was determined by using the specified start, end and length specified. At this point one has an integer between 0 and the calculated range variable. Now one simply adds the start point of the range to this number to shift it into the range requested, thus delivering a value between the requested range.

This logic is sound until one considers the maximum value of a 32-bit integer, this being 4,294,967,295, and the range within which the start, end and length are determined. Consider the function called with 0 value for start, 100,000,000 for end, and 0 length to apply no modifier to the start range. For this the range would clearly evaluate to 100,000,000, which is where the problem arises.

Applying a modulus of 100,000,000 to a number would yield a number in the range of 0 to 99,999,999. Considering that 4,294,967,295 is the maximum number a 32-bit integer can yield, this would cause an uneven distribution of integers produced by this range. The reason for this is that when the modulus operator is applied to a random integer between 0 and 4,199,999,999 it will return a number between 0 and 99,999,999. Every number in the second range can be produced by 42 separate integers in the first range, and thus the numbers will be evenly distributed. However, when one considers the range of a

[1]Comment cannot be made on other kernel versions as this was the version used for implementation.

Listing 4.1: /Linux/drivers/char/random.c get_random_int

```
1723 static DEFINE_PER_CPU(__u32 [MD5_DIGEST_WORDS],
1724                        get_random_int_hash);
1725 unsigned int get_random_int(void)
1726 {
1727         __u32 *hash;
1728         unsigned int ret;
1729
1730         if (arch_get_random_int(&ret))
1731                 return ret;
1732
1733         hash = get_cpu_var(get_random_int_hash);
1734
1735         hash[0] += current->pid+jiffies+random_get_entropy();
1736         md5_transform(hash, random_int_secret);
1737         ret = hash[0];
1738         put_cpu_var(get_random_int_hash);
1739
1740         return ret;
1741 }
1742 EXPORT_SYMBOL(get_random_int);
```

Listing 4.2: /Linux/drivers/char/random.c randomize_range

```
1753 unsigned long
1754 randomize_range(unsigned long start,
1755                 unsigned long end,
1756                 unsigned long len)
1757 {
1758         unsigned long range = end-len-start;
1759
1760         if (end <= start+len)
1761                 return 0;
1762         return PAGE_ALIGN(get_random_int()%range+start);
1763 }
```

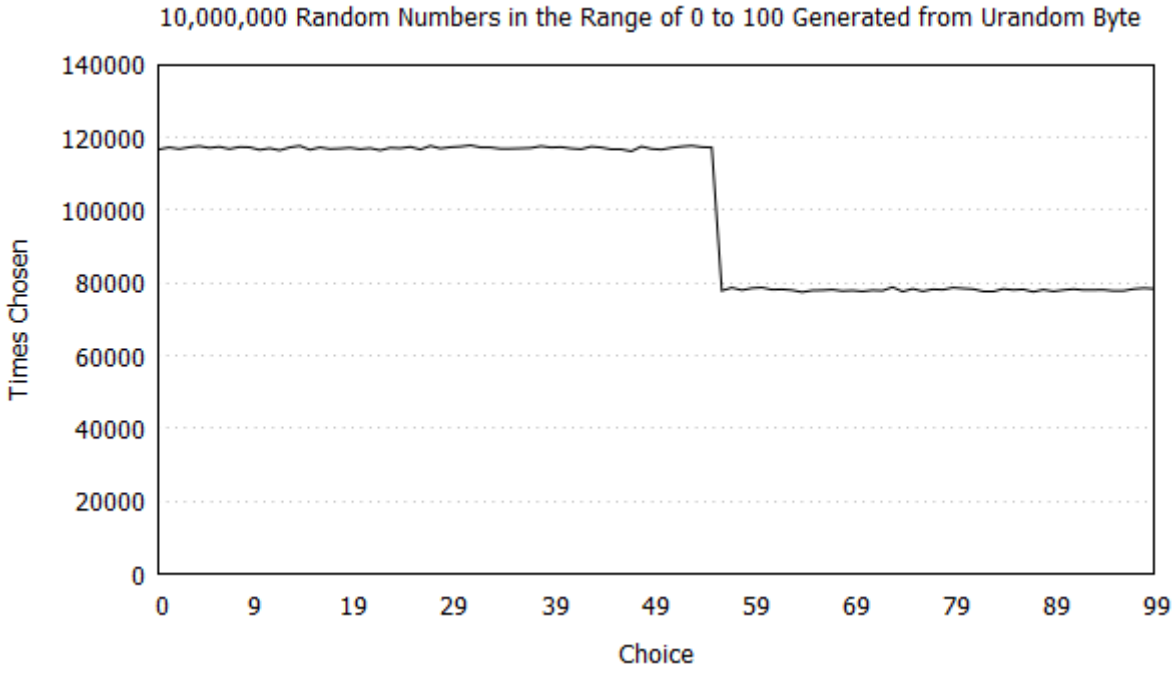


Figure 4.1: Modulus 100 applied to Single Byte for 10,000,000 Iterations

32-bit that can be produced between 4,200,000,000 and 4,294,967,295, one must notice that once the modulus operator is applied the only values produced are between 0 and 94,967,295. In short, this means that integers 0 through to 94,967,295 can be selected, integers 94,967,296 through to 99,999,999 cannot be selected for this range of a 32-bit integer.

To conclude the significance of these values yielded, this means that when a modulus of 100,000,000 is applied to the full range of a 32-bit int, as produced by **get_random_int()**, integers 0 through to 94,967,295 can be yielded by 43 unique values in the range of a 32-bit integer, where integers 94,967,296 through to 99,999,999 can only be yielded by 42 unique integers.

This means that unless the range that start, end and len produce in **randomize_range()** is perfectly divisible[2] by 4,294,967,295, one cannot expect an evenly distributed random result from this function. For this reason a function was created to evenly produce integers between 0 and 99 through randomly producing smaller integers with even distribution and combining them to produce evenly distributed integers between 0 and 99. This was required, as random percentages are usually represented as a value between 0 and 100.

[2]Yields a result with no remainder.

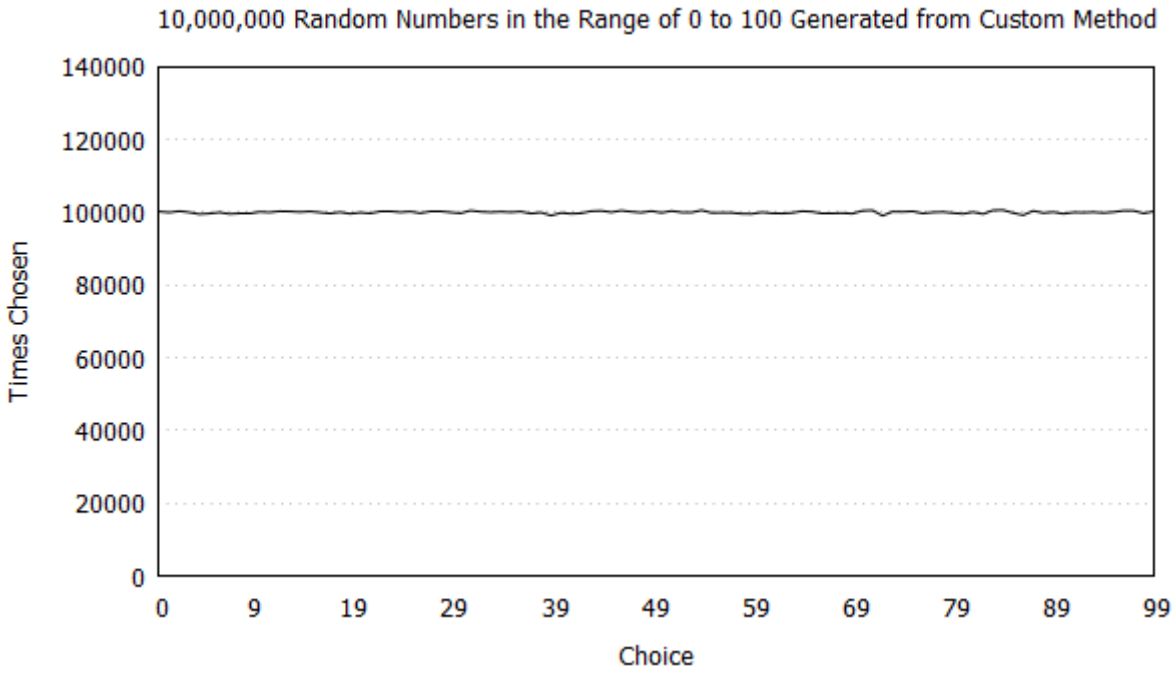


Figure 4.2: Modulus 100 applied to Single Byte for 10,000,000 Iterations Using Custom Method

In implementation, single bytes were collected and masked to produce the required smaller values. Applying a bitmask of 00111111 to a byte limits the value to a range of 0 through to 63. Similarly, a mask of 00011111 yields a value from 0 to 31 and 00000011 yields a value from 0 to 3. Although the range of numbers that each mask can produce adds up to 100, in practice this function will never yield a number over 97. For this reason two single bit numbers are produced.

The combination of these numbers, if logic is applied correctly in implementation, produces an evenly distributed integer between 0 and 100, and these results can be compared to the standard modulus function in Figures 4.1 and 4.2. This does however come at the cost of more processing time required to generate all the individual random integers. Because of this, optimization can be done through generating two random bytes and masking a high order of bits to produce one number, and the low order to produce another. If one wants to ensure an even distribution, no two numbers' bit masks should intercept on the same byte.

4.2 Radix Tries Memory Deallocation

The motivation for use of a radix trie was explained in Section 3.3.6. However, there is an issue with the library that should be addressed; this issue is the deletion of the entire data structure. The kernel library for this data structure has no destructor for the data structure, instead it only has functionality to remove a single item at a time (Velikov *et al.*, 2014). This means that a destructor for this data type has to be created by the user who wishes to perform such a task.

In practice, one would have to keep track of each node created within the radix trie, which can be done through separate data structure or other methods like continuously accessing the radix tries's root. From here, one would have to request a deletion of each individual node one at a time through the library's **radix_tree_delete()** function.

The destruction of the entire radix trie is needed at the end of a simulation, as all nodes within the simulation are stored within this data structure. This process is a slow task, in the order of minutes when dealing with networks over a million nodes large with five hundred forwarding entries per node, which is further slowed on nodes with high forwarding table entry count. The reason for this is that the forwarding tables are stored in link lists, and so each node has to be visited and freed individually.

Considering that this task of node de-allocation occurs after simulation completion, this characteristic of this simulator implementation can be overlooked unless one wishes to simulate large networks in quick succession. For this reason it was left to de-allocate in the manner described above. However, one can simply reboot the host system to achieve the same task.

4.3 Work Queue Core Scheduling

Work queues are used in the scheduling of packets within this simulator; this is detailed in Section 3.7. The main point to note about this library is that there are four functions in which to schedule work. The first two functions, **schedule_work()** and **schedule_work_on()**, schedule work for immediate processing. These do not serve the requirements to implement delay effectively within this simulator.

The second two are **schedule_delayed_work()** and **schedule_delayed_work_on()**. These functions allow one to specify at what time the work should start; this allows

Figure 4.3: System Monitor at Simulation Runtime using Round Robin Scheduler

one to implement the delay feature with ease. At implementation time the important fact to note is that **schedule_delayed_work()** does not automatically allocate a core on which to process the task; instead it queues the task for the same core which queued the work.

Schedule_delayed_work_on() takes an extra variable, this being the core identification, which tells the work queue on which core to schedule the task to be processed. This means that one has to implement one's own scheduler if one wishes to utilize all cores on a host system. Listing 4.3 contains the code snippet of the round robin scheduler used in this implementation. The core that the work is assigned to is defined on Line 17 of Listing 4.3 and it simply uses the last value of **RR_sched_core**, which is an integer that simply gets incremented, and then applies the modulus operator on it by the number of cores on the host system. After this, **RR_sched_core** is incremented so as to assign the

next set of work to the next core. This round robin's implementation's CPU usage can be seen at runtime in Figure 4.3.

Listing 4.3: schedpkt.c

```
149 // Schedule new packet
150 int8_t schedule_pkt(pkt *new_pkt){
151
152     pkt *schd = new_pkt;
153
154     INIT_DELAYED_WORK((struct delayed_work *)schd,
155                       (void *)route_pkt);
156
157     queue_delayed_work_on(((RR_sched_core++) % NUM_CORES),
158         pktqueue,
159         (struct delayed_work *)schd,
160         usecs_to_jiffies(new_pkt->nxt_hop_delay*1000));
161
162     return 0;
163
164 }
```

Round robin scheduling was a simple-to-implement, low overhead solution for task distribution over all cores in the simulation host (Silberschatz *et al.*, 2010, Page 194). The reason for this was because the task is constant; this being performing a hop after a delay. Because of this, it is fairly easy to assume that evenly distributing hop tasks among all cores would result in a fairly well distributed workload. Referring to the CPU History graph in Figure 4.3, one can see that this holds true as all cores share in the task with a fairly even load average.

4.4 Netfilter

Netfilter is described in Section 3.2.1; this section aims to point out implementation hazards and advantages that can occur when making use of this library when using the incorrect return values defined by netfilter.h (Welte and Ayuso, 2014). The definition of these values can be referred to in Listing 4.4.

Listing 4.4: Linux/include/uapi/linux/netfilter.h

```
9  /* Responses from hook functions. */
10 #define NF_DROP 0
11 #define NF_ACCEPT 1
12 #define NF_STOLEN 2
13 #define NF_QUEUE 3
14 #define NF_REPEAT 4
15 #define NF_STOP 5
16 #define NF_MAX_VERDICT NF_STOP
```

In order for a packet to continue to the next hook in netfilter, it is required that the current hook allows the packet to continue through its filter. This is performed through returning NF_ACCEPT from the hook function. If one wishes to discard a packet, one can return NF_DROP.

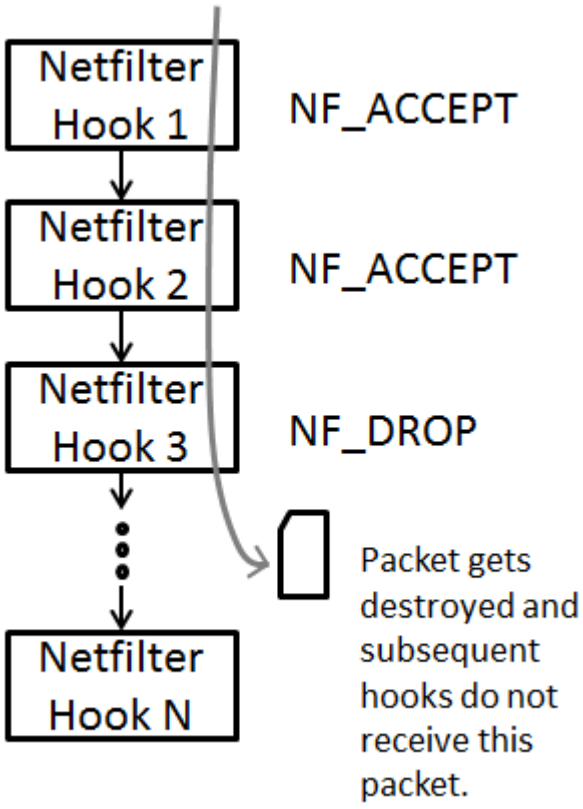


Figure 4.4: Overview of Process of a Dropped Packet in Netfilter

Discarding a packet from a netfilter hook causes the packet to be destroyed. As such, any netfilter hook after the one that discarded the packet shall not receive the packet, as

it has been destroyed, as shown in Figure 4.4. As netfilter acts as the firewall between the network interface and the applications that run on the respective host, the packet is treated as if nothing ever arrived at the interface. Essentially, a badly configured hook can lead to unintentional packet loss.

There is a further point to make upon packet modification within netfilter. When a packet arrives at the network interface, a single buffer is allocated; netfilter is given a pointer to this buffer. If one were to dereference this pointer and modify the packet that the pointer points to, one would directly modify the packet's data and/or the metadata in the packet's buffer structure. When the packet is passed onto the next hook, or introduced into the host for an application to process, the packet would represent that of the one modified by one's hook.

For this reason, packets in this simulator have to be copied to a new, separate buffer if one wishes to modify them. Furthermore, attempting to reference the packet at a later stage is a dangerous task if it is not copied to a new buffer. This is because the packet when passed out of one's hook function no longer belongs to that hook function[3]. This means that if the packet lives its life cycle within the host and is then deallocated, attempting to access this packet would yield results that can be unexpected[4]. For this reason, all packets introduced into this simulation are copied and managed by the simulator, so as to avoid potential bad dereferencing and mangling of packets for subsequent hooks and applications.

