# Narwhal: An IPv4 core routing simulator

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Alan Herbert

*Grahamstown, South Africa*
November 2012

## Abstract

Routing tables and route calculation within packet routing simulators suffer from large memory and CPU requirements. This research paper looks into reducing these requirements that are an inherent trait amongst large scale packet routing simulators by effectively allocating resources through lightweight data structures and fast routing algorithms.

This research takes on two approaches towards packet routing simulation. The first approach is a static method which calculates the routes before hand and stores the route in memory. The second approach is that of a dynamic implementation which attempts to reduce memory costs through calculation of routes at runtime. Both of these routing algorithms are software only implementations.

When considering 9991 simulated routes, this packet routing simulator manages a constant throughput of 40Mbps while consuming just under 8Mb of RAM using a static routing approach, and just under 2Mb of RAM when using a dynamic approach to solving the same problem.

**Acknowledgements**

The writer of this research paper 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, Dr 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 would like to thank the Computer Science Department at Rhodes University for allowing me access to the equipment and space required to carry out this research.

Finally I wish to acknowledge the support of the Centre of Excellence in Distributed Multimedia, within the Department of Computer Science, for the financial support for this research.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Problem Statement

Currently available routing simulators suffer from heavy resource requirements. These requirements are primarily related to high memory and CPU requirements (Xu and Ammar, 2004). Also some of these simulators suffer from low availability due to costly investments into the software and hardware needed run these simulators. The need for a memory and CPU-effective solution to routing simulation, along with ease of configuration, low cost and high compatibility, is required.

## 1.2 Research Goal

The need to simulate is often a question that arises before implementation or further study of a topic. The simplest answer is that not every environment is ideal and there are problems that can arise in these non-ideal environments. These problems usually arise from oversights that only come to light when the system is actually tested (Production Modelling, 2012).

In short, if one does not simulate a product before implementation, it can have major flaws that can lead to failure of a project completely. These flaws could have been discovered and fixed before the final product was produced through simulation. This oversight from a business point of view can lead to large net worth losses and in some cases even bankruptcy.

When relating this to routing simulators that are used in education, research and development and other systems that require testing on live networks, using real networks in these cases can lead to unforeseen side effects such as other systems failing or threats to security due to flaws in security logic arising, and thus deterring the overall positive outcome of a given task or prolonging the desired result.

Many approaches have been taken to network simulation. This research takes a more direct approach by setting its scope mainly to packet routing simulation, primarily within the IPv4 scope, and related articles in this field of study. This is because IPv4 has been in use for a longer period of time, being first introduced in September 1981 (Postel, 1981a). This may be extended into IPv6 in future related work as IPv6 (Deering and Hinden, 1998) has been developed to update IPv4 (Postel, 1981a).

Furthermore, as routing tables, as found within routing simulators, require large amounts of memory (Riley, Ammar, and Fujimoto, 2000), this implementation will look into reducing these requirements. This may come at a trade-off of CPU resources, and so delay may creep into this resource of the host system. This is ultimately what lead to CIDR (Class Interdomain Routing) which is a routing implementation designed to reduce the growth in memory requirements of routing tables (Southwestern Bell Internet Services, 2001).

This trade off will be tested through the use of a less memory effective pre-calculated routing system, a static routing system, and then compared against a more memory effective routing system that opts for dynamically working out the routes at run time.

The memory requirements of each approach will also be taken into consideration. A conclusion will be drawn to see if the memory for CPU trade-off is really worth it or not, this is also dependant on the target platform and hardware associated with it.

The testing of Narwhal will be run on a configuration of that of the top ten thousand visited sites on the Internet. These routes will be traced using traceroute, and parsed using a python script for easy upload into the simulator. This will give constant and relevant data for tests to be run against, and thus produce results that are easily related to real world applications.

Other than memory and CPU usage, throughput will also be taken into consideration. This is to test whether this routing simulator can be applied in active use and whether it is relevant or not. If the routing simulator cannot at least achieve an average usable throughput, throughput rates as found in a common household, it will not be relevant for use in real time testing.

Cost is also a factor of this research project as this routing simulator is aimed for general use. As such all libraries used and the operating system this software is intended for will be aimed at freeware implementations. Also libraries and the operating system will be selected under ease of use, compatibility and how well documented the relevant software is.

## 1.3 Document Structure

This research paper will first consider previous research performed in this field and approaches taken in advancing research in this area in the form of a literature review, this will make up Chapter 2. This will flow from why simulation is important and what packet routing is to software, hardware and hybrid implementations that currently exist, and filling in the finer details about each in between.

From here this research paper takes the observations realised from the literature review and defines what needs to be focussed on, how these areas will be approached, why they need to be considered and in what context they shall be taken in by. This is done in Chapter 3 through the design and the implementation that will form the packet routing simulator.

Following this we produce the findings of this research paper in Chapter 4. This includes whether the routing simulator works, and in what conditions it does so, if it works, followed by general testing, comparisons between routing styles, and relevance of each test in the context of real world networks.

Lastly, in Chapter 5 this research paper will conclude its findings and suggest future research, Section 5.4, propositions as to further this field of research within computer science and simulation as a whole.

# Chapter 2

# Literature Review

There is a need to understand what approaches and research have been done in this field before moving onto designing and implementing a routing simulator. This is essentially to be able to try new approaches to problems and prevent one from looking for solutions that have already been found.

Understanding what packet routing algorithms exist is essential to designing a routing simulator that is relevant to real routing replication. Being able to convert a real network into a simulated one is impossible if this basic property does not exist.

Furthermore, there are many successes and failures that have been documented, tried and tested. Taking these results and reworking the better points or redesigning failures can lead to a better routing simulator than what previous attempts had yielded.

This chapter starts by defining the importance of simulation in Section 2.1 and when and why simulations should be done. This is intended to give good footing as to why one would want to create a routing simulator.

This is then followed in Section 2.2 by the approaches to network simulation and a general overview of what exists already and how they approached such simulation. Following this in Section 2.3 is an outline on complications that arise when tackling the task of implementing and designing a routing simulator. This details the approaches taken to speed up routing simulation and what needs should exist in a network simulator.

In Section 2.4 we follow this with a look into routing protocols that are commonly used and how to go about creating a routing simulator. This also outlines libraries that exist that can help with packet transmission.

After this the use of traceroute is defined in Section 2.5 and how it can be used to collect routing data for a routing simulator. Also, other sources of such data are brought to light. Then we look at how one would create a simulated network in which to route packets, and how one would go about routing a packet in this virtual network are defined in Section 2.6.

Hardware and software to implement packet routing are then discussed in Section 2.7. This section shows what exists, but more importantly the differences between a hardware and software implementation and how a hybrid of the two may be beneficial.

We then tie up this in Section 2.8 with why packet routing simulation is important, followed by a summary of this chapter.

## 2.1 Importance of Simulation

Simulation is the process of creating a model that represents an implemented system or system that is still to be simulated and running tests against it to understand where its strengths and flaws exist (GoldSim Technology Group, 2010).

As a system becomes more and more complex, so the probability of errors or oversights increases. These can have major repercussions and lead to failures of a product or a project. Also the average cases as usually portrayed through graphs and spread sheets don't always hold in real implementations (tmn Simulation, 2011).

If one were to consider using the average request rate to design a system such as a question and answer server made open to public users, that being human, and studies towards the design of this server showed that the average number of requests made to this server was four a minute, then this means that this server would have to receive, process and respond to a question once ever fifteen seconds.

If this server were to receive twenty questions within five minutes, using the average rate would show that there was a request made every fifteen seconds and thus the system would work. The problem with this is that these requests may come in staggered bursts, or worse still, all at once, followed by nothing for the rest of the duration of the five minutes.

In this case the server would fail even though the average would still show that questions were on average received once every fifteen seconds. This where the need for simulation arises, as if this system had undergone simulation before a real implementation, one would've picked up on this rather trivial oversight and been able to account for it.

One should consider simulation when there is a high volume of production, a high cost involved, a change in an already existing system, or when there is a high level of entropy (tmn Simulation, 2011), as shown from the question and answer server which was interfaced to human input.

In the case of high volume of product or system use, this first of all usually involves large investments of time or money by a company. This is also due to the fact that if there is a problem in a wide spread product, there is simply too much of that product to ever support each one. This can lead to entire client bases moving away from the product due to the producer not being able to mitigate the problem that arose.

A high cost product is also one that needs simulation. This is normally because an investor can't afford a second development phase of the system or product in development. Failing on a first attempt can often set starting companies or ideas far back enough for them never to prosper or even make it into real world implementation.

Changes in an implemented system should also be simulated. This is mainly due to the developers behind the change not being part of the original development team. This means that subtleties or just plain misunderstanding of the current system can have negative effects when modified to work with new components or visa versa. As such, simulation should be run before updates are made live.

Lastly, there may be a high chance of random entities providing input to a system. The need for simulating this is made clear by the question and answer server example above. One needs to be especially careful with systems that deal with high levels of randomness, especially when time based. As such, simulation is a must.

Simulation is a good idea even when the above situations are not applicable. Simulations allow for error mitigation and also for a good idea of the performance of the system that is to be implemented or updated.

This being said, one should not believe that because a system has undergone simulation that it will not have any errors. Unfortunately there are some errors that can only be found when a system has already gone live and so one must remain cautious (Santner *et al.*, 2003).

## 2.2 Approaches to Network Simulation

Packet routing is simply the process of moving packets from a source host to a destination host through a network (Leighton, Maggs, and Rao, 1994). A route may simply be a direct connection to the destination host or it may involve a series of hops through routers, hubs and even load balancing systems.

Routing protocols are classified into three major classes, which are interior gateway routing through link state routing protocols, interior gateway routing through path vector or distance vector protocols, and exterior gateway routing. The purpose of a routing protocol is to prevent loops forming when routing occurs within a network, and if such loops occur, they should be able to break them up. They also deal with selecting the best routes around a network based on a predefined cost on each link which helps reduce latencies and traffic within a network (Baker, 1995).

As more hosts are added into the network so routing becomes more complex, and the chance of a single node being connected to only one host becomes negligible. This makes error testing increasingly difficult, as other systems that are not directly part of the system one may be testing can get affected by error and robustness testing on a point of the network. This can lead to loss of throughput, higher latency and even complete denial of service (Mahajan, Spring, Wetherall, and Anderson, 2003).

Network simulation software was developed to handle the needs of systems in development that required live network testing, but were not yet ready to be opened to a network due to extraneous behaviour that may affect the system or visa verse. Network simulators could allow for creation of network traffic and complex routes without ever connecting to a real network. Some hardware implementations of network simulators are Netropy and Linktropy[1] by Apposite Technologies (Technologies, 2012) and products such as Hurricane, 8400E and 4XG Network Simulator[2] by Packet Storm (Communications Inc, 2012).

Network Simulation is also required at education and research levels, as it is more secure and easier to control a network set up by an invigilator or institute than it is to use a real live network. This also reduces costs in that an institute no longer needs to fund the required amount of hardware and space needed to create a network in which a class can be educated on. NS-3 was designed as a software implementation for this purpose, and allows for scalability of the networks simulated size and also allows connection to real networks that exists (National Science Foundation, 2012c).

[1]http://www.apposite-tech.com/index.html
[2]http://packetstorm.com/psc/psc.nsf/site/index

Another specific that needs to be considered when designing Network Simulation Software is the protocols it has to support. Wireless has become increasingly popular in recent times and this area of research is still developing (Adachi, 2001). This has lead to development of many different wireless protocols, the IEEE 802.11 and 802.16 families of specifications, and as such requires different handles in simulation of these protocols.

Wired protocols are easier, as most widely used protocols have become set in their implementation. However, development over the years towards the medium in which packets are sent over have seen vast improvement. This has lead to benefits such as more reliable connections, less or no electromagnetic interference and higher bandwidth (Popov, 2009).

## 2.3 Considerations in Design and Implementation

Bandwidth, latency and packet loss also come into play when designing and implementing a network simulator. A Network Simulator should be able to handle a wide spectrum of connections to itself, of which all may have separate distances and run on different mediums. This leads to different levels of packet loss, latencies and bandwidths. This is particularly noticeable when comparing a wired to wireless connection. Wireless connections typically have higher latencies and higher rates of packet loss than traditional wired connections (Zhao and Govindan, 2003), this depending on distance and what mediums are between each wireless links. This means that the average statistical throughput in one environment can be completely different to another, and so adds to the complexity in design and implementation of wireless protocol simulation. Furthermore, it is to note that this research is intended to create a proof of concept with the intention of simulating common corporate/consumer bandwidths, delays and protocols.

Bandwidth speeds range from as low as 300 bps (bytes per second), as implemented by the ITU V.21 protocol, up to 159.25248 Gbps (Gigabits per second), as implemented by the OC-3072/STS-3072/STM-1024 protocol[1], and currently implemented in today's networks. This speed is near impossible to simulate on publicly available hardware and as such this makes simulation difficult. This is because not only does the system have to account for different synchronization speeds, but also the fact that the faster the simulated connection the more hardware intensive it becomes. Implementing such handles on a network simulator increases the cost and complexity of the system as a whole, while causing a greater level of abstraction which can lead to loss of base level features or complications in use of them.

[1]http://www.fiberoptic.com/adt_sonet_sdh.htm

### 2.3.1 Caching

There are approaches used to compensate for heavy load on Network Simulators. The first being the use of a cache. This is usually a server that stores recently or commonly accessed files and web pages within a network in an attempt to speed up networks by handling page request directly instead of requesting the page/file again from the Internet, external host or other network locations. This is implemented by storing meta data of what each cache node contains and what each other cache node contains if there are multiple caches on a network, much like one stores an entry in a database.

A request for a file, web page, or data request can then be looked up within this meta data and whether it exists on the cache node or any neighbouring cache nodes. The request can then be serviced on the local network, thus reducing the delay of requesting the data from external servers (Cieslak *et al.*, 2001). In short, caching is a method used to keep common resources close to the systems requesting them.

This has been seen to work and has been tested at network rates up to 100 Mbps (Megabits per second), but utilizing the same size caches on higher bandwidths tend to see the implementation of caches fail (Newman, Minshall, Lyon, and Huston, 1997). If one were to increase the cache servers to deal with higher bandwidth rates, this would lead to direct impact on the memory within the simulator and would result in thrashing, which would lead to an overall decrease in performance.

Implementing cache policies in this case would increase overall performance as they are designed to reduce thrashing within the cache. They do this by managing memory in a simple and logical manner (Bhuyan and Wang, 2001). These help reduce memory requirements and in all increase the performance of a network.

The use of caches reduces the bandwidth consumption of an institution or business, as requests will be handled internally rather than through an ISP (Internet Service Provider). This saves money and frees up resources on the network.

### 2.3.2 Hardware

The next approach is for one to improve the hardware of the host system running the routing simulator. A simple improvement would be to increase memory on the host system. This would allow for greater sized caches at each node and larger, more detailed routing tables to be stored. There has also been a proposed design by Wolf and Turner (2001) for hardware implementation that runs multiple network processors, each with its own cache and memory, which they suggest will overcome the hardware level bottlenecks created by conventional methods of Network Routing. This approach may be adapted into the hardware of a Network Simulator and should not be disregarded as useless in ones design of the overall system.

One may consider load sharing over multiple processors within the system as well. This will omit the need for task specific hardware and one can look into using low cost publicly available hardware to solve the problem. Even though the quantity of hardware and space requirements of the simulator will most likely increase, this is unfortunately a trade off one may have to make.

However, one may only want to simulate packets being sent and received at end points of a system. This omits the need for simulating node hops within the network and as such reduces the actual load on the simulator. This is commonly implemented by receiving a packet from one end point and transmitting it to another end point without any intermediate processing of hops in the simulated network. Another addition made to this is generation of packets from pseudo hosts within the network, by having a single host or node generate traffic as if it were an entire network, and hence create the activity of larger networks with lesser hardware requirements. This reduces the need for actual hosts and allows for testing of a specific host systems that are designed to run on large networks (Baumgartner *et al.*, 2004).

### 2.3.3 Delay

Simulating delay on a network is something that is often over looked in design and implementation. The reason for this is that most of the time the actual results are not affected by the implementation of delay, and so it is rather left out to cut down on time taken for development and time added during run time of the tests. There are however, uses for such features such as testing systems that require a certain level of throughput and need to be able to handle delays. That being said, delay is very difficult to simulate accurately as it can be generated not only on time taken to travel through the medium used, but also when processing, internally buffering and putting the data down onto the wire.

Another point that is raised by The VINT Project group is that protocols require extensive testing to ensure that they run under all conditions (Breslau, Estrin, Fall, Floyd, Heidemann, Helmy, Huang, McCanne, Varadhan, and Xu, 2000), this includes delay. This is not a problem with small networks as time taken for processing, buffering and transferring the data over a medium is negligible. The problem is that as the network becomes larger, the data takes longer to transfer between nodes. If a TCP (Transmission Control Protocol) like protocol were to implement resending of data that is not acknowledge within a time frame (Kurose and Ross, 2010), this is due to the protocol assuming the destined host did not receive the sent data in a set time frame and resends it. This implementation would work well on smaller networks but as network delay increases, there is no way to tell if the data is still being received or if it lost as the time to transfer the data could take longer than the time frame used to assume the data did not arrive. If delay is not implementing this correctly in this case, it can lead to excess traffic in the network as data is resent when it should not be.

### 2.3.4 Packet Loss

Like delay, packet loss is also a feature which is often omitted from network simulation software. The reasons are similar to that of delay as it adds time to the research project both at development of the software and run time of it. This too is very purpose-specific and mainly used in testing of systems and protocols that are sensitive to packet loss, such as file servers or more specifically the FTP (File Transfer Protocol) and similar protocols.

Packet loss comes into play with systems that stream data. These systems must be able to recover from lost data without delaying the rest of the system as a whole. Protocols too must be able to deal with packet loss and as such must also have mechanisms built in to recover from such mishaps (Breslau *et al.*, 2000). Examples of this are implementations of audio streaming and video streaming on frameworks such as HTML5 and Adobe Flash, as they can actively stream media across the Internet and have mechanisms built in for packet loss detection and recovery (03b and Sofrecom, 2012).

### 2.3.5 Packet Sniffing

Detection of routing packets entering a simulated network is another consideration that is taken in the design and implementation of a network simulator. If one relies on actual external hosts to feed information into a simulated network, the actual system that simulates the network needs to be able to discern between which packets are directed to which simulated nodes or hosts within the system. This becomes more complex if the system on which the simulator runs does not accept any packets that are not addressed to it as each address is unique (Tanenbaum, 1996).

One can set a machine to accept packets not addressed to itself by setting the systems network device into promiscuous mode. This is not available on all network adapters though. The implication of this is a higher load on the CPU and memory of the simulating system, as it has to process every packet visible on the wire, but allows for capturing of packets that would not have been before.

With this in mind one can define the addresses in which to receive packets from for simulation within a network. This is particularly useful as one can now run a routing simulator on a live network and define which packets should be simulated and which should not. This allows for only the targeted hosts to be affected and not every system on a network.

Also, one may want to sniff packets within the routing simulator. This can allow for modification or further control of the packets route while under simulation. One can implement functions to handle predefined cases such as delivery of a single packet to multiple targets instead. This can also allow for other systems to analyse packets within the network and gain further results from a simulation.

## 2.4 Routing Protocols

BGPv4 (Border Gateway Protocol version 4) defined by RFC 4271 (Rekhter, Li, and Hares, 2006) is a protocol designed specifically to handle exchange of routing information between gateway hosts. Each gateway host stores a routing table that contains all known nodes to that gateway host. Along with each router is stored the address of that router, a list of all nodes that it is connected to, and a cost that is used in determining the best path for a connection to be made on (Rekhter and Li, 1995). This protocol is used mainly in inter-country and inter-continental routing and builds up the routing backbone for the Internet.

Simulating the BGP is made easier with the implementation of CIDR (Classless Inter-Domain Routing) defined by RFC 1518 (Rekhter, 1993). This makes use of a most significant portion and least significant portion in standard 32-bit IPv4 (Internet Protocol version 4) address. The most significant part defines a subnet, a portion of the Internet to which a set of hosts belongs, and the least significant part is used to define the individual hosts. The major difference is the size of the routing table which is much larger. This however may lead to a decrease in memory usage, as if one were to store all known connections for all known nodes connected in the network to the current gateway host in the gateway hosts internal table, one could use this to direct a route for a packet rather than storing routing information by use of a separate table in each node within the network. In this way redundant data can be removed from the network. An example of redundant data would be simply shown by considering two connected nodes in a network. They each would store the connection to one another in their own routing tables, this means that there are two entries stored in total for one connection. If one were to store this data in a gateway host it would only keep one entry for the connection between the nodes and so reducing memory costs.

The company bgpVista[1] has produced a lightweight even driven BGP simulator named simBGP. Narwhal allows for configuring of link delays, packet loss, processing delays, protocols to be run on the network, as well as bandwidths. This, along with being able to set up the network as one would require for testing purposes, makes it a very well implemented simulator (BGPVista, 2009).

There are also simulators which require the ability to accept packets not addressed to it. Libpcap is a library which utilizes the promiscuous mode packet capture. Promiscuous mode packet capturing allows for packets addressed and not addressed to the host machine which it is running on, to be captured, read and manipulated down to a byte level (Jacobson, Leres, and McCanne, 2001). The process of capturing a packet of the wire that was not addressed to the host computer is known as "sniffing". This gives an easy to use interface onto any captured packets, and as such one can easily use this library to discern between routing packets or not, and as such simulate any response that one would want to.

The act of sniffing a packet however does have its problems. It becomes nearly impossible to sniff packets on a switched connection, as the switch/router directs the relevant traffic to the destined host only, and does not send it publicly onto the network like conventional hubs. This means that a promiscuous mode enabled network adapter has nothing to sniff on the wire except its own packets that are directed at it. However, data sent out can still be targeted at the host as the switch will still send the data to the destined address.

Libnet is particularly useful in doing just that. This library offers easy to use functions that allow for building a whole customized packet. This includes, source and destination

[1]http://www.bgpvista.com/simbgp.php

IP, source and destination port, payload, checksum and even changing the bits that go in the zeros part of a TCP packet (Wang, 2003). This can be useful if one wants to spoof route tracing packets and make a node or host believe that it is accessing a real time network.

