UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

Mininet Network Emulator

by

Ignatius Thomas de Villiers
17502292

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch

Study leader: Dr. H.A. Engelbrecht

November 2016

Acknowledgements

I have taken efforts in this project. However, it would not have been possible without the kind support and help of three individuals. I would like to extend my sincere thanks to all of them.

I am highly indebted to Dr. H.A. Engelbrecht for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.
I would like to express my special gratitude and thanks to Mr. I.D. de Villiers who's encouragement and contribution in terms of software structure guidance and the reviewing of all functional code, has helped me to develop a Graphical User Interface (GUI) for Mininet.
My thanks and appreciations also goes to Mr. P. le Roux who willingly helped me with his abilities.

Decleration

I, the undersigned, declare that this dissertation is my original work, gathered and utilized especially to fulfil the purposes and objectives of this study, and has not been previously submitted to any other university for a higher degree.

Signature: ....
I.T. de Villiers

Date: ....

Abstract

Network emulation is a very useful tool, that allows us to test the performance and evaluate behavior of networks, without having to physically build the network. Mininet is a fast and reliable network emulator, and often described as one of the best open source network emulators. The aim of this report is to give an overview of Mininet as a network emulator and to give information on how the emulator's accuracy, scalability was tested. To aid in testing of various networks, a Graphical User Interface (GUI) was developed. This GUI would therefore allow any user to build custom, parameterized Mininet networks quickly and easily. We will also take a brief look at how Mininet works and what it's limitations are. We effectively tested Mininet's bandwidth-, delay, packet loss and jitter consistency, to determine at which point measured values don't agree with theoretical values and why the values don't agree. A further qualitative test was to stream actual video over the emulated network, to see how different network effects, affect a streamed video's quality. We then continued to connect Mininet hosts running on different hardware devices (Raspberry Pi's), by using Generic Routing Encapsulation (GRE) tunneling to connect the two Mininet networks - This was in essence a test of Mininet's scalability. The same tests were performed, to ensure that Mininet's performance stays consistent across both hardware and software interfaces. These tests helped us to draw insightful performance graphs, which uncovered some of Mininet's strengths and limitations. A final test was to build a physical network, consisting of real hosts and switches. The GUI was then used to build a replica of this network, under the same network conditions, in Mininet. Testing whether the emulated network performs the same as the physical network, would prove that Mininet is an accurate and efficient emulator.
From test results we were able to conclude that Mininet is an accurate and scalable network emulator. However, Mininet's performance relies heavily on the host device's available resources. Following an in-depth discussion of test results, contributions of this project, some recommendations and examples of further work will be given and discussed.
[Afrikaans to Follow]

Uittreksel

Netwerk Emulasie is ’n baie handige stuk gereedskap, wat die toets van ’n netwerk se vermoë en die evalueering van ’n netwerk se gedrag moontelik maak, sonder om die netwerk fisies op te stel. Mininet is ’n vinnige en betroubare netwerk emulator, wat gereeld na verwys word, as een van die beste netwerk emulators. Die doel van dié verslag is om ’n opsomming van Mininet as ’n emulator te gee en inligting te gee oor hoe ons die akkuraatheid en skaalbaarheid van Mininet getoets het. Om die toets van verskeie netwerke te vereeneenvoudig was ’n grafiese gebruikerskoppelvlak (GGK) ontwikkel. Dié GGk maak dit dus moontelik om geparameteriseerde Mininet netwerke vinnig en effektief op te stel. Ons gaan ook ’n kort oorsig gee oor hoe Mininet werk, en wat sy beperkings/tekortkominge is. Ons het effektiewelik Mininet se bandwydte-, vertraging, pakkie-verlies en strydige pakkie vertraging (jitter) getoets, om vas te stel op watter punt Mininet nie meer ’n netwerk akkuraat kan emuleer nie. ’n Verdere kwalitatiewe toets was om regte video oor die geemuleerde netwerk te stuur, om te toets of gemete netwerk effekte kwalitatief realisties is. Ons het toe voortgegaan deur Mininet *hosts*, vanuit verskillende netwerke, wat op verskillende toestelle (Raspberry Pi’s) hardloop te konnekteer, deur gebruik te maak van Generiese Roete Omsluiting (*Generic Routing Encapsulation*) - hierdie was effektief ’n skaalbaarheids toets vir Mininet. Dieselfde toetse wat voorheen gebruik is om Mininet te toets, was weer gebruik, om te verseker dat Mininet se vermoë konsekwent bly oor ’n sagteware- en hardeware koppelvlak. Toetsresultate het dit vir ons moontelik gemaak om insiggewende grafieke te teken, wat Mininet se sterkpunte en beperkings ontbloot het. ’n Finale toets was om ’n regte netwerk, wat uit regte komponente bestaan op te stel. Die GGK was toe gebruik om ’n identiese netwerk, met identiese netwerk kondisies, in Mininet op te stel. Deur te toets of die geemuleerde netwerk dieselfde optree as die regte netwerk, kan ons die gevolgtrekking maak, dat Mininet ’n akkurate emuleerder is.
Vanuit die toetsresultate, kon ons die gevolgtrekking maak dat Mininet wel ’n akkurate en skaalbare netwerkemuleerder is. Tog, hang Mininet se vermoë baie af van die beskikbare hulpbronne op die gasheer rekenaar. Na ’n in-diepte bespreking van toets resultate word bydraes van die projek, ’n paar aanbevelings en voorbeelde van verdere werk gegee en bespreek.

Contents

List of Figures

List of Tables

List of Abbreviations

API - Application Program Interface

BGP - Border Gateway Protocol

BW - Bandwidth

DHCP - Dynamic Host Configuration Protocol

GGK - Grafiese Gebruikerskoppelvlak

GRE - Generic Routing Encapsulation

GUI - Graphical User Interface

IETF - Internet Engineering Task Force

LAN - Local Area Network

Mbps - Mega bits per second

ms - millisecond

NAT - Network Address Translation

RTP - Real-time Transport Protocol

SDN - Software Defined Networking

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

VLC - VideoLAN Client

XMPP - Extensible Messaging and Presence Protocol

Chapter 1

Introduction

1.1 Background

A network is described by Cisco.com (2011), as a system that connects hosts using two pieces of equipment; switches and routers. To understand this definition of networking, we have to understand what exactly hosts, switches and routers are and what each component's role is within the network. A network host can be defined as any device connected to a network. Each host in a network has a unique Internet Protocol (IP) address used for communication within a network. Switches, as described by Cisco.com (2011) are used to connect multiple devices on the same network. A switch therefore allows various devices to share information and communicate. Another component commonly used for connecting devices in a network, is a hub. Hubs differ from switches in a sense that instead of forwarding data from one device to another, it broadcasts the data to all of the hosts within the network. A router, according to Cisco.com (2011), is used to tie multiple networks together. It serves as a dispatcher, choosing the best route for information to travel, to ensure optimal transmission of information. Routers effectively analyzes the data it receives, repackages it and sends it to another network. Typically in any network, various network effects limit the network's performance. These effects are bandwidth limitation, packet loss, delay and jitter.

According to Opennetworking.org (2013), Software-Defined Networking (SDN) is an emerging network paradigm. This architecture enables network control to become programmable, by separating the network control plane from the forwarding plane. According to McNickle (2014), Software defined networks are most commonly implemented by the OpenFlow protocol, but in recent times other protocols like Border Gateway Protocol (BGP), Internet Engineering Task Force (IETF), Extensible Messaging and Presence Protocol (XMPP) and many more, have been implemented.

According to (Cranson and Santay, 2003) a Network Emulator can be described as a hybrid of the two most common techniques for testing SDN: simulation, which provides a synthetic environment that allows for repeatable tests under various assumptions ; and testbeds, which provide an environment for running real code. Network effects such as bandwidth restrictions, delays in transmission, packet loss and jitter can typically be simulated by an emulator. Network emulation is also used to test a product's performance under real world network conditions to test actual user experience. Network emulators are therefore handy tools that prevent expensive post-deployment operational and performance issues.

1.2 Motivation

Often Networks are set up, neglecting non-ideal network effects. This means that the network might not perform as initially expected, sometimes to the extent that the network is unsuitable for relaying information, streaming of video or playing games. The solution to this problem, is network emulation, as it provides the tools to test a network's performance, stability and functionality in the presence of various network effects, without having to physically build the network.
It is thus very important to verify that a network emulator is capable of emulating various parameterized network topologies accurately. We therefore need to test the scalability and accuracy of a network emulator, to gain information on what the emulator's strengths and limitations are. In order to ensure the best possible emulation of a network, various network emulators and simulators were considered - but as the scope of this project is to test Mininet as a network emulator, Mininet was used.

1.3 Literature Synopsis

Network effects that typically affect network conditions are; badwidth limitation, packet loss, latency/delay and jitter. Bandwidth is described by Rouse (2014) as the capacity for data transfer of an electronic communication system or the data transfer rate of a network. Bandwidth limitation is therefore the throttling of the data transfer rate of a network. Packet loss, according to Rouse (2007) occurs when one or more packets fail to reach their destination. Packet loss causes significant loss of information and can result in unintelligible video and sound or even complete signal loss. Latency or delay is described by Rouse (2007) as how much time it takes a packet to travel from one destination to another. Jitter is defined by Cisco (2006) as a variation in the delay of received packets and caused by network congestion, improper queuing or configuration errors.
According to github.com (2012), Mininet is a fast and reliable network emulator, that allows the user to create various custom topologies using virtual hosts, switches, links and OpenFlow controllers, with similar behavior to that of the hardware components. It is also possible to run the same binary code and applications on either a software or hardware platform. github.com (2012) states that Mininet uses process-based virtualization and Linux Network namespaces enabling it to support hundreds of virtual nodes on a single OS kernel. Mininet also enables the user to create custom topologies through Python's application programming interface (API).

1.4 Aims and Objectives

Evaluate the performance of Mininet as a network emulator by;

1. Creating a Graphical User Interface (GUI) that allows a user to;
   (a) Set up custom, paramaterized Mininet network topologies quickly and easily
   (b) Save custom Mininet topologies
2. Creating performance tests to;
   (a) Generate performance graphs of bandwidth-,delay-,packet loss- and jitter consistency between Mininet hosts over a (i) Software interface and a (ii) Hardware interface.
   (b) Test Mininet's scalability
3. Compare results of the emulated network, with a real network

1.5 Contributions

This project has lead to the development of a user-friendly GUI, which was used to create and save custom Mininet network topologies. Performance tests were created to evaluate Mininet's bandwidth-, delay-, packet loss- and jitter consistency over a software- and hardware platform. These tests generated data that was used to plot performance graphs and can be seen in Chapter 4 and 5. A further qualitative test was performed by streaming a video over mininet hosts using VLC. Through this test it was possible to see the quality of the video worsen as the packet loss, delay and jitter increased. Using the generated performance graphs from a real network, it was possible to test Mininet's ability to emulate a real network.

1.6 Report overview

The purpose of this report is to give information on Mininet as a network emulator. In **Chapter 2**, various concepts that will aid in the understanding of networking and network emulation are discussed. Different network emulators are compared with one another, by evaluating each emulator's advantages and disadvantages and it's explained why Mininet is the best emulator for this project. In **Chapter 3** , the GUI is explained in terms of it's front-end and back-end. **Chapter 4**, documents the software platform experiment. In this experiment Mininet's scalability and it's ability of emulating parameterized networks over a software interface, was tested. **Chapter 5** documents the hardware interface experiment. In this experiment Mininet's scalability and it's ability of emulating parameterized networks over a hardware interface, was tested. In **Chapter 6**, Mininet's performance is compared to that of a real network. **Chapter 7**, provides a summary of all test results. Finally, in **Chapter 8**, we conclude on Mininet's accuracy, scalability and reliability as a network emulator.

Chapter 2

Literature Study

This chapter gives an overview of the basic concepts of networking and different emulators available. The information provided in sections 1 to 5, will give the necessary insight to understand the report's contents. In section 6, we will compare different emulators by briefly discussing basic features, and the advantages/disadvantages of each emulator. We will also thoroughly inspect Mininet, and discuss how it can help us accomplish our aims and objectives stated in Chapter 1.

2.1 Networking basics

2.1.1 Different components of a network

Host

A Network host can be defined as any device connected to a network. Each host in a network has a unique Internet Protocol (IP) address used for communication within a network.

Switch

Switches, as described by Cisco.com (2011) are used to connect multiple devices on the same network within an enclosed area, like a university building. A switch therefore allows various devices to share information and communicate.

Hub

Another component commonly used for connecting devices in a network, is a hub. Hubs differ from switches in a sense that instead of forwarding data from one device to another, it broadcasts the data to all of the hosts within the network.

Router

A router, according to Cisco.com (2011) are used to tie multiple networks together. It serves as a dispatcher, choosing the best route for information to travel, to ensure optimal transmission of information. Routers effectively analyzes the data it receives, repackages it and sends it to another network.

2.1.2 Different network effects

Bandwidth limitation

Bandwidth is described by Rouse (2014) as an an electronic communication system's capacity for data transfer, or the data transfer rate of a network. Bandwidth limitation is therefore the throttling of the data transfer rate of a network.