Being able to drop a packet also allowed for functional isolation of a host from a network. This is needed, as one would not want traffic generated from a physical host bound into the simulation to appear onto a network that is not part of the simulation. When a packet is received into the simulation, it is checked as to whether it is bound into the simulation. If the physical host is bound into the simulation, the packet is then copied into a buffer for use in the simulation and then the original packet is dropped by returning NF_DROP. This means that if the simulation host is connected to an external network that has other gateways set up, the packet will not be passed out the connecting interface in an attempt to route the packet into the physical network. This functionality is described by Figure 4.5

[3]Actually it never belongs to the hook function, the hook function is just allowed exclusive access until done processing it.

[4]Depending on whether the memory in which the packet used to reside has been reallocated and written over by other data.

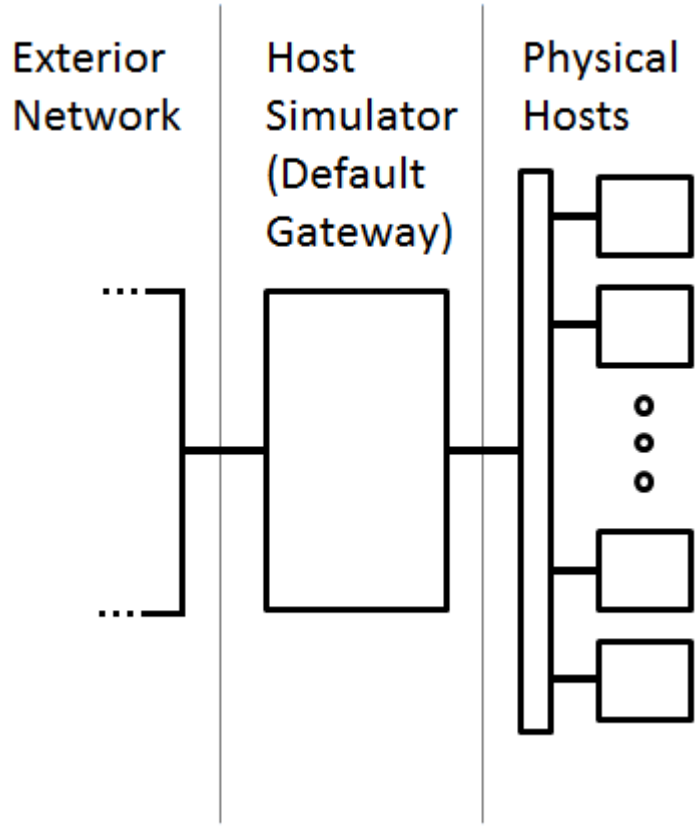


Figure 4.5: Abstract Topology of Simulation Environment

4.5 Network Configuration

In order for this network simulator to run successfully on a host and interface with other physical hosts producing and receiving traffic to and from this simulator, this simulator needs to have the external physical host's traffic routed to it. This is done through setting up the simulation host as the default gateway of the network. This means that if an IP address to which a packet should be sent cannot be found on the immediate network, the packet is instead routed out through a default route, this being the default gateway. This means that if a physical host cannot find an IP address on the immediate network, such as one that is being simulated, it forwards the packet to the simulation host, as it is seen as the default gateway, in order to forward the packet to a larger network in an attempt to locate the host in which the IP belongs.

Whether a packet is forwarded to the default gateway or not is based on the subnet mask in which a host resides. If the IP address one is requesting does not lie in the subnet that the host's IP address was assigned, then it knows immediately to search up the

network hierarchy via its default route to the default gateway. This method does have some drawbacks. If a simulated node has an IP that can reside within the subnet from which the physical host's IPs are assigned, then the physical host making the request will not forward the packet to the default gateway. Instead the physical host will attempt to address the IP address on the immediate network and if there is no response, then give up; this is not a behaviour that allows for successful network simulation.

The easiest method of turning the simulation host into the default gateway is through setting up the host to act as the default gateway on the network. When a physical host is introduced into this simulator, instead of having to manually set up the forwarding table of the host, one can also set up the simulation host to act as a DHCP to automatically assign IPs to hosts on the network. The downside to this is that one has to keep track of which IP has been assigned to each host and so IP address assignment can be hard to keep track of. A better approach, although time consuming, would be to manually assign each IP to each host that one wishes to involve in this network simulator and set their default route to the network simulator's host's IP.

4.6 Test Synchronization

Some tests in Chapter 5 required synchronization between physical hosts in order to achieve results. This task, although it may seem simple through use of a system clock, turned out to be an annoying task to achieve. Having disconnected the physical test hosts that were bound into the simulated network from access to the Internet, any form of clock synchronization on a common time base became an interesting task to achieve. Within a 24 hour period of testing, clock drift was in a range of a full second between all hosts. This caused skewing in testing results, as speeds were calculated in the total bits transferred on average per second.

To address this problem, the physical hosts kept synchronization through using a common time host attached to the network and a known delay to the host through ping. Although not perfect, a host would request the time from the time host, add the delay of the response over the network to compensate for network delay, and use this as a correction factor on the host computer's own clock in order to more accurately perform test iteration timings. In essence, this would act as a NTP server (Mills, 1991).

With this in place, each host's start and stop times of the test iterations were, for the most part, within 5 milliseconds of each other. There were outliers in the in the data,

which could be due to a delay in response from the time host; however, over the course of all data collection these outliers due to error were few and as such, mostly negligible. Other causes of data skew can be accounted to:

- Delayed start of the host's scheduler.
- Resource contention within the physical host.
- Delayed end of the scheduled task, again due to the host's scheduler.
- Failed write attempts by the host onto the network interface.

4.7 Summary

This implementation chapter gave rise to topics that were overlooked during research of this implementation. This allowed for recreation or more in-depth understanding of this implementation to be achieved. Random number generation and an analysis on the problem of unevenly distributed random number generation were discussed in Section 4.1. Sections 4.2 and 4.3 dealt with resource allocation and deallocation in the form of memory structure and task scheduling respectively. These two sections helped to explain how one can better manage resource allocation within the host simulator.

Netfilter, Section 4.4, and physical network configuration, Section 4.5, dealt with how this simulator handles incoming and outgoing traffic and further uses firewall rules to achieve simulation network isolation. These are shown to allow for more stable test environments for both the simulated network, and the network on which the simulation resides.

Finally, this chapter closes with how one can better synchronize multiple physical hosts for performing synchronous testing. The need for this is required over long periods of time as clock skew sets in; this can skew testing results presented in Chapter 5. The results collected and discussed in Chapter 5 begin by ensuring that packet routing within this implementation is sound. Testing in Chapter 5 continues by collecting and discussing data related to performance, resource requirement and limitations that are produced through these tests.

5

Routing Simulator Tests and Results

This chapter presents relevant results concerning verification of functionality and performance regarding this routing simulator. This chapter starts with Section 5.1 which applies long term testing to this routing simulator, checking that the system can stay active for prolonged periods of time without memory leaking or packet loss occurring under minimal transfer loads of under 100 Mbps. Section 5.2 questions the memory requirements of this network simulation and makes estimations as to the memory requirements of different networks within the simulation host.

Overall throughput and delays introduced through processing are then observed in Sections 5.3 and 5.4 respectively and these tests aim to see how far one can stretch the performance of a software based routing simulator. Section 5.5 brings this chapter to a close by observing the total memory usage of this routing simulator under load and aims to show the overheads incurred from buffering traffic within the simulated network.

These section's tests will be presented in the following format:

- Description of test.
- The relevant testing environment.
- Results obtained.
- Analysis of obtained results.
- Any figures will be set in a floating style.

Hardware used for testing, unless otherwise specified, was a Intel DQ77MK Motherboard with Intel Core i5-3570K Processor[1] at 3.4GHz, 8GB of 1333Mhz RAM and an Intel E10G42BT X520-T2 10Gigabit Ethernet Card[2]. The switch used was a Dell PowerConnect 6248 with a Dell P/N P623D 10Gbase-T expansion module. Other than the network interface, this was thought to be a well rounded host for testing at a mid market level at time of testing. The network interface was required to be at such a level as to test the maximum throughput of this implementation using the supporting hardware. It is also notable that only a single interface of this device was used in the stress testing; however, both of the interfaces were used in consistency testing to allow for testing the support of splitting the real bounded hosts over multiple interfaces on the simulation host.

This testing was performed on a single well rounded network configuration and is depicted at an IP connection level in Appendix A.

5.1 Consistency Testing

Consistency testing in this implementation was in the form of a long-term test consisting of multiple iterations of the following: a constant connection was instantiated, data was transferred and the connection then disconnected and verified. These connections were done over both TCP and UDP on a simple web server serving HTTP web pages over TCP. The servers' IPs are listed in Table 5.1 and one can see that the server count of each type was balanced.

[1] http://ark.intel.com/products/65520

[2] http://www.intel.com/content/www/us/en/network-adapters/gigabit-network-adapters/ethernet-x520-t2.html

Table 5.1: Server Host's Role per IP

Clients		**eb Servers**		**TCP Servers**		**UDP Servers**	
ID	**IP**	**ID**	**IP**	**ID**	**IP**	**ID**	**IP**
1	100.56.141.12	A	133.51.121.36	H	133.51.121.19	N	100.56.141.7
2	100.56.141.15	B	133.51.121.37	I	133.51.121.22	O	100.56.141.8
3	100.56.141.17	C	133.51.121.39	J	133.51.121.23	P	100.56.141.9
4	100.56.141.21	D	220.39.91.7	K	133.51.121.26	Q	220.39.91.39
5	100.56.141.23	E	220.39.91.8	L	133.51.121.30	R	220.39.91.41
6	100.56.141.25	F	220.39.91.12	M	133.51.121.34	S	220.39.91.44
7	100.56.141.26	G	220.39.91.14				
8	100.56.141.29						
9	100.56.141.32						
10	100.56.141.33						
11	100.56.141.36						
12	106.77.109.8						
13	106.77.109.11						
14	106.77.109.15						
15	106.77.109.18						
16	106.77.109.20						
17	106.77.109.21						
18	106.77.109.24						
19	106.77.109.27						
20	106.77.109.30						
21	106.77.109.31						
22	106.77.109.32						
23	106.77.109.35						
24	133.51.121.9						
25	133.51.121.11						
26	133.51.121.14						
27	133.51.121.16						
28	133.51.121.17						
29	220.39.91.18						
30	220.39.91.21						
31	220.39.91.25						
32	220.39.91.28						
33	220.39.91.29						
34	220.39.91.32						
35	220.39.91.35						

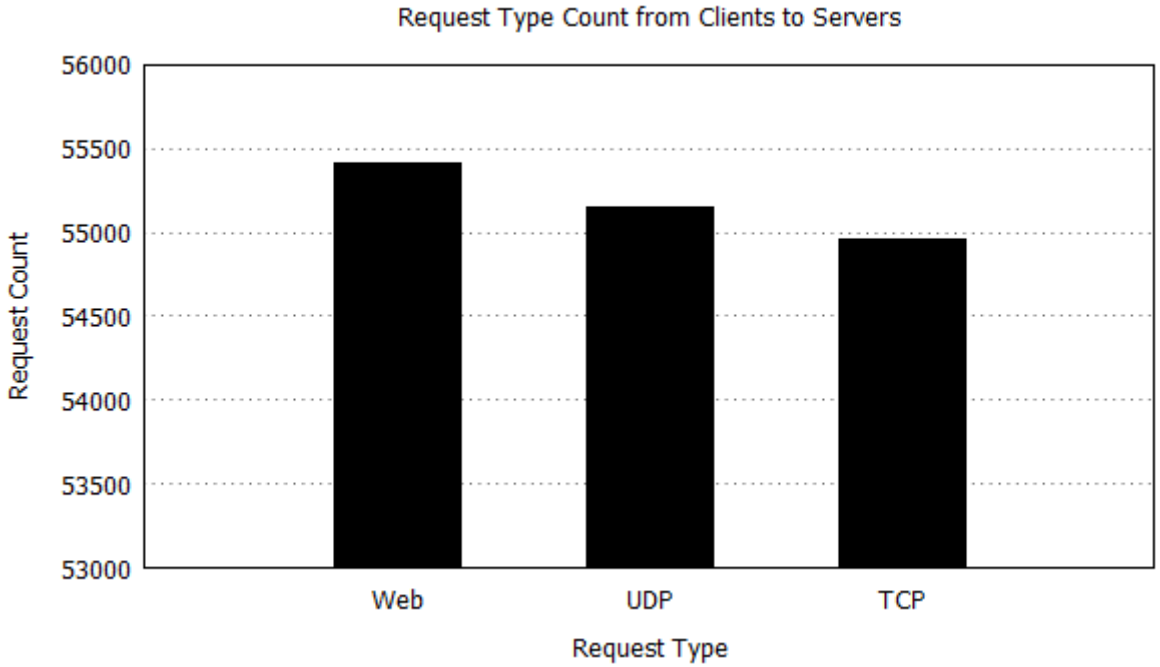


Figure 5.1: Request Type Count Performed by Clients to Servers

This test was done to ensure that multiple physical hosts can interact on multiple concurrent connections through the simulated network to which they were bound without any failures leading to data loss, connection loss, or the event of a server or client being put offline. The UDP and TCP tests consisted of a simple transfer of a payload of size 1024 Bytes from the client to the server. The web request was performed by requesting the web page served by a server; once the page was received it was checked for any errors, as the web page template was known to the client. Testing was performed over a twenty four hour period and connections were allowed to run simultaneously. Results were stored by the clients locally in the format of:

```
[Result][Type][ID][Server IP][Request Start][Request End]
```

Where result is denoted by a success or failure, type is the type of request in the test iteration which is either TCP, UDP or Web Request, ID denotes the request number, Server IP is the server in which the client connected to, and Request Start and Request End is the time stamp of the iteration's beginning and ending time. Results derived from this can be seen in Figures 5.1, 5.2 and 5.3.

5.1.1 Results

Figure 5.1 shows the count for each type of connection made by the physical client hosts bound to nodes in the simulated network to the hosts which were also physical bound into the simulated network. These results were taken to show the distribution of the types of requests made to allow for a better idea of the protocols used in testing and what proportion each made up.

Web requests made up the the largest proportion, totalling 55,415 requests made by physical clients, which amounts to 33.48% of requests performed. Following this was UDP requests requests, which totalled 55,150 requests or 33.32% of those performed. TCP was the least common request type performed and this made up the remaining 33.20% totalling 54,955 requests. This makes up the total of 165,520 requests during the 24 hour period of testing.

The results of each protocol's request count depicted by Figure 5.1 being roughly a third of the total is not surprising as each request was randomly selected with a 33.33% chance, so as to simulate a real world user. This selection was simply done through the means of a random number being generated between 0 and 2, which was then used to access an option in a switch case. This means that on average each request should be selected once for every three selections made.

Figure 5.2 shows the average time in which a request was performed. The most notable part of this was the time taken for the web request to be performed; this is due to the nature in which the web requests were served. The verification was only performed after the connection between the server and host was closed. This means that the keep-alive used in the connection in case of any further requests being made led to skewing of the data. Only once the server and client's connection timed out would the results be processed, and so the time for this type of connection is extended.

The time taken to perform a UDP request on average was 0.331 seconds, whereas the time taken on average for a TCP request was 0.527 seconds. This is also to be expected, not only because a TCP header is larger than that of a UDP, and so requires more time to send, but also because TCP requires time to set up a connection and close it due to the handshaking process. UDP, on the other hand, does not offer any securities in terms of data transfer other than a checksum for the payload. This means that there is no set up time required as an overhead for the connection; it is just expected that the IP and port in terms of which one is sending data is open and ready to receive traffic.