The use of Libnet again is a good choice as it was designed as an attempt to standardize the way in which protocols on a network are structured. With this in mind, Libnet was created to be lightweight, simple and usable over many platforms (Wang, 2003). This implies that its memory and CPU footprints are minimized and gives great functionality for little resource cost.

Using a combination of Libnet and Libpcap would make for an easy implementation of a network simulator with the ability to pick up packets off the wire and redirect them to where they need to go within the simulated network. This would be achieved by using Libpcap to retrieve a packet from the wire, break it down into its relevant parts that need to be analysed, then a decision can be made as whether the simulator should act on the packet received or not. If the simulator does act on the packet, in terms of generating more network traffic, it can use Libnet to create a packet that is relevant to the context of the packet received, and generate a response to the host machine that the packet originated from if need be. The only drawback to this is due to the network adapter running in promiscuous mode. The more hosts connected to the network simulator, the more CPU intensive the process of sniffing and processing packets becomes.

## 2.5 Collecting Routing Data

A simple tool to enumerate routes around a network on a most platform is traceroute, the implementation for a Windows platformed host is tracert. This program is used by passing it a series of options, if any, and then an IP or host name as an arguments. These options range from setting the hop limit between nodes to defining what protocol to use when performing the traceroute. These protocols range from using standard TCP, UDP and ICMP route tracing methods to using IP an alternate as defined in RFC 1393 (Malkin, 1993).