Latency/Delay

Latency or delay is described by Rouse (2007) as how much time it takes a packet to travel from one destination to another.

Packet loss

Packet loss, according to Rouse (2007) occurs when one or more packets fail to reach their destination. Packet loss causes significant loss of information and can result in unintelligible video and speech or even complete signal loss.

Jitter

Jitter is defined by Cisco (2006) as a variation in the delay of received packets and caused by network congestion, improper queuing or configuration errors. If the magnitude of the jitter is large enough, some packets may be discarded, causing packet loss in the network.

2.2 Software Defined Networking (SDN)

According to Opennetworking.org (2013), SDN is an architecture that makes Network control directly programmable, configurable, simplifies network design and operation as instructions are provided by SDN controllers. According to Rouse (2015*b*), the OpenFlow protocol is essential in building SDN solutions. In a software-defined network, a network administrator can shape traffic through a centralized control plane, without touching individual switches, and deliver services to devices regardless of its other connections. In real Networks, rules built into a switch's firmware tells the switch where to send a packet, but in SDN, rules for packet handling are sent from the controller to the switch. Controllers and switches communicate through the Openflow interface.

2.3 OpenFlow

Rouse (2012) describes OpenFlow as a protocol that allows a server to tell switches where to send packets that. According to Openflow.org (2008), OpenFlow enables researchers to run experimental protocols in the networks they use every day, without vendors having to expose the inner workings of their switches. Openflow.org (2008) also states that OpenFlow switches separate the two main functions of a traditional switch, namely fast packet forwarding (data path) and high level routing decisions (control path). The data path still occurs on the switch, whilst routing decisions are moved to a separate controller.

2.4 Network Address Translation (NAT)

In order to communicate with the local network from inside the Mininet VM, NAT was used. According to Tyson, NAT allows a single device such as a switch, to act as an agent between the inside network (LAN) and the outside network (Internet). Hosts on the inside network, are usually assigned IP addresses that cannot be routed to external networks, whereas a NAT gateway acquires an address that can connect to external networks. According to WhatIsMyIPAddess.com (2006), NAT gateways receives an outbound request from a host on the inside network, the gateway then sends the same request from its own public address to the internet source and returns the response of the internet source to the host that originally made the request . From Figure 2.4.1 we can see by using NAT the user is able to connect to the internet or other devices within the LAN from a Mininet virtual host.

How NAT will be used in this project

As one of our Objectives is to connect to machines outside of the VirtualBox environment, A NAT adapter will be used in the VirtualBox environment. This will allow VirtualBox to connect to the internet and other hosts in the local network. This will therefore enable Mininet hosts to connect to the the internet and other hosts in the local network.

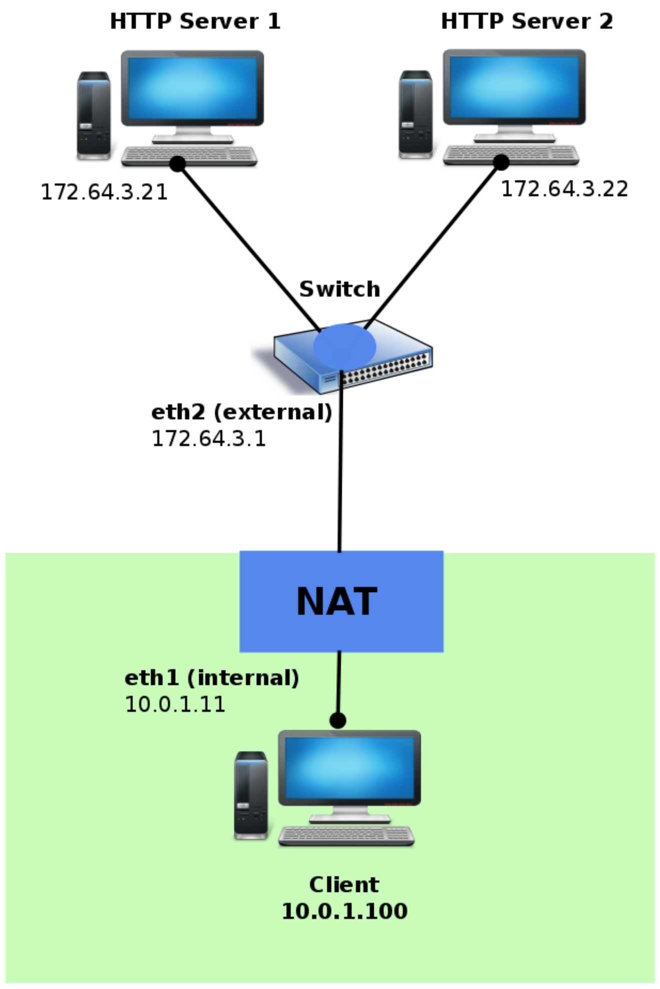


Figure 2.4.1: Mininet NAT, allowing Mininet hosts to communicate with the local network and the internet

2.5 Generic Routing Encapsulation (GRE) Tunneling

According to www.juniper.net (2012), GRE allows the source and destination switches to operate as if they have a virtual point-to-point connection with one another. One switch acts as a tunnel source, that encapsulates the data and routes it to the GRE endpoint, which is another switch, that then de-encapsulates the data and routes the data to it's final destination. A GRE tunnel therefore allows two switches to forward packets to one another using routes established in a route table. www.juniper.net (2012) provides the following description of how GRE works,

How GRE tunneling works

1. Switch acting as tunnel source receives a data packet to be tunneled and sends it to the tunnel interface.

2. The interface encapsulated the data in a GRE packet and adds a destination IP header
3. Forwards the data to the destination IP address
4. The switch acting as a tunnel remote router receives the packet from the tunnel and de-encapsulates the data.
5. The packet is then routed to it's final destination.

How GRE tunneling will be used in this project

As one of our Objectives is to connect two separate Mininet networks, located on different machines, it will be necessary to make use of GRE tunneling to allow two separate Mininet networks to exchange data. This will allow data exchange between virtual hosts on the separate Mininet networks, and enable us to test Mininet's performance across a hardware interface.

2.6 Network Simulators and Emulators

As Network Testing plays a very important role in SDN and Software Development, there are many Network Simulators and Emulators available. Before a conclusion can be made as to which emulators are accurate, a definition of a "good" emulator is needed. As emulators are used to test performance, functionality and stability of any network under real world conditions, a good emulator should have the following characteristics;

1. Easy to use,
2. Easy to install,
3. Has available documentation and support,
4. Support the emulation of large networks,
5. Allow easy topology testing ,
6. Allow the customization of network conditions,
7. Allow the creation of custom topologies, and
8. Must be able to connect to a real network

In order to ensure that the best emulation Platform is used, the following Network Emulation and Simulation platforms were considered.

1. **Cloonix**

   According to Linkletter (2016), the Cloonix network simulator provides an easy to use graphical user interface (GUI) and uses QEMU/KVM to create virtual machines. The simulator has a wide variety of pre-built file systems that can be used as Virtual machines (VM's).

Cloonix Limitations

There is no documentation for this network simulator to aid a user in the installation or use of the simulator. Therefore this network simulator was not a suitable choice for this project.

2. **Marionnet**

According to Linkletter (2016), Marionnet allows users to define, configure and run complex computer networks on a VM without any need of a physical setup. Marionnet is designed to be used for educational purposes, as it has a easy to use GUI, and samples of practice configurations available. From Marionnet.org (2007) we obtain the following information

Marionnet Provides

- Dynamic reconfiguration of the network
- Can run on virtual machines
- Can run graphical applications
- Easy-to-use GUI

Marionnet Limitations

Marionnet cannot emulate large networks (15 Hosts) and there are some concerns regarding disk usage. Marionnet was therefore not a suitable choice for this project.

3. **GNS3**

According to Linkletter (2016), GNS3 is a graphical network simulator, focused on supporting Cisco and Juniper software. This simulator is mainly used by individuals studying for Cisco exams, as GNS3 provides a variety of prepared open-source virtual applications, and allows a user to create their own custom networks.
From GNS3.com (2015) we obtain the following information

GNS3 Provides

- Comprehensive documentation
- Real-time network simulation
- Connection to any real network
- Customized topologies

GNS3 Limitations

According to packthub.com (2016), if one end of the GNS3 network is shut down, the other end is also down - this is a problem when it comes to protocol-timeouts. This is very limiting, as a user cannot shut down one interface without shutting down the entire network. Creating a cloud connection from a router directly to a host's Ethernet interface does not guarantee communication between the router and host, even if IP addressing is correct.
Even-though GNS3 is a good network simulator, these mentioned limitations make the testing of GNS3 Networks difficult, and therefore not a suitable choice for this project.

4. **CORE**

According to Linkletter (2016), The Common Open Research Emulator provides an easy-to-use GUI interface. The emulator uses Network Namespaces functionality in Linux Containers (LXC) as a virtualization technology.
From www.nrl.navy.mil (2008) we obtain the following information

CORE Provides

- Easy-to-use GUI
- Centralized configuration and control
- Real-time connection to real networks
- Highly customizable network emulation

CORE Limitations

CORE has no real limitations that hinder topology testing, and is therefore a good emulator to use in this project.

5. **Mininet**

According to Linkletter (2016), Mininet is a network emulator, that usually runs on a Linux machine, that allows a user to create a network of virtual hosts, switches, controllers and links. According to Mininet (2012), Mininet hosts run standard Linux network software, and it's switches support OpenFlow for flexible routing. Therefore, all Mininet test code can move to a real device, with minimal changes. Mininet uses process-based virtualization to run hosts and switches on a single OS kernel. Mininet uses Linux network namespaces to create virtual nodes and connects hosts and switches using virtual Ethernet (veth) pairs. This tool can support hundreds/thousands of nodes on a single machine. Through Python script, users can build custom network topologies, to test system behavior and experiment with different networks and topologies. the following information is provided by (Mininet, 2012),

Mininet Provides

- Good Documentation
- Network Testbed for developing OpenFlow applications
- Allows multiple developers to work independently on the same topology
- Regression tests
- Complex Topology testing without connecting any real hardware components
- Custom topologies, through Python API.
- Real world testing, by connecting real hosts and switches to the network.

Mininet Limitations

According to Mininet (2012), Mininet networks cannot exceed the CPU or Bandwidth available on the host PC, and in this experiment it will be limited to VirtualBox's available bandwidth and Processing power. Mininet only runs on Linux machines and cannot run non-linux-compatible OpenFlow switches or applications, it was therefore necessary to use a virtual Linux machine, on VirtualBox. According to github.com (2012) If a user needs custom routing or switches, they will have to develop a controller with all features they acquire on their own, which adds a lot of complexity. Mininet is isolated from the Local network, but can connect to the local network through a Network Address Translation (NAT) object. Mininet can not emulate very high speed networks, it is limited to about 100 Gbps. These Limitations do not inhibit customization of networks and testing of networks, and is therefore a suitable choice for this project.

2.7 Emulator of choice

After considering all of the above Emulators/Simulators, **Mininet** was selected as the emulator of choice as it is easy to install and use, allows the user to create custom topologies through python script and enables complex real world testing. Even though CORE was also a very good emulator option, the scope of this project is to test Mininet as a network emulator

Why it's Better

- Provides good documentation
- Has an Active community
- Easy to install and use
- Can support hundreds/Thousands of nodes
- Can connect to a real network

As there is no known GUI for Mininet, this project contributed, by providing an easy-to-us GUI for Mininet.

2.8 Summary

In this chapter, the basic concepts of networking are explained to help the reader understand what is meant by a 'hosts', 'switches', 'bandwidth', 'delay', 'packet loss' and jitter. Concepts like NAT and GRE tunneling are also briefly explained, along with a short discussion on how each can be used in this project. Most importantly, a list of considered network emulators and simulators is given, along with a short discussion of each emulator/simulator's strengths and limitations. In the last section of this chapter it is explained why Mininet is the best network emulator for this project.

Chapter 3

Graphical User Interface (GUI)

This chapter gives an overview of the developed Graphical User Interface. The chapter consists of two sections, the system's *Front-end*, and it's *Back-end*. In the Front-end section the GUI is explained in terms of use and functionality - this section could double as a user-manual. The Back-end section concentrates on how the GUI works, what classes are used and how a Mininet Network is created.

3.1 System Overview

Figure 3.1.1 shows a flow diagram of the system overview.