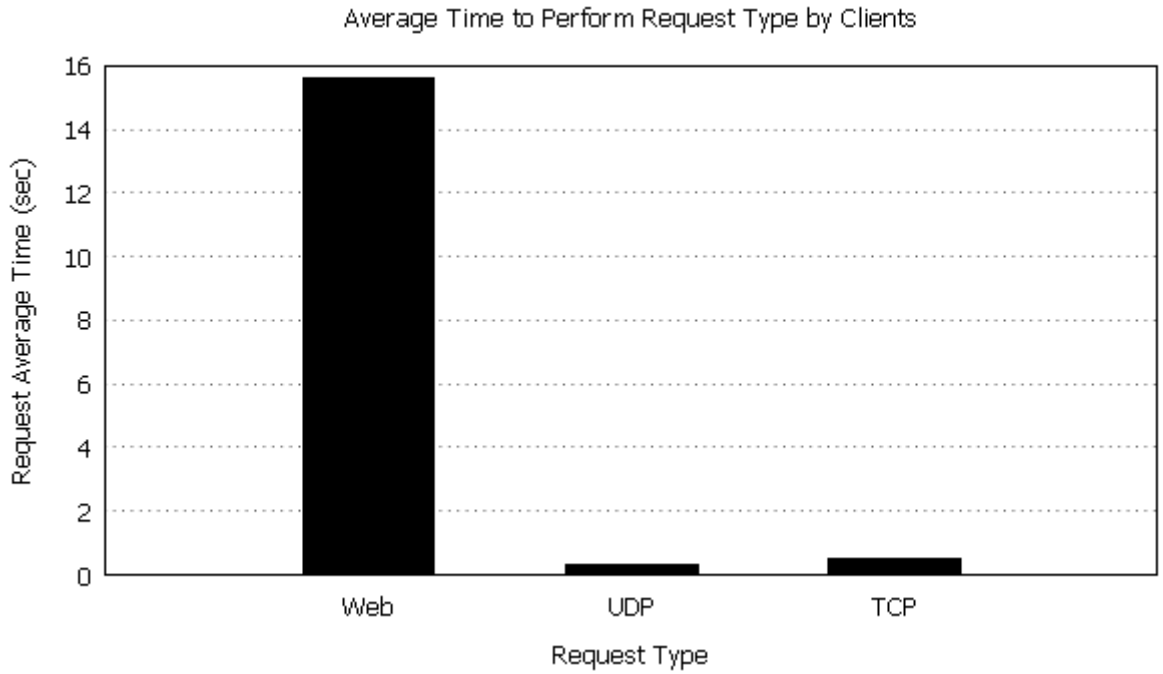


Figure 5.2: Average Time Taken to Perform an Iteration of Testing by Clients to Servers by Type

Figure 5.3: Number of Requests Made to Server IP from Clients

Table 5.2: Request Success and Failure Count by Request Type

Request Type	Success Count	Failure Count
UDP	55415	1
TCP	55150	0
HTTP	54955	0

In Figure 5.3 one can find the distribution by server IP to which requests were made by the clients. This figure was generated to show that requests were not biased towards one particular server or set of servers. The only skewing that is noted here is that the extra server allocated towards handling web requests causes the total web requests to be distributed among one more server than that of the TCP and UDP servers. This then shows a lesser connection count for HTTP servers than the TCP and UDP servers in comparison.

The number of successful requests versus failed ones is shown by Table 5.2. This graph was generated to show how many connection attempts were successful, as this directly influences whether this implementation was a success or not.

The results depicted in Table 5.2 is the most biased of the previous three. This does not come as a surprise, though, as this routing simulator's job is to successfully route packets. This does not come with a 100% result as there was 1 failure in routing, but when putting this up against the 165,519 successful requests performed during the 24 hour period shows a 99.999% success rate. This single failure is thus statistically insignificant and will be regarded as an outlier from the collected data.

5.1.2 Consistency Summary

Figures 5.1 to 5.3 and Table 5.2 show a good overview of the reliability of this routing simulator. Taking into consideration that this long term testing over a well distributed set of IPs in the requests added to the sheer number of successful responses when compared to the single failure it is concluded that this routing simulator is successful in serving its purpose of reliably routing packets.

5.2 Network Instantiation Memory Usage

Once the routing simulator was deemed sane and reliable by the test performed in Section 5.1, the need to know whether this implementation is plausible for mid range hardware use needed to be determined. This set of tests was performed to help determine just that. The amount of memory in commodity hosts[3] at the time of this research was seen to range between 4 and 16 GB. Although this amount did seem in excess of the problem's needs, it is to notable that this routing simulator is aimed at large scale network simulation, and so being able to scale one hundred thousand or one million nodes with forwarding tables into this available memory could prove to be an overzealous goal.

This test was performed by creating a simulated network containing one million nodes within the host running the simulation. Starting from zero forwarding entries, five entries were added per iteration of testing to a maximum of 25 entries. Memory usage was recorded through reading /proc/meminfo and in order to achieve the amount used a reading was taken directly before and after the nodes were added to the simulation. The difference between these two values was then taken in order to acquire the memory used in instantiating the simulated network.

As the results should be fairly linear in memory growth requirements, results can be taken and formulated into a simple linear equation to predict how much memory would be required for a required network. These results were then used to make a conjecture as to when, and if, a software based routing simulator would be able to simulate the entire Internet in memory with a set forwarding entry count.

5.2.1 Results

Figure 5.4 shows the growth in memory as forwarding entries are added to one million nodes in increments of five entries per node. Without any forwarding entries in place, the one million nodes have a base memory footprint of 23.80 MB. This means that the amount of memory required for each node is roughly 25 bytes of memory. After the first set of five forwarding entries is added to each node, the memory required rose to 193.5 MB. As one can note in Figure 5.4, this requirement grew fairly linearly to the amount of 859.4 MB. The average growth rate per five forwarding entries added to every node in this network was 167.12 MB.

[3] In terms of mid ranged hosts available in store at time of this research.

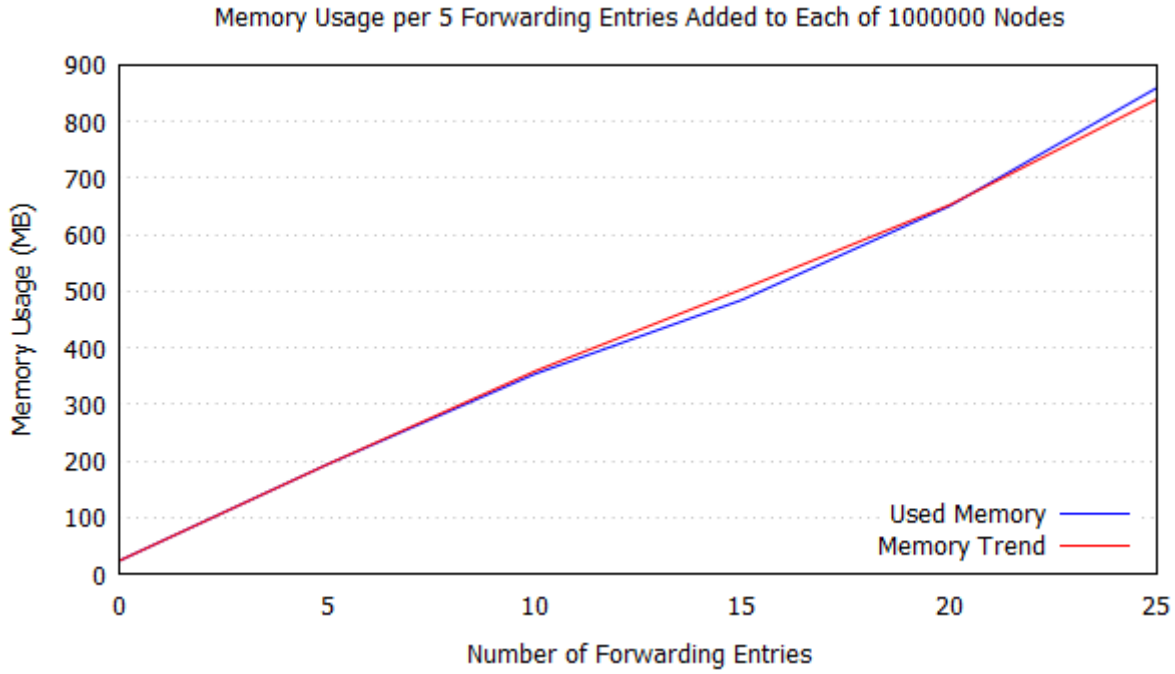


Figure 5.4: Memory Required for 5 Forwarding Table Entries Added to Each of 1000000 Nodes

To derive a formula to predict how much memory is required according to node count and number of forwarding entries per node, one needs to take into account the memory needed per node, and the amount of memory required per forwarding entry added. Using the result of 167.12 MB of memory used per five forwarding entries added to each of one million nodes, one can see that adding a forwarding entry to a node requires roughly 36 bytes of memory in this implementation. This means that the following equations should hold true to any extended predictions:

$$ForwardingEntryRequirement = ForwardingEntryCount \times 35bytes$$

$$NodeRequirement = NodeCount \times 25bytes$$

$$TotalRequirement = ForwardingEntryRequirement + NodeRequirement$$

To test this formula's accuracy one should consider the known results from the tests performed in this section. We know that there are twenty five million forwarding entries and that there are one million nodes in in the final iteration of testing. Substituting these values into $NodeCount$ and $ForwardingEntryCount$ yields the results of 858.306 MB

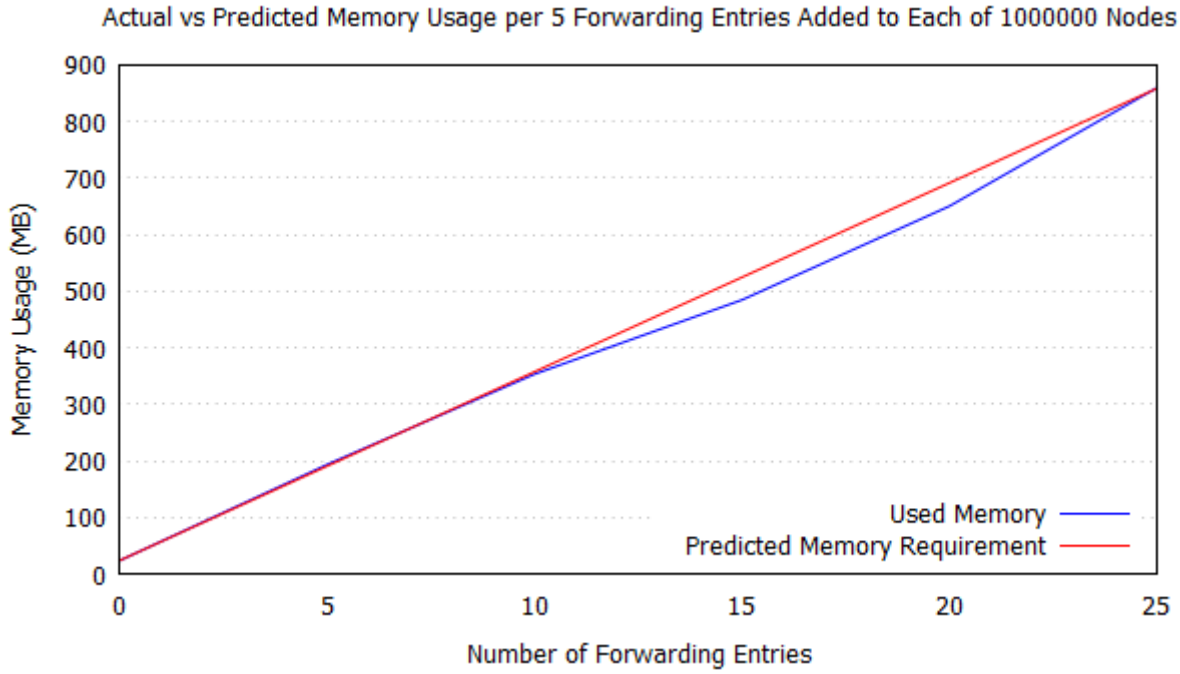


Figure 5.5: Predicted Memory Usage Versus Actual Memory Usage of Routing Simulator

Table 5.3: Node Count Growth of the Internet After ISC 2013

Date	Node Count	Node Growth Count
July 2008	570,937,778	-
July 2009	681,064,561	110,126,783
July 2010	768,913,036	87,848,475
July 2011	849,869,781	80,956,745
July 2012	908,585,739	58,715,958
July 2013	996,230,757	87,645,018

of memory required. This value is just over 1 MB more than what is actually required by this routing simulator, and given the number of nodes and entries in consideration, this is an accurate estimation of the amount of memory required by this routing simulator. Figure 5.5 shows the results of this formula against the results acquired for Figure 5.4.

Table 5.3 shows the number of nodes found annually in the Internet from July 2008 to July 2013 (Internet Systems Consortium, 2013). The average rate of growth over the past five node tallies from July 2008 show an average growth rate of 85,058,595.8 nodes a year. This is a fairly sound statistic as the growth from July 2008 to July 2009 fills in the lack of growth from July 2011 to July 2012.

With this average growth one can use the total memory requirement prediction formula

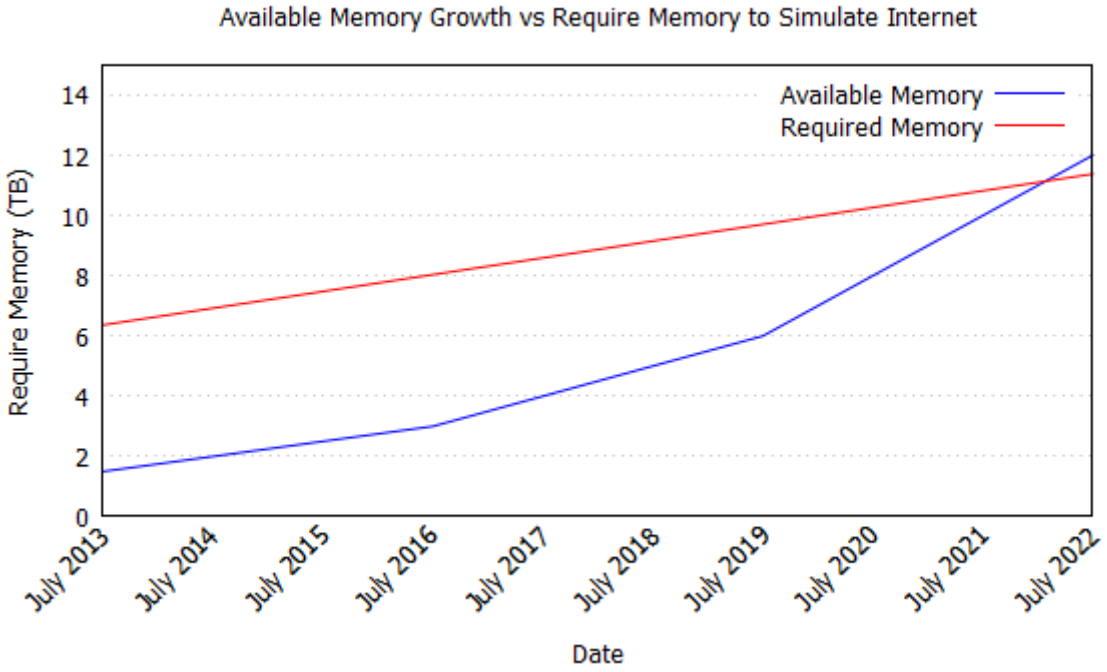


Figure 5.6: Growth of Memory to Simulate the Internet versus Growth of Available Memory

to project the memory needed in hardware to simulate the Internet at a given period of time. This combined with Moore's Law (Moore, 1965) means that one can conjecture that available memory quadruples every three years. With the existing hardware available[4], one currently has 1.5 terabytes of memory available to one's self [5].

Using the memory available for use with the predicted growth rate of memory using Moore's Law, and assuming an average forwarding entry count of two hundred forwarding entries per node, one can predict the amount of memory one will have available against the amount of memory required by a simulated network of size in the current year in consideration; this prediction is shown in Figure 5.6. First it is notable that when using these predictors to model the two growth rates against each other, they will in fact intersect. This means that in the future, we will have enough memory to fully simulate the Internet in a single host.

The second notable point is that the date when enough memory will be available is predicted to be around end 2021. This kind of simulation will not be achievable on commodity hardware at this date, but the day on which this scale of simulation becomes available to

[4] As of mid 2013.
[5] http://ark.intel.com/products/61022/Intel-Server-Board-S4600LH2

the general public should not follow to long after this date, that is if trends continue as is.

Another point that must be raised is that this is an IPv4 network simulator and that it can be extended into IPv6 space. With the introduction of IPv6's vast address space, this being 2^{96} times larger than that of IPv4, one can expect a spike in the growth of the Internet. This growth spike will serve to throw off this prediction if it becomes a reality, thus disproving Figure 5.6.

One should also realise that the prediction represented by Figure 5.6 considers the sum total of actively reachable nodes made available to public interaction. This statistic also includes the endpoints of the network. This serves as a mitigating factor for this simulator in terms of memory requirement growth as this simulator does not consider endpoints as these are externally bound into the simulated network. Instead, this simulator only needs to be concerned about the core of the network, that being non-endpoint nodes that provide the routing of the packet from endpoint to endpoint. This effectively reduces the full 32-bit address space of IPv4 to a 24-bit address space as most endpoints are allocated an IP in the last 8-bits of the 32-bit IPv4 address (RIPE NCC, 2014). It is highly uncommon for an endpoint to be allocated an IP in a subnet that allows more than 8 bits of addressing.

5.3 Throughput

The fundamental function of a successful network routing simulator is the ability to route packets through the simulated network. Once this feature is functioning in a sane manner, the next step is to route packets at a large enough volume to make it viable for large scale routing simulation. This set of testing aims to see what volume of packets can be routed simultaneously, in other words, to find the maximum throughput this routing simulator can support. Referring back to Section 1.2, the throughput this implementation aimed to achieve was 1 Gbps.