To perform a traceroute using ICMP one can pass the command "traceroute -I -m 5 www.google.co.za" to a terminal. It would execute a route trace to "www.google.co.za" with options "-I" and "-m 5". "-I" tells traceroute to use ICMP as the protocol for tracing with, and "-m 5" tells traceroute that the maximum number of hops to trace to is five. This is shown in Figure 2.1.

```
test@Rhodes: $ 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 strubencore-maincampus-1.net.ru.ac.za (146.231.2.25) 2.283 ms 2.290 ms
2.345 ms
3 datacentres-1-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 2.1: Use of traceroute with ICMP and Maximum Hops Set to 5

Similarly one can perform a traceroute using TCP or UDP by simply specifying the option "-T" or "-U" respectively instead of the -I option. These are done like so, "traceroute -T -m 5 www.google.co.za" and "traceroute -U -m 5 www.google.co.za". It is to note that traceroute usually performs its traces using UDP and that setting the option to "-U" is basically forcing traceroute to use its default option.

These programs result in showing every hop in the route taken, along with delays and any IP address changes along the way. This becomes particularly useful when looking into recreating live networks, one can simply trace the routes to every known host machine on a network and then recreate the network using the routes returned. Caida[1] is an association that can aid in doing just that. They collect data about the Internet from average delays and packet loss to tracing routes, furthermore they make this data available for research on request (CAIDA, 2012) which will be used in testing of the simulator produced in this paper in Chapter 4. This can be particularly useful in recreating a large portion of a network based on the Internet as a model which can lead into DoS (Denial of Service) attack testing and worst case scenario recovery tests in disaster simulations.

The releases of Microsoft equivalent of traceroute is a program called tracert. This application yields the same function of that of traceroute, however it does have a subtle difference. This is found when looking at the default protocols used to perform route tracing.

Traceroute uses UDP as its default protocol for route data collection. This is done through sending UDP packets to a port that is expected to be unused (Jacobsen, 2001). When

[1]http://www.caida.org/data

the destination is reached the program gets a response that the destination port is closed, as it is not in use, symbolizing that the destined host did indeed receive the packet.

Tracert instead defaults to ICMP. This traces routes in a similar fashion to that of traceroute. It sends out ICMP echo packets with incrementing TTL's (Time To Live) (Cisco, 2005). As each packet expires tracert receives back ICMP packets representing that the TTL Expired, the packets TTL reached zero. The host in which the TTL Expired packet is received from gets marked in the route according to the original TTL set on the outbound packet. This is exactly the same as traceroutes UDP method.

The part that differs is at the destination, where instead of the destined host responding with an ICMP packet stating that the port is closed, it instead responds with an ICMP packet representing an echo response. Tracert then detects this and it knows it has reached its destination.

## 2.6 Network Creation and Packet Routing

When creating the nodes of a simulated network within the network simulator, a general approach is to use an object orientated design to create nodes within the network, each with their own routing table of which nodes they are connected to (Wang, Chou, Huang, Hwang, Yang, Chiou, and Lin, 2003). As stated in Section 2.2 this becomes very memory taxing but as it turns out it also takes up a lot of CPU time (Wang *et al.*, 2003). A solution for this is creating a over-lying super routing table which contains the meta data and state of every node in the simulated network. If this design is implemented with a database, such as SQL (Date and Darwen, 1987), it can also lead to decreases in both memory footprint, as unused nodes will be left on hard disk, and CPU usage, as there is no longer a need to service all nodes in memory at once. The overheads of reading a node to memory though, will have to be taken into consideration when calculating delays, as access times on a hard disk are slower than access times to memory.

The IP (Internet Protocol) also needs to be factored in when creating a network, as it is now becoming more and more common for a network to run a hybrid of IPv4 (Internet Protocol version 4) and IPv6 (Internet Protocol version 6). The IP handles routing around networks and more specifically the Internet. Every device that is connected to the Internet needs to have its own unique IP. This is because an IP address is used to locate the device within the network. If two devices were to have the same IP address on a network, this would cause a conflict when routing the packets to the correct host, as the host which the packet is actually destined for could be either one.

The main difference between IPv4 and IPv6 is the address space. IPv4 allowed 32 bits for an IP address, this allowed for 4294967296 (or $2^{32}$) unique addresses (Postel, 1981a). At the time this was thought to provide more addresses than would ever be required until February 2011 when IPv4 address exhaustion occurred as a result of the IANA (International Assigned Numbers Authority) allocating the last available IPv4 blocks between the five RIRs (Regional Internet Registry) (Smith and Lipner, 2011). IPv6 was created int the late 90's to solve this problem by introducing a larger address space by allowing 128 bits for an unique address(Deering and Hinden, 1998). This is equivalent to $2^{128}$ unique address which is $2^{96}$ times larger than that of IPv4's address space. This vast amount of extra address space solves the problem with IPv4 running out of unique addresses.

As such, network simulators have implemented not only support for IPv6, but also for hybridized networks which consist of both types of addressing protocols. The National Science Foundation who implemented NS-3[1] manages this by defining each node as an object, each node then extends from this object, forming up its protocols that it supports, what IP family it belongs to and any other needed information (National Science Foundation, 2012a,b). This allows for an object orientated design and communication between nodes on an object level until any sort of network simulation is required.

As such one also has to consider scaling a simulator as objects are added and connected within the routing simulator. As more nodes and connections are created to and within the simulator, so the need for more hardware to support the added load is required. This is not always the case however, as Moore's law also comes into play. This was a simple observation made by Gordon E. Moore that every two years the number of transistors that can inexpensively be placed on an integrated circuit doubles (Schaller, 1997). This plays a big part in reducing the costs and space requirements needed to set up any system that needs to deal with heavy loads.

Even though this helps when scaling up a network simulator in terms of connections and nodes within the network, one can't go about doubling the size of the simulated network every two years. This is mainly due again to the implementations of routing tables and the structure of the network leading to a $O(n^2)$ complexity (Stone, DiBenedetto, Mills Strout, and Massey, 2010). The result of this complexity means that every time one adds a node,

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

the load cost for adding that single node gets larger and larger because it has to tie back into the routing table and structure of every other node. This means that if one implemented a global routing table containing all nodes in the network. For *n* nodes, each node would have to know about the other *n-1* nodes.

If one were to take the approach of assigning each node their own routing table, one could also spawn a thread to handle that node. There will be no need for mutexes within each node as each node will have an instance in memory and would only deal with its own requests and responses. This conceptually is a very neat way to implement a network simulation of any scale as it removes global variables and focuses more on each node as a separate system rather than a network as whole. This also allows for adding of nodes very easily to the network, as one can simply spawn a new thread, give it its connections and let it work out its routing table for itself.

## 2.7 Hardware and Software Development

Another key point to this approach is that the development of multi-cored systems, heavily threaded applications are becoming less strenuous to run. Also cheap commercial processors available to the public have features such as hyper-threading which allows multiple threads to run on a single core in parallel. This, combined with multiple cores on one processor, enhances the ability for modern day inexpensive processors to deal with multiple thread based applications (Intel, 2012) and can be used to up-scale network simulation cost effectively. This makes complex simulations a viable option on commodity hardware.

Research and development into simulation, and more specifically network simulation, has shown more and more use, not only in research, but also in testing and application in commercial and public sectors. As such, it is continuing to grow, and with development not only in better algorithms but also in hardware, application specific hardware and low cost hardware with high availability will continue to push this field of research, and will lead to more efficient and secure networks in the future.

Furthermore the approaches taken can be broken up into three main categories, being hardware, software and hybrids which are a combination for both hardware and software. Hardware approaches, as produced by Apposite Technologies (Apposite Technologies, 2011) and Packet Storm (Communications Inc, 2012), are in most cases less cost effective but easier to set up as the hardware is designed to handle network simulation.

Software approaches are slower in execution due to the fact that the hardware in which the simulator runs on is not designed specifically for network simulation. Although this is a more cost effective approach, it can lead to complicated software, as configuring hardware for which the original intention was not for network simulation can get tedious. However, a software implementation does allow for modularization, as it is far easier to pull out a a software module and stick in a new one than to go about designing a new piece of hardware to fill the required gap.

A hybrid implementation does have its advantages as well as its disadvantages, being able to modularize software written in a language that compiles to byte codes specifically optimized for a hardware instruction set. This can always be used to offload tasks, such as checksum calculation, to hardware specific devices to free up the core of the system. The disadvantages of this however, are bottlenecks, specifically the time it takes to offload the data required for processing to another device. This may be unnoticeable under light load but can cause major slow downs as the systems takes on a heavier load.

## 2.8 Reasons for Packet Routing Simulation

The need for routing simulation rather than emulation is that emulation deals with replicating a platform, where simulation allows one to change features about a platform. This becomes useful as one can customize a simulation to help achieve result that by emulation would take longer periods of time, or in the worst case, no results at all.

These network based simulators are usable in a multitude of ways. The areas of use are primarily education, business and research.

In terms of education, using a routing simulator in place of a live network has its benefits as mentioned in Section 2.1. This is extended when one considers security. There are many sensitive systems in place on a live network and within these systems there is sensitive data. Allowing for modification of these real networks by untrained people can result in security flaws and loss of data.

Further more it is far easier to set up a lesson or specific environment via a network simulator. This also gives options of resetting or restoring to a point in simulation. Features that are not as easily acheived through real time hardware.

Touching on the business environment, one would not want systems to crash because of a new system being tested. This can result in major losses to the business as a whole.

Instead, using a network simulator to test a new system on before it goes live, gives the flexibility required to develop new systems and maintain them all while having the peace of mind that all bugs can be corrected before the new system goes live.

It also helps in development, as recording network traffic is far easier to create as a feature for a network simulator than try and record on a live hardware level network. This will allow for easy tracing of errors and bugs that arise in the systems being tested.

These simulator can also be used to replace real networks. They can be used in setting up networks with firewalls and access rights, replacing the need for switches and hubs throughout a building. Paths and specific controls can be managed from a single point rather than having to access multiple nodes throughout a building or institute, a task that can get very convoluted very quickly.

Knowing about the route a packet takes within a network is not knowledge to be taken lightly. Inefficient routing can lead to excess traffic and increased delays. Also, this knowing about routes within a network can help with redundancy, either creating redundant connections to increase reliability of a network, or removing them where they hold no benefit. It can also help in selecting the best algorithm use within a network to increase efficiency. This is all due to the fact that networks are formed from links, and built up from these links, and if the links are inefficient in the way they communicate, the results compound into the structures built upon these links.

## 2.9 Summary

This chapter takes a look at what packet routing is and how it is done. As such we find that routing algorithms have grown over the years. This is due to more effective algorithms being created and the need to reduce memory requirements within nodes in a network.

It has also shown methods to speed up network transactions through caching, hardware and software based solutions. We see how delay has an effect on the workings of a network and how it is important to simulate this common feature within all networks to some extent. We also find how packets that arenâĂŹt addressed to a host can be received anyway using a library such as libpcap.

The use of traceroute is also shown and how one can use the data enumerated from it in collection of data for real network configuration. This then leads into node creation and connection and what effects this has on the memory and CPU of the host system running the simulator.

In this chapter's closing we touch on the approaches taken through hardware, software and hybrid implementations of packet routing. Finally we gain some understanding of the uses and needs for packet routing simulators and other network based simulators through a few example cases.

The next chapter builds on the knowledge obtained from this literature review and presents a means for designing and implementing a packet routing simulator.

# Chapter 3

# Design

From review of the approaches taken in the previous chapter and problems that arise with each implementation towards routing simulation, it is to note that memory usage is a common problem. This, as well as the need for large amounts of computational power in some implementations, so much so that hardware specific devices have been made for the task.

With this observation, the approach taken in this research is to address the issues of memory usage, CPU usage and general throughput under different sized network simulations and network load. This will also be tested under a software approach and use of hardware that is readily available to the general public.

It was expected that there would be a trade off between memory and CPU load, as opting for a more dynamic system rather than a pre-calculated system would result in a heavier load on the CPU. This, as it is non-static, will not be stored in memory and thus be less taxing on a low memory host.

On the topic of memory, there are some objects that are difficult to remove, compress or in some cases do anything about except take the memory or performance hit. Such objects in creation of a routing simulator are node IP addresses, node routing tables and the actual packets being passed within the simulated network.

With these problems defined, this chapter will be structured around these observations. This will start by first looking at how a node will be created and accessed. The means in which its data structure is also represented will also be discussed.

This will be followed in Section 3.2 by how an appropriate choice of data structure can allow for added functionality and development of such features. This will then lead into Section 3.3 which discusses how to best utilize a multi-threaded system and how packets are transmitted by interfacing to the host system's hardware.

Configuration of a network will be addressed in Section 3.5 followed by how a packet gets routed through a configured network in Section 3.6. The architecture in which this routing simulator is intended for and what operating system on which it is designed to run on is discussed in Section 3.7.

The language in which it is to be programmed in is also taken into consideration here and reasons for its choice are given in Section 3.8. We then follow on to define the boundaries of this routing simulator through defining its environment on which it is designed to run in Section 3.10.

This will tie into why a terminal based approach was taken in Section 3.12 and how threads are spawned from this main thread, that being the terminal. This chapter finishes up by touching on the finer points of interfacing to the network and threads that the host system has access to. Lastly touching on the actual physical environment that it is intended for and how it should respond to other systems on it.

## 3.1 Node Memory Structure and Access

A problem to be addressed is access of each individual node within the simulation. Depending on the approach taken, one can save memory with simpler forms of node reference in trade-off for accessibility or time taken to access a node directly.

As it will be more useful to access a node directly, a hash table was used when approaching this system. This allows for near instant time access to any node within the network, although there is a slightly higher CPU load in finding the hashed node, and a higher memory footprint, as a bucketed hash table has a more complex structure than that of a one to one variable to object reference to a node in the network.

However, when comparing the look up times between that of a link list implementation to a hash table, it is clear that there is a reduction in time needed to look up a node in the simulated network. Furthermore the use of a tree structure such as a binary search tree allows for O(n.log(n)) complexity when searching for a node, but also this holds much of

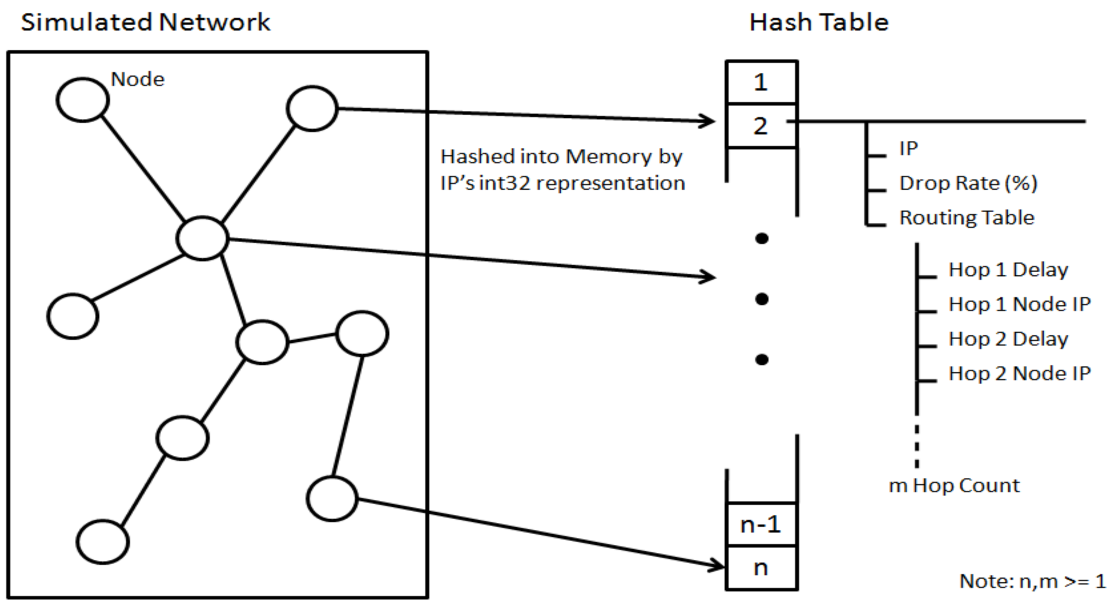

Figure 3.1: Hash Table Structure of Simulated Nodes

the characteristics of that of a link list. The only difference is that the data is ordered logically by smaller, larger or equal, but essentially a binary search tree is a link list with two connections rather than just one.

When comparing the look up times of a binary search tree to a hash table, the hash table is still quicker. However, the memory footprint of a binary search tree, like a link list, is less than that of a hash table.

Furthermore if one were to take a simpler approach by defining a static array to hold all node objects within the simulator, this would lead to a waste of memory. This is because space for every node in a 32-bit range, due to IPv4 addressing, would be created where every node would not be used. If one were not to declare space for all nodes, this would lead to a limitation on which IP addresses could be used and trying to access anything out of the bounds of the given range would result in critical failure or unexpected behaviour as one would access memory locations beyond the scope of the array.

### 3.1.1 Architecture Specific Optimization

One must also keep in mind, however that omitting the use of arrays could lead to a performance reduction, as when one declares an array, the array is stored in a block of memory without any gaps. This knowledge can furthermore be exploited to increase performance when passing nodes up to the CPU through the memory hierarchy, through spacing nodes in the array to allow blocks that are loaded up to the CPU to be done in a more effective manner.

This however, is architecture specific, and as such requires thorough knowledge of the architecture for optimization. This becomes even further troublesome when one needs to modify such exactness for ever platform in which the software is intended for use.

What really deters use of such optimization is the general running of the simulation software, as nodes visited in a route are non-linear, and so the chance of accessing a node in the same memory block on the next hop is highly unlikely. Also, one cannot determine the IP of the next hop before hand for every network configuration that has and will exist, and so optimization in that respect is also nearly impossible without a lot of guidance.

With this in mind, the approach taken will be towards optimizing the general cases and prevention of bottle necks, rather than getting down to optimization of specifics for set platforms and environments.

The reason why a hash table approach was taken was for this reason, as it runs across all platforms, and performance is relatively constant. It allows for a higher level of memory management and does not require platform specific optimizations. This, and the fact that overall it is the most balanced data structure when compared to a static array as it uses less memory, and when compared to a link list, it has far lesser CPU load requirements on a node look up.

## 3.2 Functionality

Functionality will also be addressed in this routing simulator. The reason for this is that being able to directly reference a node at any point in execution, and change parameters such as packets drop ratio or processing delay, makes for a more realistic simulator, and as such more data that is producible by a wider range of tests.

Delay is a preloaded-loaded average that then takes on added realism through the use of entropy. This is done through firstly taking into account the processing delay within the host system, and then adding or removing a randomly generated number.

Each node in Narwhal will contain the route to itself from a top down point of view. In other words, as this is a core routing simulator, we are only concerned about the packets delivery from the core of the network out, and not in its route to the core of a network. The delay to the core however, should not be omitted, and as such will be dealt with as a general hop with delay that will represent transferring a packet to the core. Discussed later in this section, a more dynamic approach is taken and comparisons are made to this static routing system.

As this simulation works on an IPv4 bases, each IP within the route is stored as a 32-bit integer representing the IP addresses followed by a 32-bit integer to represent the delay between hops. This means a total of 8 bytes is required to store a hop in the route to a node. To this will be added the nodes general settings, such as processing delay and chance of packet loss. These will be represented by 32-bit integer values as well, and stored in a global table, removing the need to store a value multiple times in every route that uses a specific node.

## 3.3 Multi-Thread Implementation

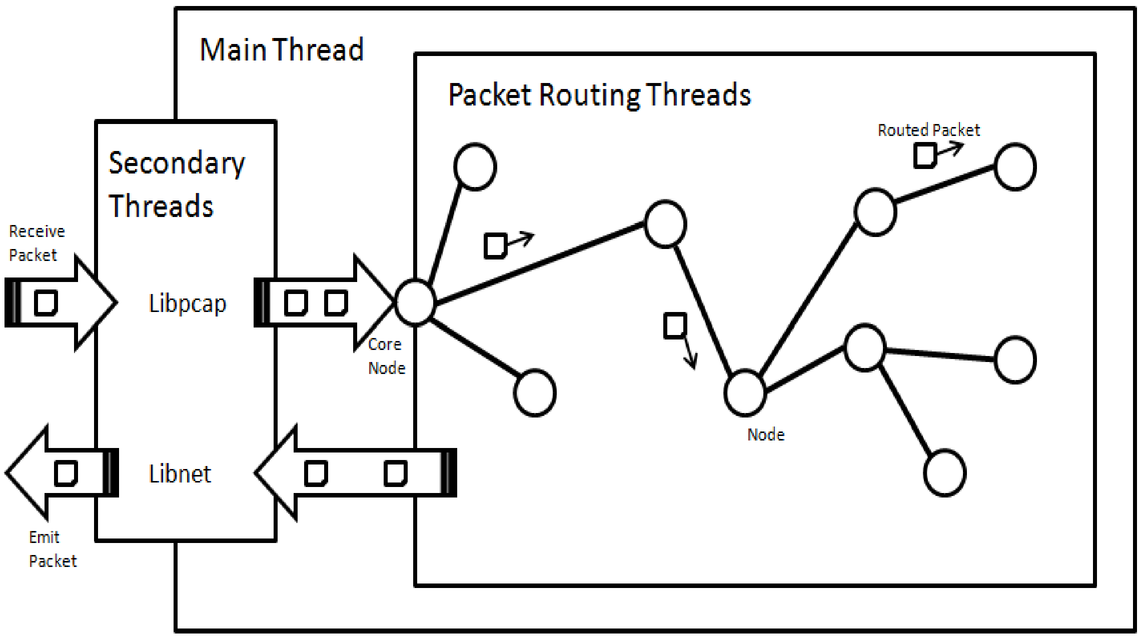

Figure 3.2: Simulation Run Time Thread Overview

To make use of modern multi-threaded processors, Narwhal takes on a threaded approach to packet handling. Each received packet is allocated a worker thread, or packet routing thread as labelled in Figure 3.2, that simulates each hop on the route to the destination

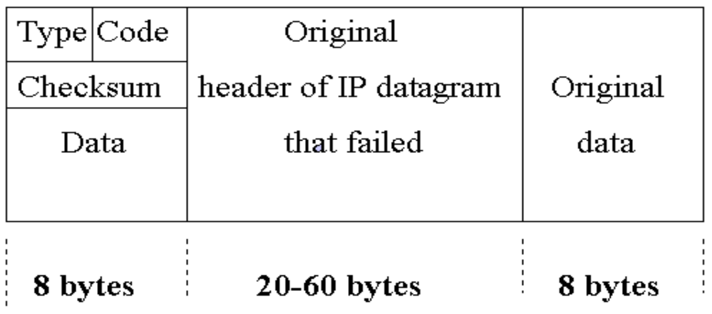

Figure 3.3: ICMP Error Message Layout as Defined by RFC 792 (Postel, 1981b)

host. At each hop the chances of packet drop, processing delay and hop delay are taken into consideration. In the event that a packet is dropped or the destination port is unreachable, the relevant ICMP message is returned to the packet's source host.

The TTL (Time To Live) of a packet is calculated through each hop and, if it reaches 0, the packet is then dropped and a ICMP packet, as shown in Figure 3.3, is returned to the host with a message telling the host, that the TTL of the packet has expired (Postel, 1981b).

This allows for programs such as ping and traceroute to run successfully on Narwhal, as they largely make use of the ICMP through exploiting the information in the IP header that is attached to the ICMP packets returned to it.

When a packet has finished being routed through the simulated network and an appropriate outcome has been produced, the thread will de-allocate its memory and return back to the thread pool in wait for a new packet to be handled.

Also shown by Figure 3.2 are secondary and main threads. Secondary threads are assigned to the libpcap, which receives packets, and libnet, which emit packets, onto the network. In this way they do not contend for resources within within a single thread but instead are assign each their own. The main thread is directly linked to the terminal, later discussed in Section 3.12, and handles the creation and managing of both secondary and worker threads.

## 3.4 Packet Transmission

Packet ejection is also something that should be considered in the simulator. This is approached through binding a host on the network to a node within the simulator. This way one can change rolls of a node by simply rebinding a host to it, or nodes can share a single functionality from one host by binding multiple nodes to the single host with the desired functionality.

This method works by routing a packet until it reaches the destination, this is assuming it does reach its destination, and then looking up if the node is bound to a host connected to the simulator. If a host is bound, the IP is traded out for the now destined bound host, and the checksum in the IP header is then recalculated as to keep the packet consistent. The host now accepts the packet, and if the host responds to the node in the network, the process is repeated when and if the response reaches the now destined node.

This was mainly implemented to address the fact that if hosts were connected to the simulator directly, and referenced directly by nodes on the simulator that had the same IP as the host. This will result in a packet being sent straight to the a host, as they can see each other, rather than the packet being retrieved by the routing simulator and then ejected from the simulator to the destined host. Introducing host to node binding allows for one to mitigate this problem by not sending packets addressed directly to destined host that the source host can communicate with directly on the network.

Another benefit of using a host bound system is that this abstracts the need to know exactly what device is connected to the IP. All that it has to be able to do is handle a network connection by receiving and sending packets. This allows virtual machines to be used instead of actual hardware based hosts. The virtual machines can even reside on the host that is running the simulator, as each virtual machine, if set up correctly, is seen as host outside of the system hosting the simulator.

To further touch on the use of a multi-threaded system, both packet retrieval and packet ejection are handled in their own threads. This was done to draw away any further delays from the routing algorithm. Even though there should not be any added delays from packet retrieval and packet ejection by handling it in the packets routing thread, it allows for drawing out a global packet sniffer and packet transmitter without needing to recreate handles for every packet that is received or ejected.

This further allows for robustness of the system, as, if an unforeseen problem arises, the system can shutdown the thread and allow for safe recovery from that point, where as blindly creating new threads, as in the example of using the routing thread to create handles to receive and send packets, might result in the whole system falling over if the network device gets locked by one of the other crashing threads.

This allows for scheduling and easier monitoring of packets received and ejected from the simulator. Also it removes any form of resource contention in the creation of handles, and instead allows for a clean interface from, and back onto, the physical network.

## 3.5 Configuration Script Generation

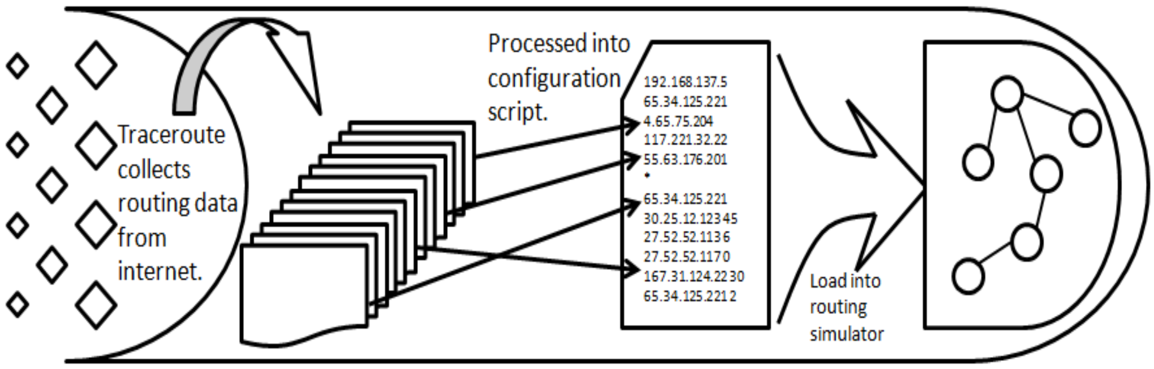

Figure 3.4: Flow Diagram of Configuration Script Generation and Loading

As the network will have multiple threads handling multiple packets at a time, the network's configuration was to be kept as read-only as possible by the threads accessing it. This task was made relatively easy by the fact that a packet being routed only needs to be able to read data of the network to figure out where it is going. A thread handling a packet should not ever have a need to write data or change information about a node or route within the simulated network.

This being said, there are global functions that allow for adding and changing paths in the network. This can be done at run time of the network, and is kept safe by the fact that as a route to the addressed node is kept by the addressed node, all information about how to get to it is stored by it. This means that the route to the node at the given time either exists, or it does not and so can be checked once and then run. This removes the error that may arise from removing a node while it is in route, as if the node is removed, the thread holds its own copy of the node and processing finishes as if the node still existed. Any thread that tries to access that node however, after it has been modified, will get the new modified value of the node.

As for the host the node is bound to, there wont be a problem either, as the binding table is kept separate from the actual network and so, until the host is rebound from the node, it'll continue to receive packets from the given nodes IP.

There are varying methods of configuration and set up of simulators currently available for use. Narwhal makes use of configuration scripts that can be loaded in at run time to quickly setup a network to be simulated. There are also equivalent commands that can be entered to create and modify nodes and routes within the simulator, allowing for finer details to be modified and optimized for the needs of the test being run.

Traceroute also produces easy to read data, and as such a script has been created that translates a route generated as a result of tracing the route of a host into the form that the simulator understands. This can be run over multiple traceroute results to create a single configuration script, and so one can go about simulating real networks that exist today.

The script also allows for translation of the generated scripts into the DOT language for easy graphical representation of the network that is to be generated by a configuration script.

This greatly helps in reducing the times needed to create networks and collect data by hand, as one can now simply automate a script to traceroute a given network and then generate the networks configuration script, and also get a graphical layout of it. This is all done by the computer and as such man power can be spent on other activities while the network is being automatically generated.

## 3.6 Routing

The format of these scripts are as follows. Each node that exists in the network is listed first, followed by a separator to define between this section and the next. This allows for each node to be prepared and memory to be allocated, so that routes and other details can be loaded into them. The next section is the actual routes which are of the form node first, followed by each hop to the node and the delay of the hop.

This format was taken over the initial approach of listing all nodes in the network ,followed by each nodes routing table which consisted only of each node connected to it. This is logically a lot harder to route with, as a node now only knows of the next hop and nothing

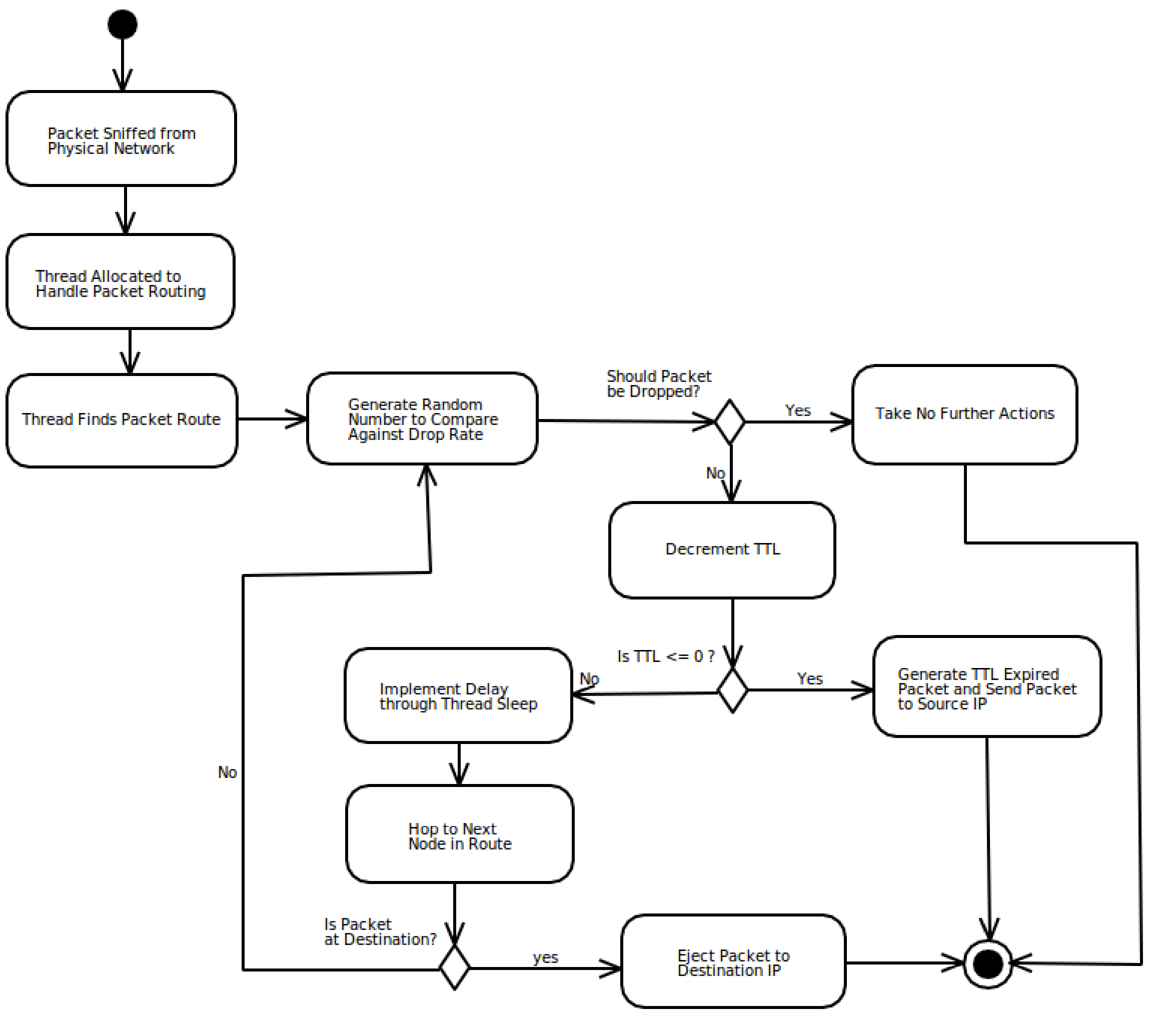
Packet Sniffed from Physical Network
Thread Allocated to Handle Packet Routing
Thread Finds Packet Route
Generate Random Number to Compare Against Drop Rate
Should Packet be Dropped?
Yes
Take No Further Actions
No
Decrement TTL
Is TTL <= 0 ?
No
Yes
Implement Delay through Thread Sleep
Generate TTL Expired Packet and Send Packet to Source IP
Hop to Next Node in Route
No
Is Packet at Destination?
yes
Eject Packet to Destination IP

Figure 3.5: UML Flow Diagram of Packet Routing Logic

further. Also, this form of routing produces a lot more CPU load to determine routes within the network, as a route is worked out in real time. As such, without a BGP nodes or global routing table, this method becomes very hard to determine the correct path without looking ahead in the network, which will produce further CPU load. So this method soon falls apart when trying to simulate parts of the Internet, as it stands currently without the presence of BGP simulation or using extra memory to hold global routes.

Such routing however is not disregarded in this paper, as the memory footprint should be far less than that of the current static routing method. This is largely due to the fact that a connection is not stored multiple times in a network, but rather once by the each node in the connection.

To show this, consider a five hop route to a destination. If one were to use the static method then each node would store every hop to it, so node three would store node one and two to get to itself. When we look now at node five as a destination with static routing, nodes one, two, three and four would be stored in node five's route. As we can see we are repeating connections in the routes to nodes.

Now, if one were to use the dynamic approach, node one would only have to know that it is connected to node two, node two would only need to know that it is connected to node one, and three and so on. With this method it is plain to see that the amount of memory required for a dynamic approach should be less than that of a static one.

In terms of the actual routing logic, both the static and dynamic approaches use the same flow of logic, this is described by Figure 3.5. When a valid packet is introduced into Narwhal, it gets assigned a worker thread to handle the packets routing. The first task of this thread is to find the first hop to occur in the packets route. By doing this the worker thread checks that the packet does indeed have a valid route.

From this point a random number between zero and one hundred is generated and compared to the current nodes packet drop rate. If the generated number is less than that of the nodes packet drop rate number, then the packet is dropped and no further action takes place. This was implemented first as one does not want to process the packets route and then just before the hop is to be made, drop the packet. This would result in a waste of CPU resources.

Next, the TTL is decremented and then checked to be equal to, or for safeguard less than, zero. If either of these cases occur, the packet is then drop, and a TTL Expired ICMP packet is generated and sent back to the source IP address.

Finally, if both packet drop and TTL expire tests fail, then the packet successfully gets routed to the next node in its route. At this point the packets destination IP address is checked against the current nodes IP address. If they are equivalent then the packet has reached its destination and the packet is ejected from the simulated network to the current node. If it is not equivalent, the process is repeated by generating a new random number to compare to the now current node's packet drop rate and continuing from there, as shown again in Figure 3.5.

## 3.7 Architecture and Operating System

The environment for this simulation software took into consideration compatibility, cost and ease of access to the hardware required to run this software. As Intel[1] compatible hardware and Intel Processors are readily available to the general public, and is found in almost every computer hardware related store, it was chosen as the platform for this software to run on.

This is backed up by the wide range of hardware solutions that Intel provides, and their compatibility with many third party hardware devices and software. From a hardware perspective this includes RAM, GPU's, Motherboards and many other common components mass produced by many companies. From a software point of view, many compilers have been, and are still being written to compile software to Intel's native instruction set.

This software includes many operating systems such as Unix (Bodenstab, Houghton, Kelleman, Ronkin, and Schan, 1984) based and the popular Microsoft Windows (Microsoft, 2012) line of operating systems. This means that the likely hood of this programs intended platform should be supported for a significant amount of time, and so be more relevant and useful to further studies in this area.

As such the operating system that this software is intended for is the Unix based Linux. This operating system is well documented and supported well by the Intel architecture. It even extends to many versions, depending on your needs and experience with the operating system.

Linux allows for easy tweaking, when compared to other operating systems, of the general running of the system, such as the maximum simultaneous open port connections (Goodacre, 2011). This is particularly useful as this routing simulator will be making major use of multiple connections. It goes further to allow modification of the kernel to provide optimal functionality and performance on the given architecture.

Furthermore all distributions of Linux are free and easily obtainable through download from the relevant distributions hosted mirror. Installation is also straight forward and so setup of such a software environment is relatively straight forward.

[1]http://www.intel.co.za/content/www/za/en/homepage.html

## 3.8 Programming Language

As the Intel architecture is compatible with many languages, one has to consider which is best for the task at hand. Most languages that can be run on the Intel architecture also have the ability to spawn multiple threads as required, and interface with a network device in one way or another.

This means that choosing a language is more about how quickly it can execute on a given architecture rather than weather it has the basic functionalities to achieve the requirements of this given software.

As such one should consider common wide spread languages that are in themselves well documented and supported, as well as having a well supported library base and following as to ensure their future support. Languages such as C, C++ and Java hold these properties, as well as other languages that are becoming more popular, like Python and Ruby.

### 3.8.1 Interpreted Languages

Interpreted languages such as Python and Java are great for compatibility with different architectures as these languages don't run natively on a host, but are rather interpreted to the native instructions at run time. This, however, does have an added overhead that the instructions need to be looked up and converted to the native architectures byte codes through a commonly implemented look up table. This significantly slows down the execution process and so should be avoided if possible.

Ruby however is known as a interpreted language through the MRI (Matz's Ruby Interpreter). However, one can compile ruby code to a native form, or to an optimized form that is easier to interpret at run time citeruby. This optimized step however, does still not give the full performance of native byte codes as yielded by compiled executable as produced by compiled C and C++ sources.

### 3.8.2 Compiled Languages

C and C++ are more ideal for the task, as the operating system and architecture is already known. C and C++ can be compiled down to native byte codes for this environment, and so ensure maximum performance in that respect.

Furthermore the GNU[1] (GNU's Not Unix) compiler collection, GCC for short, allows for levels of optimization to be specified at compile time. This allows for re-arrangement of data structures and segments of code to best suit the architectures version, and tidy up any inefficient code one may have written in the early hours of the morning.

### 3.8.3 Supporting C as the Language Used

C was chosen when developing this project as the ideals and intended use of C was more suited to the task than that of C++. C follows a procedural style of programming where C++ combines both the procedural style of programming but adds in object orientation as well (cplusplus, 2012). When relating the two to the requirements of this system one can argue that each packet can be seen as an object and should be treated as such however, this is not completely true.

Each thread receives a packet and is then handled in the simulation process but, no thread knows of any other threads state or existence in the network. As such there is no need for a thread to handle multiple objects, or even create any objects in system. Instead, it follows a procedure and makes calls to functions within the simulator before terminating when the thread completes.

In this way the system models the C procedural paradigm and leans more towards being function driven than that of a typical C++ program, which is object driven. This, as well as the program taking a top-down approach rather than the C++ style of a bottom-up approach that consists of building a system from smaller sub-systems, helped to motivate the use of C as a language of choice.

Lastly, this routing simulator works with network packets and network interfaces at a byte level. This is a very low level of programming and allows for little error. C provides such functionality and gives one the ability to work at such a level without to much interference from the language itself, also it easier to port to a hardware implementation then other languages. This being said, unlike a higher level language like Java, there is no bounds checking at compile and at run time which can lead to errors creeping in, that may have been completely avoided when using a higher level language.

[1]http://gcc.gnu.org/

### 3.8.4 Thread Interfacing

Another reason for the use of C is the library pthread. This is a standard C library intended for Unix based systems. This allows for threaded programming through POSIX (Portable Operating System Interface) Threads which allows for easy creation of new threads and handling of threads within a system, this gives threads within Narwhal portability.

## 3.9 Network Interfacing

The C programming language supports network connections at a socket level, and there have been many libraries built around this support. A socket created in C allows for support of both the IPv4 and IPv6 addressing as well as being able to define whether the connection is TCP or UDP based. This is further extended by the ability to accept connections from any address instead of waiting on a specific IP address to contact the server or client. This functionality lets one extend a service to many clients simultaneously without the need to reconfigure a server application.

### 3.9.1 Application of Libnet and Libpcap

Libnet builds on the socket interface that C provides by creating packet frameworks that allows one to customize individual fields of a packet before it is transmitted onto the wire. This means that one can use this library to spoof packet addresses and the data contained within. Also, one has the ability to choose the type of packet that is going to be sent, which allows for spoofing of packet loss through creation of ICMP packets and other network control messages.

In this way Libnet stays true to its motive of bringing a standard for network communication, and speeds up development of network intended applications such as this one.

Also extending this socket interface provided by C is libpcap. This will be the library used to retrieve packets from a network. The reasons for using this library are that it is easily configured, in that it basically only requires a socket handle and a range to listen to, and also installation of this library on a system is trivial.

An added benefit to both these libraries is that they are freely distributed and well documented, both officially and unofficially through example codes that come with the package, and tutorials around the Internet. This makes use a lot easier and keeps costs of this system to a minimum.

### 3.9.2 GPU and FPGA Extensions through OpenCL

OpenCL is also worth mentioning as it is based on the C standards. It is a library that allows for easy portability onto GPUs and FPGAs, which will be most useful if this implementation ever takes a step into hardware implementation. The library is designed for C and is accessable like any other library in C. One includes the header and from there uses the functions provided by the library to create and pass variables for processing by the CPU or other hardware, such as a GPU or FPGA.

This, however, is not considered in the scope for this research as Narwhal is aimed at a software only implementation, however use of OpenCL a viable extension and has been listed in Section 5.4.

### 3.9.3 Application of Structs in C

The C language also allows for casting of structs onto objects in memory. This is very useful as one can define a template struct with the offsets of the different protocols to be simulated. An in-bound packet can then be cast to the relevant template struct after the type of packet is determined. This allows one to reference parts of the template struct, thus accessing fields within the packet stored in memory without having to calculate offsets for every access.

## 3.10 Physical Network

The physical network that Narwhal runs on is also important, as this is where all packets will be received from and transmitted to. This can bring up problems in configurations and lead to erroneous events that may occur when running a simulation under the wrong conditions. As such, it is better to make a physical interface that can fit in with other hosts on a network, rather than one that requires dedicated links.

It is also to note that if a network based simulator can run on a network with other devices and hosts generating traffic, then it will be all the more fit to run a network dedicated for its own simulation traffic. In this way the simulator as a whole should be more robust.

Libpcap has functionality that allows for such a simulator. If one passes libpcap a configuration string into libpcap's filter, it tells libpcap to listen to the IP addresses that meet the filter strings criteria.

This means that the simulator is able to pick out the IP range or hosts that it should listen to for any packets that need to be routed. In this way any noise on the network that is being listened to should be ignored, the packets generated by other hosts outside of the simulators defined scope.

This does not mean that one should not consider running a network based simulation on a network with other host traffic on it. The reason for this is that a packet sniffer still has to discern between what is a valid packet and what is not, when compared to the defined filter expression. This means that all packets are actually still processed, but only the ones that meet the filter expressions criteria are the ones that get passed up to the system for use.

As such, one should seriously consider running the simulator on a dedicated network, as this will increase performance and also drop miss rates of packets which may occur if the host system is under to much load to handle the amount of packets that have to be filtered.

## 3.11 Configuration of Narwhal

This routing simulator is designed to be set up and run with as little user requirements as possible. For this a lot of automation has been included in design.

This routing simulator will run from the fore mentioned configurations scripts. These are generated through conversion of enumerated data from the traceroute program. The traceroute collection of data from traceroute is also automated, and all that is required is to pass a list of all URLs and/or IP addresses to the bash script and the script will handle running the traces.

Once this is done the traces are converted into a usable configuration script by another python script. This script runs through every traceroute output file generated by the bash script and determines whether it is usable, and if so, writes it out to the configuration script.

After this, all that is required is to load the routing simulator and enter the command "build 'configuration file name'". This will then go about building all traced routes that were placed into the configuration file into a ready to route state within the simulator.

All that is left from this point is to bind hosts to nodes within the simulator using the "bindnode" command, followed by the "run" command which initiates the packet transmission threads.

## 3.12 Terminal

In order to achieve easy handling of this multi-threaded system, that allows for modifying of routes and host bindings during run time, a terminal approach was taken. This terminal runs in its own thread and handles requests from a user in terms of the configuration and general running of the simulator.

The interface is simple to use and thus further allows for easy integration of any new features, as well as real time debugging of the system, as one can monitor what is happening within the simulation and respond to any errors that arise during run time.

As the terminal handles configuration and running of the simulator, it is appropriate to assign it the main thread of the system. If the terminal closes, the simulator no longer has a means of user end base of collecting or inputting data, and thus the point of the system becomes naught. As such the simulator and all its child threads should be ended at this point and any handles and resources de-allocated and released, followed by the system shutting down.

As this is the only interface to the user, this terminal should be intuitive and responsive. Responsiveness is trivially achieved by the fact that it runs in its own thread, this being the main thread of the process. However, an easy to use interface is not so easy to implement.

The approach taken to accomplish an easy to use interface was to keep things simple. For this reason the commands should be made powerful, but at the same time the number of commands were kept to a minimal. A help function was also put into the design of this interface, which displayed all available functions along with what parameters were required by each.

Another aim was to allow for a fast build and run to allow for simple, but effective use of Narwhal. As such, one can simply load a configuration script into the simulator, through the use of the "build" command followed by the configuration scripts file name, and then use the "run" command. This would start the simulator in two simple commands that requires no more effort than pre-compiling a re-usable configuration of the routes one wishes to simulate.

This terminal can also view the routes to nodes within the simulated network. This is done through the use of the "bindnode" command that brings up all details about the node. This includes all nodes in the route to the queried node. The delay in each hop between nodes in the route to the queried node is also displayed.

Also included is a function to set the drop rate of packets getting routed through the queried node. This requires the IP of the node one wishes to set the drop rate of, followed by the percent drop rate one wants to apply to the node.

Other commands include the "stop" and "clear" commands. The "stop" command forces a stop on the simulation where each thread that is currently routing a packet is allowed to finish its routing of the current packet but no new packets are introduced into the simulation. The "clear" command is used as a means to clear the screen of any text to allow for easier readability.

## 3.13 Summary

This chapter covers the basic building blocks of the routing simulator and design choices after considering other possible solutions. This starts by defining how a node will be allocated in memory as to have best access times, as well as low as possible memory usage.

CPU requirements also come into play, and so we took into consideration the use of threads giving the rise to multi-threaded processors. This also allows for easier integration of functionality into the system with the combination of easy node access and multiple threads. This chapter then moved onto how packets were transmitted on the network. This lead into how a simulated network should be configured and the different approaches to routing within this network. From here we show why the decision for a main thread in the form of a terminal was chosen and the logical reasons for such an approach is defined.

Next was a move towards a more environmental look in designing this system. Operating system, architecture and programming language were chosen here, all with there benefits and reasoning taken into consideration before choice. Interfacing to the physical network and threads within the system was explained and methods in how to do so were made. This then lead onto how this routing simulator should interact on a physical network, before concluding with how one would go about using this routing simulator.

# Chapter 4

# Routing Simulator Tests and Results

This chapter deals with what tests are going to be run on this network simulator to define how the system performs as a whole under different conditions. This includes varying simulation loads and size to obtain accurate results of memory and CPU usage, as well as other relevant results.

A dynamic and static approach will be taken to into consideration in this routing simulator. Testing of the dynamic approach however, will only be done where comparison is relevant, that being on CPU and memory requirements.

The networks simulated in these tests are constructed from routing data generated from tracing real routes found in the Internet. These routes were collected from results produced from the traceroute program.

The reason for use of these routes from the Internet is simply because these tests need to model a real configuration that is used everyday. The Internet is maintained constantly and always up to date. It is the largest network on Earth and as such is the best source for collecting data in terms of routes to hosts.

All configurations used in the testing of the static routing approach will be applied to that of the dynamic approach as well. All tests were run a total of five times and the average of these results were taken.

These tests are required as this routing simulator needs to firstly be checked if it works. Following that, it needs to be decided whether it works well enough to be usable in real implementations. These results can also be used to compare this routing simulator to other like simulators, and thus to weigh up whether this implementation is a success and in what regards it is, if it is a success.

This chapter will start in Section 4.1 by testing out whether or not this system can route packets successfully. This will be through sending packets that will get sniffed into the routing simulator, having it routed, and finally ejected at the destination.

Following this the tests will continue into memory and CPU requirements under tests that include dedicated packet routing and routing under load, these are shown in Sections 4.2 and 4.3 respectively.

Throughput and packet drop rates will be taken into consideration next in Section 4.5 and 4.6. Section 4.7 through to Section 4.9 will test inaccuracies introduced through processing delay and unserviced packets through hardware, or software failures.

## 4.1 Basic Simulation Test

This first test was conducted to test whether this routing simulator could in fact route packets. If this basic property does not exist, then there is no further tests that can be conducted and considered accurate. The ability to route will be tested with a simple UDP based trace of a route from an attached hosts. The routing simulator will have the route to be traced loaded into it.

Narwhal has the ability to display routes to nodes in the simulated network via its terminal. With this functionality, one can display the route as is in the simulator and compare each hop directly to the results of the real route trace performed by traceroute, which will be run on the connected host.

A successful routing of a packet in this test would result in the route described by the routing simulator being the same as that of the traceroute result. Also, the delay should line up with that of the delay stored in the routing simulator.

Figure 4.1 shows the IP addresses of the nodes in the route to the destined node. The IP addresses are in the order that they are connected to the destination node. Also in Figure 4.1 is the average delay between each hop in the route.

Figure 4.2 shows the results from traceroute that was run on the real connected host. The layout of the results produced by traceroute starts with the header, which displays the