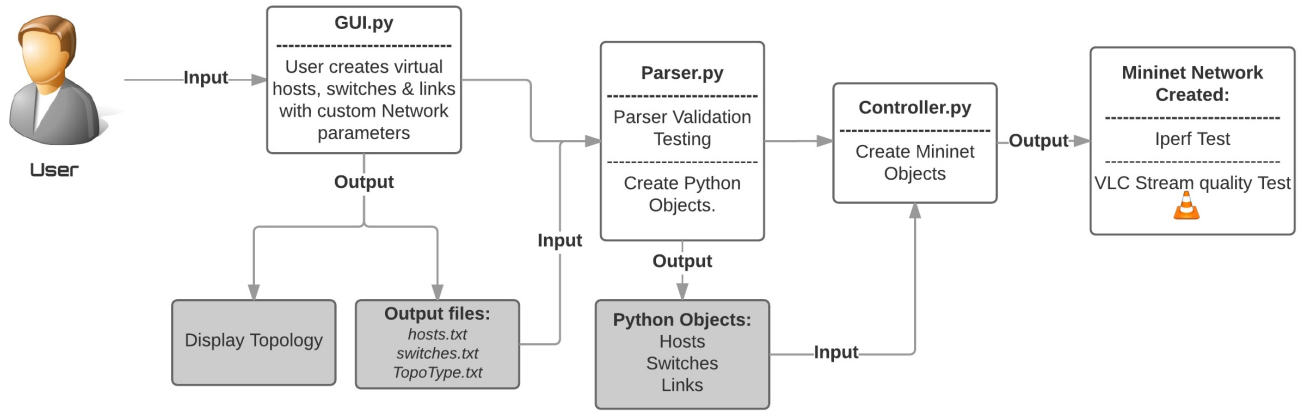


Figure 3.1.1: System Overview

3.2 Front-end

The GUI was designed using tkinter, as tkinter has very good documentation, which will help with the development of the GUI. Using Python's GUI toolkit also simplified front-end to back-end integration.
To ensure that the user gets the most out of the Mininet Network Emulator, it is important to allow the user to specify the network Topology and set various network conditions, like bandwidth, delay, packet loss and jitter. When the program is started the first window to appear, is the *Working Directory* Window, shown in Figure 3.2.1. This window prompts the user to enter a save location for the topology they are about to create. As seen in figure 3.2.2, the main GUI window consists of three widget frames; the **Hosts Widget**, **Switches Widget** and the **Links Widget**. Each of these widgets allow the user to add hosts and switches and to customize the link parameters. Figure 3.2.3 shows the inner workings of the GUI.

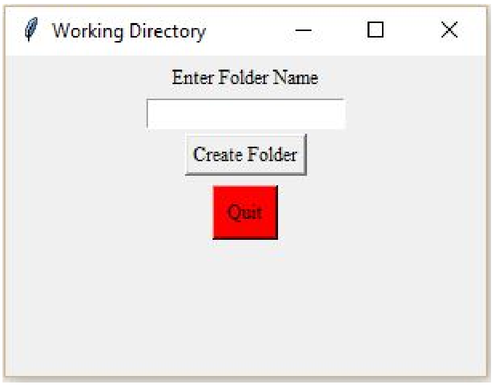


Figure 3.2.1: Initial Window on starting the Mininet Network Emulator

Mininet Network Emulator GUI

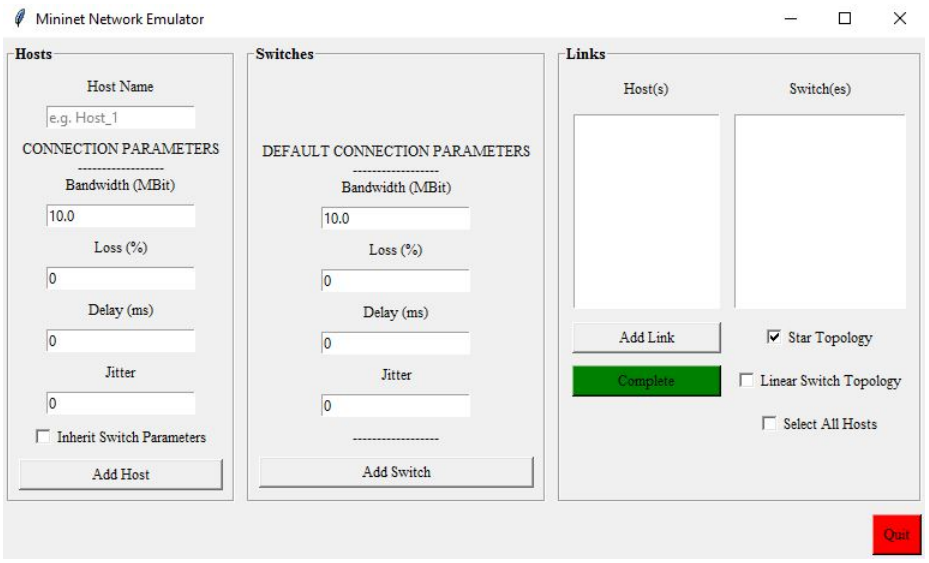


Figure 3.2.2: Mininet Network Emulator GUI Frame

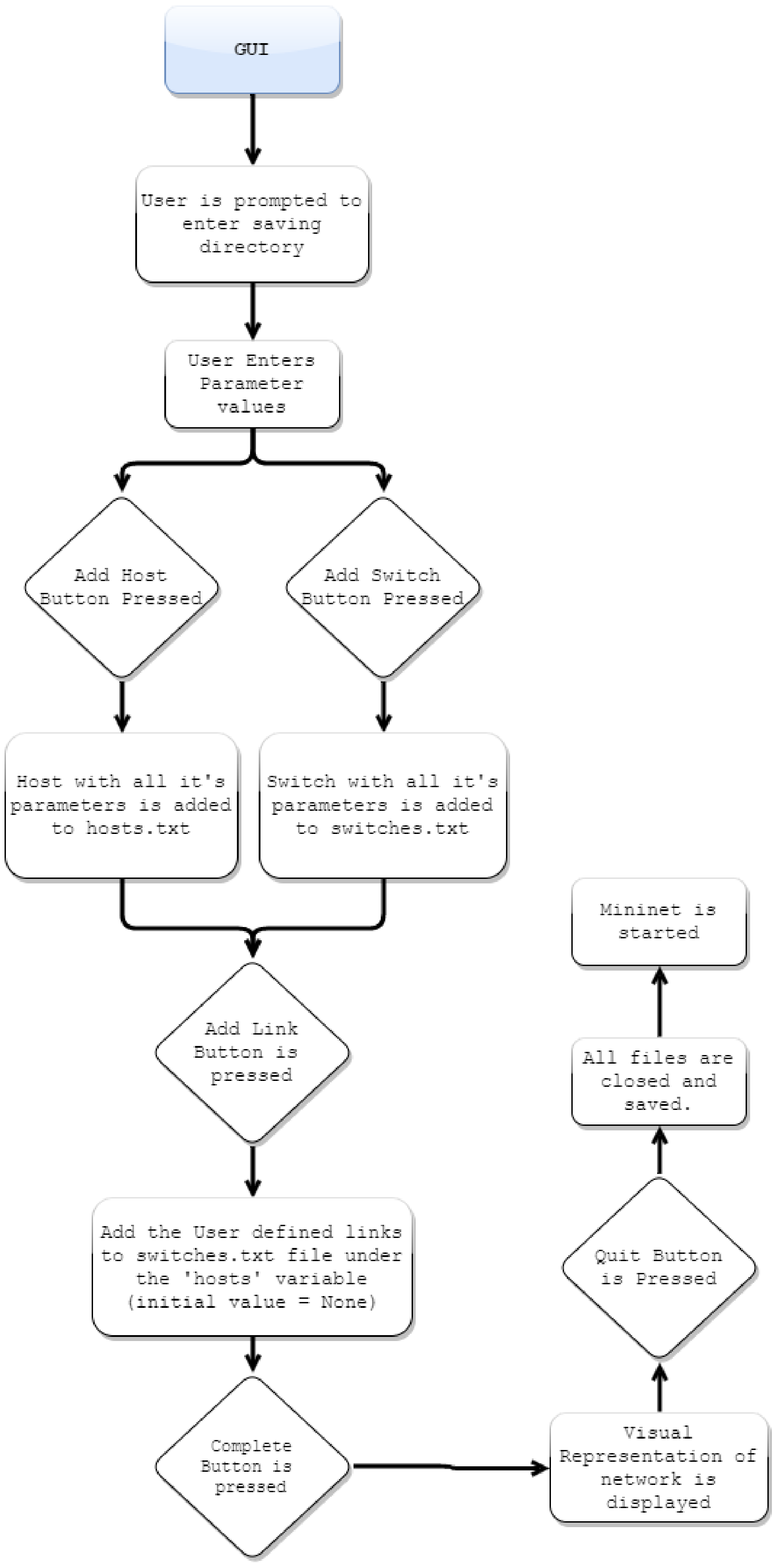

GUI
User is prompted to enter saving directory
User Enters Parameter values
Add Host Button Pressed
Add Switch Button Pressed
Host with all it's parameters is added to hosts.txt
Switch with all it's parameters is added to switches.txt
Add Link Button is pressed
Add the User defined links to switches.txt file under the 'hosts' variable (initial value = None)
Complete Button is pressed
Visual Representation of network is displayed
Quit Button is Pressed
All files are closed and saved.
Mininet is started

Figure 3.2.3: GUI flow chart - showing all steps the during the creation of a topology

3.2.1 Hosts Widget

The Host widget, as seen in Figure 3.2.4, allows the user to add hosts to the Mininet network. The user can customize the Host name by entering a name into the first text-field called Host Name. In the case that the user leaves this text-field blank, a default name *'h1', 'h2', 'h3', etc* will be used. The user can also add connection parameters/network effects for each individual Host, namely bandwidth, delay, Packet loss and jitter, to simulate a real life connection. Value testing prevents the user from entering invalid values into text-fields, like text and other ascii values, to ensure only valid entries are saved. The default values for Connection parameters are: bandwidth = 10 Mbps, Loss = 0 percent, delay 0 ms, jitter = 0 ms. The radio button **Inherit Switch parameters**, instantly changes the Host link parameters to inherit the switch's connection parameters when it is selected. When the **Add Host Button** is pressed, the host name and all its connection parameters are written into a text file *'hosts.txt'*, as seen in Figure 3.2.5. This file is then later used to create Mininet host objects - this will be discussed later in this chapter.

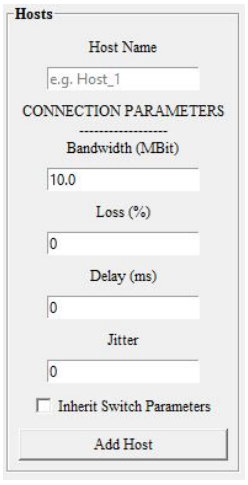
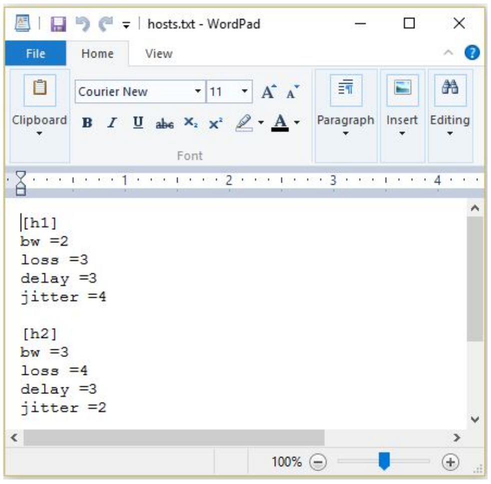


Figure 3.2.4: Hosts widget, which allows the user to add Hosts to the network, and customize it's connection parameters



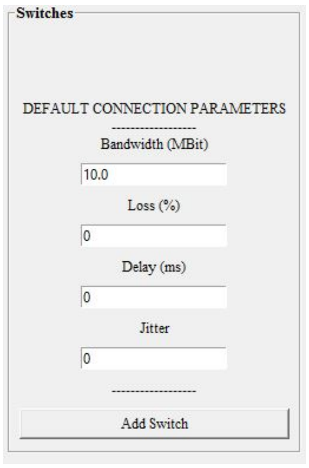
```
[h1]
bw =2
loss =3
delay =3
jitter =4

[h2]
bw =3
loss =4
delay =3
jitter =2
```

Figure 3.2.5: Example of created Hosts text file

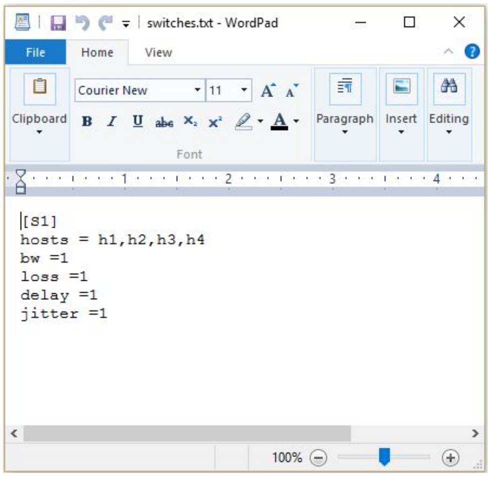
3.2.2 Switches Widget

The Switches widget, as seen in Figure 3.2.6, allows the user to add switches to the Mininet network. A default name *'S1', 'S2', S3', etc* is automatically assigned to each switch. The user can also add connection parameters/network effects for each individual switch, namely bandwidth, delay, Packet loss and jitter. These parameters are similar to the Host's connection parameters, but are only used if the user wants to tie link parameters with a switch rather than a host. In the case that the Host connecting to the switch's parameters are zero, a link will inherit the switch's connection parameters. Value testing prevents the user from entering invalid values into text-fields, like text and other ascii values, to ensure only valid entries are saved. The default values for Connection parameters are: bandwidth = 10 Mbps, Loss = 0 percent, delay 0 ms, jitter = 0 ms. When the **Add Switch Button** is pressed, the switch name and all its connection parameters are written into a text file *'switches.txt'*, as seen in Figure 3.2.7. This file is then later used to create Mininet switch and link objects - this will be discussed later in this document.