This set of testing was performed through a forty 48-switch with each port allowing for a 1 Gbps negotiation and transfer rate[6]. To this switch a 10 Gbps expansion card was added to allow 10 Gbps of traffic to the simulation host. Twenty physical hosts were then

[6]Network negotiation is different to network transfer. Negotiation refers to the rate in which the network controller can synchronize at with the network, where as transfer rate refers to the actual throughput of data that the network controller can actually saturate the medium with.

connected to 1 Gbps interfaces on the switch, each host supported a 1 Gbps negotiation and transfer rate. The simulation host was then attached to the 10 Gbps expansion card on the switch to allow it to reach its full negotiation and transfer rate of 10 Gbps.

The twenty hosts were then bound into the simulated network and paired in a server-to-client set up to form ten pairs. The procedure that followed was a simple one gigabit transmission from client to server over UDP which should take a second to process on a gigabit link. The server simply listens for traffic from its paired client and counts the total number of bytes received. This testing method synchronizes all clients to wall time (The Jargon File, 2013) and thus all clients would send their test data to their respective paired server in parallel. This means that the total network load reaching the routing simulator would be at 10 Gbps for the second long test.

From this point it is a matter of re-iterating the test multiple times followed by adding up the total data received collectively for each iteration of testing performed. The total then gets compared to the known expected total of ten gigabits. From here one can see how many packets got routed successfully. This will show the total throughput rate of this routing simulator.

These tests were performed with the routing simulator software to schedule to a single core on the routing simulator host. These tests were then repeated to schedule for all four cores on the routing simulator host. This would give an indication of where a bottleneck occurs in the event that the routing simulator host does not saturate the 10 Gbps transfer rate.

5.3.1 Results - Single Core

Figure 5.7 displays the results of the single core scheduled throughput rate of this routing simulator. These results were generated over 71 iterations of the 10 Gbps transfer test. The results showed a global maximum of 5.813 Gbps, and a global minimum of 4.737 Gbps. The average throughput achieved by this routing simulator scheduling for a single core is 5.253 Gbps which exceeds results collected from pre-optimization test iterations (Herbert and Irwin, 2013).

This average throughput produced by the testing shown in Figure 5.7 is a fairly sizeable result on a single core, though it is clear from these results that the routing simulator is reaching its limits on a resource. To attempt to find this resource, we introduce all four available cores for scheduling in Figure 5.8.

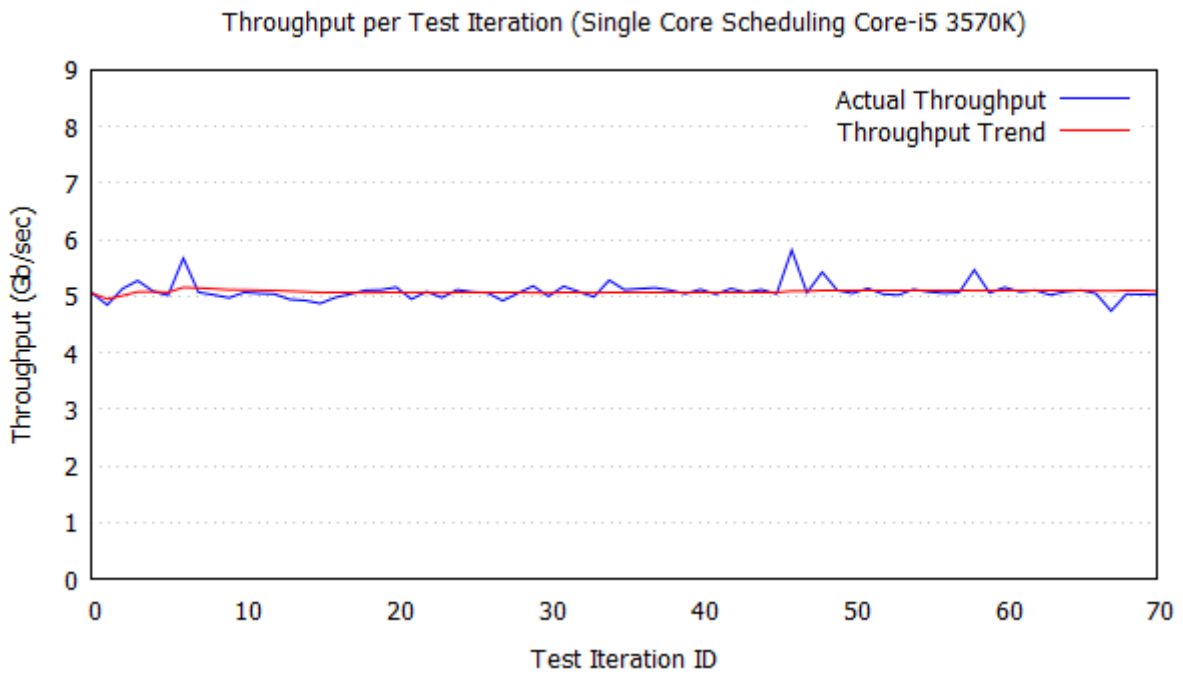


Figure 5.7: Average Throughput over Multiple Iterations Scheduling for One Core

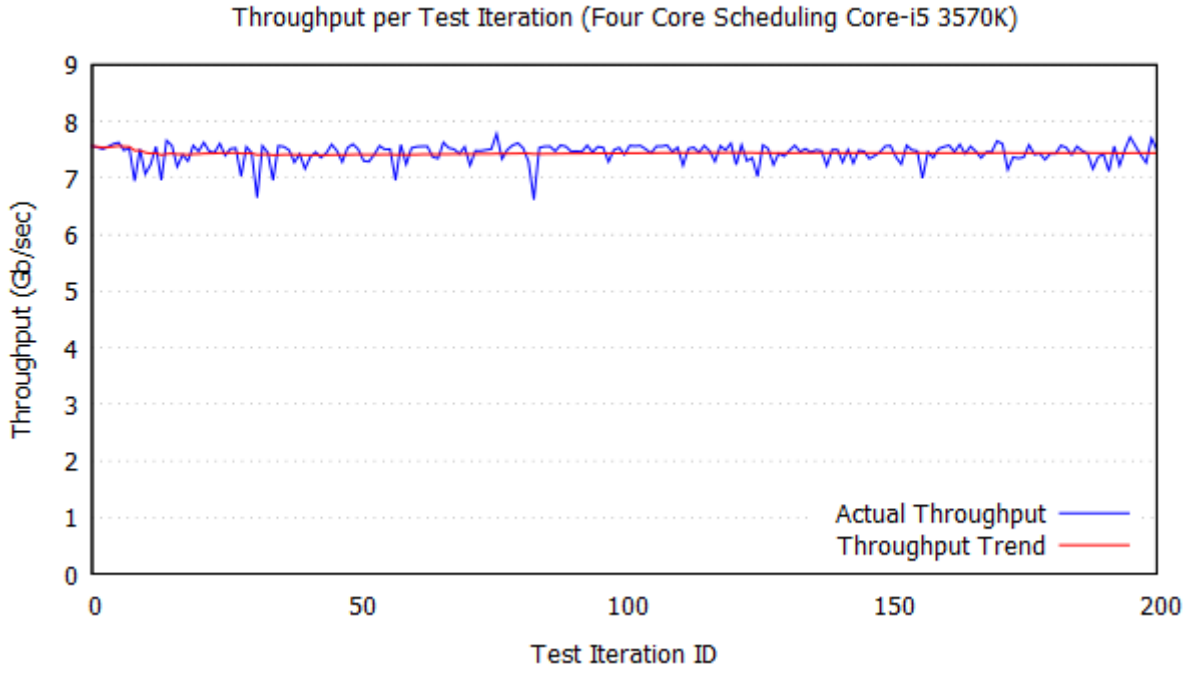


Figure 5.8: Average Throughput over Multiple Iterations Scheduling for Four Core

5.3.2 Results - Four Cores

In Figure 5.8 one can observe the results of the tests performed in Figure 5.7 when scheduled for four cores as explained in Section 4.3. This shows an overall improvement in performance, leading to the conclusion that the tests in Figure 5.7 were in fact bottlenecking because not enough CPU resources were available. The global maximum of the test results shown in Figure 5.8 is 7.782 Gbps, and a global minimum of 7.223 Gbps. The average throughput over the two hundred and one iterations of testing is 7.443 Gbps.

To extend on identifying the system bottleneck, one needs to take into consideration the speed-up that can be yielded form a multi-core approach to this problem. Although Amdahl's Law can lead one to believe that the order of magnitude speed up of this multi-cored implementation is in fact in line with the expected speed up when compared to a single-core approach (Amdahl, 2007), one should first consider the part of this implementation that can be run in parallel.

Both reads and writes of this implementation, although most operations at simulation time are reads, are all performed in parallel. This is because the reads to the dataset are performed as accesses to routing tables, and writes are evenly distributed in a disjoint manner across all cores. This means that the part of this implementation that can be run in parallel lies above 95%. According to Amdahl's Law, this means that the expected speed up should top out at a minimum of a 20 times, regardless of whether the cores assigned to this task exceed 20 in count.

5.3.3 Throughput Summary

With this in mind, one can trivially conclude that a four-core approach to this simulation should provide a fairly significant speed up to this task; one that exceeds that of a single-core approach at least threefold. In comparing the average throughput of this four-core approach to that of the single-core one, the factor speed up is 1.571. This factor serves to point towards the CPU resource no longer being the bottle neck in this simulation.

The CPU usage during this second phase of testing averaged 65% to 70% under periods of load. This leads to out the conclusion that the test results depicted in Figure 5.8 are not bottlenecked by CPU resources but rather by a supporting feature of the CPU. This could either be the load on the system bus, a limitation within the network interface or read/write speed of the memory on the host system.

With these results, it is fairly safe to draw the conclusion that with a 7.443 Gbps average throughput rate, this routing simulator easily exceeds the goal of 1 Gbps set in the problem statement, Section 1.2.

5.4 Delay Accuracy Loss Over Multiple Hops

Accurate hop delay is one of the key features of this routing simulator; as such one needs to consider the time taken to process a hop before scheduling and how much time this adds onto the overall delay in a hop. This added delay is due to the time between spawning a thread for the job from the work queue, processing where the next hop is located if the destination is not reached, and then queueing the packet on the scheduler again. This processing delay in the routing simulator is not accounted for and so can add unforeseen delays at runtime even though a correct network configuration was supplied.

This test was done through the use of a real life example and iterated seventeen times; two were dud runs due to network path change at run time. First the output of a traceroute iteration was converted into a configuration script for this routing simulator using the tool described in Section 3.10. The delay results for each hop from the traceroute output were averaged and then used for the delays of each hop. This means that the real delays are known and used as a control in this test.

A host was then bound into this simulated network by the IP of the original host IP that performed the traceroute test. This allowed the host to perform the same traceroute test as if it were the original host performing the traceroute on a real network; the only difference being that now the traceroute is performed through the simulated network instead. This test was repeated twenty times and the results presented are an averaged value.

The delay results from the traceroute performed through the simulated network were then averaged and compared to the original known delays from the real network traceroute result averages. From here conclusions were drawn as to the delay skew caused through process delay within this routing simulator.

5.4.1 Results

Table 5.4 shows the comparison of a traceroute performed through a physical network as found in the Internet with that of the simulated network which represents the physical

Table 5.4: Real vs Simulated Hop Delay Averages over 15 Test Iterations

Hop Number	Real Time (ms)	Simulation Time (ms)	Difference (ms)
1	1	1	0
2	20	21	1
3	33	33	0
4	33	33	0
5	36	35	-1
6	214	216	2
7	207	207	0
8	217	217	0
9	302	303	1
10	352	353	1
11	380	380	0
12	371	373	2
13	382	382	0
14	359	357	-2
15	366	367	1
16	363	365	2

one. The physical network's results are then subtracted from the simulated network's results to produce the difference in delays.

The delay difference between the real network results and the simulated ones does not vary by more than two milliseconds. This means that delay caused through processing the next hop or whether a destination is reached or not is negligible. Furthermore, this slight skew of the actual delay in the simulation is due to the load on the simulation host and how the scheduler within it is programmed to deal with states such as idle and heavy load scheduling.

The reason for delays getting not only larger, but also smaller as the hop count gets greater is that two packets destined to the same host do not have to take the same path. This means that although Hop 13 was located on the way to Hop 14 in the route, it does not mean that the packet sent to Hop 14 has to take the the route on which Hop 13 exists. This basically means that traceroute cannot actually be trusted to always return the exact route, that is node for node, to the destined IP that is in question.

5.4.2 Delay Summary

The average delay skew over all hop results produced by traceroute is 0.438 milliseconds. When considering that the the delays skew in both directions, that being more and less than that of the real network, and that the delays become as large as 363 milliseconds, 0.438 milliseconds is a negligible result. This delay could arise due to the network simulator's host being under load, waking up from an idle state or even the traceroute iteration doing the testing being under load. In conclusion, the delay skew caused by processing the logic in determining the route of a packet being simulated is negligible and thus has no weighting in the outcome of the packet's total delay to a destination.

5.5 Memory Usage Under Load

Memory during simulation runtime isn't only used by the network configuration, but also by the packets being simulated. This is not a straightforward one-to-one memory use ratio due to metadata being pulled out of a packet when it is received. On top of this, one also has to consider the pointers associated with the memory location, and the memory taken up by the structure that stores the packet[7].

To gain insight into the memory usage during testing, the following test was performed. Using the same test as found in Section 5.3 on throughput, a tool was written that simply checks the available memory of the simulation host every quarter of a second. The same throughput test then occurs, in which ten paired systems transfer one gigabit of data between each other for a second, thus totalling 10 Gbps in order to saturate the 10 Gbps interfaces found on the simulation host.

5.5.1 Results

The results of the actual rate of data transfer are known already from the results of Section 5.3 and can be used to estimate how much memory is required per 1 Gbps of throughput. This test data was produced from the single core test depicted by Figure 5.7. Although this testing is done through use of a single-core, the calculation leads to a conclusion in terms of memory and so results to this section are applicable over all scheduling and hardware configurations.

[7] This struct being sk_buff.

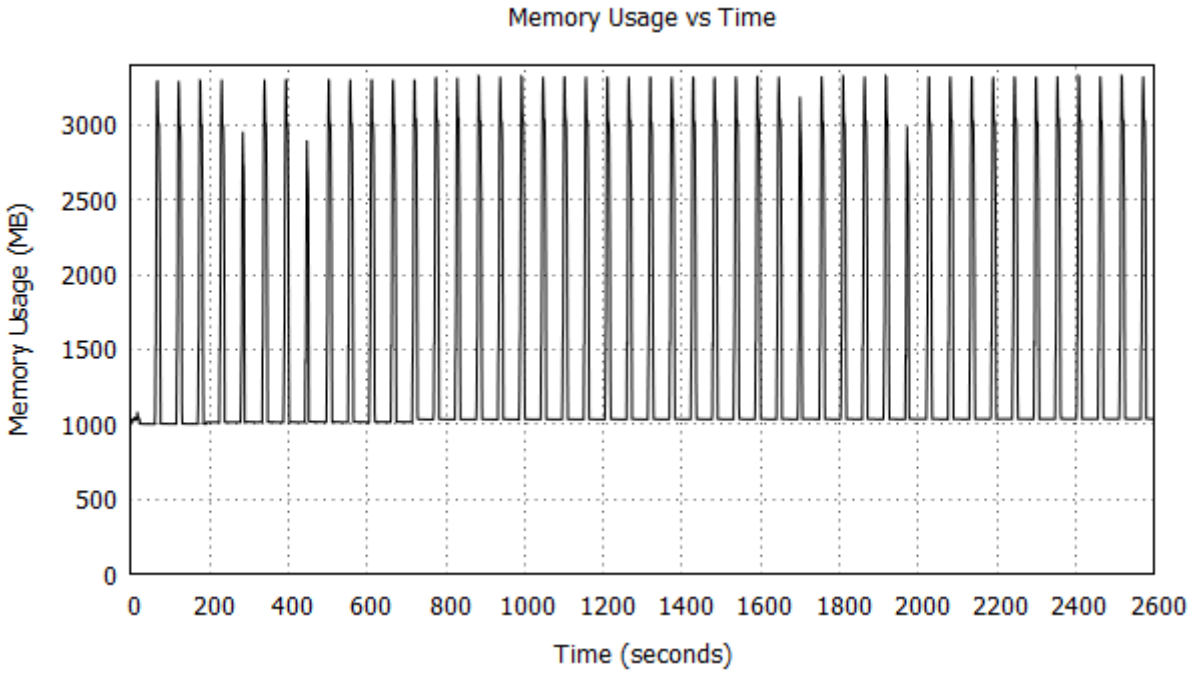


Figure 5.9: Memory Usage During 10 Gbps Transfer Testing

To estimate the amount of memory needed per 1 Gbps of throughput introduced to this routing simulator, one can refer to Figure 5.9. One can observe the tests occurring every minute, which is true to the method in which the throughput testing was performed. This means that the spikes in memory usage are due to the simulation taking on packet load rather than another external cause.