```
Delay IP

1     146.231.120.1
0     146.231.2.21
2     146.231.2.10
0     146.231.0.2
7     192.42.99.1
22    155.232.6.85
171   196.32.209.45
0     83.231.146.65
0     129.250.3.46
119   129.250.8.138
0     4.69.139.120
0     4.69.153.133
67    4.69.137.66
7     4.69.148.50
0     4.69.148.37
33    4.69.137.121
0     4.69.151.165
0     4.69.151.138
2     4.69.151.169
19    4.69.148.121
0     4.53.106.146
3     64.30.227.118
```

Figure 4.1: Route To Destination Node with IP 64.30.227.118 Produced from Narwhal Stored in Routing Simulator

chosen configuration, followed by each node in the route to the destination node. These discovered nodes are described by a number, which signifies the position in the route, then the node's IP, and finally the delay to the node.

There are no missing nodes when comparing the IP of each discovered node produced by the simulator, as seen in Figure 4.1, to that of the traceroute results produced by the connected host, as seen in Figure 4.2. Furthermore, the order of each of the nodes which were discovered by traceroute were in the order in which they were intended to be visited by the routing simulator.

The delays in each hop in the route are also accurate. This is observed from comparing the additive total of the node hops defined in the simulators route to the destined node to that of the delays produced by the traceroute.

These results point strongly towards this routing simulator accurately fulfilling all its