Bandwidth (MBit): 10.0
Loss (%): 0
Delay (ms): 0
Jitter: 0

Figure 3.2.6: Switches widget, which allows the user to add Switches to the network, and customize the default connection parameters



```
[S1]
hosts = h1,h2,h3,h4
bw =1
loss =1
delay =1
jitter =1
```

Figure 3.2.7: Example of created Switches text file

3.2.3 Links Widget

The Links widget allows the user to link network entities, namely x amount of hosts to y amount of switches. As shown in figure 3.2.8, two list-boxes are used, the Host-listbox and the Switches-listbox. The user selects one or more hosts from the Host-listbox and then selects the switch they want to connect the hosts to, from the Switch-listbox. After a user has decided which links they want to make, the user can click the **Add Link Button** which will then save the Hosts connected to each specific switch and into the text-file, *'switches.txt'*, as seen in Figure 3.2.7. The 'hosts' variable in the *'switches.txt'* text-file, indicates which entities are connected to one another. The two radio buttons **Star Topology** and **Linear Switch Topology** allow the user to either create a Star Topology or a Linear Switch Topology.
A Star topology, is a network that consists of a single switch and multiple hosts. A linear topology is a network that consists out of multiple hosts and switches, where all switches are linked to one another.
Once a radio button is selected, a unique topology identifier (Star Topology - 'Single' / Linear Topology - 'Linear') is written into a text file *'TopoType.txt'*. In the case of a Star Topology only one switch is allowed, thus all existing hosts are connected to the Star. For the Linear Topology, multiple switches and hosts are allowed and additional links between switches are automatically created. The **Select All Hosts** radio button, activates (selects) all of the hosts in the host-listbox when selected.

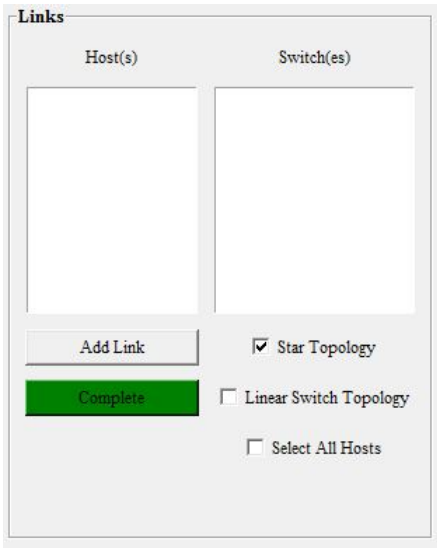


Figure 3.2.8: Links widget

Error messages will occur in the following situations: (See Appendix C.1 for details)

- No hosts or switches added and the Add Link button is pressed
- No hosts or switches added and the Complete button is pressed
- Hosts and switches are available, but no links were added and the Complete button pressed
- Hosts and switches are available, but no links were added and the Add Link button pressed

- Hosts are added, but no switches found in the network and the Add Link Button pressed
- Single Topology Radio Button active, user tries to add more than one switch
- Host without a link to a switch in the topology and the Complete Button pressed
- User Tries to link a device which is already linked or tries to make the same connection twice
- User Tries to add a Host, with a Host Name that already exists

3.2.4 Visuals Widget

Once all network entities were added, the necessary links were created, and the **Complete Button** is pressed the Visuals widget is created, which provides the user with a simple canvas representation of their topology. Figure 3.2.9 (Example 1) shows the visuals created for a basic Star Topology example and Figure 3.2.10 (Example 2), shows the visuals created for a basic a Linear Topology. A blue squares represents a hosts, a turquoise circle represents a switch and a red circle represents the controller. Lines drawn between these shapes, represents a link between the entities.

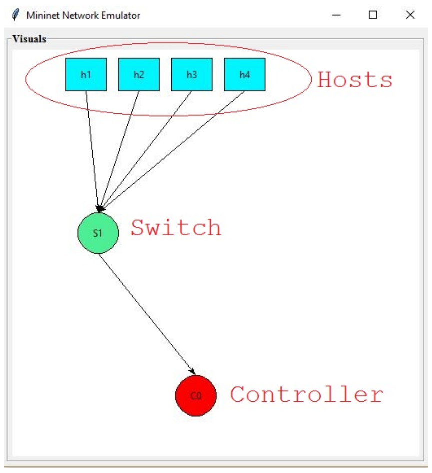


Figure 3.2.9: Example 1

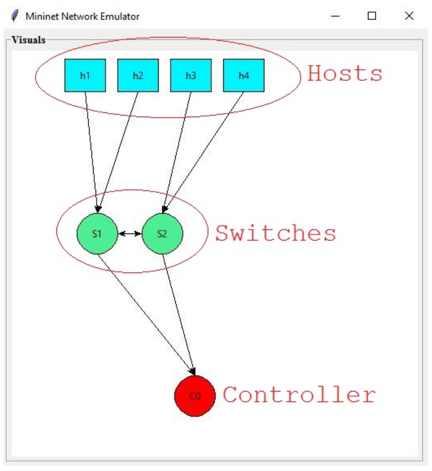


Figure 3.2.10: Example 2

3.3 Back-end

3.3.1 Parser.py

In order to create mininet objects, data needs to be read from the text-files created by the GUI. To achieve this, the Parser.py class was developed. This class uses python's configparser, to read the text-files section by section, to create python Objects of Hosts, Switches and Links. The Host's, Switch's and Link's classes all have built in functions that will allow easier access of each individual network entity's, network parameters.
Figure 3.3.1 shows a flow diagram of the parser.py class

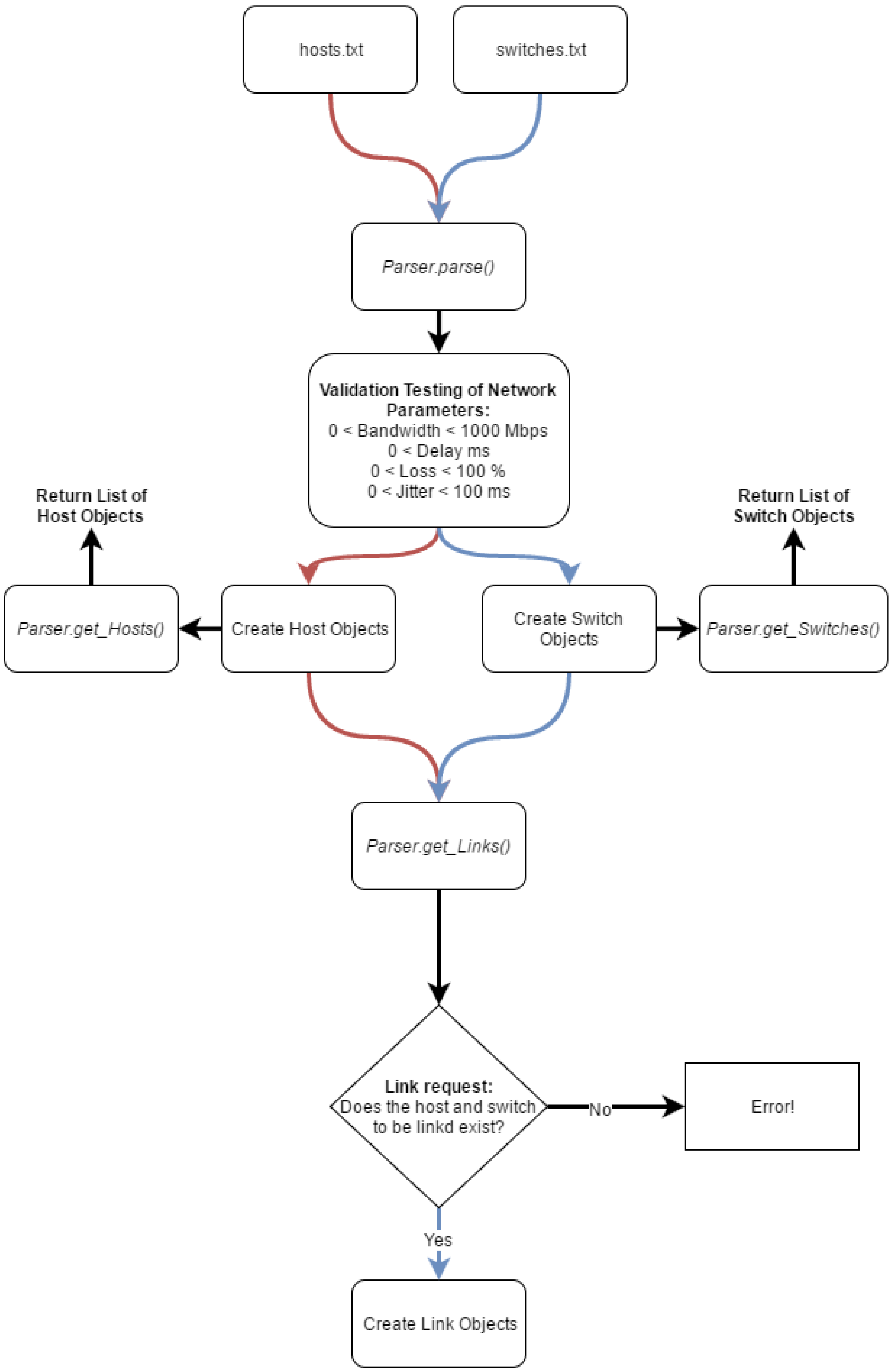


Figure 3.3.1: Parser.py Flow chart

3.3.2 Creating Host Objects - Host.py

The Host class is used in both the Controller and GUI classes to create host objects. Each host object has the following parameters as local variables: bandwidth, loss, delay, jitter and host name. These parameters are used, when linking a specific host to a switch, to define the network effects of the link. Get functions are used to return these parameters. On initiation all parameter values are passed from the Parser.parse function. Host init function:

Host(self, hostname, bandwidth, loss, delay, jitter, hosts)

3.3.3 Creating Switch Objects - Switch.py

The Switch class is used in both the Controller and GUI classes to create switch objects. Each switch object has the following parameters as local variables: bandwidth, loss, delay, jitter, switch name and hosts. If the switches network parameters are left blank, the link will automatically use the host object's network parameters. The switch's link parameters are only used as default network parameters. Get functions are used to return network parameters, the switch's ID and all the hosts that are linked to the switch. On initiation, a switch ID is assigned and all parameter values are passed from the Parser.parse function. Switch init function:

Switch(self, bandwidth, loss, delay, jitter, hosts)

3.3.4 Creating Link Objects - Link.py

The Link class is used in the Controller classes to create link objects. Each link object has the following parameters as local variables: bandwidth, loss, delay, jitter, host and switch. These parameters are used to add a link between the object's host and switch variable with it's network parameter values passed through on initiation. Link init function:

Link(self, swith, host, bandwidth, loss, delay, jitter)

More information on the Parser.py, Host.py, Switch.py and Link.py class functions are available in Table C.2.1 - C.5.1, Appendix C.

3.3.5 Creating the Mininet Network - Controller.py

The Controller class is responsible for creating Mininet objects through standard Mininet functions. The Parser class is used to create Host, Switch and Link python objects. These objects are then used to create the exact user defined topology, with all it's specified network parameters.
Mininet supports parameterized topologies, thus custom Mininet topologies can be built within the Python API. The base class for Mininet topologies is the *Topo* class. The Topo class enables the use of the addSwitch, addHost and addLink functions which creates mininet switches, hosts and links. Important functions include:

addSwitch(): adds a switch to the topology, returns switch name
addHost(): adds a host to the topology, returns host name

addLink(): adds a bidirectional link to the topology, returns name of linked devices and link parameters.

TC links are used as they allow for performance parameters like link bandwidth, delay, packet loss and jitter.

3.4 Summary

This chapter gives and overview of the GUI. From the information provided in this chapter, it is evident that the GUI enables a user to build and view custom parameterized networks, easily and efficiently. The GUI makes use of validation testing to ensure a user does not enter incompatible values for entry fields. As an intermediate step, the Parser.py class makes use of object orientation, to create Host, Switch and Link objects. These objects allow the Controller.py class to easily create the custom Mininet network.

Chapter 4

Software Interface Experiment

This chapter focuses on Mininet's performance across a software interface. Tests were conducted to help test Mininet's capability of emulating paramaterized networks over a software interface, by testing Mininet's bandwidth-, delay-, packet loss- and jitter consistency with a varying amounts of hosts. These tests would also prove whether different Mininet networks can communicate over a software interface. The data captured from these tests helped in plotting performance graphs. In section 1, an in depth overview of the test setup is discussed - this setup was used for all tests. Section 2 provides us with a motivation, to why it is necessary to test Mininet's performance over a software interface. In sections 3 the different test networks are introduced. In sections 4 to 6 the different network effects are tested for their consistency. Each test is discussed in terms of; motivation to perform the test, setup, results and why or why not measured values differ from theoretical values. Section 7 provides us with a further qualitative test, by testing video streaming over the emulated network, and how certain parameters affect the video quality.