The peak of the spikes are also determined by the external hosts. The packets introduced into the system cause memory to be allocated. If a host transfers ineffectively due to its resources for the task being allocated elsewhere, or due to a operating scheduling delay the task starts slightly later than expected, which would cause memory to be allocated to a lesser degree over a longer period of time. These occurrences can be seen in Figure 5.9 at 0 seconds, 240 seconds, 480 seconds and 1,980 seconds.

The peak memory usage in every test performed is within 20 MB of each other if one omits the four outliers in the data listed above. The average memory usage of these peaks, excluding the outliers, is 3220.882 MB and the usage outside of a test iteration is 1,034.328 MB. This means that the average memory required to hold the packets in transit during a throughput testing iteration is 2,186.554 MB of memory.

5.5.2 Summary

From the results for the single core throughput testing in Section 5.3, an estimation of how much memory is required per 1 Gbps of throughput can be drawn. The average throughput for this method of scheduling was 5.253 Gbps, dividing this number by the 2186.554 MB of memory required leads to the average memory requirement of 416.249 MB of memory per 1 Gbps of throughput. As this set of testing was performed with an average round trip of 200 milliseconds, one can also factor the average round trip time into an estimation, as the duration of the packet within the simulator is the amount of time the memory has to be allocated for the packet. This means that the follow equation applies when predicting the amount of memory required for testing of a known throughput rate.

$$MemoryRequirement = GigabitsPerSecond \times 2.081MB \times AveragePacketRoundTrip$$

Using this with the equation given in Section 5.2 would give a good approximation of the hardware needed by the simulation host in order to successfully simulate the given network.

5.6 Summary

This chapter started by detailing the manner in which tests were performed and what hardware was used to obtain these results. The first objective was to perform a long term test to check whether the system could sustain long up times while servicing the main forms of network communication, this being TCP, UDP and HTTP which runs over TCP. After the consistency of the network simulator was checked in Section 5.1, a look into memory usage, in terms of RAM, was tested next in Section 5.2.

In Section 5.3, the maximum throughput of this network simulator was tested to further obtain results to conclude whether this network simulator was successful. Finally the added delay to a packet being processed on the CPU was analysed and the extra memory required by the packets within the simulated network was shown in Sections 5.4 and 5.5 respectively.

In Chapter 7 these results are weighed up against each other and real life network examples. From this point, a conclusion is drawn as to whether this routing simulator is

suitable to real life application, or use as an alternate for network simulation. Finally, ideas for future work and application are detailed and whether or not this research was a success is ultimately decided, with respects to scope of this research.

The following chapter serves to round out this research by taking the results produced in Chapter 5 and using them as a guideline to apply this implementation to real world scenarios. These real world scenarios are discussed in Chapter 6 where it presents this research in environments that can benefit from the ability to prototype, test and aid a process that is due for real world application, or mimic real world processes, in the case of education.

6

Use Cases

This chapter explains five major areas in which this simulator could be used and provides reasoning as to why one would want to use such a tool as this. Furthermore, real world application examples are provided, and more importantly examples of where this routing simulator can be placed within a simulation environment to improve the effectiveness of it.

Sections 6.1 and 6.5 take a look into how students and staff can be better prepared for the situations that may arise in industry through proper training. Not only is this simulator a cost effective replacement to hardware, but ease of use allows for it to better educate through pseudo-live environments that one may find in a physical setting. These are further detailed in the following sections.

The description of uses of this simulator begins with a educational approach in Section 6.1. This takes a look into how this simulator can be used to educate students by allowing access to resources that an educational institute may not possess. Section 6.2 takes a look at how one can defend a network from a network design point of view and how it can be used as a defensive mechanism through simulation of a pseudo-network. Section 6.3

Table 6.1: Router vs Simulation Costs

Accounted Hardware Require	Total Cost (USD)*
1 Router	24.99
10 Routers	249.90
100 Routers	2499.00
System Used in Testing	1170.35

* based on NETGEAR WNR1000 from Newegg.com

takes a look at the other end of the spectrum, this being offensive, and shows how this simulator can be used for both internal and external attack planning.

Discussion of the application of this routing simulator is then extended into the context of the development of infrastructure in Section 6.4. Use of this routing simulator is described within the practices of power grid control and wireless communications. Finally the benefits which the corporate sector can gain from this simulator is discussed in Section 6.5.

6.1 Education

Network simulators in an educational context can be invaluable as network education tools. Acquiring the equipment necessary to represent an office, a building, or even a section of the Internet can be a costly endeavour for any education institute. Furthermore, the time spent receiving and configuring the equipment can add to the lead time before the course is available to teach a group that is interested in the knowledge area. To add to the costs, one still has to power and run these systems.

6.1.1 Cost Mitigation

Use of this network simulator can replace the need to purchase such a costly system. (Such costs can be compared in Table 6.1 while keeping in mind the capabilities of this routing simulator.) The ease of configuration of this network simulator also mitigates the lead time required for any course planning. This simulator is also easily scalable and as such, both small scale and large scale networks are easy to configure and reproduce within the required scope.

Table 6.2: Time from Cold Start to Able to Route Network Traffic

Test Case	Time to Able to Route (Seconds)
Netgear DGN2000	21.110
D-Link Wireless N 300	68.803
Billion BiPAC 7300W	21.533
Single Simulation Node	$\ll$0.001
1,000,000 Simulation Nodes	1.027

Access to this kind of tool can allow for better education within a multitude of system environments. This allows for graduates better prepared to enter the work space as they will be able to complete tasks given to them with less system specific training necessary from the groups that require their skills. This leads to faster system development, better maintenance and faster error recovery within a system.

Finally, from an educational point of view, more content can be covered in a course due to the ease of configuration of this simulator. This further enables a student to prepare themselves for the area to which they may be employed one day.

6.1.2 Isolated Environments

This network simulator also provides isolation from physical networks. This means that the environment in which one is educated in provides more freedom to test the system to its limits; if one were to break a component of the system, the simulated environment would be easily restarted with little down time when compared to restarting a physical network. The typical boot times for some devices can be found in Table 6.2.

Practical testing of students within a course can be better assessed through individual simulated networks. An educator now has the ability to create a network for each student in the course in order to test them; something that is not easy to achieve through a physical means. As this environment is also isolated, one cannot affect another's network which means that a student cannot prevent other students from answering their test due to system malfunction or malicious intent.

6.1.3 Common Tools in Education

There is a wide array of tools that exist for network traffic creation, recording and analysis; these are usually made available for download from the respective authors of the

software. There are, however, tools that are installed by default on common OSs such as Microsoft Windows or the multiple distributions of Linux. The tools referred to are that of nmap, tracert/traceroute, nslookup and ping. These tools are often used to provide insight into what ports a host has open, the route a packet takes to arrive at a destination, whether a host exists on a network and what IP it can be addressed at.

As this routing simulator routes traffic and simulates the internal nodes of the network as to their real world counterparts, it allows for results yielded from these applications used within the simulated environment to reflect exactly as if they were used within the physical network that the simulated one represents. For this reason, one can continue to use the aforementioned tools in the same manner in a simulated network as in the physical one that the institute had in place before. Furthermore, this allows for institutes that could not afford large scale networks for purpose of education to represent a larger network, or one that is specific for a case scenario for specific needs training.

6.2 Defensive

Communication is fundamental to information sharing. When an assault occurs on a group of people, the ability to work together through means of successful information relaying can prove invaluable in any situation (Sun, 2013). In short, successful and timely communication allows a group of individual parts to interact to achieve a greater potential than that of each part alone (Koffka, 1935); an idea conceived by Aristotle (although never published in his works) but still relevant today.

6.2.1 Network Communications

Networks within a system are used as a fundamental communication channel and protection of these communication lines is required in order to ensure information can be relayed without falter. With this in mind one can use this network simulation to test the fundamentals of a systems network, or create defensive characteristics for a network in order to mitigate a future attack on the system.

To use network simulation for testing a network's defence, one can simply simulate the system's network that needs defending. From here one can look into several aspects that ensure a healthy network. First and foremost, one should have access to all major nodes

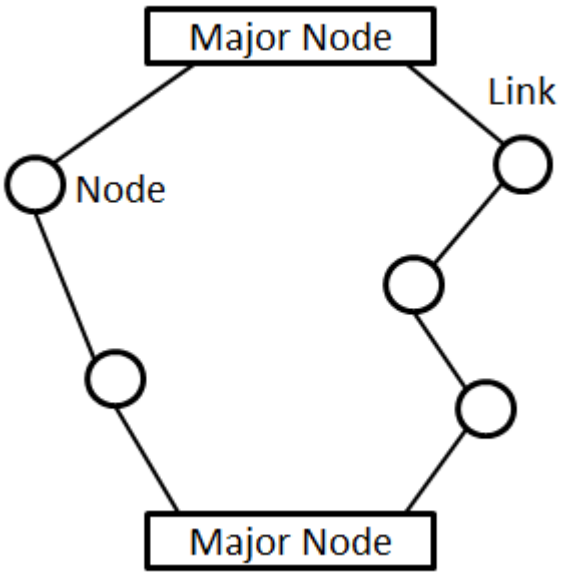


Figure 6.1: Example of a Redundant Link Between Two Major Network Nodes

of a network at any point, and in the worst case, all major systems should be able to communicate. With this in focus one can look into preventing isolation of a network if certain links or nodes happen to go offline. This can be done through removal of links within this routing simulator and observation of traffic flow. One can then gain insight into where redundant links need to be created or fail safe points need to be implemented within a network.

Figure 6.1 shows a simple redundant link between two major nodes. Another point is that if any one link is broken or if any one node is taken offline within this network, all other nodes can still communicate with each other. More importantly, the two major nodes can still communicate with each other; this is basic network design.

Communication round trip times also need to be considered within a system; data needs to arrive in a timely manner in order for the information it carries to be current. This should also be taken into account when a network is restructured to be more enduring. Monitoring of the type of data passing through nodes and links can be used to create better firewall rules and better traffic prioritizing on a network, which will increase major node bandwidth and reduce communication times. Tools like Wireshark and Tcpdump can aid in collection of this data. Analysis of the collected data is dependant on the network and network architects, who decide what needs to be prioritized, what needs to be monitored and what is considered a threat.

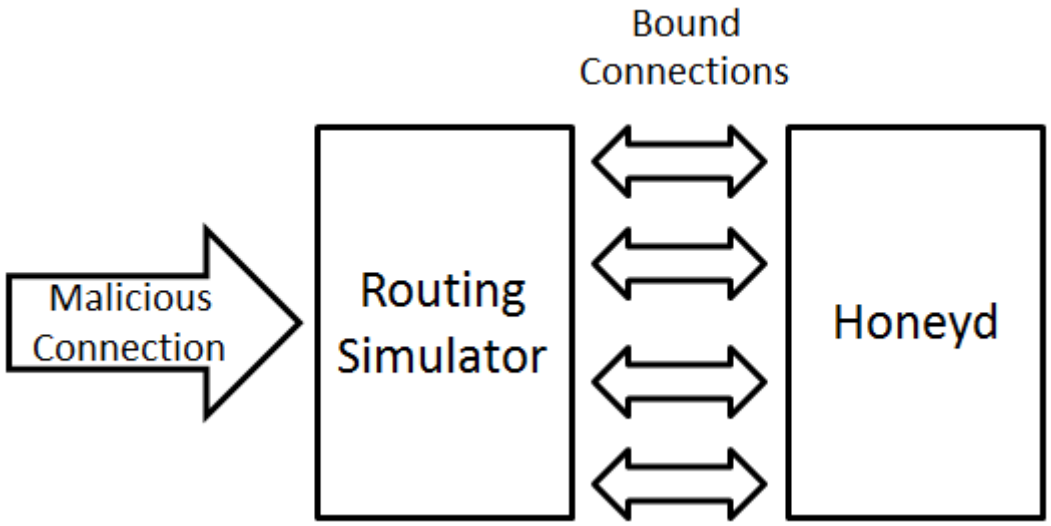


Figure 6.2: Binding This Routing Simulator to Honeyd

6.2.2 Honeypots

One can also apply this routing simulator as a sub-system of a real network to act as a diversion for malicious traffic. When inbound traffic is deemed irregular or malicious, the overlying system can redirect the traffic to this routing simulator within the targeted system. From the attacker's point of view, it would seem as if they have penetrated the network and have access to resources which may appear valuable to the attacker. In reality this attack can be monitored and either left to run its course, or acted against.

Honey pots are a good example of this approach to system defence, as introduced in Section 2.11. This implementation can easily take on the roll of a honey pot through simulation of a network with hosts in the form of VMs acting as endpoints. An example of an application to use as these VMs is Honeyd. Honeyd is a daemon that creates a set of virtual hosts within a network. These hosts can then be configured to mimic services that are commonly found on a network and that an attacker may be targeting (Provos, 2008). Combining Honeyd and this routing simulator, one can create a virtual network that can act as an attractive entity for an attacker, thus mitigating damages to the actual targeted network.

Listing 6.1 is a sample configuration file used for configuring this routing simulator. This configuration binds all traffic to interface eth3 within the simulator host. The simulated nodes all exist on the 106.77.109.0/24 subnet and all IPs used by Honeyd lie on the 10.42.0.0/24 subnet. The configuration matches traffic that arrives at endpoints found on

Listing 6.1: Simulator Binding Configuration

```
1  eth3  106.77.109.8   10.42.0.85  00:22:4d:56:32:3f  0 0 0 0 0
2  eth3  106.77.109.11  10.42.0.47  00:22:4d:56:3a:cc  0 0 0 0 0
3  eth3  106.77.109.15  10.42.0.29  00:22:4d:56:11:08  0 0 0 0 0
4  eth3  106.77.109.18  10.42.0.75  00:22:4d:56:3a:e8  0 0 0 0 0
5  eth3  106.77.109.20  10.42.0.25  00:22:4d:56:37:3a  0 0 0 0 0
6  eth3  106.77.109.21  10.42.0.33  00:22:4d:56:38:a8  0 0 0 0 0
7  eth3  106.77.109.24  10.42.0.86  00:22:4d:56:38:dd  0 0 0 0 0
8  eth3  106.77.109.30  10.42.0.43  00:22:4d:56:3a:6d  0 0 0 0 0
9  eth3  106.77.109.31  10.42.0.23  00:22:4d:56:3a:50  0 0 0 0 0
10 eth3  106.77.109.32  10.42.0.31  00:22:4d:56:3a:be  0 0 0 0 0
11 eth3  106.77.109.35  10.42.0.76  00:22:4d:56:23:fd  0 0 0 0 0
```

the 106.77.109.0/24 subnet and emits the relevant traffic to the IPs held by the respective Honeyd daemon. One can see an overview of this concept depicted in Figure 6.2.

6.2.3 Malware Analysis

One can also use this routing simulator to analyse contagions within a network; these contagions are commonly known as viruses or malware. To elaborate on this idea (and referring to the cycle depicted by Figure 6.3), consider the case of a virus being unleashed onto a simulated network. One can log data about the spread of this virus through means of actions performed by users within the network, common applications present on the infected system, communication that occurred pre- and post-infection and systems that were initially targeted.

Using this kind of logged data, one can then form a point from which to analyse the contagion. One can analyse patterns and develop better firewall rules with which to prevent spread of the infection. Furthermore, one can form means of better detecting whether a system is infected or not through analyses of known symptoms that are present within the network or system (Barthélemy *et al.*, 2004). Use of this simulator in this regard can aid in better virus signature database development and prevention of malware attack.