```
traceroute to 64.30.227.118 (64.30.227.118), 30 hops max, 60 byte packets
1 146.231.120.1 0.450 ms 0.191 ms 0.000 ms
2 146.231.2.21 0.001 ms 0.000 ms 0.293 ms
3 146.231.2.10 2.171 ms 2.180 ms 2.193 ms
4 146.231.0.2 2.185 ms 2.200 ms 2.074 ms
5 192.42.99.1 8.459 ms 8.405 ms 8.294 ms
6 155.232.6.85 29.109 ms 28.858 ms 28.819 ms
7 196.32.209.45 184.191 ms 184.078 ms 183.951 ms
8 83.231.146.65 184.133 ms 184.111 ms 183.866 ms
9 129.250.3.46 184.129 ms 183.918 ms 183.688 ms
10 129.250.8.138 292.042 ms 292.183 ms 292.026 ms
11 4.69.139.120 292.217 ms 291.967 ms 292.104 ms
12 4.69.153.133 292.082 ms 292.843 ms 292.698 ms
13 4.69.137.66 353.648 ms 353.397 ms 353.685 ms
14 4.69.148.50 360.148 ms 359.981 ms 359.722 ms
15 4.69.148.37 360.016 ms 360.870 ms 360.779 ms
16 4.69.137.121 390.335 ms 390.138 ms 390.669 ms
17 4.69.151.165 390.489 ms 390.322 ms 390.605 ms
18 4.69.151.138 390.551 ms 390.651 ms 390.299 ms
19 4.69.151.169 392.380 ms 392.153 ms 392.226 ms
20 4.69.148.121 409.924 ms 409.735 ms 409.680 ms
21 4.69.106.146 409.522 ms 410.169 ms 409.901 ms
22 64.30.227.118 413.312 ms 413.602 ms 413.106 ms
test@Rhodes: $
```

Figure 4.2: Traceroute to IP 64.30.227.118 Results from Real Host Attached to Simulator

intended purposes. It manages to sniff a packet from the physical network, simulate routing of the packet, emit ICMP messages where required back to the source host, and finally emit the actual sniffed packet to the destined host. From this we can now move onto further testing as the basic functionality required to successfully simulate the routing of packets is in place.

## 4.2 Progressive Memory Test

As one of the key difficulties within a network based simulation is keeping memory usage to a minimum, this will be addressed by looking at total memory usage on the host system by the simulator.

The test will be run through stages consisting of different sized networks of growing complexity. This will consist of tests that will contain both high and low node counts, with a variation of hops in each route.

This test will only test the usage of memory by nodes within a configured network, and results will not include the memory overheads needed for temporarily storing the routed packets within the network. As such, the packet sniffing and packet ejecting threads will not be activated, to allow for more accurate results from this test.

It is predicted that as the simulated network grows, the memory requirements of the network being simulated will reduce in the amount of memory needed to route a new path. This is because in the Internet, routes to a region of the world usually result in a repeated default route up until the region in which a set of nodes reside is reached. At this point the packet being routed route changes as the node it is addressed to is no longer part of a default route and rather an individual one.

This means that the path to nodes in a default route will not be duplicated. This is because a network does not allow for duplicate IP addresses, as packets will not know which node they are destined to. Narwhal omits creation of duplicate nodes by checking if the node exists and then if a path has been defined to it. If both of these requirements are fulfilled the new path is ignored. In this way a default route is created once, and so any further requests to recreate the path are rejected and thus further memory is not used.

Furthermore, in the Internet it is common practice to host multiple URL's on a single IP. This is due to a single host hosting multiple websites, a NAT (Network Address Translation) or even a node acting as a load distributors to handle larger amounts of requests.

This means that as node again cannot be duplicated, that all the directed URL's will be directed at a single node, and so reduce the memory requirement as a single path will be stored to the given node.

This test will be conducted on the dynamic routing approach, exactly the same manner in which the static routing test was conducted. It will run through stages of both high and low node counts of growing complexity. These routes again will contain a variation in hop count.

It is expected that using a dynamic approach will result in a reduced memory requirement, as there won't be a need to store every node involved in the route to a destination in the

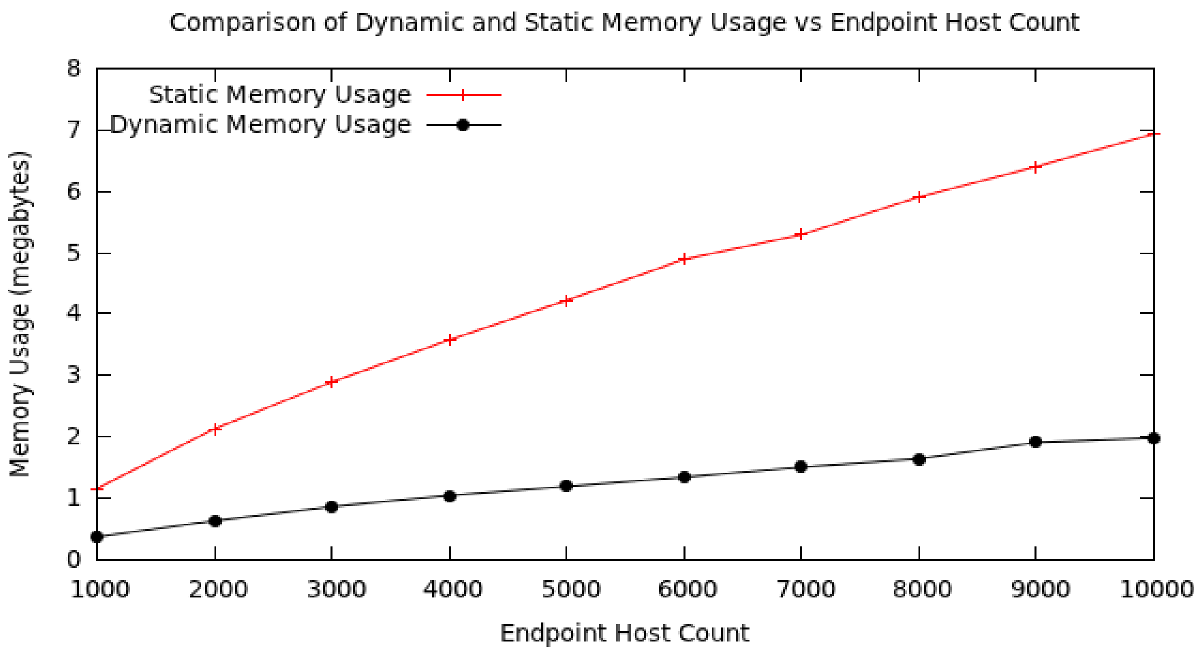

Figure 4.3: Memory Usage of a Dynamic and Static Routing Approaches

destined node. Instead, only the neighbouring nodes connections are stored and so there should be a decrease in the memory required to store routing information.

Figure 4.3 shows the memory usage of this routing simulator in both dynamic and static routing approaches as more endpoint hosts are added to the simulated network. This shows that in both routing approaches, the more hosts that are required to be routed to, the more memory is required. This is a trivial and expected result.

Furthermore, Figure 4.3 shows the beginning of a logarithmic trend in both approaches. In the static approach, this is due to the fact that repeated routes are not stored in the routing simulator and as such are not added multiple times. A route is only stored from the point it deviates from a known route. This results in memory requirements dropping as more nodes are added to the network, as there is a higher chance that part of a route already exists within the simulated network.

Also, memory usage is reduced by the fact that some hosts map to the same IP and so share a node, and therefore share an entire route. This means that the path won't be added as it already exists, and so no memory is used in this case.

In the dynamic approach no two links are repeated. This is done during creation of the network by checking what links exist at a node before adding the new link to it. If the link already exists then don't recreate it.

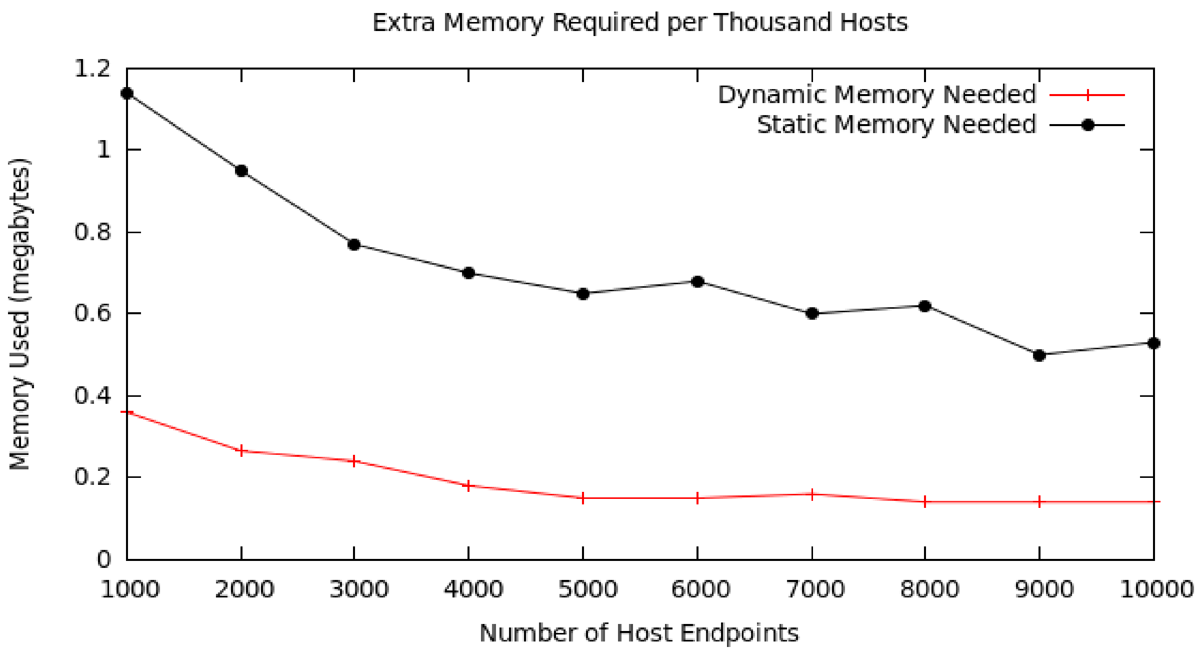

Figure 4.4: Memory Usage Difference of Dynamic and Static Routing Approach

In this way one can see how a dynamic routing approach would require less memory as well. So in both cases, the more routes that are added, the less memory is required overall.

Also shown in Figure 4.3 is that the dynamic approach to routing in Narwhal uses a lot less memory than a static routing approach. This result was expected, resulting in the dynamic approach needing only about a third of the memory that the static approach needed.

However as the routes in the dynamic approach now need to be calculated at run time. It is expected that the memory trade off seen between the static and dynamic approach will effect the amount of CPU resources required to run the simulator in its dynamic mode. This should result in an overall increase of CPU resources required by the dynamic approach. The results for this will be shown in Section 4.2.

Figure 4.4 shows the amount of extra memory required by the routing simulator for every thousand hosts added to the simulated network. The first thousand is derived from the amount of memory initially taken to add the first thousand routes. Figure 4.4 shows both static and dynamic approaches and each are labelled accordingly.

It is easier to see the reduction in memory required by the routing simulator to add new routes in Figure 4.4 when compared to Figure 4.3. One can see a general downward slope

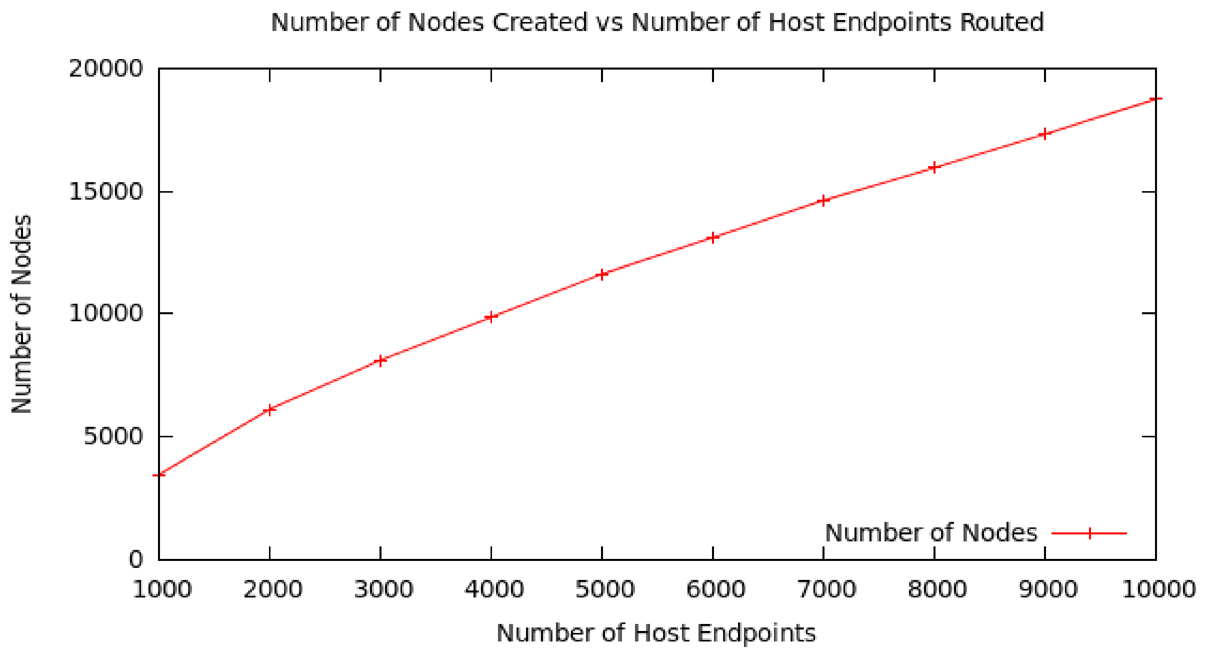

Figure 4.5: Total Nodes per Route

following a reverse exponential trend, which suggests that adding further routes to this network will result in further reduction of memory usage per route added.

A point of interest from Figure 4.4 is that the negative gradient in the requirement of memory in the static routing approach is greater than that of the dynamic approach. This is probably due to the fact that the dynamic routing approach uses less memory to begin with. This makes it more difficult to actually save on memory, and so would yield less drastic results than that of the static routing approach.

This is due to the route representation differences between the two approaches. Not repeating a node in the dynamic approach would save on the memory allocation of that node and the connection to it. In the static approach this would result in not only saving the memory needed to create the node, but also the entire precalculated route to it.

Figure 4.5 shows the amount of nodes present in the simulated network within the routing simulator for every thousand routes added to the network.

This graphs trend, like Figure 4.3, follows a slight logarithmic trend. This relates to the memory usage of the routing simulator, as the more nodes that are created, the more memory that is required to support these routes. So as less nodes are being introduced into the routing simulator due to nodes already existing, so less memory is required to create the nodes as not all nodes in the route will be created due to them already existing.

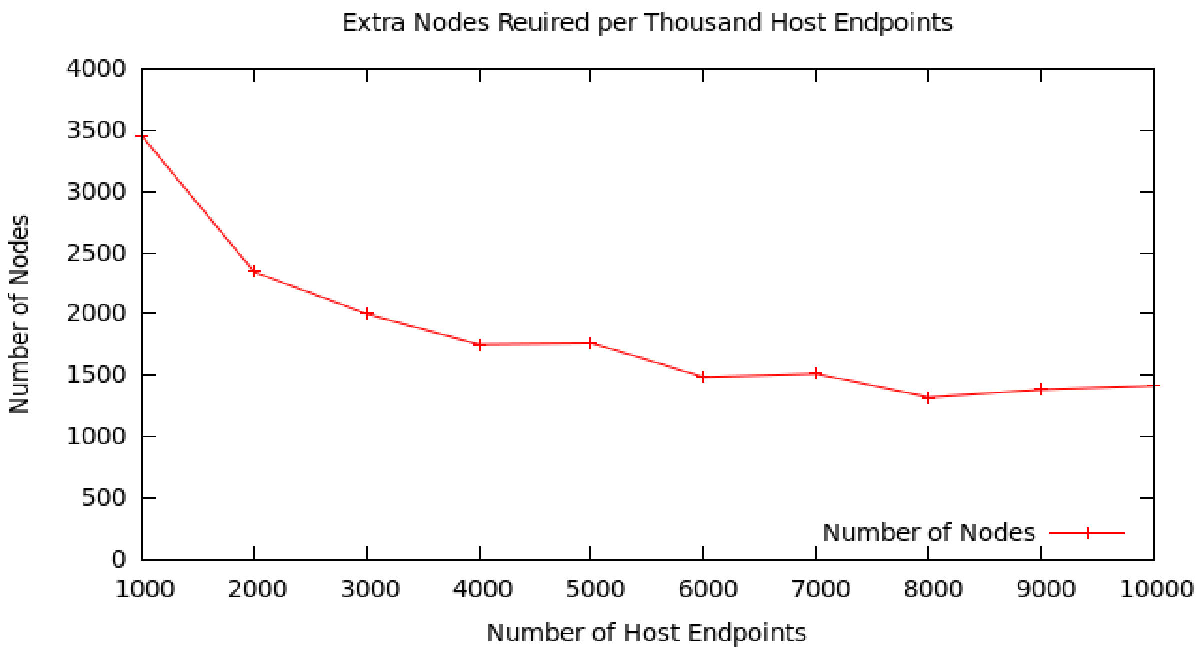

Figure 4.6: Total Nodes Difference per Route

This also suggests that further increasing the size of a simulated network will require less memory. If this trend continues adding a similar amount of nodes to this routing simulator will have an almost negligible memory requirement.

Figure 4.6 was produced to show the correlation between the amount of extra memory required, as shown in Figure 4.4, and that of the amount of nodes added to the routing simulator.

Figure 4.6 shows more clearly the reduced number of nodes being created with respect to added routes to the routing simulator. However, the dip seen in Figure 4.4 at the seven thousand host endpoints point is not depicted in Figure 4.6.

This suggests that the routes added to the routing simulator were shorter than the other additions to the network simulator, and so even though an expected amount of nodes were created in the simulator, as when compared to prior results, there was not as much of a memory requirement, as the routes in average required less hops and so were shorter.

These figures confirm the fact that as more routes are added to Narwhal, less memory is required due to repeated nodes not being created in memory. This as, well as host endpoints not being repeated where routes lead to the same host endpoint.

## 4.3 CPU Load During Simulation and Configuration

As more packets are introduced into Narwhal, it is expected that load on the CPU will increase. This is mainly due to the fact that there is no external hardware being used to route the packets and so the CPU has to do this task by itself.

This series of tests will aim to stress the CPU by increasing the number of packets to be routed in fixed steps. Each test will be run on the a series on the same large scale network fixed network. This is so no inaccuracies creep in due to differences in the way the networks route.

Packets will be of a fixed size and will only differ in the number introduced to the system. The time span which the allotted packets are to be introduced into the routing simulator will be fixed as to more accurately test the stress on the CPU.

It is expected that a large network with simple routes should run at roughly the same CPU load as that of a network with fewer nodes but complex routes. This is due to the fact that the amount of packets routed on a large network should balance out with that of smaller network with complex routes due to the time taken to route the packet. This being said, the test has opted for a fixed network to make sure that there are no differences that can arise from changing networks.

Again, this test will be conducted under the same conditions as that of the static routing test for the dynamic case. It will consist of introducing a larger and larger packet load to the routing simulator in evenly spaced increments equalling that of the static routing test.

The network that these tests will be run on will again be fixed as not to let any error arise from subtleties in the routing of a configured network. This will allow for testing of the stress on the CPU of the host only, and will result in accurate results.

It is expected that, due to the nature of calculating the next hop at each node, the CPU usage of dynamic routing will be higher than that of a static approach. This is due to each node having to look through their routing table to find a match before making the next hop. This requires more comparisons and thus more CPU time than a simple go to next node in list approach.

Figure 4.7 depicts peak CPU usage during full simulation undergoing routing of increasing number of packets over a set time frame. Each packet used the standard TCP protocol

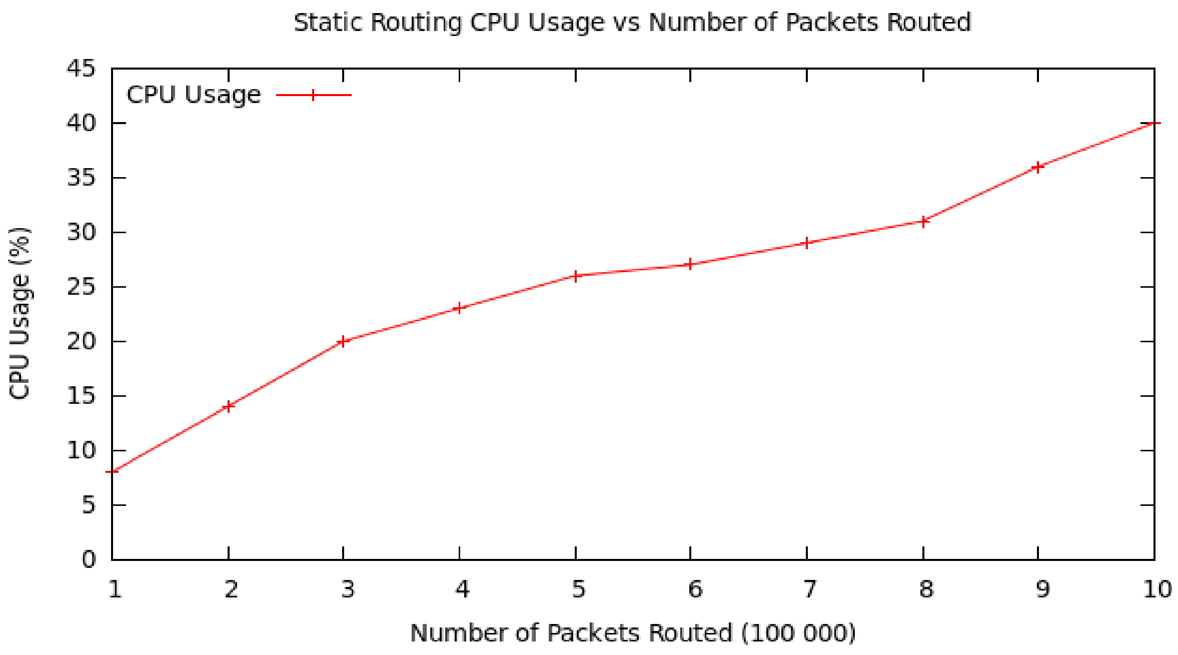

Figure 4.7: CPU Usage per Hundred Thousand Packets Routed Using a Static Routing Approach

and had a payload of 8 bytes. As scheduling varies depending on tasks active on the host system at a given time, these tests were repeated several times and averages taken.

One would expect the load on a CPU undergoing this task to be fairly linear. These results do not directly adhere to this prediction, but a linear trend can be loosely fitted to the results shown in Figure 4.7.

In these results the CPU shows adaptive behaviour. The CPU allocates resources as a task becomes harder to handle in an allotted CPU time. This then leads to less CPU load for similar larger tasks until the task becomes to large to be handled in the allocated resources, This is shown by the sudden growth in CPU Usage again. The CPU then allocates more of its resources to the task, and the growth of CPU usage then reduces until the task becomes too much again.

With this in mind, one can map a linear trend to the results of Figure 4.7, as further results will continue to oscillate around the plotted linear trend line. In short, this means that CPU growth is linear, which is to be expected, as handling twice the amount of packets should only take twice the amount of resources.

Figure 4.8 is the dynamic CPU usage results for that of a full simulation run using a dynamic approach on this network simulator. Each packet was again transmitted using

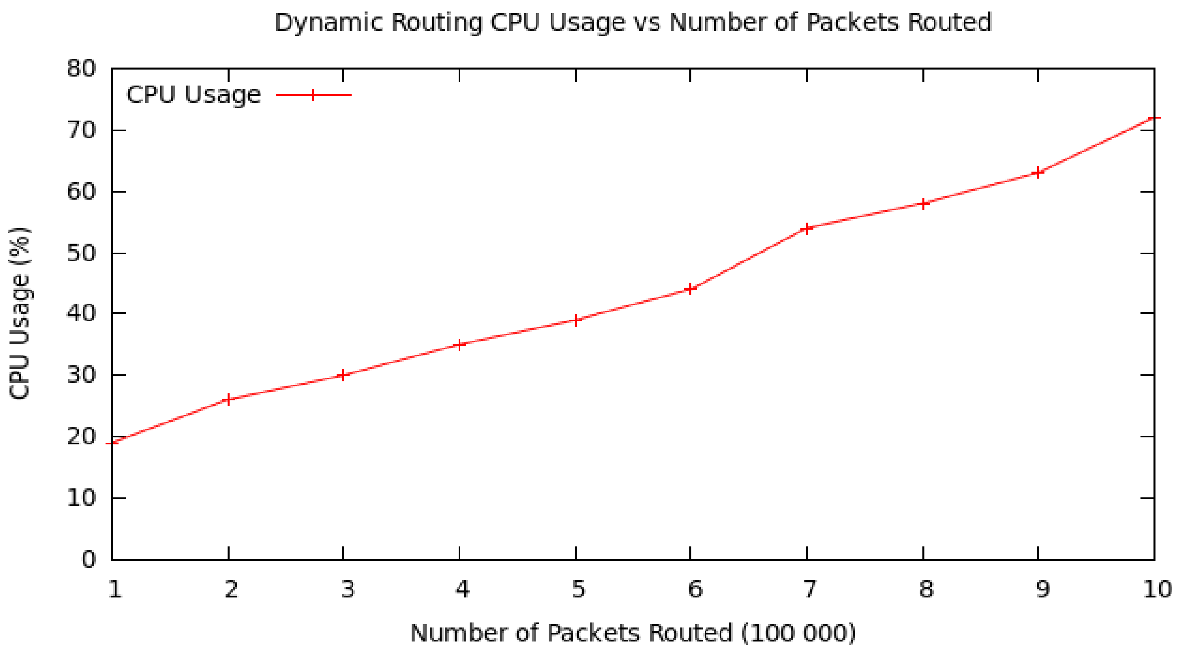

Figure 4.8: CPU Usage per Hundred Thousand Packets Routed Using a Dynamic Routing Approach

the TCP and had a payload of 8 bytes. Again, as scheduling of a CPU's tasks can obscure results, these tests were repeated and averages taken.

Again the results show a fairly constant growth in CPU requirements. This is again expected, as if one were to double the amount of packets being routed, one would expect to double the CPU load.

It is also to note that this graph does not show any major signs of adaption as observed in Figure 4.7. This may be because of the higher requirements from the dynamic approach, and thus there is no obvious stepping of allocated resources.

The graph in Figure 4.9 was produced to show the difference in CPU requirements between a dynamic and static routing approach. Both tests used the same routing configuration and stepping of packets routed to allow for results to be compared directly.

As expected, the dynamic approach requires more CPU resources than that of the static approach. Also the requirements of the dynamic routing approach climbs more aggressively than that of the static routing approach.

It is interesting that comparing the memory usage of each approach in Figure ?? to that of CPU usage of each approach in Figure 4.9, the memory for CPU requirement trade off

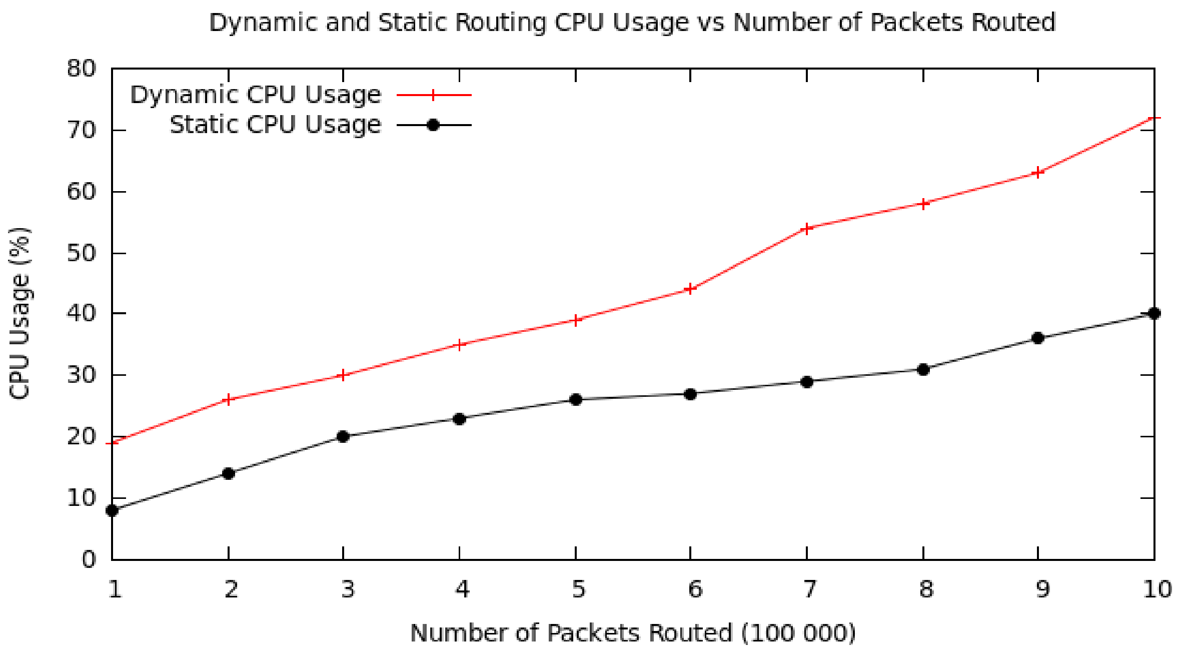

Figure 4.9: CPU Usage Comparison of Static and Dynamic Routing Approaches

seems to fall in the dynamic approaches favour, as the extra requirement for CPU is less than that of the extra memory required by the static approach.

From Figures 4.7, 4.8 and 4.9 one can come to a good conclusion about what the host systems CPU usage would max out on. Following the linear trend of the static approach, this estimate arrives at roughly 2.2 million packets.

Following the linear trend set by the dynamic approach, one comes to a figure around 1.2 million. With these figures its plain to see that a static routing approach out-does a dynamic one by about a million packets. This is quite a significant number, but if one were to again compare this ratio to that of the memory usage's ratio, one can clearly see that CPU trade off falls short of the saving one gets in memory.