4.1 Setup Overview

As Mininet only runs on Linux machines, VirtualBox was used to create a Linux Virtual Machine on a Windows computer. Using a VM made the project more modular, by allowing use of the VM on other computers. Mininet was installed on the VM, which allowed us to create custom parameterized topologies and run various network performance tests. Figure 4.1.1 represents the test environment for this experiment.

Unfortunately, as a result of working on a VM, Mininet's performance was throttled due to resource limitations. This could possibly result in Mininet's performance being lower than expected, and should be taken into account in the discussion of each test. Table 4.1.1 shows the Virtual Machine's available resources.

Resource	Available
Base Memory	5082 MB
Processors	2
Video Memory	128 MB

Table 4.1.1: VM available resources

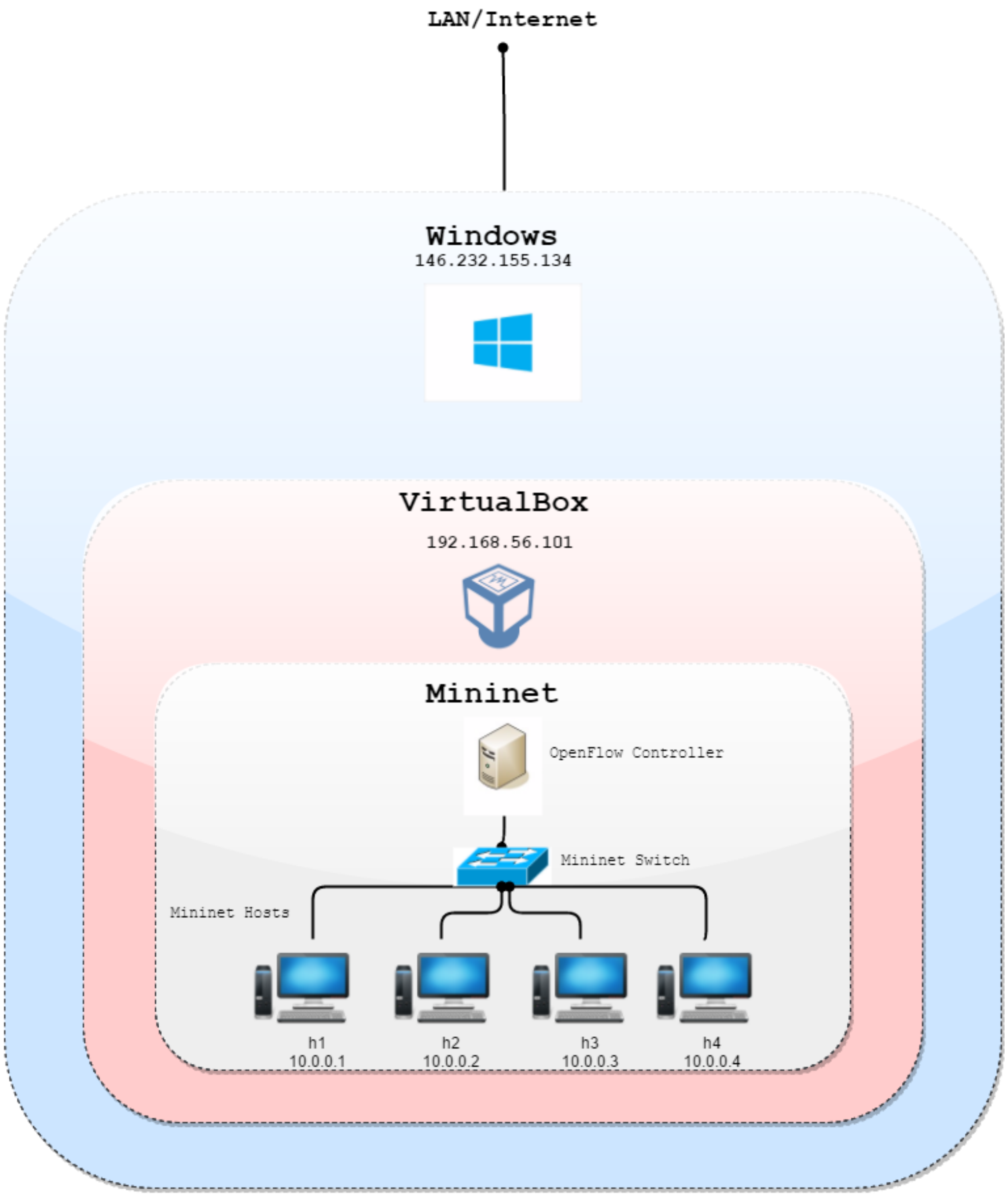


Figure 4.1.1: Software Interface setup

4.2 Motivation

To prove that Mininet is an efficient and scalable network emulator, the following had to be tested:

1. **Bandwidth consistency**
2. **Delay consistency**
3. **Loss consistency**
4. **Jitter consistency**

Results from these tests prove mininet's ability to emulate various parameterized network networks. Results also show whether network conditions set by the user, is in fact the network conditions that are present in the network. These tests will also highlight any limitations Mininet has in terms of parameterized network emulation.

4.3 Test Networks

Six networks, consisting of Mininet's standard hosts and switches, were used for each test case. Figures 4.3.1 to 4.3.6 shows the GUI's output for each network. Star-topology networks were implemented, as this is the most commonly used network topology. Furthermore, it was decided to only test network size up to 50 hosts, as a network with more than 50 hosts will most likely be simulated inaccurately, due to resource limitations caused by the virtual machine.

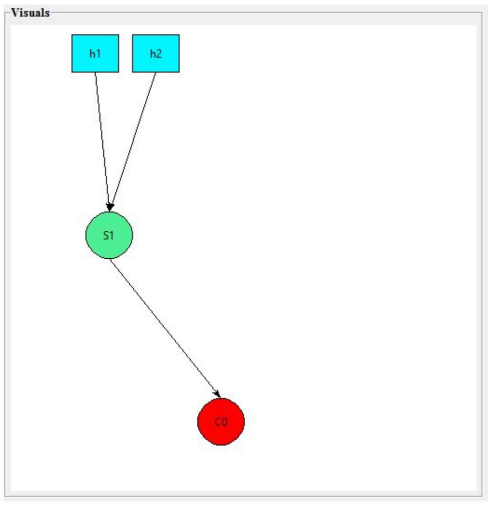


Figure 4.3.1: Test Network 1: 2 Hosts

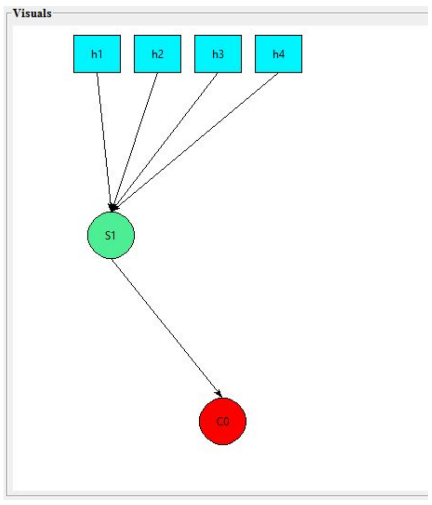


Figure 4.3.2: Test Network 2: 4 Hosts

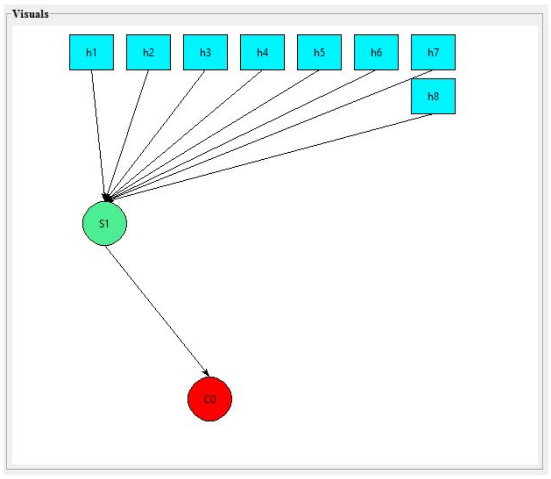


Figure 4.3.3: Test Network 3: 8 Hosts

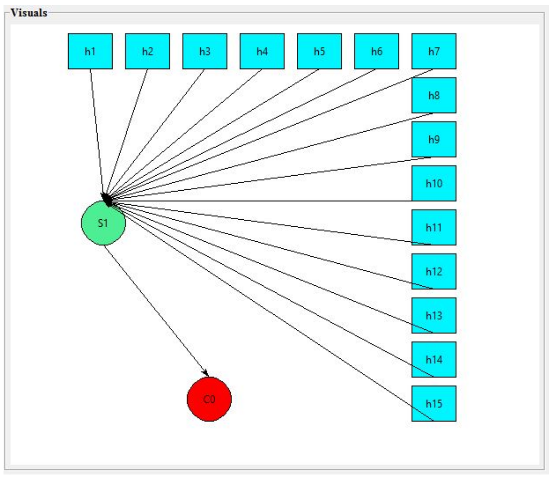


Figure 4.3.4: Test Network 4: 15 Hosts

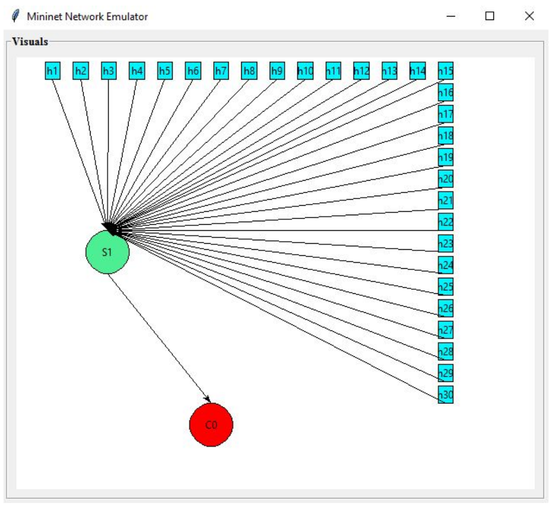

Mininet Network Emulator
Visuals
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30
S1
C0

Figure 4.3.5: Test Network 5: 30 Hosts

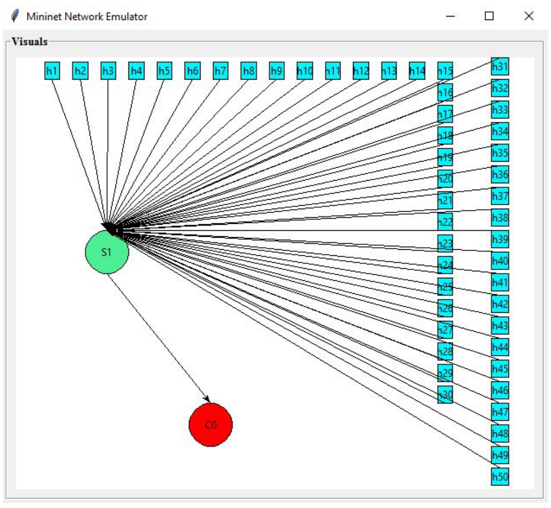


Figure 4.3.6: Test Network 6: 50 Hosts

4.4 Bandwidth Test

4.4.1 Motivation

When setting up a network, it is very important to know how consistent the bandwidth of the network is. The bandwidth of the network determines how efficient hosts on the network can communicate with one another. Bandwidth inconsistency can cause quality dips in the network - this is especially bad for networks designed for video streaming and online gaming. It is therefore very important to know how consistent the bandwidth of Mininet networks are.
The objective of this test is to see how consistent, measured- vs user-defined, bandwidth values are, as the size of the network increases.