This simulator can also be used to model cyber-warfare between malware and security systems (Kotenko, 2005). Use of this routing simulator can help model environments and in these models re-enact both malicious attacks (such as DDoS or a security breach) and attack prevention mechanisms in order to produce data for later analysis as they run

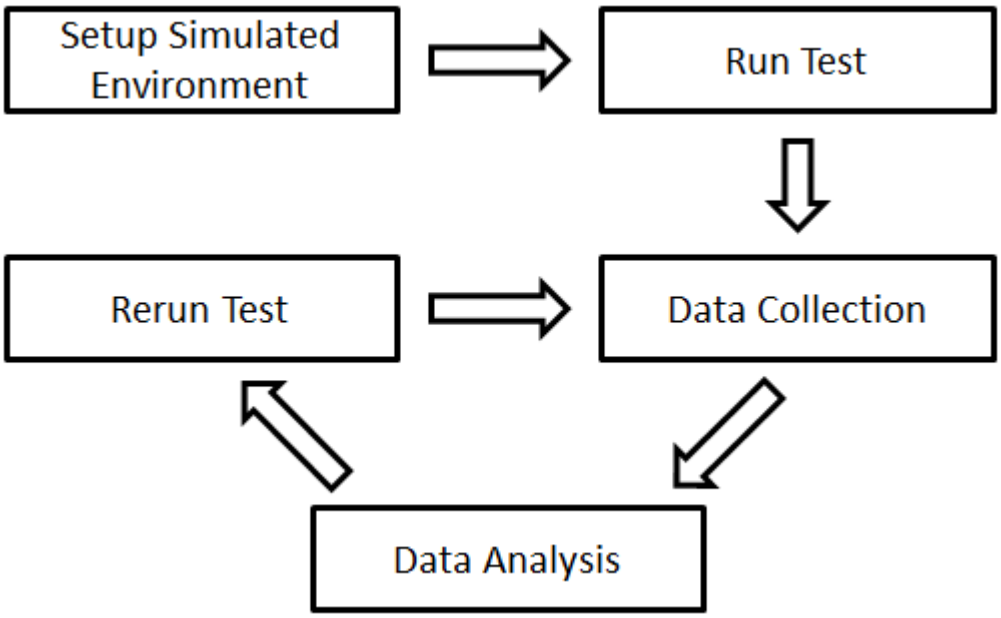


Figure 6.3: Malware Simulation and Analysis Cycle

against each other within the environment. Again, once the data is analysed, security systems can benefit through better development with this safe and reusable source of data collection.

6.3 Offensive

Defensive measures are only one side of the battle when it comes to an attack on a system; one needs to consider the use of this network simulator from a offensive point of view too. The use of this simulator as a tool to aid in the assault of a system can be performed from an external or internal standpoint.

From an external standpoint, one can simulate links that provide inbound and outbound connections. From here one can look towards targeting nodes and connections that are required to ensure successful communication. With this in mind, one should not aim for target links to achieve the task at hand without considering the systems that could be caught in the crossfire. Instead one should look towards bringing down links and nodes that would have the least effect on systems not involved in the targeted system in order to reduce the chance of detection.

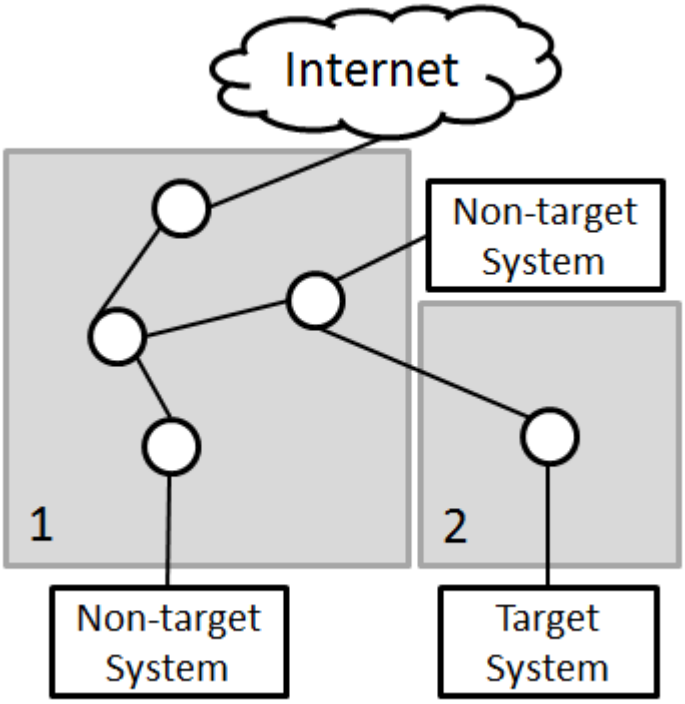


Figure 6.4: Target System External Links that Cause Least Collateral Damage

6.3.1 Attack Preparation

Figure 6.4 depicts a subset of a part of the Internet. This subset is then divided into two areas. In order to prevent collateral damage occurring during an attack of the Target System, one should aim to bring down either of the links or the node within area 2. If one were to disconnect any links or bring down a node in area 1, this would cause the Non-target Systems to become isolated from the Internet, and thus they will no longer be able to perform their tasks on the Internet.

For prevention of this, a simulated attack can be a very useful tool. Using this routing simulator one can better analyse (through multiple iterations of testing) which links would cause the least collateral effect on surrounding systems. Once the best result can be achieved for the planned attack, one can then go about executing it with a greater chance of success as the procedure of the attack would have been simulated before hand.

If one is fortunate enough to know the topology of the network which one wishes to attack, one can benefit through simulation of the network. One could either wish to collect valuable information, or one could wish to bring the system to a halt. To bring a system offline, one can take an approach similar to that of the aforementioned external attack.

One can find what nodes and links are crucial to the systems execution and plan ahead for the process through multiple iterations of such an attack.

In the case of information collection or data alteration[1] one could use this simulator to create the topology of the network. One could then go further to link in physical hosts to act as hosts within the system that one wishes to attack, the hosts containing the targeted data with this routing simulator.

6.4 Infrastructure

Many day-to-day systems run on complex networks within buildings, cities and even countries. These systems require current and reliable communications to ensure that the system as a whole works as expected. Some of these systems are listed below:

- Cellphone networks.
- Large scale networks.
- Power controls.
- Cables.
- Traffic light controls.
- Water system controls.

6.4.1 Smart Grid

Smart Grid is an implementation of a power system that monitors sources and sinks of electricity and aims to provide enough electricity for the demand of all electronic devices on the grid. Furthermore, this Smart Grid is a network of millions of nodes that use advanced communication protocols developed for secure and reliable communication (Wang and Lu, 2013). If such a system suffers from either unreliable requests through malformed communication or malicious attacks due to attacks on the system, it can have widespread affect on the users of the power grid that it services.

[1] Such as including ones self on a payroll system, or modifying values of a nuclear testing calibration.

Similarly to a power grid, the water system of a city, country or neighbouring countries is also subject to supply and demand characteristics. In this case, supply comes from both the pump stations producing the water and the consumer as they produces waste water (Ormsbee and Lansey, 1994). These systems and the systems within each station are often controlled remotely as active feedback is fed to the supplying infrastructure. Again, like Smart Grid, these systems require reliable and secure data so as to ensure correct responses to the demands of the water system that is in question.

Both power and water systems rely on network communications and specialized protocols in order to provide the relevant information required for correct response by each respective system. Thus there is a need for development and continued maintenance of such a system in order to prevent attacks that could otherwise harm devices connected to these systems, or prevent malformed data from producing system responses that could prove harmful. In the event of an unforeseen event, these systems need a reliable means by which to mitigate any problems that occur. It is in this development that this routing simulator can be used to provide a means in which test such protocols and systems in development.

Such simulation can be achieved by using this implementation to replace the underlying communications medium used to relay data within these systems. Real endpoint hardware that is used to provide this service can still serve as endpoints and can be included into the simulation through connection to this routing simulation via its real world integration capabilities. From here, development can occur in an isolated environment as if the connected hardware is attached to the real infrastructure over a pre-determined distance made available through delay simulation. Other factors like lossy connections and bad links can also be simulated through this routing simulator, thus allowing for the developing system to be further prepared for events that may occur before actually facing them in physical implementation.

6.4.2 Wired and Wireless Communications

Other systems, such as networks provided by telephone or cellphone companies, also rely on a infrastructure in order to provide their services. Telephone companies in the past relied on copper cables as their communications carrier in order to provide their service from point-to-point. At the time of writing, telephone companies are making a move towards fibre optic solutions that rely on digital communications rather than the analogue

signals provided over copper that are often subject to noise. These fibre optic replacements up the available voice channels for telephone communication from 24 connections, to more than 32,000 connections on a single fibre link (The Fiber Optic Association, Inc., 2005). Application of this network simulator in a telephone company's context would be in line with Section 6.2, with focus on throughput and major node connections through redundancy.

Cellphone companies require tall towers to allow for optimal distribution of their wireless network. These towers are often placed strategically to allow for maximum bandwidth while ensure good coverage. This routing simulator would prove useful in this regard when applying the jitter, packet loss and packet mangling features provided by it. These features would allow for statistical retry and drop rates to be provided between entities and thus quality of service testing can be applied within a proposed cellphone network.

6.5 Commercial Training

Companies exist on the Internet that provide services that are available through the Cloud. These services need to act interact in a non-malicious manner not only with systems within the internal networks which the company owns, but also with other services provided by other companies. For this one should consider simulation of the product in a private context before it becomes available for public consumption.

6.5.1 System Prototyping

This simulator can be used to provide an interface to a pseudo-public environment that can allow for testing of the system. Furthermore, other physical systems that the new system requires to interact with can be introduced at a later stage through physical host bindings into the simulator. This allows for security of a system to be bolstered as a developer can further ensure the system acts as one expects it to.

This simulator can also provide a means in which maintenance can be provided. Any updates or modifications to the working parts of a system can be tested through the means of a pseudo-live simulated environment. If one were to develop for a VoIP software company and were to attempt optimization of the underlying custom protocol in which

the software transfers its audio, such as Skype, Teamspeak[2] or RaidCall[3], one would rather not have to roll out the modification to all active clients and wait for any errors to occur. Instead, one can set up the simulated environment and pretest the updated protocol within an offline environment which does not involve the clients in testing of this updated protocol. This allows for a better product to be available for access at the time a service goes public.

6.5.2 Training

Such an approach also allows developers to more easily come to grips with a system, as breaking the system in a simulated environment would not yield any problems to the real-time service provided to the public domain. This same quality allows a developer to more easily try a new approach to a solution, or modify parts of a system without the concern of deteriorating a user's experience in the public domain.

Lastly, having a simulated network with the major services the company provides in the Cloud can have benefits among the staff of the company. Many companies have to train new staff that enter the company's work force to develop and maintain these Cloud services. With a simulation in place running the services provided by the company, a company can better prepare their new staff through means of demonstration and pseudo-maintenance within this environment.

6.6 Summary

This chapter shows that this routing simulator implementation extends past a research endeavour and can further extend itself to serve as a tool in defence, offence, infrastructure development, education and commercial sectors. It was initially shown to serve as a useful simulation environment for network design through means of simulation of link and node removal. This was then taken further to use this simulator as a honey pot in order to divert a potential attacker away from one's valuable information.

An offensive approach was also considered as a contrast to defence; simulation was also shown to be a key tool in effectively preparing for an attack. Furthermore, it was shown

[2]http://www.teamspeak.com/
[3]http://www.raidcall.com/index.html

that simulation can help prevent collateral damage that may be caused to other systems on a network.

The use of this routing simulator was shown in an infrastructural context in Section 6.4 in terms of power and water system development. The application of this implementation's features were then shown when considering a wireless network such as one a cellphone company may wish to instantiate.

Education and commerce are then shown to achieve positive results through simulation in sections 6.1 and 6.5. Overall, simulation can help better education through ease of set up and configuration, all while keeping costs to a minimum (as networking hardware can prove costly in large quantities). Commerce can also benefit from simulation. Use of this simulator helps commercial endeavours (both in development and staff training for maintenance).

7

Conclusion

This research sought to implement a routing simulator and through testing conclude whether this implementation achieved its purpose effectively. This routing simulator was aimed at TCP, UDP and ICMP protocols with the objective of achieving at least 1 Gbps of throughput on any, or a combination, of these protocols.

This chapter aims to summarise major testing of this routing simulator in Sections 7.1. The key points referring to the compatibility, scalability and future performance of this implementation are addressed in Section 7.3.

Section 7.2 brings a close to this research through contrasting the goals achieved by this implementation in Section 7.1 and compatibility detailed in Section 7.3 with the goals and timeline set out for this research in Section 1.2. Finally this document sees its closing with ideas for future extensions to this research and research area in Section 7.4.

7.1 Key Aspects

The goals of this research initially set out in Section 1.2 were as follows:

1. An attempt to enable simulation of reliable IPv4 based routing in a software defined environment and the ability to interface with physical hosts outside of the simulation host through this simulated environment.

2. The ability to route major protocols that rely on IPv4, these being TCP, UDP and ICMP.

3. Creation of realistic simulated characteristics of physical events through introduction of delay, network jitter, packet loss and packet mangling.

4. Finally, optimization of this simulator to achieve a speed of at least 1 Gbps of throughput on commodity hardware.

In essence, this research aimed to achieve a considerable packet throughput, all while keeping the requirements on CPU and memory usage low enough to allow for deployment on commodity hardware. With this in mind, we look towards drawing conclusions about this research based on the results gathered in Chapter 5.

7.1.1 Consistency and Protocol Support

Section 5.1 evaluated whether this routing simulator produces consistent results without discrimination between protocols. This was dealt with through testing three protocols, these being HTTP[1], TCP and UDP. TCP and UDP had similar request time averages with TCP taking slightly longer due to its three-way handshake[2] and larger header, thus generating more packets than UDP for the given transfer. HTTP showed long connections due to the basis on which HTTP requests are served (this was explained in Section 5.1).

Each protocol proved to pass each test iteration consistently over the 24 hour testing period. More than 54,900 test iterations were performed for each of the three protocols in this time period, which further steadies the results acquired in these tests. These results also behaved as expected over the three tested protocols: this being that HTTP took the longest, whereas UDP was the quickest due to the nature of each protocol's behaviour. Given that this implementation involved receiving, routing and sending traffic produced by real physical hosts serves to further re-enforce the legitimacy of the support of HTTP, TCP and UDP.

[1] HTTP relies on TCP.
[2] standard SYN, SYN/ACK, ACK.

Considering the request distribution (as depicted in Figure 5.3) and the single failure in more than 160,000 requests across all protocols that were produced by the physical hosts bound into this simulation, this implementation of a network routing simulator holds firm in its consistency and proves as a viable medium in which to securely replace a physical network.

7.1.2 Throughput

Other than the ability to support major network protocols found within networks, the ability to sustain an acceptable throughput of a protocol, or mix of protocols, is key in the success of a network's simulation. If a simulator cannot effectively fulfil its purpose in a timely manner, it would fail to be a meaningful tool for future use within the areas of research to which the simulator applies.

This routing simulator's ability to fulfil its purpose in a timely manner is based on the data throughput which it can achieve. The tests performed in Section 5.3 shows this to be 5.253 Gbps on a single core of the Intel i5-3570K CPU, and when testing occurs involving usage of all four cores, a total of 7.443 Gbps of throughput is achieved.

An average host's NIC is rated to 1 Gbps of throughput at the time of this research (this being full-duplex traffic). Considering that many legacy devices are still based on 100 Mbps Ethernet and wireless devices synchronizing between 54 and 108 Mbps which consist of half-duplex connections over air; the throughput that this simulator can handle serves to host multiple physical hosts in a simulated environment and is able to process in excess of most Internet uplinks.

Considering that the entirety of this simulation is performed in software, these results are notable as a vast improvement to prior proof of concept work which achieved slightly less than 40 Mbps of throughput (Herbert, 2012; Irwin and Herbert, 2013). Given this improvement, and taking into consideration the throughput rates of physical hardware that exists at this time of research, these results are more than satisfactory when considering this research aimed to achieve 1 Gbps of throughput when it started.

7.1.3 Memory Utilization

There are benefits to using a lot of memory within a simulation's buffers. These benefits are mostly in the storing of calculations that are frequently used in tables or other such

data structures discussed in Section 3.3. This method frees up CPU time for the host computer due to the fact that the calculation happens once and is then stored, allowing for the result to be recalled at a later stage. Although a useful method of computation within a simulation, this can lead to a host's memory becoming full and thrashing[3] may occur on the host. This can result in a significant slow down in one's implementation, more so than if the result was calculated on request.

Considering the result of Section 5.2 and the memory available to commodity host computers, the ability to store every result is severely limited. To elaborate in terms of a routing simulator, the ability to store every route from every node to any other node is clearly far greater than what one could expect to be within the requirements of a standard desktop computer. This means that every node would have to know the route to every other node from itself. The result of this is each node needing to remember N-1 routes, where N is the number of nodes in the simulated network. This means that the total routes that need to be stored is the number of nodes multiplied by the number of routes to each other node, thus resulting in $N \times (N-1)$ routes. From this formula one can observe an $O(n^2)$ growth in memory requirements by the simulated network for each node added to the network. For this reason the decision to process results at runtime allows for more space to instantiate a larger network, and more importantly, more memory with which to route packets within the simulated network.

Considering the hardware used had 8GB of memory available and managed to route 7.443 Gbps of data within the instantiated network, the memory requirements of this routing simulator appear to fall within the range of a commodity hardware range of computing power (at the time of this research). Also, due to the throughput this simulator can achieve by doing calculations at runtime, and that the host does not bottleneck on the CPU's resources, this simulator does not need to store results in memory. This further adds to the memory available for simulated network instantiation and packet routing. The projections made in Section 5.2 also suggest that instantiation of the entire Internet within software is not an impossible task in the scope of IPv4 within five years of the time of this research.