Figure 4.10 shows peak CPU usage during creation of routes in the routing simulator. The results of this test are an average of CPU usage across all cores of the host systems CPU. It is observed that for every thousand nodes, CPU usage increases by roughly twenty percent. As scheduling varies depending on tasks active on the host system at a given time, these tests were repeated several times and averages taken.

One must keep in mind that there are multiple nodes involved in creating a route to a single endpoint. This means that routing to and creating the node that handles the host at an endpoint is not the only node created when building the route.

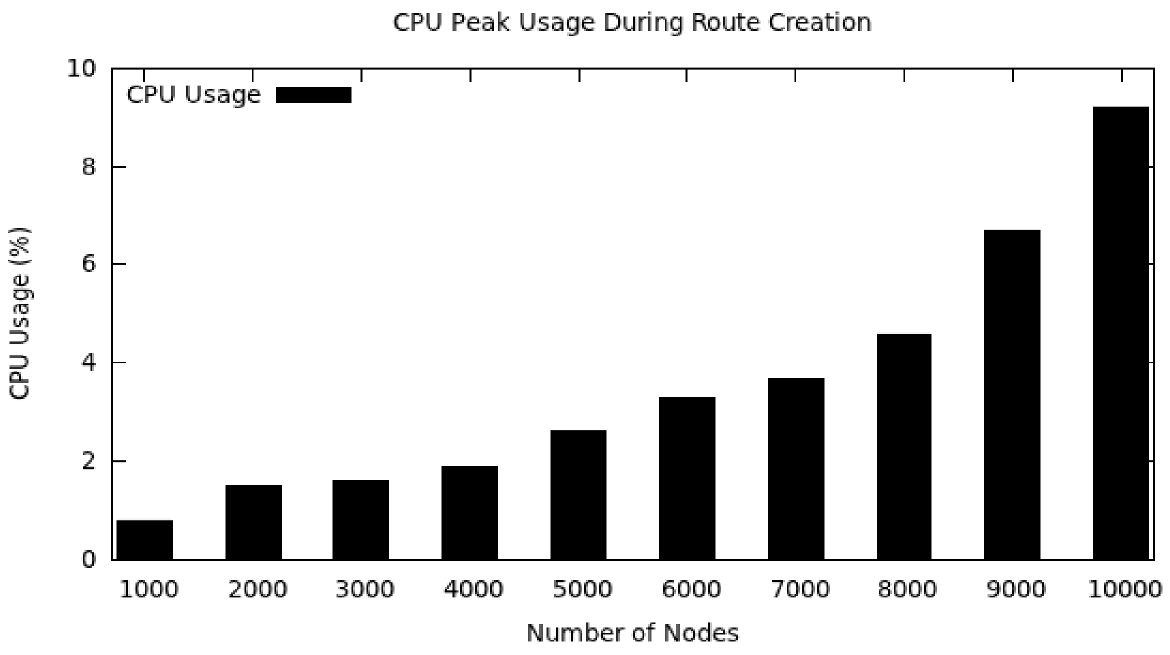

Figure 4.10: CPU Usage During Creation of Routes within Narwhal

With this observation CPU usage can vary depending on the network configuration that is to be routed. If there are many repeating endpoints that require a general route, less nodes will need to be created and so less CPU usage will be required. This is however the opposite if each endpoint host requires its own unique route, and will result in greater CPU usage.

Table 4.1 is a projection of CPU usage under creation of larger networks in increments of a thousand as to keep to the stepping of the original sample set. With these figures it is estimated that to utilize one hundred percent of this hosts CPU would require a network consisting of around 22000 to 23000 host endpoints.

This prediction only holds if the nodes configuration of further nodes follows the same configuration as that of the original endpoint hosts routed to in the original sample set. If these were to change too dramatically then it would introduce too much entropy to accurately predict any form of CPU usage by the host system.

Table 4.1: Projected CPU Usage as More Endpoint Host Routes are Added

| Number of Nodes | Projected CPU Usage (percent) |
|---|---|
| 11000 | 11.04 |
| 12000 | 13.25 |
| 13000 | 15.90 |
| 14000 | 19.08 |
| 15000 | 22.90 |
| 16000 | 27.47 |
| 17000 | 32.97 |
| 18000 | 39.56 |
| 19000 | 47.47 |
| 20000 | 56.96 |
| 21000 | 68.36 |
| 22000 | 82.02 |
| 23000 | 118.12 |

## 4.4 Memory Use Under Varying Packet Loads

This test will focus on the extra overhead required by packets received and routed within the simulated network. As such the packet sniffing and packet ejection threads will be activated to allow for packets to be introduced and removed from the simulated network. This is required as one cannot assume that, just because a network fits into the allocated memory on the host system, it will run effectively. This is due to the fact that packets being routed are stored temporarily in memory and thus take up memory.

Testing this memory usage will allow for an estimated figure of how many packets can be active on the simulated network at a given time. This will be roughly proportional to the amount of memory left for the system to route such packets. It is hard to give an exact figure for this as, although packet headers within the network are of fixed sizes, the payloads attached to these packets change, and so add entropy to the calculation.

As such, the best way to estimate this number is to take the average size of a each simulated packet protocol that exists on a common network. This can be made more accurate through the use of ratio's of the packet protocol types found on a network, and using this to further increase the accuracy of the estimate by multiplying through by each respective ratio (average packet size multiplied by packet occurrence ratio).

Unfortunately, packets cannot scale logarithmically as each packet is seen as a unique object. This means that each packet has to have its own segment of memory allocated to it and so leads more to a one to one ratio of packet to memory allocation.

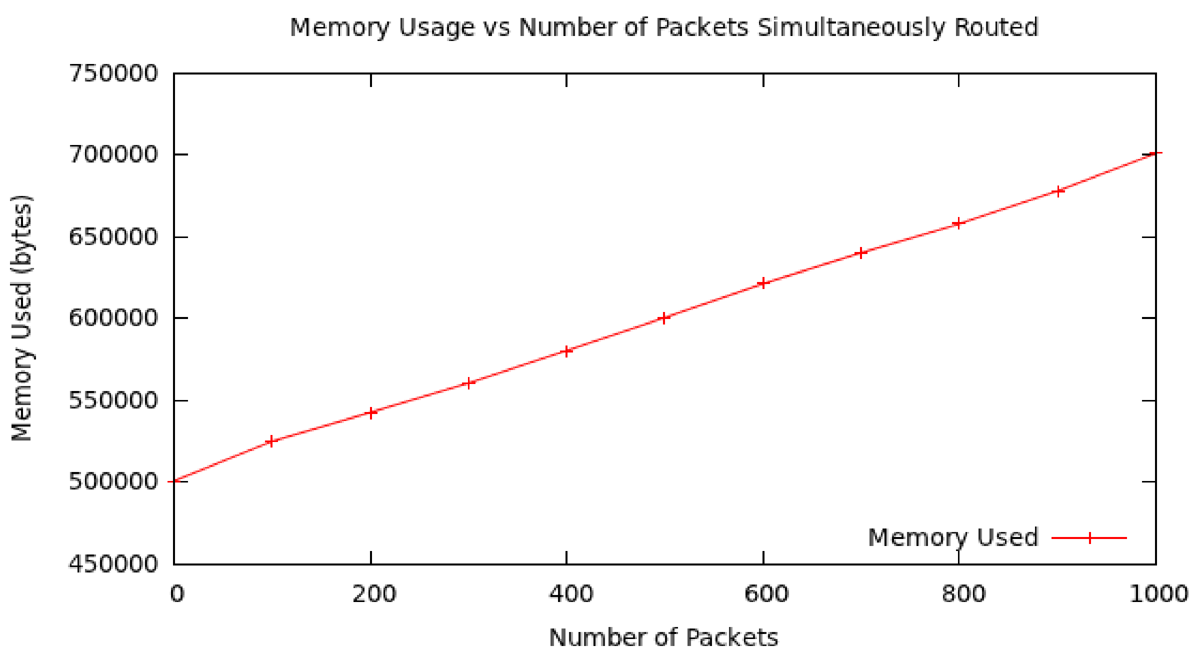

Figure 4.11: Memory Used per Packet Routed Simultaneously

One could go about checking for exact replica's of packets within a network simulator and using some clever referencing tricks to reduce memory usage. The problem with this is that the over head required by checking for such duplicates in terms of CPU load is in most cases not beneficial to the general running of the simulator.

Figure 4.11 shows the memory used per packet simultaneously routed within the routing simulator. This graph follows a fairly linear trend and is expected, as packets are fairly evenly distributed in size.

This means that doubling the amount of packets being routed in the simulator should result in the memory requirement needed for twice the amount of packets being roughly double. This is clear from Figure 4.11 as the growth per hundred extra packets simultaneously routed is constant.

The initial required memory to start packet capture and transmission threads is around 500Kb (Kilobytes), and every hundred packets introduced to the simulator requires a further 20Kb of memory.

Figure 4.12 shows the amount of extra memory required to route a further hundred nodes simultaneously within the routing simulator. This graph follows a fairly linear trend oscillating around 20000 bytes of extra memory required to simultaneously route an extra hundred packets.

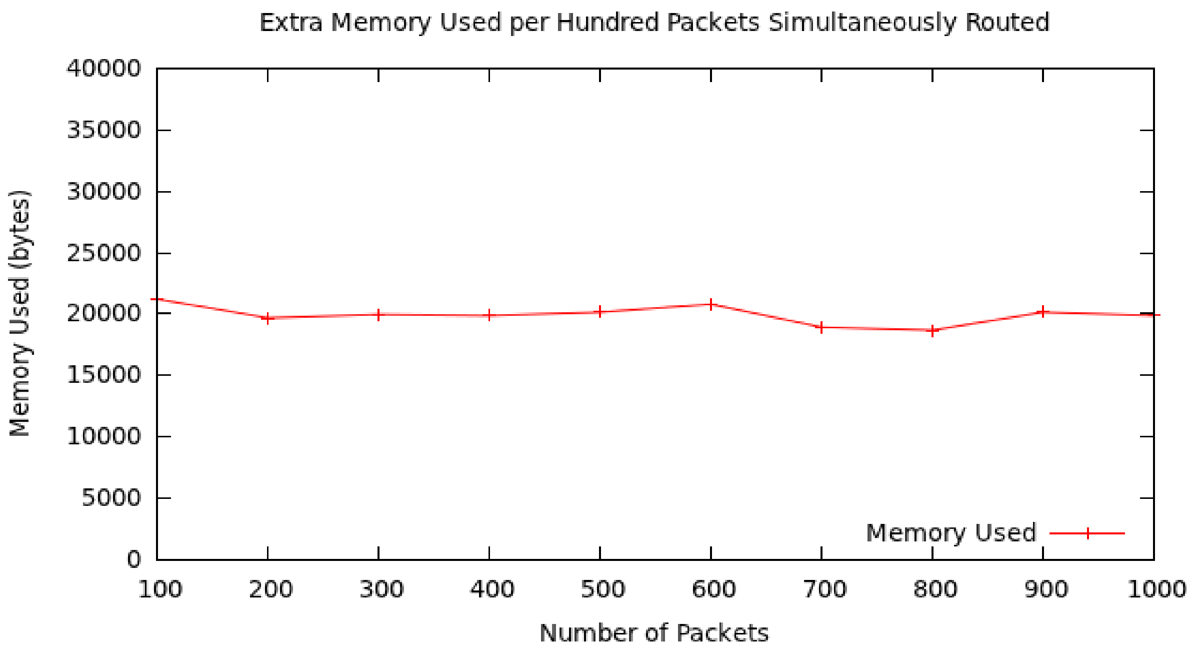

Figure 4.12: Memory Difference per Packet Routed

Figure 4.12 is a break down of Figure 4.11 intended to show more clearly the linear growth in requirements of memory by the routing simulator per packet routed in parallel with other routed packets.

The reason for the extra memory requirement not being precise is due to the fact that the packets being routed were randomly generated around a set size of 2048 bytes. This allowed for the simulator to show that it could handle different sized packets with different payloads in parallel, and that it can dynamically allocate memory and not allocate a memory block of a fixed size. This allows the routing simulator to save memory with this dynamic memory allocation characteristic rather than leave inaccessible, unused memory blocks with a static approach.

## 4.5 Throughput

The need to test the throughput of Narwhal is required as one of the growing problems faced world wide is that of bandwidth. As websites become more complicated and media grows larger in data size, the need to be able to handle the transfer of such larger amounts of data within the same time constraint as their smaller sized predecessors is a very real problem. This means that Narwhal has to be able to handle such transfer rates too, if it is to be relevant to solving current day problems.

A common LAN (Local Area Network) nowadays runs at at least 100Mbps (Megabits per Second). The Internet however does not hold such speeds world wide. As this routing simulator is based on routing parts of the Internet, it will be enough for it to reach the bandwidth of the country with the fastest average bandwidth between all its users.

This country is South Korea which has an average connection speed of roughly 17Mbps (Pingdom AB, 2011). Using this figure and rounding it up to the nearest ten leaves a target figure of 20Mbps. This figure, when converted to standard file size as read from disk, is equivalent to 2.5MBps (Megabytes per Second) transfer speed. Note: There are 8 bits in a byte.

These tests used TCP to handle the transfer of a file across the simulated network. This is because this is the most commonly used protocol for file transfer and that which other protocols are based upon.

Furthermore, these tests will be be repeated with files of varying size, as the load on the system can affect the speed at which packets are routed through the network. This would directly affect the throughput of the file being sent through the simulator.

The data transferred will range between two and twenty megabytes. These tests will undergo two main repetitions, these being transfer under load and transfer under no load. The transfer under load will be conducted by transferring the same file multiple times simultaneously from separate hosts. The transfer under no load will be conducted as transfer of a single file from one host only. This is to further test the effect of multiple file transfers, and whether routing multiple files simultaneously would effect the throughput of an individual transfer.

These tests were conducted through the use of a simple server client implementation written in C. This was kept as simple as possible to ensure the least interference from other aspects of the program. The server would simply wait for a connection, receive the data from the client and confirm the success by outputting the amount of data received and what was contained in the file.

This allowed for control over the network communication and allowed for the program to be specifically tailored for the test. This further insured that other sub-processes or subtleties could not creep into the file transfer, introducing entropy to the transfer and thus obscuring the results of this test.

Table 4.2: Throughput: Single Data Transfer

| Transfer Size (MB) | Time (s) | Throughput (Mb/s) |
|---|---|---|
| 2 | 0.453 | 35.32 |
| 4 | 0.783 | 40.87 |
| 6 | 1.198 | 40.07 |
| 8 | 1.666 | 38.42 |
| 10 | 1.923 | 41.60 |
| 12 | 2.493 | 38.51 |
| 14 | 2.731 | 41.01 |
| 16 | 3.198 | 40.03 |
| 18 | 3.637 | 39.59 |
| 20 | 3.997 | 40.03 |

Table 4.2 shows the times taken to transfer data through this routing simulator of varying size from a single host. The size of the data transferred is then divided by the time, and finally multiplied through by eight, as there are eight bits in one byte, to get a result in Mbps (Megabits per Second).

Other than the first result, the transfer of two megabytes, the rest of the results tend towards a transfer rate averaging 40Mbps. This result exceed expectations set at 20Mbps.

The first result of Table 4.2 is lower than the others and this can have many contributing factors. The main one of these is the time it takes to set up the connection. When data is being transferred it can be streamed in multiple parts. However, setting up a connection cannot be done as easily as it requires a three-way handshake to take place before data is transferred.

This delay in starting the transfer can lead to a significant drop in the total throughput of the simulator.

Figure 4.13 is a graphical representation of the size versus time results in Table 4.2. This graph shows a linear trend and is expected, as throughput remains fairly constant throughout the transfer of different sized data segments.

Table 4.3 shows the times taken to transfer data through this routing simulator of varying size from multiple hosts in parallel. This puts more strain on the routing simulator as it now has to transfer segments of data of the same size from multiple hosts. Transfer Size in Table 4.3 refers to the size of data each host is transmitting. Ten hosts were used in this test.

Again the first result in Table 4.3 shows a low result when compared to the rest. This

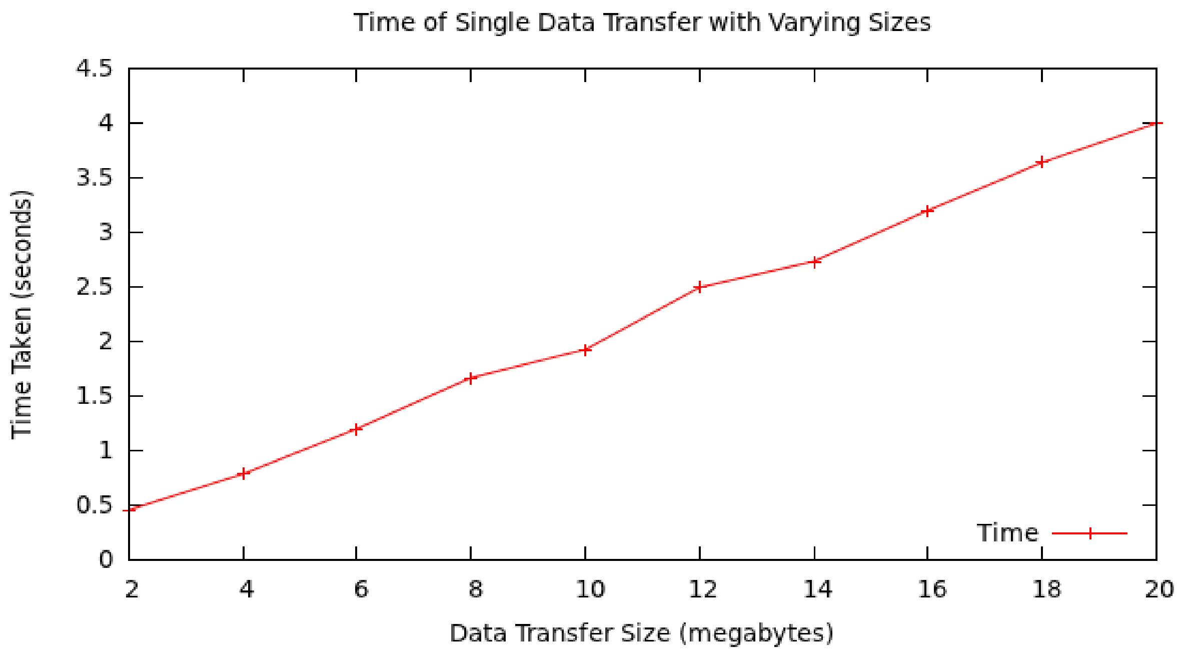

Figure 4.13: Throughput of Single Data Transfer

coincides with the results from Table 4.2. This can also be linked to the same reasons as discussed in the single data transfers result.

Multiple data transfers are slower than a single data transfer, but not by a big margin. In comparison, a single transfer results in an average throughput of 39.54Mbps, where transferring the same sized data from ten clients in parallel results in 38.98Mbps. This is a difference of 0.56Mbps which equates to roughly 70KBps (Kilobytes per Second).

This means that this routing simulator can effectively deal with multiple simultaneous connections through use of threads. Furthermore, essentially the data used to obtain the results of Table 4.3 if brought down to a single host, is the equivalent of transferring ten times the data size. This shows that the system can scale well from small transfer to that of a hundred times larger than the first test of a single transfer.

Figure 4.14, like Figure 4.13, is a graphical representation of the size vs time results in Table 4.3. This graph shows a linear trend as well, and this is also expected, as throughput remains fairly constant throughout the transfer of different sized data segments from multiple hosts.

Table 4.3: Throughput: Multiple Data Transfers

| Transfer Size (MB) | Time (s) | Throughput (Mb/s) |
|---|---|---|
| 2 | 0.486 | 32.92 |
| 4 | 0.812 | 39.41 |
| 6 | 1.203 | 39.9 |
| 8 | 1.731 | 36.97 |
| 10 | 1.964 | 40.73 |
| 12 | 2.375 | 40.42 |
| 14 | 2.751 | 40.71 |
| 16 | 3.180 | 40.25 |
| 18 | 3.671 | 39.23 |
| 20 | 4.074 | 40.27 |

## 4.6 Throughput Under Varying Packet Drop Rates

Packets can under go many unforeseen circumstances that can lead to them being dropped or lost in transit to the destined node. Depending on the protocol used for transfer of such packets, different levels of error recovery, if any, are instantiated.

Although it is primarily up to the hosts involved in the transaction to provide such data loss mitigation, the network too must be able to not get in the way of the hosts efforts to fix the problem at hand. The network essentially must act as a medium for which two hosts can communicate on, rather than what would seem as a more intelligent form of message delivery.

This will be tested by setting the level of packet drop rates at nodes within the route to which the packet is destined. If all works as it is intended to, a ten percent drop rate of packets should result in roughly an ten percent decrease in the throughput of a connection. This means that a transfer should take roughly an ninth (this is legit maths) longer than it would in an ideal environment.

This test is kept relevant by the very fact that packets are being dropped due to queue overflows, lossy connections, disconnected routes and more, within the Internet and general networks. So, if this routing simulator cannot deal with packet loss in the correct manner, it can lead to a big impact on the results produced by further testing and use in a real world environment.

Figure 4.15 shows the relationship between time taken to transfer a four megabyte data segment, and percentage of packets dropped on a simulated network. The results follow a exponential trend.

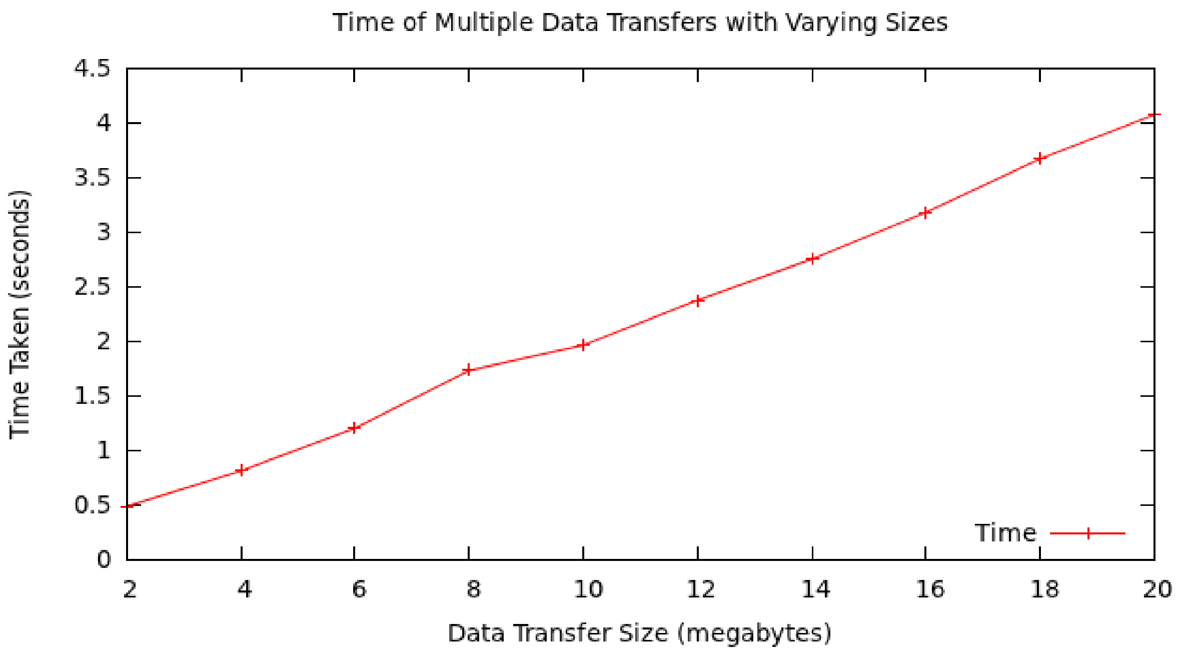

Figure 4.14: Throughput of Multiple Data Transfer

This is particularly interesting as one would have expected the amount of packets dropped to be directly proportional to the time taken to complete a data transfer across a network. Instead the time taken is significantly larger from the forty percent mark, and continues to grow from there.

The results from anything larger than sixty percent drop rates fluctuate to much to give an accurate reading and so were not recorded. A ninety percent drop rate resulted in failures in coping in a few cases, but times taken to transfer the four megabyte segment exceeded one minute and took up to three minutes in the worst case.

Figure 4.16 displays the drop in throughput as the packet drop percentage rises. This graph is based on the results from Figure 4.15 and shows the unexpected drop in throughput over the range of drop rates.

This is made more clear when comparing the throughput of a zero percent drop rate to that of a sixty percent drop rate. These values are 40.87Mbps and 3.27Mbps respectively. One would expect a sixty percent drop rate to result in a sixty percent drop in in throughput. However, this would yield the predicted result of 16.35Mbps which is significantly off the mark.

Instead with a 60% drop rate the simulator only retains eight percent of its total throughput at a zero percent drop rate. A contributing factor to this may be lost ACK (Ac-

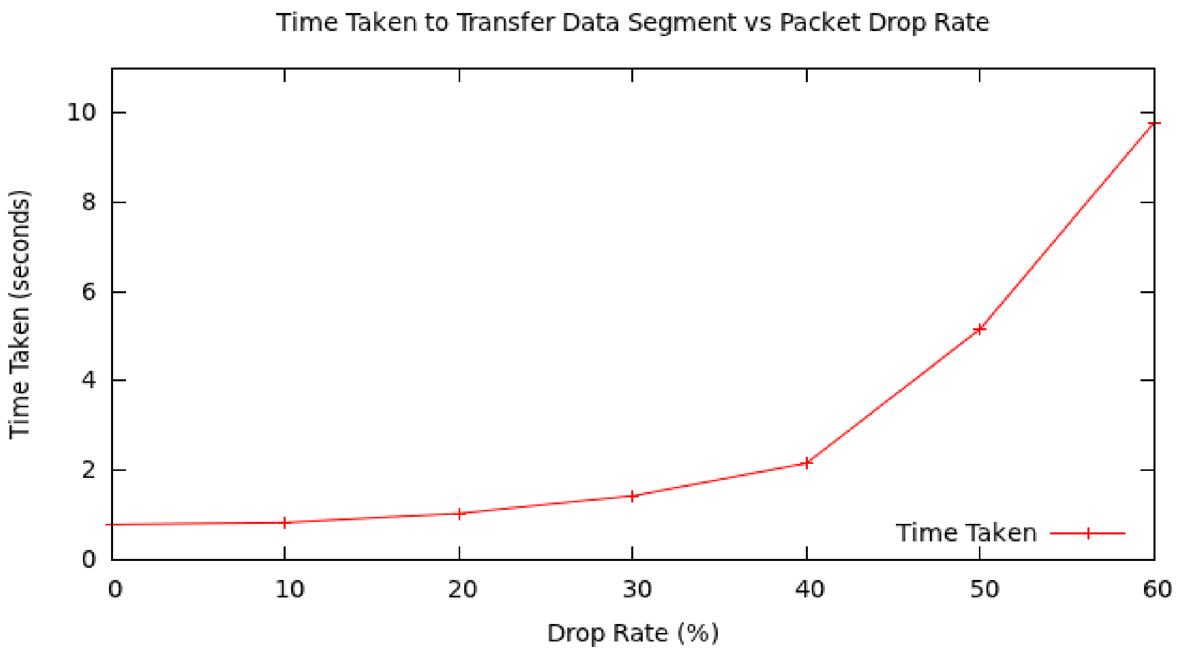

Figure 4.15: Time Taken to Transfer Set Data Under Varying Drop Rates

knowledgement) packets, resulting in packets being transmitted twice when the original was received.

This means that the data transfer not only has to deal with transmitting actual lost packets, but also packets that had actually been successfully transferred, but the ACK packet had been lost and so resulted in that packet being retransmitted.

## 4.7 Delay Accuracy Loss

Delay is part of a network and is largely due to four main components of most networks. These are queue times and processing delays within a node, and propagation delay and transmission delay caused by sending data over a medium. These tests were performed with the static routing approach as this is the most CPU effective method as shown by Section 4.2.