4.4.2 Test Setup

To test the bandwidth consistency of the emulator Iperf Transmission Control Protocol (TCP) data stream tests were used to measure the bandwidth of each network topology recursively, increasing the link bandwidth after each run. To ensure that no other network effects, affect the network, a delay of 0 ms, packet loss of 0% and jitter of 0 ms was used. Table 4.4.1 shows the bandwidths that were tested:

Run no.	Bandwidth
1	1 Mbps
2	2 Mbps
3	4 Mbps
4	8 Mbps
5	12 Mbps
6	20 Mbps

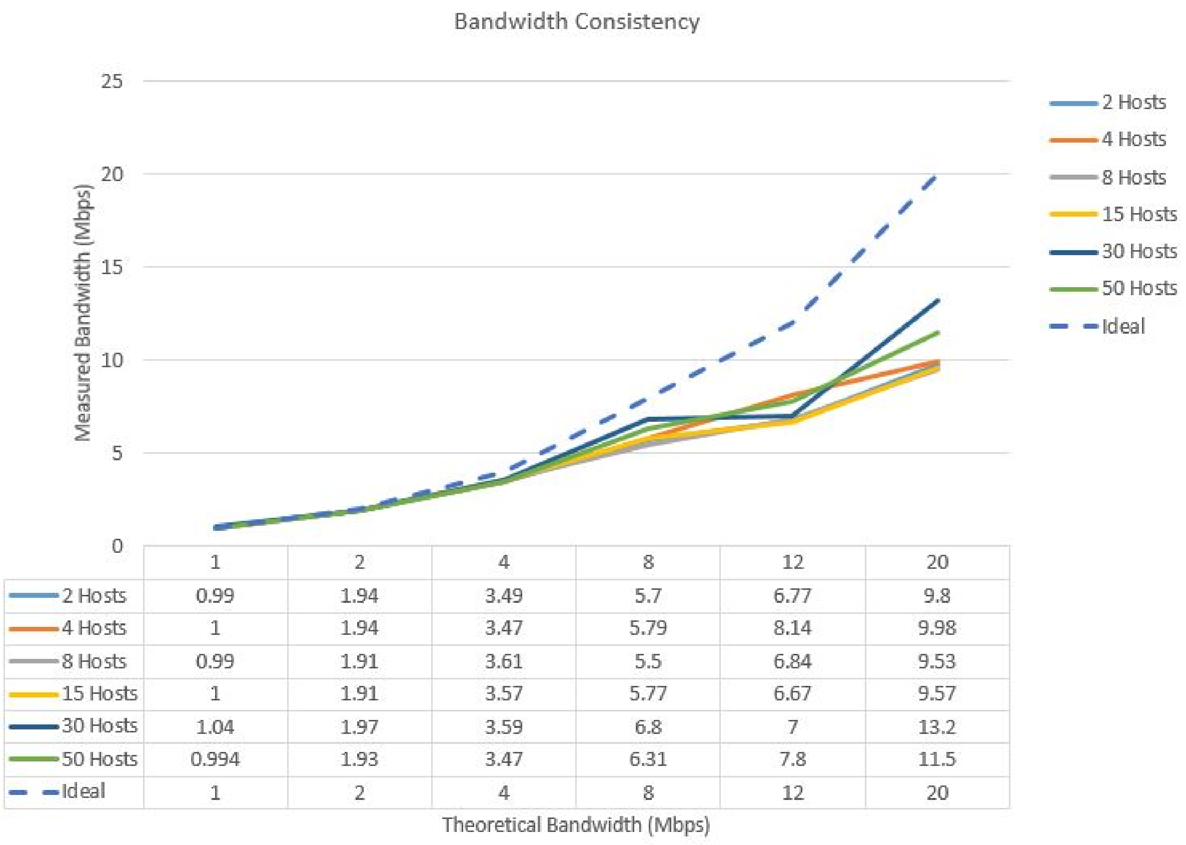
Table 4.4.1: Bandwidth Test

Number of Hosts	Packet Loss	Delay	Jitter
2	0 %	0.171 ms	1.675 ms
4	0 %	0.170 ms	2.01 ms
8	0 %	0.266 ms	1.83 ms
15	0 %	0.21 ms	1.89 ms
Average	0 %	0.204 ms	1.85 ms

Table 4.4.2: Average of measured values - ideal network conditions

To ensure that the values of delay, packet loss and jitter were in fact zero, Test Networks 1 to 6 (Figure 4.3.1 - 4.3.6) were used to run preliminary Iperf tests, to measure the actual delay, packet loss and jitter values. Table 4.4.2 show the measured values for each of the parameters. Graphs of the measured results can also be found in appendix D, Figures D.2.1 - D.2.3. It is clear that the measured values of delay and jitter are not zero - and should be considered in further discussions. These non-zero conditions were assumed for all tests.

4.4.3 Results



	1	2	4	8	12	20
2 Hosts	0.99	1.94	3.49	5.7	6.77	9.8
4 Hosts	1	1.94	3.47	5.79	8.14	9.98
8 Hosts	0.99	1.91	3.61	5.5	6.84	9.53
15 Hosts	1	1.91	3.57	5.77	6.67	9.57
30 Hosts	1.04	1.97	3.59	6.8	7	13.2
50 Hosts	0.994	1.93	3.47	6.31	7.8	11.5
Ideal	1	2	4	8	12	20

Figure 4.4.1: Bandwidth Consistency results

4.4.4 Discussion

From Figure 4.4.1, we can see test results for Mininet's bandwidth consistency. From this figure it is clear that Mininet can emulate a small bandwidth accurately, regardless of the amount of hosts within the network. For larger bandwidths, (>5 Mbps), it was evident that Mininet could not produce the desired bandwidth for each host in the network. A graphical representation of each bandwidth test was also generated using Xjperf (Iperf's graphical user interface for Linux), and can be found in appendix D, Figures D.1.1 - D.1.24. The fact that Mininet cannot emulate large bandwidths, proves one of Mininet's limitations - Mininet cannot exceed the bandwidth available on the host PC. In this case Mininet is limited by the Virtual machine, which itself does not have a lot of resources. It is therefore not surprising that Mininet cannot supply all it's hosts with a high bandwidth.

4.5 Delay Test

4.5.1 Motivation

Network delay is a measure of the amount of time it takes a packet to reach it's destination. In a network, delay can cause connections to become slow and in extreme cases, almost unusable. It is therefore very important to know how consistent the delay of Mininet networks are.
The objective of this test is to see how consistent, measured- vs user-defined, delay values are, as the size of the network increases.

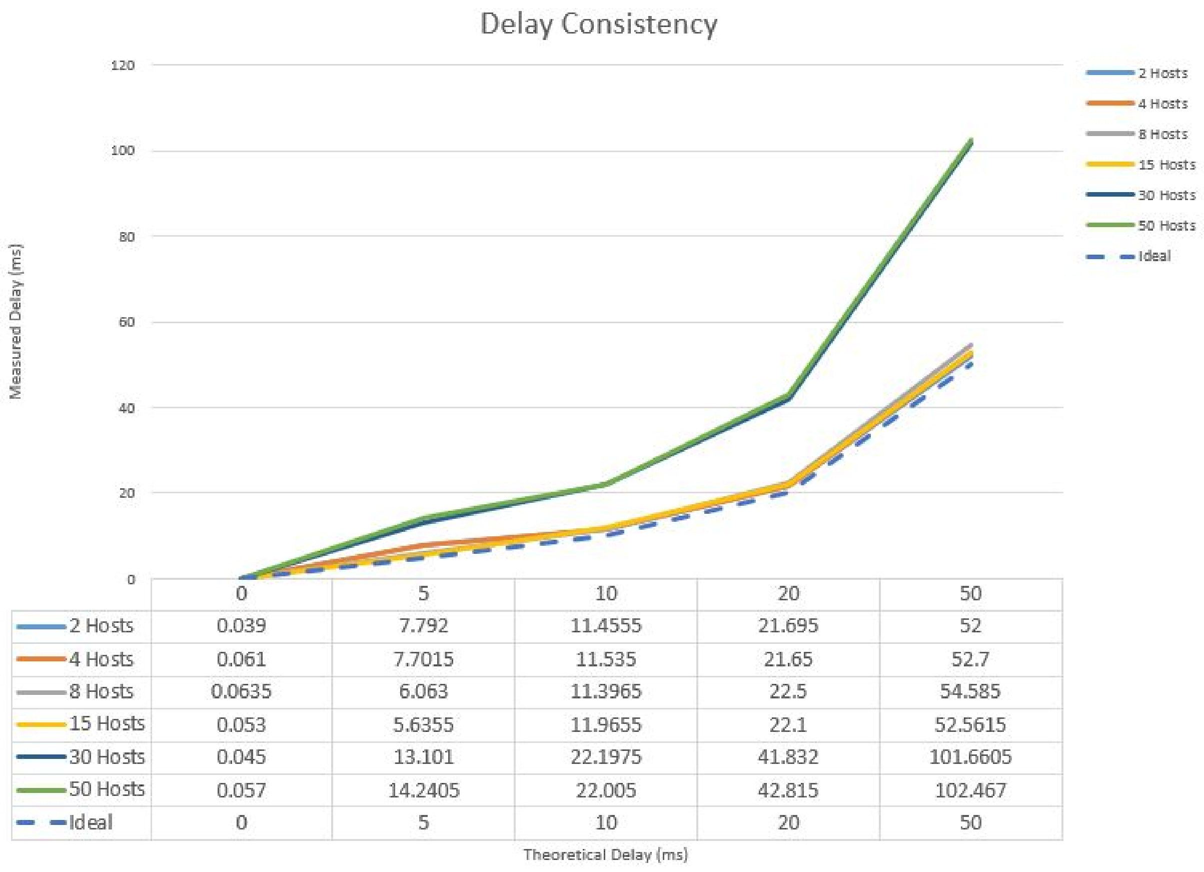
4.5.2 Test Setup

To test the delay of each network topology, a ping function was used, increasing the delay after each run. To ensure that no other network effects, affect the delay, a bandwidth of 2 Mbps, packet loss of 0% and jitter of 0 ms was used (non-ideal zero values were assumed). A ping test was performed between two hosts in the network, sending out a ping every second for 50 seconds. It is important to know that the measured delay is the travel time to-and-from the destination, thus the actual delay time to one host is the measured value, divided by 2. The average of the 50 pings was accepted as the delay for each network. For each of the test networks, the following delay times were tested:

Run no.	Delay
1	0 ms
2	5 ms
3	10 ms
4	20 ms
5	50 ms

Table 4.5.1: Delay Test

4.5.3 Results



	0	5	10	20	50
2 Hosts	0.039	7.792	11.4555	21.695	52
4 Hosts	0.061	7.7015	11.535	21.65	52.7
8 Hosts	0.0635	6.063	11.3965	22.5	54.585
15 Hosts	0.053	5.6355	11.9655	22.1	52.5615
30 Hosts	0.045	13.101	22.1975	41.832	101.6605
50 Hosts	0.057	14.2405	22.005	42.815	102.467
Ideal	0	5	10	20	50

Figure 4.5.1: Delay Consistency results

4.5.4 Discussion

From Figure 4.5.1, we can see tests results for Mininet's delay consistency. It is clear that Mininet can simulate delay very accurately for smaller networks ($<$ 20 Hosts), despite a very small offset delay of about 2 ms in each measurement. This offset delay is most likely caused by the resource limitations on the VM and the non-zero values for delay and jitter. For larger networks ($>$ 30 Hosts), Mininet could not accurately emulate the network delay. Measured delay values proved to be almost double the theoretical value. As Mininet failed to provide all it's hosts with the necessary resources, the desired network conditions couldn't be emulated.
We can therefore conclude that smaller Mininet networks can emulate delay very accurately, regardless of resource limitations and small offset values. However, larger networks are greatly affected by resource limitations and will most likely cause the actual network delay to differ from it's desired value.

4.6 Packet Loss Test

4.6.1 Motivation

Packet loss occurs when packets fail to reach their destination. Packet loss in a network is very undesirable, as it can cause significant loss of information. If a network is designed to have any packet loss, it is very important that the emulated packet loss value does not change, as it would cause the network to be unreliable. It is therefore very important to know how consistent the packet loss of Mininet networks are.
The objective of this test is to see how consistent, measured- vs user-defined, packet loss values are, as the size of the network increases.

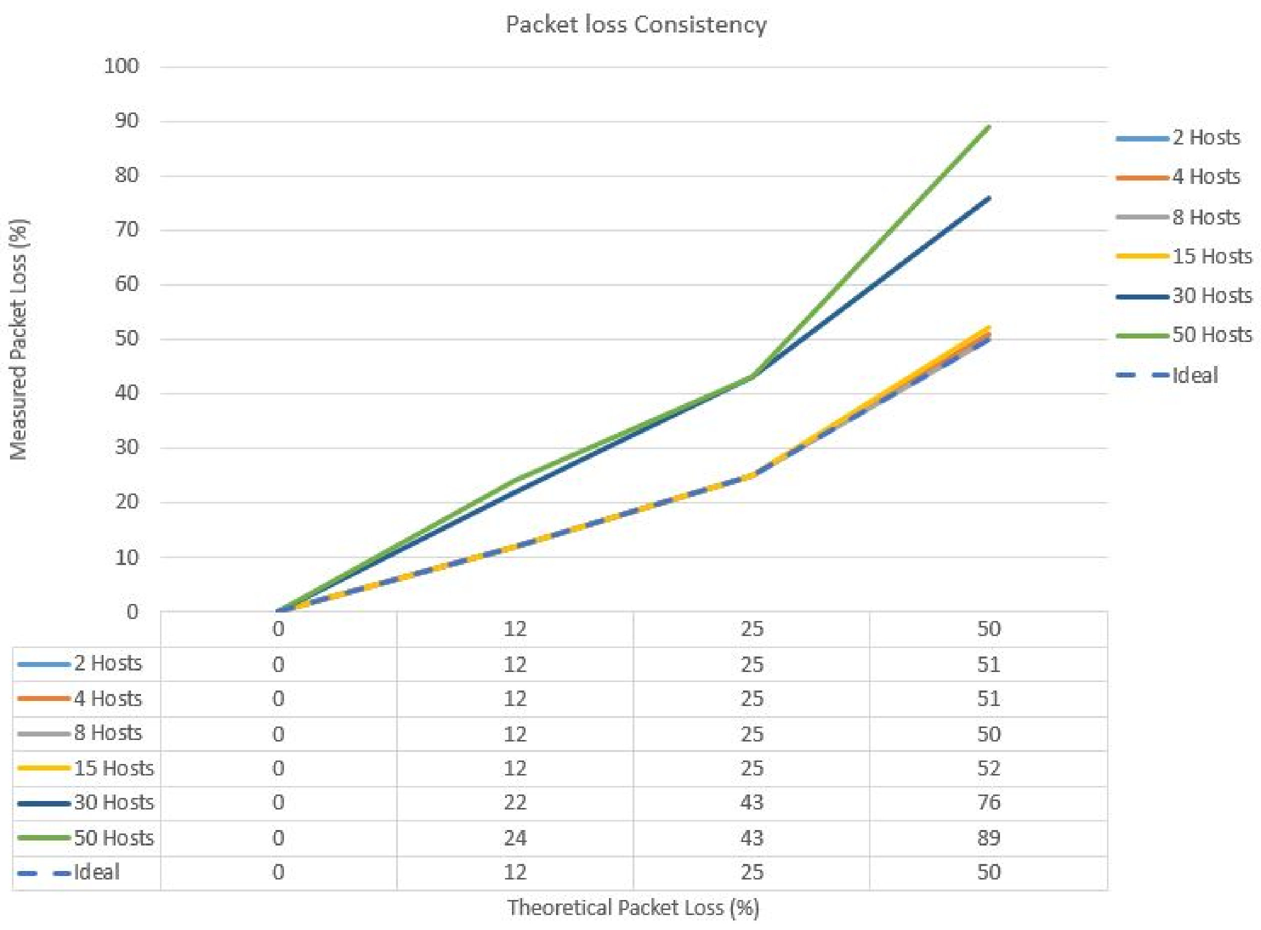
4.6.2 Test Setup

To test the packet loss for each network, Iperf User Datagram Protocol (UDP) data stream tests were used, one host as the Iperf server and another as a client. The Packet Loss of the emulated network was tested recursively, increasing the packet loss after each run. To ensure that no other network effects, affect the packet loss, a bandwidth of 2 Mbps, delay of 0 ms and jitter of 0 ms was used (non-ideal zero values were assumed). Packet loss up to 50% was tested, as any packet loss higher than 50% would cause Iperf tests to fail.
Table 4.6.1 shows the Packet loss percentages that were tested:

Run no.	Loss
1	0 %
2	12 %
3	25 %
4	50 %

Table 4.6.1: Packet Loss Test

4.6.3 Results



	0	12	25	50
2 Hosts	0	12	25	51
4 Hosts	0	12	25	51
8 Hosts	0	12	25	50
15 Hosts	0	12	25	52
30 Hosts	0	22	43	76
50 Hosts	0	24	43	89
Ideal	0	12	25	50

Figure 4.6.1: Packet Loss Consistency results

4.6.4 Discussion

From Figure 4.6.1, we can see test results for Mininet's packet loss consistency. From this figure it is clear that Mininet can simulate packet loss very accurately for smaller networks ($<$ 20 Hosts). For larger networks ($>$ 30 Hosts), Mininet could not accurately emulate the desired packet loss. Measured packet loss values proved to be almost double the theoretical value. As Mininet failed to provide all it's hosts with the necessary resources, the desired network conditions couldn't be emulated.

We can therefore conclude that smaller Mininet networks can emulate packet loss very accurately, regardless of resource limitations and small offset values. However, larger networks are greatly affected by resource limitations and will most likely cause the actual packet loss within the network to differ from it's desired value.

4.7 Jitter Test

4.7.1 Motivation

Jitter is defined as delay inconsistency between packets. Jitter can cause packet loss if the magnitude of network jitter is large enough and can cause problems in audio and video playback. It is therefore very important to know how consistent the network jitter of Mininet networks are.
The objective of this test is to see how consistent, measured- vs user-defined, jitter values are, as the size of the network increases.

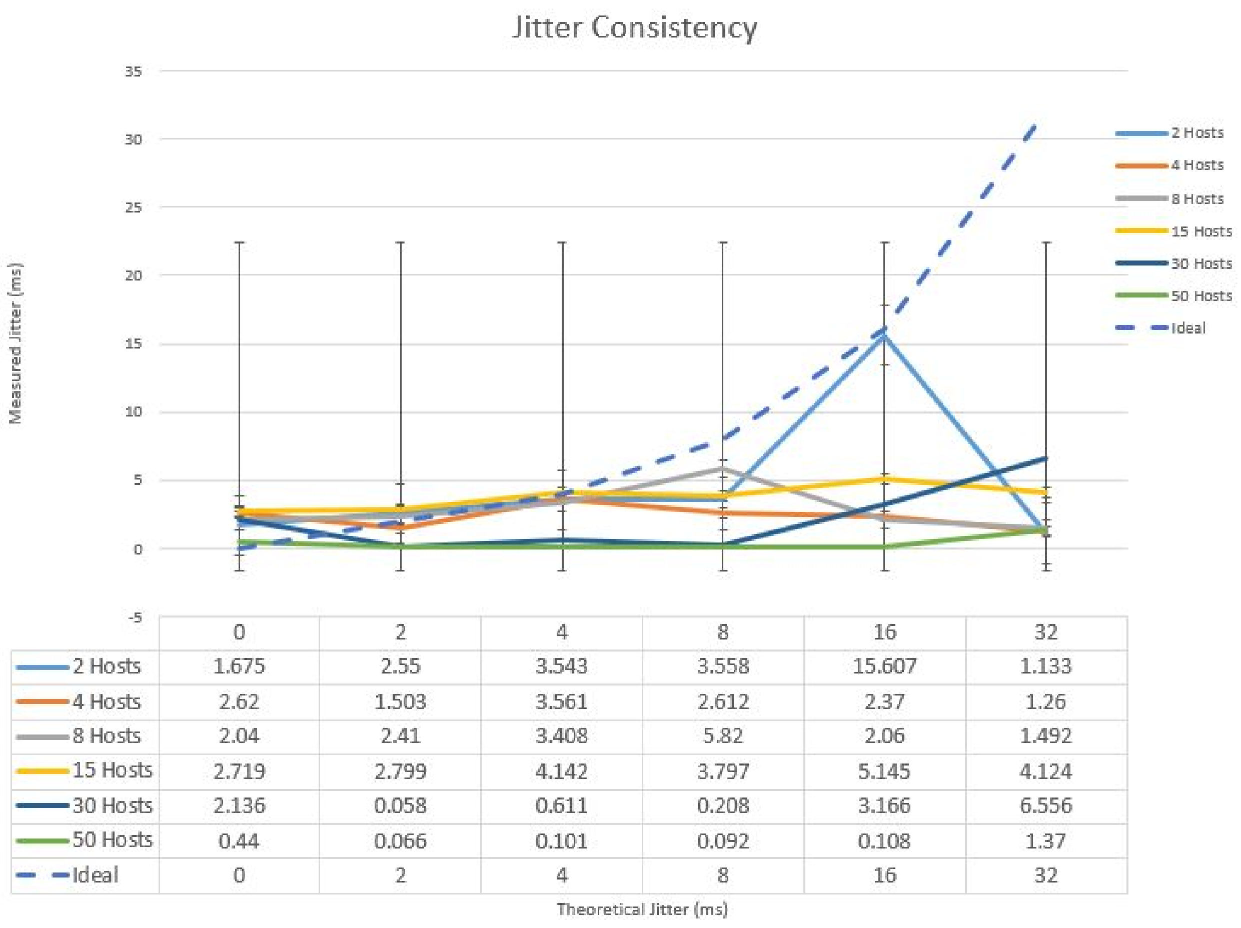
4.7.2 Test Setup

To test the Jitter for each network, iperf UDP data stream tests were used, one host as the iperf server and another as a client. Each test ran for approximately 300 seconds to help improve accuracy of the measured results. The jitter of the emulated network was tested recursively, increasing the jitter after each run. To ensure that no other network effects, affect the packet loss, a bandwidth of 2 Mbps, delay of 0 ms and jitter of 0 ms was used (non-ideal zero values were assumed). A jitter up to 32 ms was tested, as a jitter larger than 32 ms can cause Iperf tests to fail.
Table 4.7.1 shows the jitter values that were tested:

Run no.	Jitter
1	0 ms
2	2 ms
3	4 ms
4	8 ms
5	16 ms
6	32 ms

Table 4.7.1: Jitter Test

4.7.3 Results

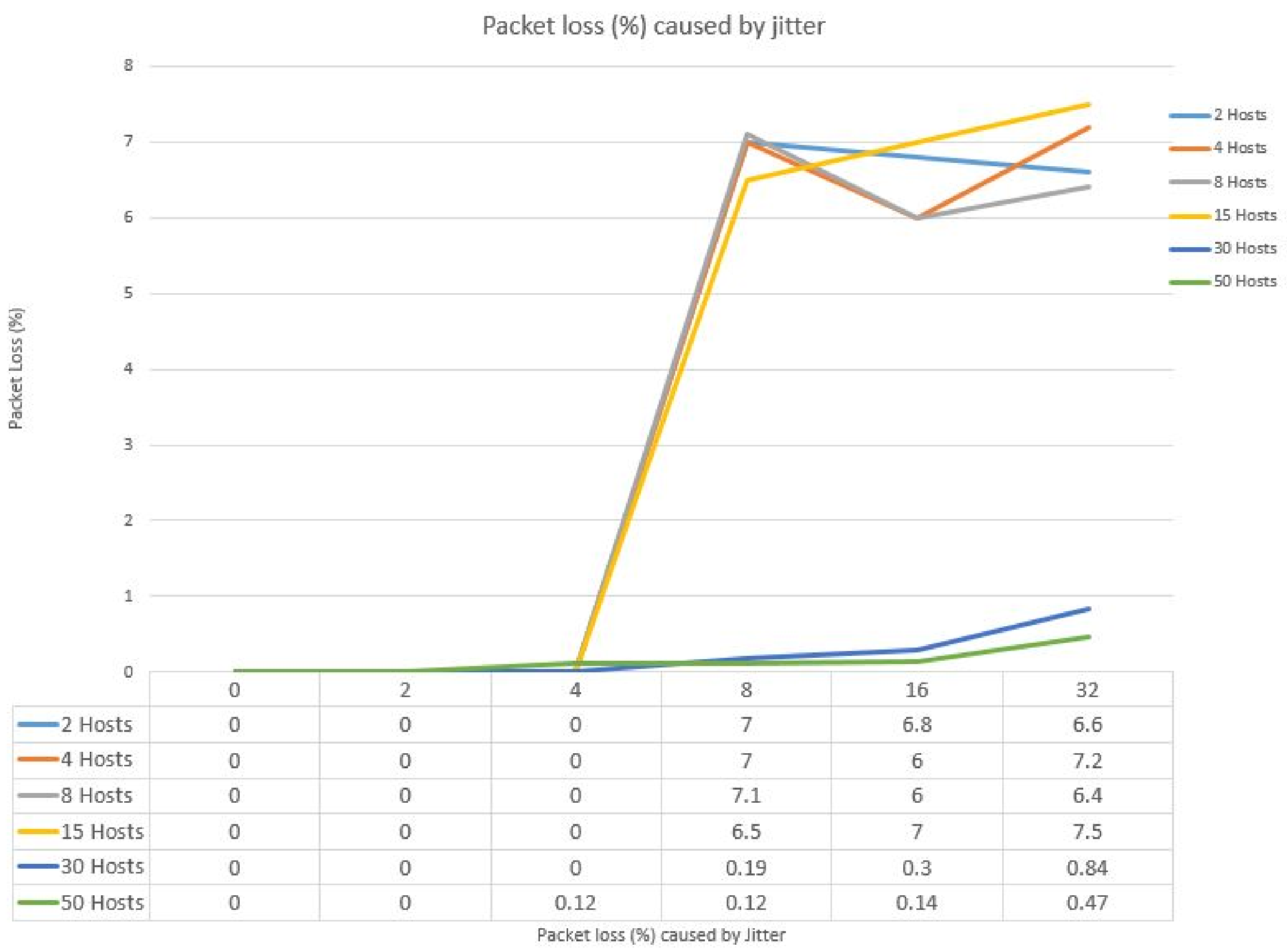


	0	2	4	8	16	32
2 Hosts	1.675	2.55	3.543	3.558	15.607	1.133
4 Hosts	2.62	1.503	3.561	2.612	2.37	1.26
8 Hosts	2.04	2.41	3.408	5.82	2.06	1.492
15 Hosts	2.719	2.799	4.142	3.797	5.145	4.124
30 Hosts	2.136	0.058	0.611	0.208	3.166	6.556
50 Hosts	0.44	0.066	0.101	0.092	0.108	1.37
Ideal	0	2	4	8	16	32

Figure 4.7.1: Jitter Consistency results

4.7.4 Discussion

From Figure 4.7.1, we can see test results for Mininet's jitter consistency. From this figure it is clear that Mininet can simulate a low jitter fairly accurately, despite a small offset. For larger network jitter, the measured results were very inconsistent and inaccurate to the theoretical jitter. This could be a result of packet loss caused by a large jitter. Figure 4.7.2 plots the measured packet loss vs theoretical jitter. This graph proves that when network jitter is larger than 4 ms, packets are dropped. For large networks, the jitter is very inaccurate, but it is however more consistent due to less packet loss. According to a previous report by Handigol *et al.* (2012), there are some problems with UDP Synchronization in Mininet networks. This has lead to some concern surrounding Mininet's ability to accurately emulate jitter.