7.1.4 Processing Skew

Processing skew is the delay added to a hop due to processing delay (in addition to the desired delay). Essentially this delay is due to the time between the work queue spawning

[3]The act of writing memory out to disk.

a thread to perform a hop, calculating whether there is another hop and rescheduling it, and ejection of a packet from the simulation. If this delay is excessive, it can skew the actual intended delay of a packet. Furthermore, the more hops the packet has to take in a route, the greater the skew would become.

In Section 5.4, one can see that over multiple hops the processing delay is not significant as it introduces delay skew by between -2 and 2 milliseconds. This delay can also be attributed to the time waiting in queue of a NIC or bus delays of the motherboard due to bus congestion. There is no doubt that processing delay makes up part of this delay time too, however it is not enough to warrant concern as this delay simply adds a slight entropy to a packet's delay which serves to mimic the behaviour of a physical networks.

7.2 Evaluation of Research Goals

This section aims to draw a conclusion about the success or failure of this implementation and research. For this reason this section refers back to Section 1.2 to draw a conclusion to what degree this implementation of a routing simulator achieves these goals. As these goals are clearly defined in numerical order, they will simply be referred to by their accompanying number in this section.

1. This research initially sought to simply route packets on a IPv4 address space and to be able to interface with physical hosts connected into the simulation via a NIC present on the simulation host. This was achieved, found to be reliable, and thus laid the grounding on which the transport layer protocol support could be built upon.

2. Transport protocol support was extended on top of IPv4 to support TCP, UDP and ICMP, as many applications rely on these for protocols for transport layer communication. Features were added to these protocols after the support of TCP, UDP and ICMP was stable.

3. Having tested reliability of the transport protocols to be lossless under normal operating conditions[4], features were then added. These features included delay, packet loss, packet mangling and network jitter. The addition of these features created a

[4]Normal referring to throughputs within the accepted range of the NIC and traffic without malicious intent.

more realistic network simulation that represents characteristics and behaviours in physical networks due to bad links, interference and wireless connections.

4. Finally, this research aimed to reach a throughput of 1 Gbps. Having achieved 7.443 Gbps average on commodity hardware readily available to the public consumer (as defined in Chapter 5), this routing simulator surpassed the objective it sought to achieve. Furthermore, this routing simulator is compatible with a variety of hosts and can scale to the needs of the user.

In closing, this research aimed to create a software based routing simulator as a viable tool for network simulation while keeping set-up costs to a minimum. Having achieved the goals set out for this research in Section 1.2, this research shows promise in supporting further research in the fields of networks and simulation.

7.3 Compatibility

This routing simulator was compiled under Linux for the Linux version 3.2.0-4-amd64 kernel. As such, this routing simulator is compatible with any system that supports this kernel version. Drawing system requirements from testing done in Chapter 5, one can find that the memory requirement is based on the size of the network one wishes to simulate and the throughput one wishes to handle. Furthermore, CPU requirements are based on the throughput one wishes to introduce into the system. The equations to estimate memory requirements are detailed in Sections 5.2 and 5.5.

This compatibility creates the ability to customize a host computer to the requirements that one's simulation needs. This allows for cost effectiveness in terms of a small simulation, and upward scalability in terms of a larger simulation. It is notable that, unlike in a hardware solution, a software approach continues to scale up in time. This is due to the fact that software runs on hardware that is improving as time continues and so the software runs better because of this. On the other hand, a hardware solution only has the computational power of when it was manufactured and time does not increase nor reduce this.

This allows this routing simulator to be a viable option in the future with this increased lifetime and malleable nature in terms of compatibility; this routing simulator serves as a solution in a wide scope of hardware environments.

7.4 Future Work

As the field of information technology is ever growing, so are the sub-fields within it. As such, network technology is seeing new protocols, hardware and software advances regularly. Because of this, this field of research (and this research directly), can see growth through creation of these new technologies and through extension of currently existing ones. For this reason, the following list of possible ideas for future work on this research has been documented.

- Extension into support of the IPv6 address space. Although not a ground breaking advance in this research, the ability to provide support for and simulate IPv6 based networks opens more opportunities for this research and will further add to this routing simulator's utility. This being said, the simulation of the entirety of IPv6 address space at a single moment is a stretch, as currently this address space is 2^{96} times larger than that of IPv4. However, relatively low address allocation count is feasible in the IPv6 address space.
- Addition of further transport protocol support for less frequently used protocols that exist. This would improve the utility of this routing simulator and aid in further extending this field of research. Examples of such protocols are DCCP (Kohler *et al.*, 2006), SCTP (Stewart *et al.*, 2000) and RSVP (Braden *et al.*, 1997).
- Porting of this routing simulator to other OSs, or create a user context implementation to analyse bottle necks that occur between a kernel and user context.
- Design of network generation and packet generation tools for application on this routing simulation platform. This will help further increase the utility of this routing simulator and provide a basis on which to stress test network layouts, or hardware that is physically connected to this routing simulator.
- Collection and processing of data sets from institutes like CAIDA for route configuration and analysis would provide the means by which to create a more accurate routing simulator. This kind of data would allow for one to more accurately simulate the behaviours and qualities of a network and adjust the functional code of this implementation to better represent this physical data.

These ideas can be applied across many similar routing simulators, and even other simulation implementations. As such, any further development in this field of research will

help to forward networks, simulation, network simulation and Computer Science as a whole.

References

Adachi, F. *Wireless past and future-evolving mobile communications systems*. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E Series A*, 84(1):55–60, 2001.

Amdahl, G. M. *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*. *Solid-State Circuits Society Newsletter, IEEE*, 12(3):19–20, 2007.

Apposite Technologies. *Wan emulation made easy*. 2013. Accessed 2 February 2014. URL `http://www.apposite-tech.com/index.html`

Aycock, J. *Computer Viruses and Malware*, volume 22. Springer, 1st edition, 2006.

Ayuso, P. *Netfilter*. 2010. Accessed 9 October 2013. URL `http://www.netfilter.org/`

Barthélemy, M., Barrat, A., Pastor-Satorras, R., and Vespignani, A. *Velocity and hierarchical spread of epidemic outbreaks in scale-free networks*. *Physical Review Letters*, 92(17):178701, 2004.

Bautts, T., Dawson, T., and Purdy, G. *Linux Network Administrator's Guide*. O'Reilly Media, Inc., California, 3rd edition, 2005, 123-124 pages.

Berners-Lee, T., Fielding, R., and Frystyk, H. *Rfc 1945: Hypertext Transfer Protocol HTTP/1.0, may 1996*. 61, 1996.

Black, P. E. *Array*. online, January 2004. Accessed 30 September 2013. URL `http://xlinux.nist.gov/dads//HTML/array.html`

Black, P. E. *Complete binary tree*. online, January 2008. Accessed 30 September 2013. URL `http://xlinux.nist.gov/dads//HTML/perfectBinaryTree.html`

Bolot and Jean-Chrysotome. *End-to-end packet delay and loss behavior in the Internet*. *ACM SIGCOMM Computer Communication Review*, 23(4):289–298, 1993.

Bovet, D. P. and Cesati, M. *Understanding the Linux Kernel*. O'Reilly Media, Inc., California, 1st edition, 2000.

Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. *RFC 2205: Reservation Protocol (RSVP) Version 1*. Technical report, IETF, September, 1997.

Burns, R. and Dennis, A. R. *Selecting the appropriate application development methodology*. *ACM Sigmis Database*, 17(1):19–23, 1985.

CAIDA. *CAIDA: The Cooperative Association for Internet Data Analysis*. January 2014. Accessed 16 January 2014.
URL `http://www.caida.org/home/`

Cisco. *Using the traceroute command on operating systems*. August 2005. Accessed 31 Octoboer 2012.
URL `http://www.cisco.com/c/en/us/support/docs/ip/ip-routed-protocols/22826-traceroute.html`

Cisco. *The Cisco Networking Academy*. 2009. Accessed 3 November 2014.
URL `http://www.cisco.com/web/EA/solutions/en/computer_networking/index.html`

Coltun, R., Ferguson, D., Moy, J., and Lindem, A. *RFC 5340, OSPF for IPv6*. *IETF. July*, 24, 2008.

Comer, D. E. *Computer Networks and Internets*. Prentice Hall, Inc., New Jersey, 2008.

Communications Inc. *Network emulation with data rates up to 10 Gbps*. 2014. Accessed 2 February 2014.
URL `http://packetstorm.com/psc/psc.nsf/site/index`

Deering, S. E. *RFC 2460: Internet Protocol, version 6 (IPv6) specificat*. 1998.

Demichelis, C. and Chimento, P. *RFC 3393: IP packet delay variation metric for IP performance metrics (IPPM)*. *IETF, November*, 2002.

Devin, J. and Warren, D. *From shafts to wires: Historical perspective on electrification*. *Journal of Economic History*, 43:2, 1983.

Digilent. *Digilent c-mod boards reference manual*. Technical report, Digilent, June 2004.

Douligeris, C. and Mitrokotsa, A. *Ddos attacks and defense mechanisms: classification and state-of-the-art*. *Computer Networks*, 44(5):643–666, 2004.

Doyle, J. and Carroll, J. D. *Routing TCP/IP 1*, volume 1. Cisco Press, Indianapolis, 2005.

Dunham, D. *Hash tables*. April 2009. Accessed 1 October 2013.
URL `http://www.duluth.umn.edu/~ddunham/cs4521s09/notes/ch11.txt`

Edelkamp, S. *Dictionary of algorithms and data structures: Patricia tree*. December 2010. Accessed 7th February 2014.
URL `http://xlinux.nist.gov/dads/HTML/patriciatree.html`

Emerman, M. and Malik, H. S. *Paleovirology: Modern consequences of ancient viruses*. *PLoS biology*, 8(2):e1000301, 2010.

Fishwick, P. *Why do simulation?* 1995. Accessed 6th February 2014.
URL `http://www.cise.ufl.edu/~fishwick/introsim/node2.html`

Ford, W., Topp, W., and Ford, W. H. *Data Structures with C++ Using STL*. Pearson Education, India, Computer Science Department, University of the Pacific, Stockton, California., 2nd edition, 2002. ISBN-10: 0130858501.

Garcia, L. M. *Tcpdump & libpcap*. January 2014. Accessed 19 January 2014.
URL `http://www.tcpdump.org/`

GNU. *Options that control optimization*. 2013. Accessed 16 October 2013.
URL `http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`

Heikkinen, T. and Järvinen, A. *The common cold*. *The Lancet*, 361(9351):51–59, 2003.

Herbert, A. *Narwhal: An IPv4 core routing simulator*. Technical report, Rhodes University, 2012.
URL `http://www.cs.ru.ac.za/research/g09H1151/presies/thesis.pdf`

Herbert, A. and Irwin, B. *A kernel-driven framework for high performance internet routing simulation*. In *Information Security for South Africa, 2013*, pages 1–6. IEEE, 2013.

Huston, G. *IPv4 Address Report*. February 2014. Accessed 6th February 2014.
URL `http://www.potaroo.net/tools/ipv4/index.html`

Hyun, Y. and Broido, A. *On third-party addresses in traceroute paths*. 2003.

Intel Corperation. *IA-32 Architectures Software Developer's Manual. Volume 3[a]*, 2001.

Internet Systems Consortium. *The ISC Domain Survey*. Jan 2013. Accessed 22nd December 2013.
URL `http://www.isc.org/solutions/survey`

Internet World Stats. *Usage and Population Statistics*. 2014. Accessed 3 November 2014.
URL `http://www.internetworldstats.com/stats.htm`

Irwin, B. and Herbert, A. *Deep routing simulation*. In *Proceedings of the 8th International Conference on Information Warfare and Security: ICIW 2013*, pages 97–105. Academic Conferences Limited, 2013.

Ixia. *Breakingpoint application and security test solutions*. 2014. Accessed 30th March 2014.
URL `http://www.ixiacom.com/breakingpoint`

Jacobsen, V. *traceroute. man page, UNIX, 1989. See source code: ftp://ftp. ee. lbl. gov/traceroute. tar. gz, and NANOG traceroute source code: ftp://ftp. login. com/pub-/software/traceroute*, 2001.

Jacobson, D. *Iseage project overview. Iowa State Univ.. Ames, IA*, 2007.

James, J. *Low-cost computers for education in developing countries. Social Indicators Research*, 103(3):399–408, 2011.

Jelsim Partnership. *esim builder 2.4 training*. 2009. Accessed 6th February 2014.
URL `http://www.jelsim.org/training/introduction/simulation.html`

Jim Kurose, K. R. *Computer Networking A Top-Down Approach*. Pearson Education, Inc., fifth edition edition, 2010.

Knizhnik, K. *Patricia tries: A better index for prefix searches. Dr. Dobb's Journal*, 2008.

Knuth, D. *Fundamental Algorithms*. Addison-Wesley, Massachusetts, 3rd edition, 1997.

Koffka, K. *Principles of Gestalt psychology*. New York: Harcourt, 1935, 176 pages.

Kohler, E., Handley, M., Floyd, S., and Padhye, J. *Datagram congestion control protocol (DCCP)*. 2006.

Koonin, E. V., Senkevich, T. G., and Dolja, V. V. *The ancient virus world and evolution of cells. Biology Direct*, 1(1):29, 2006.

Kopp, M. *The Top Java Memory Problems Part 2*. December 2011. Accessed 25 September 2013.
URL `http://apmblog.compuware.com/2011/12/15/`

Kotenko, I. *Agent-based modeling and simulation of cyber-warfare between malefactors and security agents in Internet*. In *19th European Simulation Multiconference Simulation in wider Europe*, pages 56–73. 2005.

Leighton, F., Maggs, B., and Rao, S. *Packet routing and job-shop scheduling in O (Congestion+Dilation) steps*. *Combinatorica*, 14(2):167–186, 1994.

Leiserson, C. E., Rivest, R. L., Stein, C., and Cormen, T. H. *Introduction to algorithms*. The MIT Press, 2001.

Lindholm, T., Yellin, F., Bracha, G., and Buckley, A. *The Java virtual machine specification*. Pearson Education, 2014.

Locke, J. *An introduction to the internet networking environment and SIMNET/DIS*. 1995. Accessed 3 Novemeber 2014.
URL `http://dailyitinfo.com/bca/sem1/networking%20environment.pdf`

Louck, J. D. *Unitary symmetry and combinatorics*. World Scientific, New York USA, 2008.

Love, R. *Kernel Korner: the new work queue interface in the 2.6 kernel*. *Linux Journal*, 2003.

Mackall, M. and Ts'o, T. *Linux cross reference: Linux/drivers/char/random.c*. April 2014. Accessed 21 April 2014.
URL `http://lxr.free-electrons.com/source/drivers/char/random.c`

Mahajan, R., Spring, N., Wetherall, D., and Anderson, T. *User-level Internet path diagnosis*. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 106–119. ACM, 2003.

Malkin, G. *RFC 1393: Traceroute using an IP option*. 1993.

Malkin, G. *RFC 2453: Rip version 2. Request for Comments*, 2453, 1998.

Maria, A. *Introduction to modeling and simulation*. In *Proceedings of the 29th conference on Winter simulation*, pages 7–13. IEEE Computer Society, 1997.

Marques, P., Guichard, J., Raszuk, R., Bonica, R., Patel, K., Fang, L., and Martini, L. *Constrained Route Distribution for Border Gateway Protocol/MultiProtocol Label Switching (BGP/MPLS) Internet Protocol (IP) Virtual Private Networks (VPNs)*. 2006.

Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A., and Upton, M. *Hyper-threading technology architecture and microarchitecture. Intel Technology Journal*, 6(1), 2002.

Martinez, V. P., Bellomo, C., San Juan, J., Pinna, D., Forlenza, R., Elder, M., Padula, P. J. *et al. Person-to-person transmission of andes virus*. *Emerging infectious diseases*, 11(12):1848–1853, 2005.

McDowell, M. *Understanding denial-of-service attacks*. 2009. Accessed 20th March 2014.
URL `http://www.us-cert.gov/ncas/tips/ST04-015`

Microsoft. *A description of the differences between 32-bit versions of windows vista and 64-bit versions of windows vista*. 2011. Accessed.
URL `http://support.microsoft.com/kb/946765`

Mills, D. *RFC 904: Exterior Gateway Protocol Formal Specification, DARPA Network Working Group Report*. Technical report, M/A-COM Linkabit, 1984.

Mills, D. L. *Internet time synchronization: the network time protocol*. *Communications, IEEE Transactions on*, 39(10):1482–1493, 1991.