The delay recorded within the simulator is the sum of the average of all four of the above mentioned factors. With this in mind one does not want to introduce any further delay from processing the next hop in a routing simulator. However, it is nearly impossible to

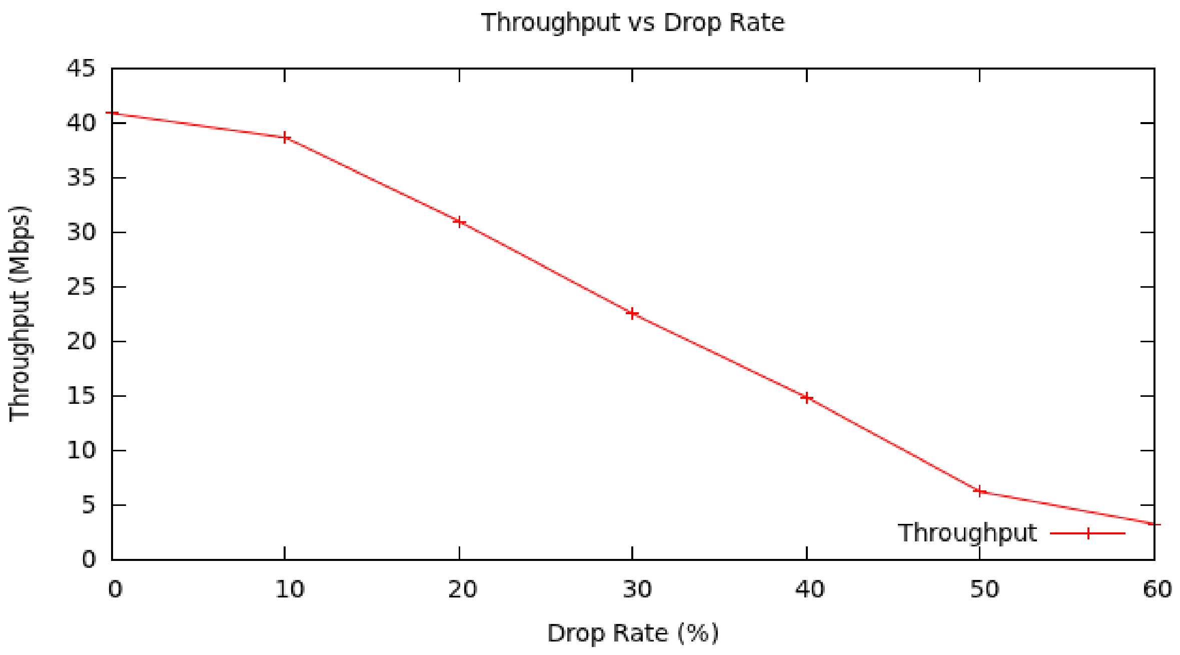

Figure 4.16: Throughput of Set Data Transfer Under Varying Drop Rates

calculate how long it will take to process the next hop in a simulator, and so it is more beneficial to use the simulators processing delay in introducing realism into the simulator.

This is done by using the simulators processing delay as entropy in the delay of a hop. As this simulation aims for a realistic simulator, there will be changes in the delay of a hop each time it is tested. This, as mentioned earlier, is done by generating a random integer in a realistic range and adding or subtracting it from the set delay of a node. If one were to further enhance the entropy of a hop by introducing the processing delay of a system into this calculation, it would remove the problem of accurately calculating the delay caused by simulation processing delay and removal of it by rather including it in the delay of a hop.

This test will be done under a constant network that will not change its configuration for the duration of testing. The only variable of this system will be that of the varying number of packets being introduced to the interface that the simulator is sniffing from on the host system.

Load in these tests is defined as having multiple data transfers occurring in parallel. Similarly, no-load is defined as having only a single transfer of data occurring during the running of the tests.

Table 4.4: Delays to Each Node of Traceroute Load vs No Load and Difference

| Hop Number | Time No Load (ms) | Time Load (ms) | difference (ms) |
|---|---|---|---|
| 1 | 0.191 | 0.393 | 0.202 |
| 2 | 0.293 | 0.463 | 0.170 |
| 3 | 2.180 | 2.107 | -0.073 |
| 4 | 2.185 | 2.151 | -0.034 |
| 5 | 8.405 | 8.497 | 0.092 |
| 6 | 28.858 | 31.533 | 2.675 |
| 7 | 184.078 | 184.642 | 0.606 |
| 8 | 184.133 | 184.659 | 0.524 |
| 9 | 184.129 | 184.684 | 0.526 |
| 10 | 292.042 | 292.565 | 0.523 |
| 11 | 292.104 | 292.505 | 0.401 |
| 12 | 292.698 | 292.378 | -0.320 |
| 13 | 353.648 | 353.756 | 0.108 |
| 14 | 359.722 | 359.972 | 0.250 |
| 15 | 360.779 | 360.035 | -0.744 |
| 16 | 390.335 | 390.357 | 0.022 |
| 17 | 390.489 | 390.383 | -0.106 |
| 18 | 390.551 | 390.587 | 0.036 |
| 19 | 392.226 | 392.474 | 0.8 |
| 20 | 409.680 | 409.529 | -0.151 |
| 21 | 409.522 | 409.675 | 0.153 |
| 22 | 413.312 | 412.596 | -0.716 |

Table 4.4's results are taken from delays to nodes of a single twenty two hop route. This was repeated under conditions of the routing simulator taking no extra load, only the packets from the traceroute program were being routed, and with extra load. There were ten hosts generating packets that were being routed at the time of the traceroute program running under load.

The difference was then calculated by taking the time for a run with load subtracted from a time for a run with no load. Negative values represent the run with load running faster than the run with no load by the given amount after the negative.

Other than Hop 6 in Table 4.4 there is no significant change in delay to nodes in the route. Even Hop 6's delay difference is not significant enough to be noticed by a human. These results also distribute fairly evenly between the run with load and the run without load being faster. This means that the routing simulator as a whole handles servicing multiple requests fairly well and that delay is not affected by the simulator taking on a load.

Figure 4.17 shows the comparison between delays of each hop between nodes in a set route

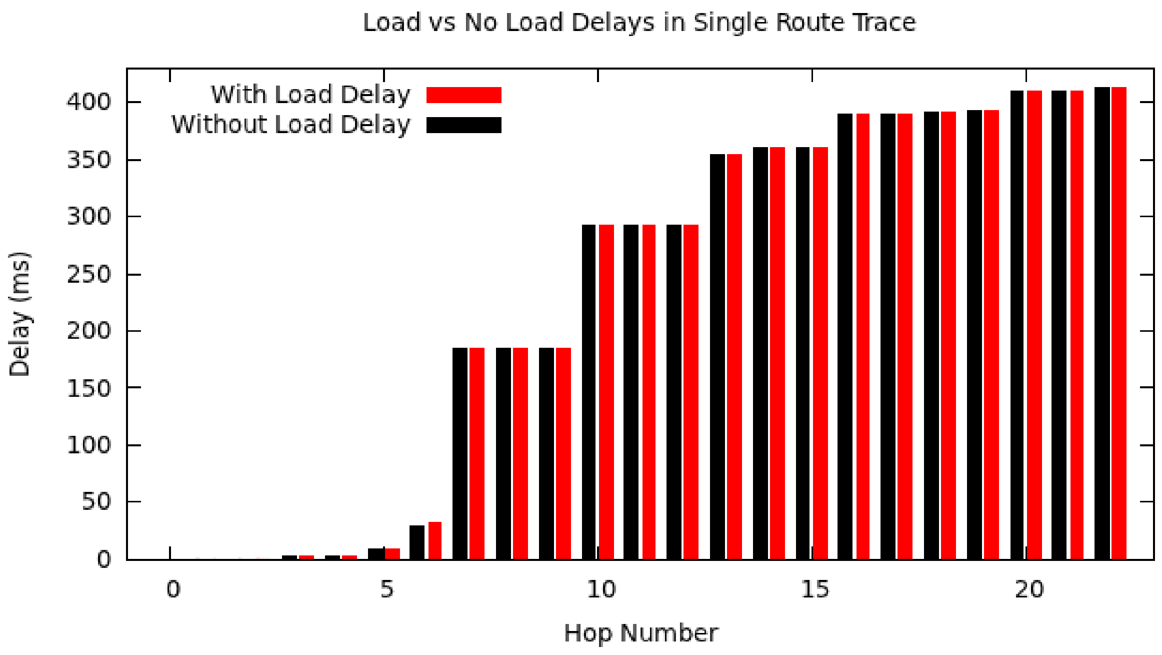

Figure 4.17: Results if Delays to Each Node in Route of Load and No Load Runs

as determined by a run with load and a run without load. This graph more clearly shows how negligible the difference is between the delays of a run with load and a run without load.

Even the significant difference seen in hop 6 makes little difference to the over all delay in the route. This can further be justified by the fact that even in the run with load, the final hop is actually processed quicker than the run with no load on the routing simulator.

## 4.8 Hops per Second with No Delay

This test is aimed at benchmarking a network. This will be done by removing all network packet transfer delays and only considering the time for processing delay of the simulator. A network's efficiency can be tested directly by seeing what the average source to destination host packet transfer time is.

The idea behind this test is to further see if different configurations of a network are indeed better or worse. For example a network with many hosts but smaller routes, compared to that of a network with longer routes but more services hosted on a node. Delay in this kind of test brings in inaccuracy, as a path with little delay may seem better than

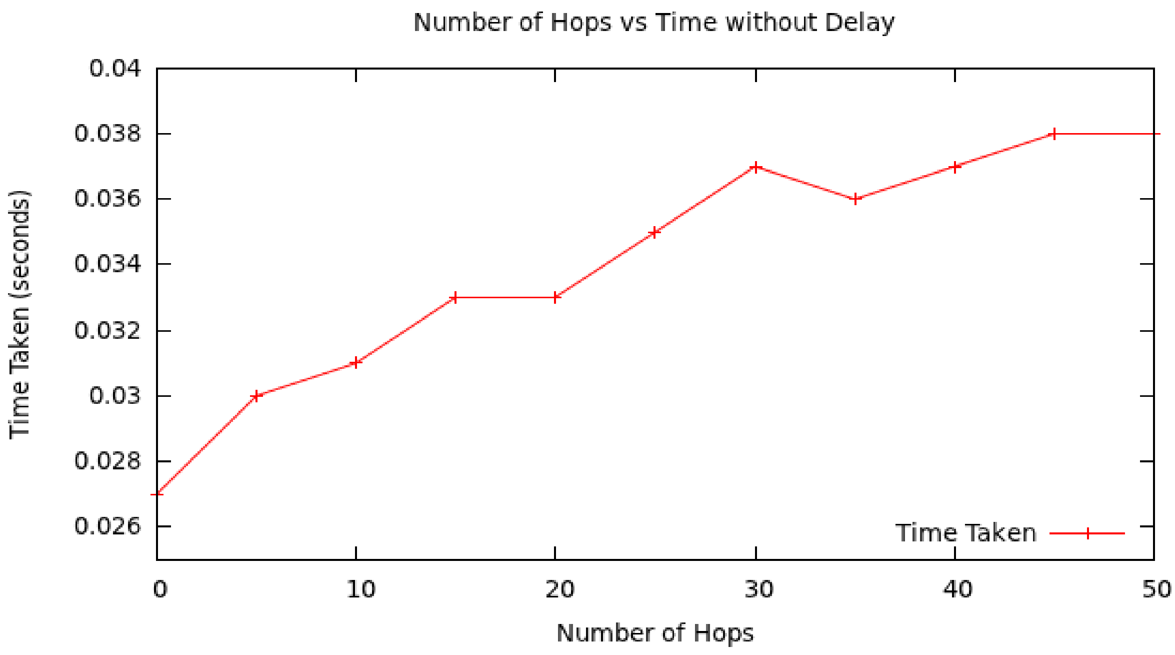

Figure 4.18: Time Taken to Complete Set Hop Count without Delay to Test How Many Hops the Simulator can Handle per Second using Static Routing

that with large delays. However, the processing delay of the longer route may be less but obscured by the fact that there is a large delay.

This test can also be used to see how compatible a host system is with this software. The slower the average transfer rate on a set network, the less compatible the host system is. This is due to the routing capabilities mostly relying on the hardware of the actual configured network.

The test on the dynamic approach will again be conducted in the same way, and with the same hop increments as that of the static tests. This is so that a direct comparison can be made in the results. The aim of this test is again to see how many hops the routing simulator can perform with no delay applied.

It is again predicted that this dynamic approach will yield a lesser amount of hops per second than that of the static approach. This is due to the nature of the algorithm, as the dynamic approach trades off less memory requirements for higher CPU requirements. As such, it should take longer to calculate the next hop, and so result in less hops occurring per second.

Figure 4.18 shows the times taken from zero hops to fifty hops, in steps of five, taken to complete routing a single packet within the routing simulator. This test is done with no

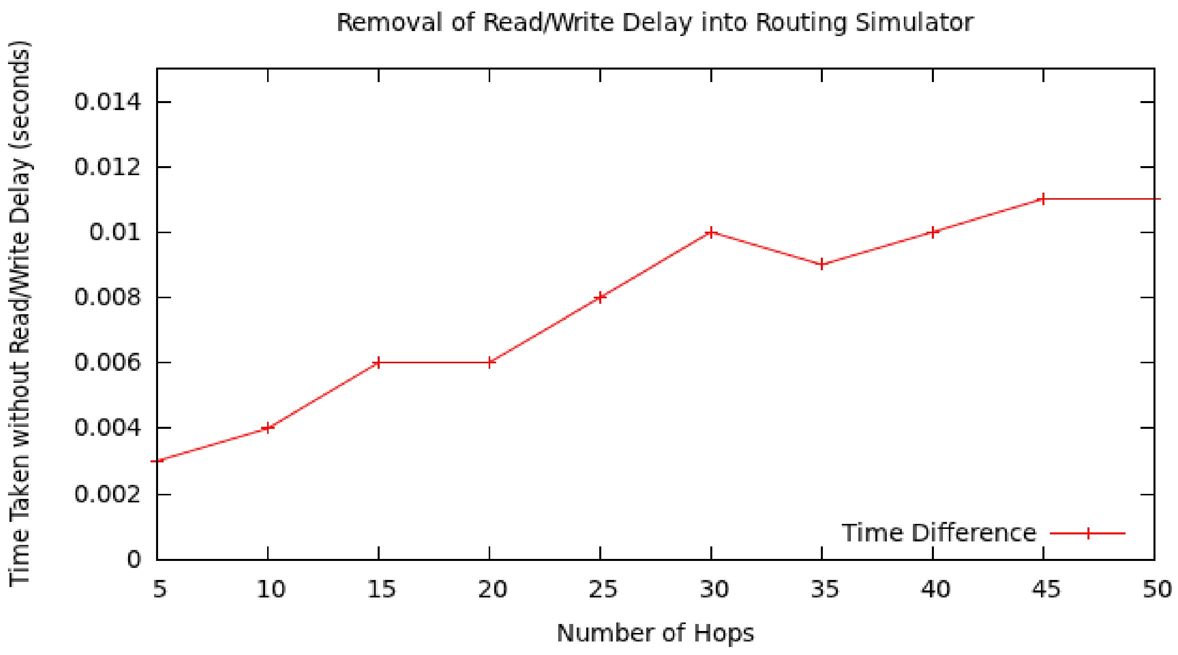

Figure 4.19: Time Taken to Complete Set Hop Count with removal of Read/Write Delay using Static Routing

delays and was intended to see what is the peak amount of hops the routing simulator can achieve per thread per second.

This graph shows an upward trend following roughly a linear trend. The entropy in the results is most likely caused by CPU scheduling. This is because the task is performed so quickly that minor scheduling behavioural changes can bubble up into the results and cause this randomness to occur.

It is still, however, clear that as the number of hops increase, the overall trend of the time taken to complete the task increases, and so an upper limit can be roughly determined as to what the hops per second will amount to on average.

Also, a point to note is the zero hop point in Figure 4.18. This is a network where the entry node into the core is the destination node. This means that there is no node transfer during simulation, and thus only the times taken to read in the packet to be routed, and write out the packet back onto the wire, are taken into account. This allows for this number to be removed, and thus help in producing a more accurate result.

Figure 4.19 shows the extra time taken to process the extra hops delegated to the processor in the routing of a single packet within this routing simulator. This graph was produced to help in calculating the predicted hops per second by a thread in this routing simulator.

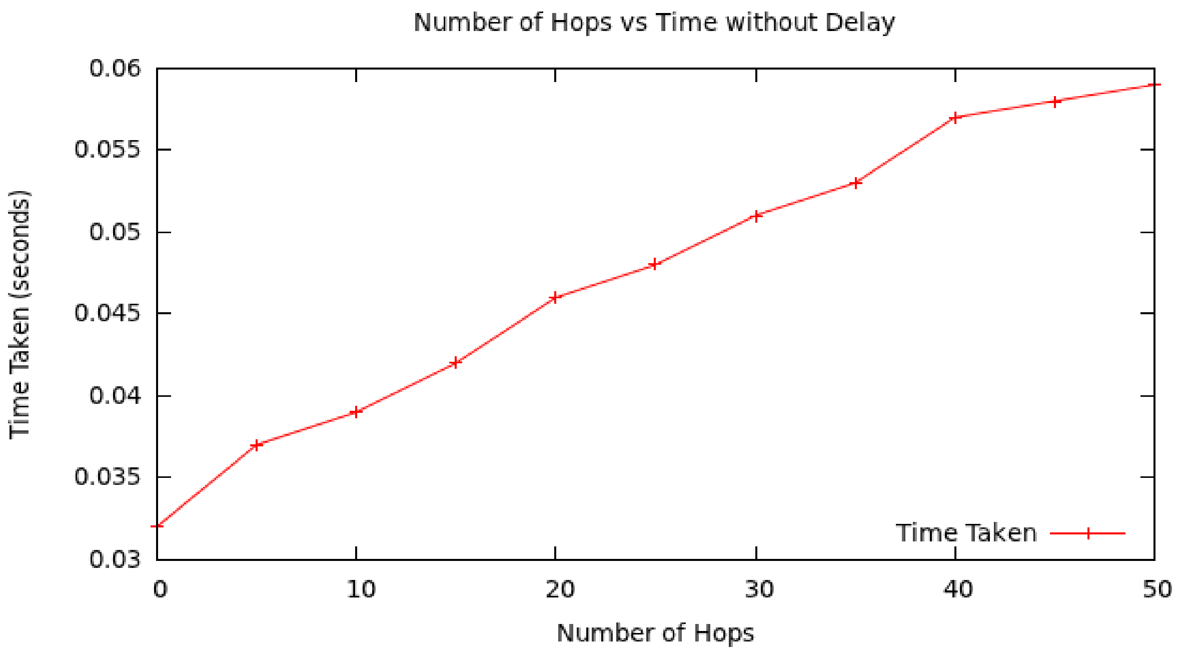

Figure 4.20: Time Taken to Complete Set Hop Count with No Delay using Dynamic Routing Approach

Taking the average of a single hop is what is required first in this calculation. This is done by adding up all the times taken without read/write delay onto the wire performed by the routing simulator. This equates to 0.072s. Next one has to divide this by the total number of hops performed throughout testing. The total number of hops is 275 and the quotient comes to 0.00026(18)s, bracket symbolize recurring digits.

So, in a second this routing simulator can achieve a total estimate of 3819 hops per second. This means that 3819 hops will introduce roughly an extra second of delay in processing time. This number however, must be taken in mind that most networks hardly ever surpass twenty hops in a route, even in the Internet today. For that reason almost every packet is transmitted with a TTL (Time to Live) of sixty four as it is not expected to ever need to hop more than sixty four times in a route.

What this means is that in the worst case that there is such a CPU intensive route introduced into the system with an average of twenty hops, there will be no more than six milliseconds of CPU delay introduced into the system.

Figure 4.20 is intended to show the same results as Figure 4.18 for a dynamic routing approach to a routing simulator.

Like Figure 4.18, Figure 4.20 also shows an upward trend. However, it is with less noise

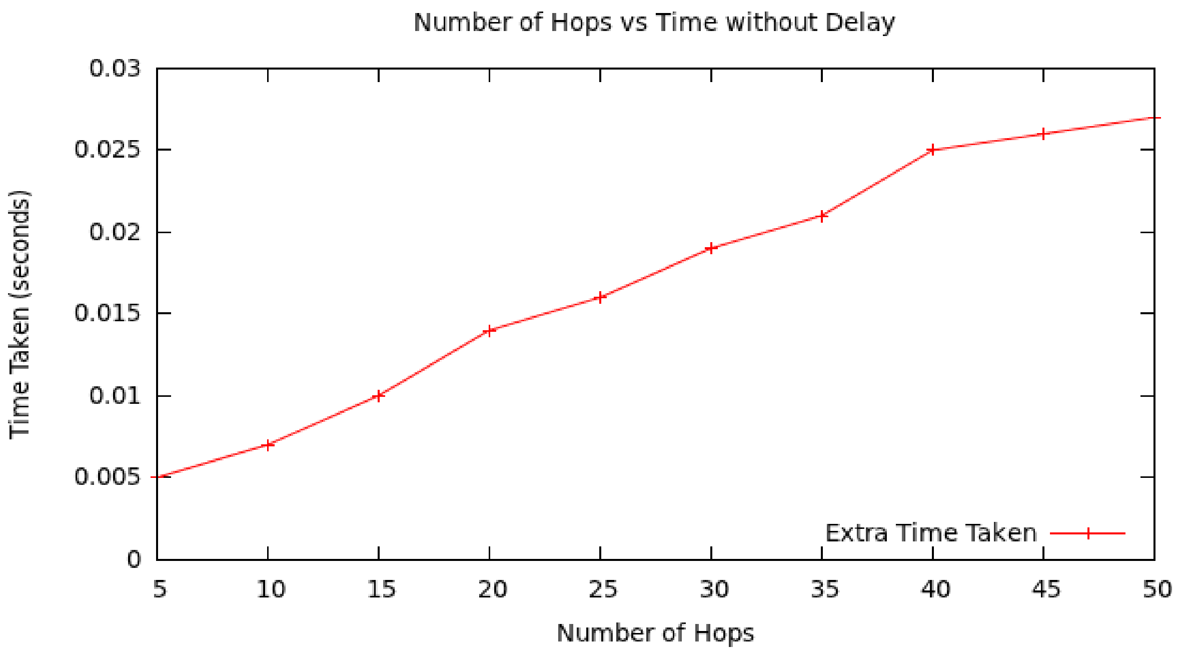

Figure 4.21: Time Taken to Complete Set Hop Count with No Delay using Dynamic Routing Approach

than that of Figure 4.20. As such, Figure 4.20 shows a linear trend more clearly than that of Figure 4.18.

The reason for less noise is probably because, as a whole, the routing task with a dynamic approach took longer than that of a static one. With this, scheduling of this given task is more regular on the CPU than that of shorter tasks. This is probably the reason for smoother results than that of the Figure 4.18.

This being said, other factors may come into play beyond the scope of what is visible directly from tests run on Narwhal, or that can be managed. Figure 4.20's results however, are more consistent and support the estimate of that of Figure 4.18.

This graph in Figure 4.21 was produced to show how much extra time it takes to complete a further five hops without delays created by writing to or reading from the network adapter. As in the static routing case, these results will be used to help calculate an estimate on the maximum hops per second that this dynamic routing approach may achieve.

Following the same calculation as used in the static routing approach, first one adds up all times taken without the read/write delay, this comes to a total of 0.17s. The next step

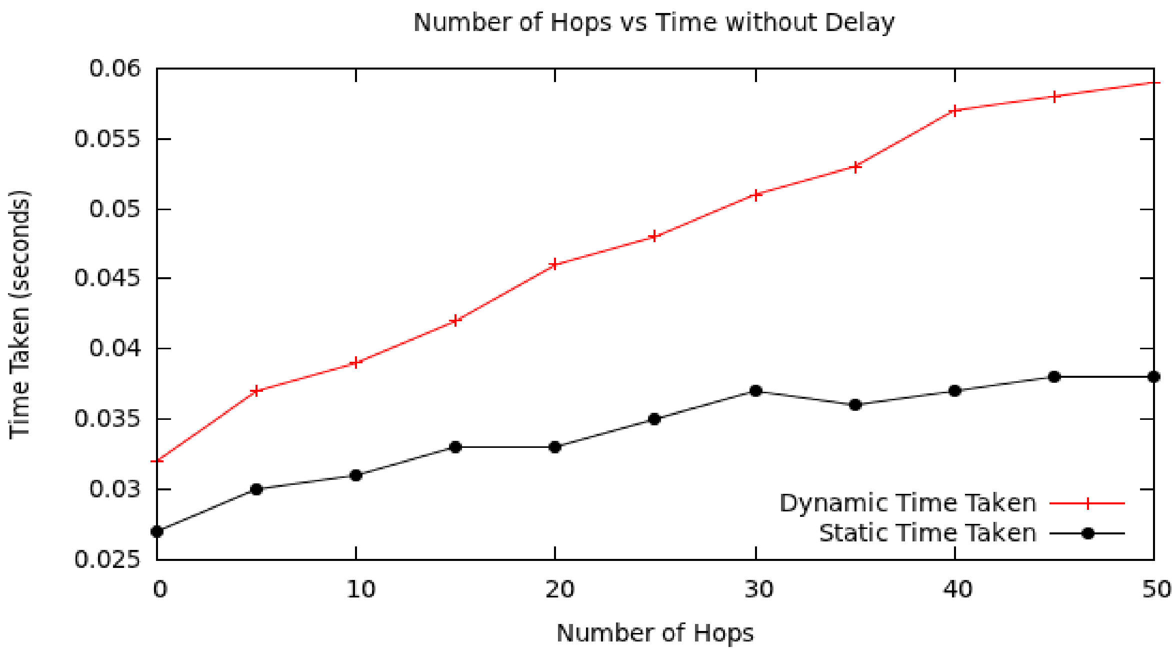

Figure 4.22: Time Taken to Complete Set Hop Count with No Delay Difference between Dynamic and Static Routing Approach

is to divide this total by the total number of hops to get the average taken to perform one hop. The total number of hops again is 275, and dividing the delay total by this gives the result 0.0006(18)s, where brackets represent recurring digits.

The last step is to divide the previous answer by a second to find out the estimated number of hops per second that this approach can achieve. This results in 1618 hops per second and will produce no more than an estimated thirteen milliseconds of delay per an average 20 hop route.

This is less than half of the 3819 hops per second that the static approach is estimated to achieve. It was however expected, and so this result shows again that the dynamic approach is a lot more CPU intensive than that of the static approach, and thus the dynamic approach takes longer to route.

This Figure 4.22 was produced to show the difference in time taken to route a set number of hops between a dynamic and static routing system. This graph further shows the speed difference between that of a dynamic and static approach, where a static approach takes less than half the time of that of a dynamic one.

This graph also shows that the estimation calculations are indeed correct, as it is estimated that a static approach can route more than twice the number of hops of that of a dynamic one.

## 4.9 Unserviced Packets

A packet being unserviced is because of the queue of packets coming into the simulator overflows and packets get dropped. This is also a result of the load on the systems network interface being too high and packets not being picked up at all.

A network simulator should be able to handle all packets at the speed of the actual network bandwidth rate of the interface that it is connected to. If a network simulator cannot handle at least this, it will lead to errors within communications between hosts and inaccuracies in results produced by Narwhal.

To test this, the simulator will be pushed to the limit of the connection speed of the interface it is sniffing from. From here packets will be put onto this network from multiple hosts until the connection is fully saturated. The number of packets on the network interface will be known and the number of packets sniffed will be recorded. This will result in the number of packets that are serviced as well as those which are unserviced.

It is, however, expected that if the total in-coming traffic does not exceed that of the bandwidth provided by the network adapter, there should not be any loss of packets.

Upon testing, the results were as expected. Unless the amount of data put onto the wire by all connected hosts exceeds that of the maximum bandwidth available by the network adapter used by the routing simulator to listen to incoming packets, all in-bound packets are serviced.

The reason for the simulator handling such load is because the host, based on which the simulator runs, can process the packets as fast as they are sniffed off the wire. This allows for no delay in sniffing the next packet, and so there is no chance of any over flows in the network adapters buffers.

These results may not hold if the host on which the routing simulator is running does not have enough CPU power to process data as fast as the network adapter is able to read the data. This will result in delay and loss of packets, as buffers will fill and overflow, at which point old packets will be lost, or any new packets will be ignored, depending on design.

## 4.10 Summary

This chapter shows the capabilities of this routing simulator as a full scale routing mechanism that can be used to replace hardware within a real network. The tests performed on the routing simulator were explained in detail, and information was given on how the tests were performed and what tools were used in performing them, where tools applied.

The results were presented in a graphical or tabular manner, followed by interpretation of what they meant. These were followed, in some cases, by further predictions and behaviour of the routing simulator, and estimates as to peak performance were drawn out from these interpreted results.

These tests were kept to a guideline of what was relevant in today's real networks. This mainly refers to throughput, resources required and delay introduced by the use of a routing simulator instead of the actual hardware.

Extra emphasis was given to the trade-off between memory and CPU requirements between the dynamic and static routing approaches. This was detailed as to why this happens, and some further detail was given into how the system works for further understanding of why this trade off occurs.

Also outlined was how, as more nodes were added to the simulated network, the less extra memory required to do so was needed. This lead to a logarithmic requirement in memory as nodes were added to the simulation when using memory requirement as the dependant variable. This too was explained.

# Chapter 5

# Conclusion

This research to implement and test a routing simulator that could route packets on an IPv4 datagram level implementing both TCP and UDP along with ICMP implementation. The routing simulator discussed in this paper achieves all of the above objectives.

Other issues addressed after such fundamental properties of a routing simulator were achieved, was that of both CPU and memory requirements. During design and implementation these were both kept as key points and as such any unnecessary waste of both CPU and memory was tracked and then corrected as to keep both requirements to a minimum.