Moore, G. E. *Cramming more components onto integrated circuits*. 1965.

Moy, J. *RFC 2328: OSPF Version 2*. 1998.

National Science Foundation. *NS-3*. 2013. Accessed 1 March 2012.
URL `http://www.nsnam.org/`

Naval Research Laboratory. *Coreemu: Common open research emulator*. 2014. Accessed 10 June 2014.
URL `https://code.google.com/p/coreemu/`

Netronome. *Redefining networks*. September 2014. Accessed 8 October 2014.
URL `http://www.netronome.com/`

Null, L. *The essentials of computer organization and architecture*. Jones & Bartlett Learning, 2nd edition, 2006.

Ormsbee, L. E. and Lansey, K. E. *Optimal control of water supply pumping systems*. *Journal of Water Resources Planning and Management*, 120(2):237–252, 1994.

Ostinato Open Source Project Group. *Ostinato: Packet/traffic generator and analyzer*. May 2013. Accessed 19 January 2014.
URL `http://code.google.com/p/ostinato/`

Popov, M. *Everything converged–A flexible photonic home*. In *Microwave Photonics, 2009. MWP'09. International Topical Meeting on*, pages 1–4. IEEE, 2009.

Postel, J. *RFC 768: User Datagram Protocol*. Technical report, IETF, 1980.

Postel, J. *RFC 791: Internet Protocol*. Technical report, IETF, 1981a.

Postel, J. *RFC 792: Internet Control Message Protocol*. Technical report, IETF, 1981b.

Postel, J. *RFC 793: Transmission Control Protocol*. Technical report, IETF, 1981c.

Postel, J. *RFC 765: File Transfer Protocol specification*. Technical report, IETF, 1985.

Potter, S. and Nieh, J. *Reducing downtime due to system maintenance and upgrades*. In *19th LISA*, pages 47–62. 2005.

Provos, N. *Developments of the honeyd virtual honeypot*. July 2008. Accessed 30 September 2014.
URL `http://www.honeyd.org/`

Python Software Foundation. *Extending Python with C or C++*. October 2013. Accessed 21 October 2013.
URL `http://docs.python.org/2/extending/extending.html`

Quoitin, B. and Uhlig, S. *Modeling the routing of an autonomous system with C-BGP*. *Network, IEEE*, 19(6):12–19, 2005.

Ramanathan, R. *Intel® multi-core processors. Making the Move to Quad-Core and Beyond*, 2006.

Randelzhofer, M. *Gop-xc3s200 user's manual 0.91*. Technical report, OHO-Elektronik, 2009.

Rekhter, Y., Li, T., and Hares, S. *Rfc 4271: Border gateway protocol 4*. 2006.

RIPE NCC. *Understanding IP Addressing*. Jan 2014. Accessed 18th April 2014. URL http://www.ripe.net/internet-coordination/press-centre/understanding-ip-addressing

Robinson, S. *Simulation: the practice of model development and use*. John Wiley & Sons Chichester, 2004.

Santa Clara University. *skbuff struct reference*. September 2004. Accessed 7th February 2014. URL http://www.cse.scu.edu/~dclark/am_256_graph_theory/linux_2_6_stack/structsk__buff.html

Savage, S., Anderson, T., Aggarwal, A., Becker, D., Cardwell, N., Collins, A., Hoffman, E., Snell, J., Vahdat, A., and Voelker, G. *Detour: Informed Internet routing and transport. Micro, IEEE*, 19(1):50–59, 1999.

Siewiorek, D. P. and Swarz, R. S. *Reliable computer systems: design and evaluation*. Digital Press, Newton, 1992.

Silberschatz, A., Galvin, P. B., and Gagne, G. *Operating System Concepts*. John Wiley and Sons, Inc., Hoboken, 8th edition, 2010.

Smith, R. D. *Converting a large simulation system to a 64-bit computer*. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, volume 2004. NTSA, 2004.

Sommers, J. and Barford, P. *Self-configuring network traffic generation*. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81. ACM, 2004.

Spitzner, L. *Honeypots tracking hackers*. Pearson PLC, London, 2002.

Spitzner, L. *Honeypots: Catching the insider threat*. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

Stanescu, B. *Top 5: Corporate losses due to hacking*. 2011. Accessed 28 March 2014. URL http://www.hotforsecurity.com/blog/top-5-corporate-losses-due-to.-hacking-1820.html

Stanford University Engineering Computer Science. *NetFPGA: The NetFPGA is*. 2013. Accessed 30th March 2014. URL http://netfpga.org/index.html

Std, I. *VHDL standard: Language reference manual*. Technical report, IEEE, 1988.

Stevenson, A. and Lindberg, C. A. *New Oxford American Dictionary*. Oxford University Press, 3rd edition, August 2010.

Stewart, R., Xie, Q., Mornmeault, K., Sharp, H., Taylor, T., Rytina, I., Kalla, M., and Zhang, L. *V. Paxson, Stream Control Transport Protocol*. Technical report, RFC 2960, October, 2000.

Sun, T. *The art of war*. Orange Publishing, 2013.

The Fiber Optic Association, Inc. *Copper or fiber? what's the real story?* 2005. Accessed 10 June 2014.
URL `http://www.thefoa.org/tech/fo-or-cu.htm`

The Jargon File. *Wall time*. 2013. Accessed 10 November 2013.
URL `http://www.catb.org/jargon/html/W/wall-time.html`

The Linux Information Project. *Context switch definition*. May 2006.
URL `http://www.linfo.org/contextswitch.html`

The UNIX and Linux Forums Content. *Man pages: Panic(9)*. 2013a. Accessed 19th October 2013.
URL `http://www.unix.com/man-page/FreeBSD/9/panic/`

The UNIX and Linux Forums Content. *Man pages: Proc(4)*. 2013b. Accessed 29 October 2013.
URL `http://man.cat-v.org/unix_8th/4/proc`

TMN Simulation. *What can simulation do for me?* 2011. Accessed 13 April 2012.
URL `http://tmnsimulation.com.au/simulation/why-simulate/`

Turner, A. and Bing, M. *tcpreplay Tool*. 2012.
URL `http://tcpreplay.synfin.net/`

Velikov, M., Hellwig, C., Piggin, N., and Khlebnikov, K. *Linux cross reference: Linux/drivers/char/radix-tree.c*. April 2014. Accessed 21 April 2014.
URL `http://lxr.free-electrons.com/source/include/linux/radix-tree.h`

Wang, W. and Lu, Z. *Cyber security in the smart grid: Survey and challenges*. *Computer Networks*, 57(5):1344–1371, 2013.

Welte, H. and Ayuso, N. *Linux cross reference: http://lxr.free-electrons.com/source/include/uapi/linux/netfilter.h*. April 2014. Accessed 22 April 2014.
URL http://lxr.free-electrons.com/source/include/uapi/linux/netfilter.h

Wheeler, D. A. *Why open source software/free software (oss/fs, floss, or foss)? look at the numbers*. 2005.

Wireshark Foundation. *Checksums*. 2013. Accessed 6th February 2014.
URL http://www.wireshark.org/

Xilinx Inc. *Xilinx: All programmable*. 2014. Accessed 15th February 2014.
URL http://www.xilinx.com/

Ylonen, T. and Lonvick, C. *The Secure Shell (SSH) transport layer protocol, RFC 4253*. 2006.

Yuanji, G., Fengen, J., Ping, W., Min, W., and Jiming, Z. *Isolation of influenza c virus from pigs and experimental infection of pigs with influenza c virus*. *Journal of General Virology*, 64(1):177–182, 1983.

Zdrnja, B., Brownlee, N., and Wessels, D. *Passive monitoring of dns anomalies*. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 129–139. Springer, 2007.

Zhang, Y., Li, T., and Qin, R. *Computer virus evolution model inspired by biological dna*. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, pages 943–950. Springer, 2008.

Zheng, L., Zhang, L., and Xu, D. *Characteristics of network delay and delay jitter and its effect on voice over IP (VoIP)*. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 1, pages 122–126. IEEE, 2001.

Zou, Y. and Black, P. E. *Perfect binary tree*. online, January 2008. Accessed 30 September 2013.
URL http://xlinux.nist.gov/dads//HTML/perfectBinaryTree.html

A

Test Network IP Level Connections

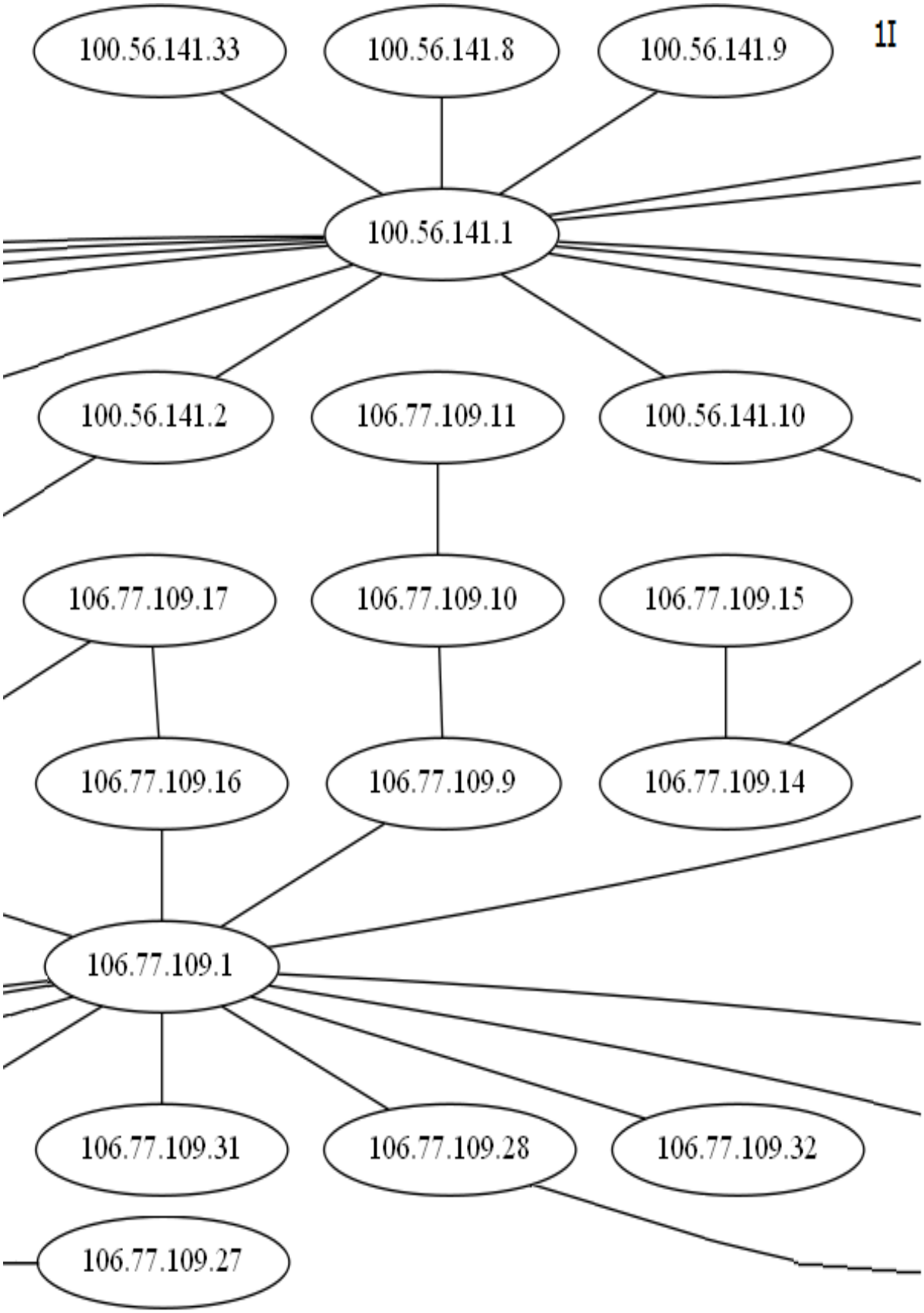
This appendix exists to show the layout of the test network used in this research. It is shown node-for-node on a network layer level, thus using IP addresses for each node within the network. The purpose for this appendix is so that the test network that yielded these results can be recreated and used for confirmation of such.

This graph was created using the Dot format and then parsed into a graphical representation with GIMP. This appendix is intended to allow for simple reconstruction of a visual overview of this network configuration. The image depicted in Figure A.1 is a condensed version of the network used in the testing of this routing simulator. Full scale images are available from the online repository in which this implementation resides and are labelled by algebraic chess notation for reconstruction.

Figure A.2 is an expanded view based on /Gen Net/cfg in the online resource. As one can see, this figure shows each node represented by a circle with its associated IP and a line showing a link between nodes.

Figure A.1: Test Network Map Reference

Figure A.2: Test Network Map Reference - Online Resource Sample



B

Simple Configuration Files for Working Example

In this appendix one can find a sample of what one is expected to produce to configure this implementation of a routing simulator. Configuration of this routing simulator requires two input files. The first input file describes the nodes within the network, what anomalies apply and how they are linked, whereas the second defines where physical hosts are bound into the simulation.

B.1 Network Configuration File Format

Network configuration follows these rules:

1. The start of a new node is defined by the ">" character.
2. The node's IP is then defined.
3. The start of a node's forwarding table rules is defined by "*".

4. The rules are then listed in the form "IP to match on" "Resultant Hop Destination" "Subnet Mask" "Delay of Hop" "Chance to Drop Packet" "Chance to Mangle Packet" "Jitter Chance" "Delay Amount to Jitter" "Max Jitter Retries".

Following these rules for a packet arriving at a node, one would match the packet's destination IP with the "IP to match on" by using the "Subnet Mask" to mask the destination IP. If this matches, one can then fetch the next hop's address by the "Resultant Hop Destination". Anomalies are calculated as per specification in an expected manner. A configuration for a basic 8 node network can be viewed in Listing B.1.

Listing B.1: Sample Network Configuration

```
1  >
2  123.0.0.0
3  *
4  10.0.0.1  10.0.0.1  32  15  25  0  0  0  0
5  10.0.0.0  57.5.0.0  16  10  0  0  20  10  5
6  >
7  10.0.0.1
8  *
9  10.0.0.0  123.0.0.0  8  15  0  0  0  0  0
10 >
11 57.5.0.0
12 *
13 10.0.0.1  123.0.0.0  32  10  0  0  0  0  0
14 10.0.0.0  120.5.0.0  16  10  0  0  0  0  0
15 >
16 120.5.0.0
17 *
18 10.0.0.1  57.5.0.0  32  10  0  50  40  10  3
19 10.0.0.1  83.5.0.0  24  5  30  25  0  0  0
20 10.0.1.0  103.5.0.0  24  10  0  0  0  0  0
21 >
22 83.5.0.0
23 *
24 10.0.0.1  120.5.0.0  32  5  0  0  0  0  0
25 10.0.1.0  120.5.0.0  24  5  0  0  0  0  0
26 10.0.0.0  10.0.0.5  24  10  0  0  0  0  0
27 >
28 103.5.0.0
```

```
29 *
30 10.0.0.0 120.5.0.0 24 10 0 0 0 0 0
31 10.0.1.0 10.0.1.5 24 10 0 0 0 0 0
32 >
33 10.0.0.5
34 *
35 10.0.0.0 83.5.0.0 16 10 0 0 0 0 0
36 >
37 10.0.1.5
38 *
39 10.0.0.0 103.5.0.0 16 10 0 0 0 0 0
```

B.2 Physical Host Binding Configuration File Format

Rules for binding a physical host into this simulation are as follows:

1. Each line contains a single binding rule.
2. The form of this rule is "Name of NIC" "Simulated Node IP Address" "Physical Host IP Address" "Physical Host MAC Address".

To bind a physical host to the simulation network one first must define on which interface the physical host can be found. From here one simply creates a link between the simulated node's IP address and the physical host's IP address. The last requirement is the MAC address of the physical host by which to complete communication. A sample configuration file for the network described in Listing B.1 can be found in Listing B.2.

Listing B.2: Sample Binding Configuration

```
1 eth3 10.0.0.1 10.42.0.78 a0:36:9f:0d:4b:6e
2 eth3 10.0.0.5 10.42.0.35 c0:ff:ee:c0:ff:ee
3 eth2 10.0.1.5 10.42.0.50 70:54:d2:ad:52:68
```

C

Online Resource Access

Access to online resources is not public and as such should be requested from the author of this document. To contact the author to request access please send an email to msc@eskerfall.com. Both repositories containing this document and implementation reside on Bitbucket and can be cloned through the relevant address below where username is defined by the account used by the reader as registered on Bitbucket:

Resource	Address
Thesis Document	https://username@bitbucket.org/twelfthletter/msc_thesis
Implementation	https://username@bitbucket.org/twelfthletter/msc