## 5.1 Key Aspects

The key aspects in implementation of this routing simulator was memory, CPU and throughput. The goal was to keep memory and CPU resource requirements as low as possible while trying to maximize on the overall throughput of packets being routed. A conclusion of these results follows.

### 5.1.1 Memory

This routing simulator keeps memory usage down to a minimum, as it requires less than seven megabytes of RAM to store just under 19000 reachable nodes with routing paths in a static approach as tests show in Section 4.2. This figure is bested by that of the dynamic approach needing less than two megabytes of RAM to achieve the same task at the expense of extra CPU requirements.

In July of 2012 there were an estimated 900 million reachable nodes in the Internet (Internet System Consortium, 2012). Using this figure, it is estimated that the above dynamic routing approach would require 92GB of memory, this is estimated at two megabytes for every 19000 nodes. This means that one could simulate an instance of every mappable node in the Internet within a computer with a 128GB of RAM. Technology that can achieve this figure is available to the public in the higher end spectrum of commodity hardware.

With the ability to map the largest network known to humans into a readily available technology, this software implementation of a routing simulator achieves its task of being memory effective. This however does come at a trade off for CPU requirements, as calculating routes in real time brings a toll onto the host system.

### 5.1.2 CPU

This being said, results from Section 4.8 show that, on a standard network, no more than thirteen milliseconds of delay will be introduced into the system in the very special case of no hop delay. This style of routing adds significantly more load onto a CPU via a routing algorithm, as there are no breaks between each route calculated, which means every packet entering the routing simulator will be contending for CPU resources.

Thirteen milliseconds added delay can also be accounted for by prioritizing shorter routes to minimize the effect of this delay and by subsidizing longer routes delays according to the CPU load. In all though, in standard routing of a real time network such as the Internet, these delays won't be anywhere near this figure. So one can conclude that this routing system brings in no significant noise under routing in normal conditions. Furthermore this makes this routing system even more successful.

If one were concerned about this extra delay, this delay can be reduced by using the static routing approach which involves precalculating all routes. This, as mentioned, requires more memory, but reduces CPU requirements significantly. This then reduces the delay introduced by routing to no more than six milliseconds, and so allows for more accurate routing.

### 5.1.3 Throughput

From a throughput point of view, this routing simulator under-performs. Although it does surpass the mark set in testing of 20Mbps, the actual throughput of roughly 40Mbps, from results in Section 4.5, that this routing simulator achieves does not even meet the throughput of a common network, this being a 100Mbps Ethernet connection.

Although a positive is that this routing simulator does support multiple connections at very little loss of throughput, it maintains an average of 40Mbps while servicing multiple connections.

This being said, even though Narwhal reaches a speed that is more than enough to simulate a localized portion of the Internet, such as a country or part of a continent with multiple connections, it is not recommended that one should attempt to use this software to simulate any major continental interconnects or local area network connections.

Therefore this software has limited use and should only be applied to lower speed networks as found in use within the Internet. Any network faster than 40Mbps should keep its size to a minimum as to allow for a slight improvement in throughput due to less strain on the host system running this software. The other approach to mitigating this problem would be to upgrade the hardware within the host system.

## 5.2 Cost and Packet Servicing

In terms of cost, this routing simulator makes use of free libraries namely, libpcap, libnet and makes use of the standard pthread library found in C. These libraries are easily installed and well supported by both creators and the community as discussed in Sections 3.8.4 and 3.9.1.

The operating system used, Linux Ubuntu 11.10 as discussed in Section 3.7, is also freely distributed, and so reduces the cost of this software to zero. All that is required in terms of cost is that the hardware in which this software is intended to run on supports above mentioned libraries and operating system.

In terms of servicing packets generated by hosts on the network being simulated, Narwhal can handle a throughput of packets up to the point of that which the network adapter is rated to. This must also be compared to the hardware limitations of the host system, as if the CPU and memory can't perform up to the rate at which packets are being introduced into the system by the network adapter, then there will be unserviced packets introduced from this factor as well.

As originally intended, see Section 1.1, this routing simulator is completely compatible with commodity hardware that is readily available to the public. The result of this is an increase in the availability of this software to anyone who requires routing simulation.

## 5.3 Closing Statement

With all the above results considered, this routing simulator is applicable in effectively simulating the routing of packets up to a speed of 40Mbps. This is done while keeping delays introduced from processing to a minimum. Also this routing simulator has the ability to route packets through simulated networks on a large scale, as memory requirements as a network configuration grows is kept as low as possible. This allows it to simulate every node in the Internet within the limitations of hardware available to the public. With all this considered, Narwhal manages to meet all goals set out in this research.

## 5.4 Future Research

Future research potential in this field is abundant and is mainly sourced from ideas that came up during implementation. Also these came from features that were originally intended for this system but the time frame for which this project was allocated was too short. There is much improvement that can take place within this software implementation of this routing simulator. Future research ideas and propositions are as follows.

- Introduction of fake hosts within the routing simulator. These fake hosts go about generating traffic as a normal host would. This can be done through duplication of data received from a real hosts or through relying on an individual thread within the simulation system to generate the traffic from a node.
- Extending protocol capabilities by introducing IPv6 (Internet Protocol version 6) and allowing for support of all already existing protocols using IPv6 routing. This will further to secure this software's future use and allow for further testing and experimentation.

- Allowing interfacing of routing onto a hardware interface such as a co-processor or GPU is a feasible idea although this routing system was originally intended for software use only. The benefits of such an implementation will most likely result in vast improvement in throughput and packets routable within the simulator. These will come at a cost though but the trade off may be worth it.

- Rewriting the thread driven routing system to one that is asynchronous. This allows for a single thread of execution and thus less managing needs to take place. The recent results produced by web serving as done by node.js[1] shows much promise in rewriting Narwhal to use asynchrony rather than a thread based approach. This also removes the limitation of the number of threads that can be spawned within a process.

- Adding more realistic queues to nodes. This will allow for simulating buffer overflows and also allow for easier prioritization of packets being routed in the simulator. Will also help keep the order of packets being ejected out of the simulator.

- Introduction of load balancing. This can be done within a thread based system by monitoring how much load is on each core of the host systems CPU and distributing load according to this. If applied to an asynchronous approach this can be done by balancing load across the multiple asynchronous nodes.

- Use of OpenCL to handle routing. This will allow for easier transfer onto a GPU solution as but not remove the option of running simulation on a common multi-core CPU.

- Introduction wormhole routing or flow control mechanisms. This will allow for more realistic routing and added functionality into the routing simulator. Introduction of such a feature will allow further packet control within the routing simulator.

- Implementation of BGP and CIDR protocols to allow for full Internet routing simulation accross multiple continents. This will also improve dynamic routing but may come at a cost of further memory requirements.

Improvements in throughput. Possible rewriting of the routing algorithms in place in this routing simulator implementation can and probably will see improvements in throughput achieved by this routing simulator.

[1]http://nodejs.org/

These are just some ideas for future research in this area, these ideas can be applied across many similar routing simulators and as such further the achievements in this field of Computer Science.

# References

**03b and Sofrecom**. *Why latency matters to mobile backhaul*. 2012. Accessed 15 October 2012.
URL `http://www.o3bnetworks.com/media/45606/latency%20matters.pdf`

**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.

**Apposite Technologies**. *Wan emulation made easy*. 2011. Accessed 2 February 2012.
URL `http://www.apposite-tech.com/index.html`

**Baker, F.** *RFC 1812. Requirements for IP version*, 4:1995–06, 1995.

**Baumgartner, F., Scheidegger, M., and Braun, T.** *Simulating router-and domain characteristics*. In *International Workshop on*, volume 117, pages 139–145. 2004.

**BGPVista**. *bgpvista*. 2009. Accessed 1 March 2012.
URL `http://www.bgpvista.com/simbgp.php`

**Bhuyan, L. and Wang, H.** *Execution-driven simulation of ip router architectures*. In *Network Computing and Applications, 2001. NCA 2001. IEEE International Symposium on*, pages 145–155. IEEE, 2001.

**Bodenstab, D., Houghton, T., Kelleman, K., Ronkin, G., and Schan, E.** *Unix operating system porting experiences*. *AT&T Bell Laboratories Technical Journal*, 63(8):1769–1790, 1984.

**Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., Huang, P., McCanne, S., Varadhan, K., and Xu, Y.** *Advances in network simulation*. *Computer*, 33(5):59–67, 2000.

**CAIDA**. *Caida data - overview of datasets, monitors, and reports.* 2012. Accessed 1 March 2012.
URL `http://www.caida.org/data/overview/`

**Cieslak, M., Mayes, J., and Lowe, R.** *Methods and apparatus for caching network data traffic.* May 29 2001. US Patent 6,240,461.

**Cisco**. *Using the traceroute command on operating systems.* August 2005. Accessed 31 Octoboer 2012.
URL `http://www.cisco.com/en/US/tech/tk364/technologies_tech_note09186a00801ae32a.shtml`

**Communications Inc**. *Network emulation with data rates up to 10 gbps.* 2012. Accessed 28 February 2012.
URL `http://packetstorm.com/psc/psc.nsf/site/index`

**cplusplus**. *A brief description.* October 2012. Accessed 30 October 2012.
URL `http://www.cplusplus.com/info/description/`

**Date, C. and Darwen, H.** *A Guide to the SQL Standard*, volume 3. Addison-Wesley Reading (Ma), 1987.

**Deering, S. and Hinden, R.** *RFC 2460: Internet Protocol.* 1998.

**GoldSim Technology Group**. *What is simulation.* 2010. Accessed 25 October 2012.
URL `http://www.goldsim.com/Web/Introduction/Simulation/`

**Goodacre, B.** *Increase simultaneous/concurrent TCP connections (Linux).* January 2011. Accessed 30 October 2012.
URL `http://ben.goodacre.name/tech/Increase_simultaneous/concurrent_TCP_connections_(Linux)`

**Intel**. *How operating systems can do more and perform better.* 2012. Accessed 22 May 2012.
URL `http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`

**Internet System Consortium**. *The ISC Domain Survey.* July 2012. Accessed 27 October 2012.
URL `http://www.isc.org/solutions/survey`

**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, V., Leres, C., and McCanne, S.** *pcap-packet capture library. UNIX man page*, 2001.

**Kurose, J. F. and Ross, K. W.** *Computer Networking.* PEARSON, 2010, 261–268 pages.

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

**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.

**Microsoft**. *Windows.* September 2012. Accessed 30 October 2012.
URL `http://windows.microsoft.com/`

**National Science Foundation**. *Ipv4 class defnition.* 2012a. Accessed 19 May 2012.
URL `http://www.nsnam.org/doxygen/classns3_1_1_ipv4.html`

**National Science Foundation**. *Ipv6 class defnition.* 2012b. Accessed 19 May 2012.
URL `http://www.nsnam.org/doxygen/classns3_1_1_ipv6.html`

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

**Newman, P., Minshall, G., Lyon, T., and Huston, L.** *IP switching and gigabit routers. Communications Magazine, IEEE*, 35(1):64–69, 1997.

**Pingdom AB**. *Pingdom.* 2011. Accessed 11 Octboer 2012.
URL `http://royal.pingdom.com/2010/11/12/`

**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 791: Internet protocol.* 1981a.

**Postel, J.** *Rfc 792: Internet control message protocol. InterNet Network Working Group*, 1981b.

**Production Modelling**. *Why simulate?* 2012. Accessed 25 October 2012.
URL `http://www.simulation.co.uk/solutions/why-simulate.html`

**Rekhter, Y.** *Rfc 1518: An architecture for ip address allocation with CIDR.* 1993.

**Rekhter, Y. and Li, T.** *RFC 1771: A border gateway protocol 4 (BGP-4).* 1995.

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

**Riley, G., Ammar, M., and Fujimoto, R.** *Stateless routing in network simulations.* In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on*, pages 524–531. IEEE, 2000.

**Santner, T., Williams, B., and Notz, W.** *The design and analysis of computer experiments.* Springer, 2003.

**Schaller, R.** *Moore's law: past, present and future. Spectrum, IEEE*, 34(6):52–59, 1997.

**Smith, L. and Lipner, I.** *Free pool of IPv4 address space depleted.* 2011. Accessed 1 March 2012.
URL `http://www.nro.net/news/ipv4-free-pool-depleted`

**Southwestern Bell Internet Services**. *Classless Inter-Domain Routing (CIDR) Overview.* 2001. Accessed 30 October 2012.
URL `http://public.swbell.net/dedicated/cidr.html`

**Stone, A., DiBenedetto, S., Mills Strout, M., and Massey, D.** *Scalable simulation of complex network routing policies.* In *Proceedings of the 7th ACM international conference on Computing frontiers*, pages 347–356. ACM, 2010.

**Tanenbaum, A. S.** *Computer Networks.* Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, third edition, 1996, 416 pages.

**Technologies, A.** *WAN Emulation Made Easy.* 2012. Accessed 11 May 2012.
URL `http://www.apposite-tech.com/index.html`

**tmn Simulation**. *What can simulation do for me?* 2011. Accessed 25 October 2012.
URL `http://tmnsimulation.com.au/simulation/why-simulate/`

**Wang, P.** *Libnet documentation.* February 2003. Accessed 19 May 2012.
URL `http://libnet.sourceforge.net/libnet.html`

**Wang, S., Chou, C., Huang, C., Hwang, C., Yang, Z., Chiou, C., and Lin, C.** *The design and implementation of the nctuns 1.0 network simulator. Computer networks*, 42(2):175–197, 2003.

**Wolf, T. and Turner, J.** *Design issues for high-performance active routers. Selected Areas in Communications, IEEE Journal on*, 19(3):404–409, 2001.

**Xu, D. and Ammar, M.** *Benchmap: Benchmark-based, hardware and model-aware partitioning for parallel and distributed network simulation.* In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pages 455–463. IEEE, 2004.

**Zhao, J. and Govindan, R.** *Understanding packet delivery performance in dense wireless sensor networks.* In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 1–13. ACM, 2003.