	0	2	4	8	16	32
2 Hosts	0	0	0	7	6.8	6.6
4 Hosts	0	0	0	7	6	7.2
8 Hosts	0	0	0	7.1	6	6.4
15 Hosts	0	0	0	6.5	7	7.5
30 Hosts	0	0	0	0.19	0.3	0.84
50 Hosts	0	0	0.12	0.12	0.14	0.47

Figure 4.7.2: Packet loss caused by jitter

4.8 Quality Test - Video Streaming

As a further qualitative test and to prove that Mininet virtual host's can communicate and run real software, a simple VLC stream test was used. For this test, a simple star topology, 2 host network was used, where one host would stream the video over the network and the other would receive the incoming Real-time Transport Protocol (RTP) data stream. From this test we would be able to see how different network effects, affect a video stream. The video stream was tested for the effects of packet loss, delay and jitter on the video quality. Bandwidth limitation was ignored for this test, as bandwidth limitations would not severely affect the streaming of a low quality video. The video that was tested is *Big Buck Bunny*.

4.8.1 Ideal Video stream:

In this test, ideal network conditions were used, as seen in Table 4.8.1. From Figure 4.8.1 we can see that the two frames look very identical in terms of quality, despite a small delay on the client side.

Parameter	Value
Bandwidth	2 Mbps
Packet Loss	0 %
Delay	0 ms
Jitter	0 ms

Table 4.8.1: Ideal video stream

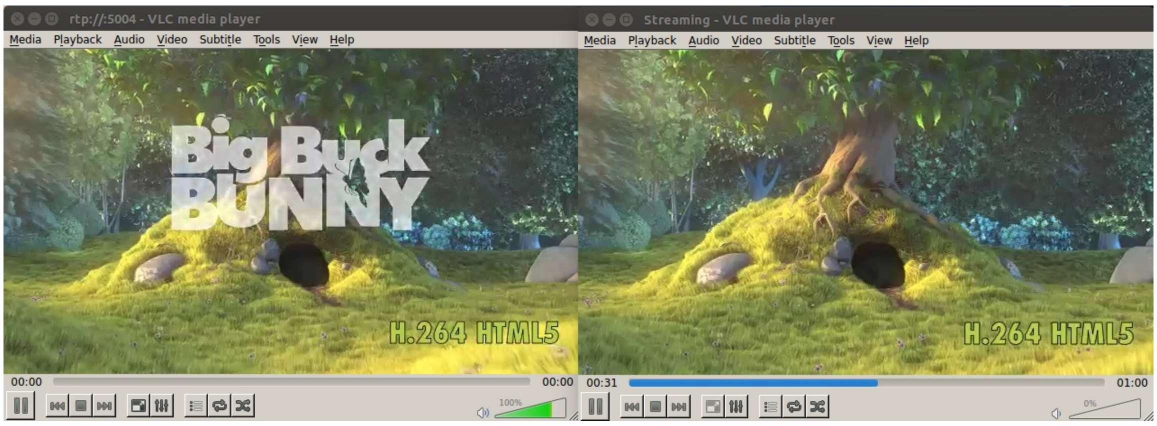

rtp://:5004 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
Big Buck BUNNY
H.264 HTML5
00:00
00:00
100%
Streaming - VLC media player
Media Playback Audio Video Subtitle Tools View Help
H.264 HTML5
00:31
01:00
0%

Figure 4.8.1: Video stream - ideal network conditions | LEFT: Client, RIGHT: Server

4.8.2 Video stream with packet loss:

In this test, the effect of packet loss on the video quality was tested. Table 4.8.2 shows all test parameters. From Figure 4.8.2 we can see that the two frames differ in quality, as the packet loss causes lower video quality on the client side.

Parameter	Value
Bandwidth	2 Mbps
Packet Loss	5 %
Delay	0 ms
Jitter	0 ms

Table 4.8.2: video stream with packet loss

To verify that the quality effects are in fact caused by packet loss, we analyzed the rtp stream for packet loss, using Wireshark. From figure 4.8.3 we can clearly see that the measured packet loss stayed consistent with the theoretical packet loss.

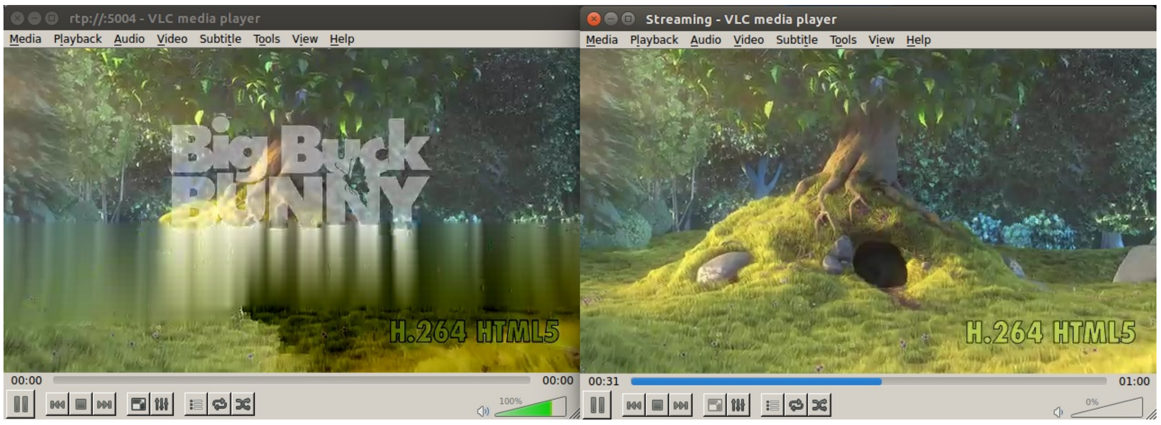
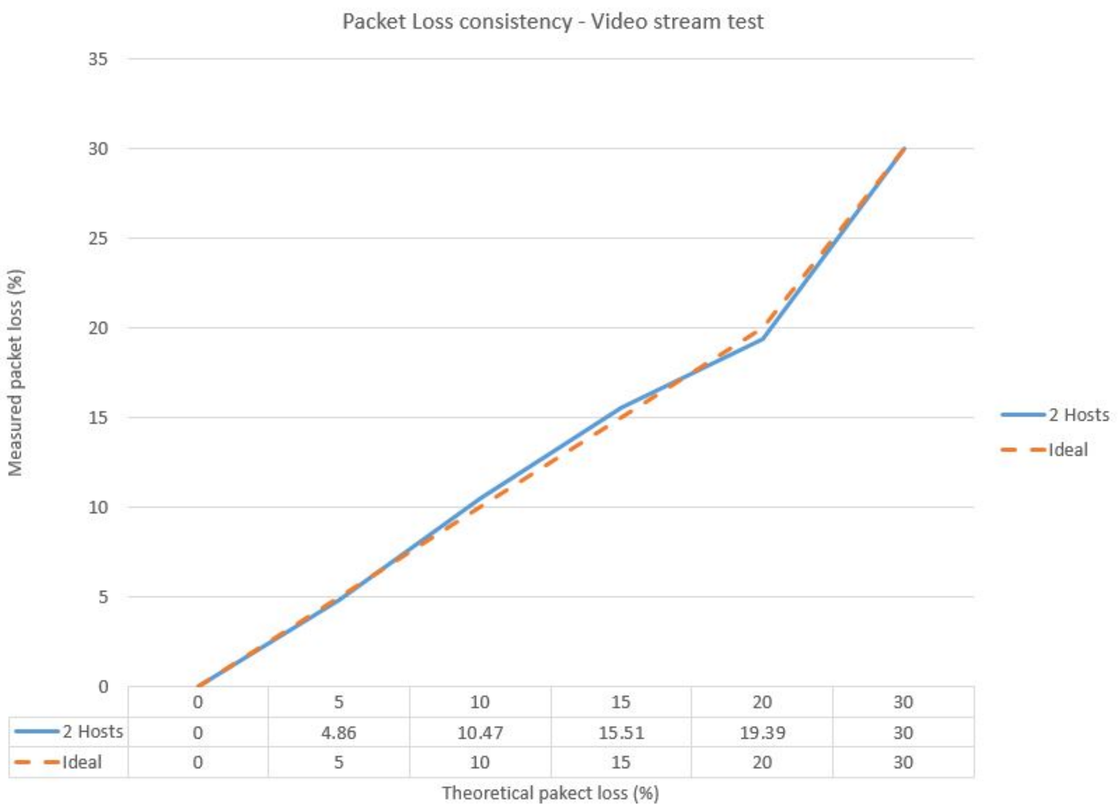

rtp://:5004 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
00:00
00:00
100%
Streaming - VLC media player
Media Playback Audio Video Subtitle Tools View Help
00:31
01:00
0%

Figure 4.8.2: Video stream with packet loss | LEFT: Client, RIGHT: Server



	0	5	10	15	20	30
2 Hosts	0	4.86	10.47	15.51	19.39	30
Ideal	0	5	10	15	20	30

Figure 4.8.3: Video stream with packet loss measured results

4.8.3 Video stream with a delay:

In this test, the effect of delay on the video quality was tested. Table 4.8.3 shows all test parameters. From Figure 4.8.4 we can see that the two frames are nearly identical in quality, with a large delay in the clients video. It was easy to see the 5 second delay in the client's video.

Parameter	Value
Bandwidth	2 Mbps
Packet Loss	0 %
Delay	5000 ms
Jitter	0 ms

Table 4.8.3: Video stream with a delay

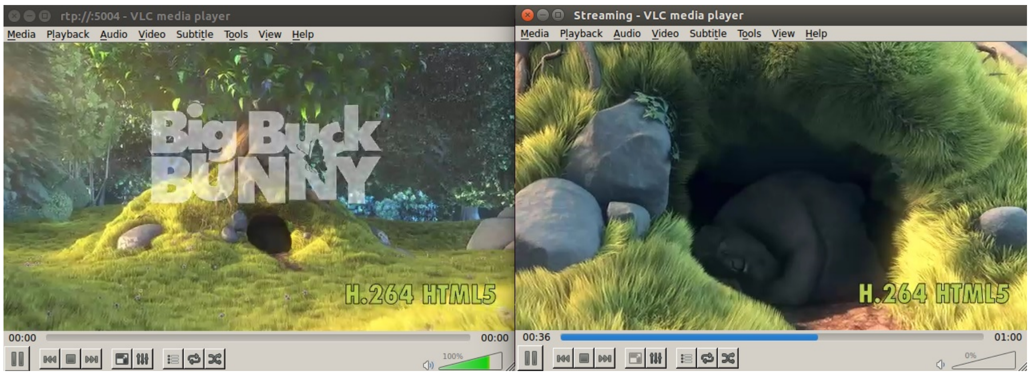

rtp://:5004 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
Big Buck BUNNY
H.264 HTML5
00:00
00:00
Streaming - VLC media player
Media Playback Audio Video Subtitle Tools View Help
H.264 HTML5
00:36
01:00

Figure 4.8.4: Video stream with a delay | LEFT: Client, RIGHT: Server

4.8.4 Video stream with jitter:

In this test, the effect of jitter on the video quality was tested. Table 4.8.4 shows all test parameters. From Figure 4.8.5 we can see that the two frames are nearly identical in quality, despite a small delay in the clients video. Jitter did thus not have any noticeable affect on the video's quality. This is not however a result of the jitter not being present in the network, but a result of VLC's jitter buffer, that gets rid of any jitter in the network. Even after turning off VLC's jitter buffer, the effects of jitter was still not visible.

Parameter	Value
Bandwidth	2 Mbps
Packet Loss	0 %
Delay	0 ms
Jitter	32 ms

Table 4.8.4: Video stream with jitter

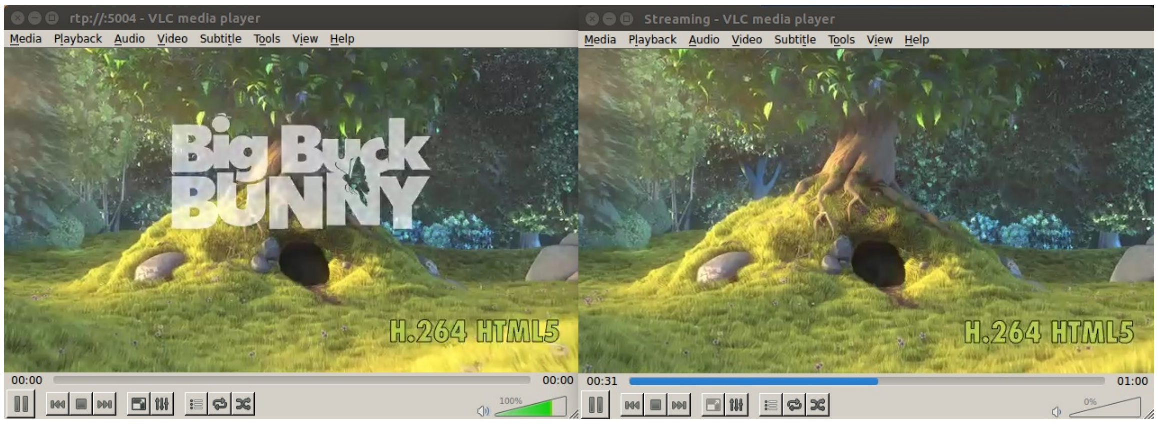

rtp://:5004 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
Big Buck BUNNY
H.264 HTML5
00:00
00:00
100%
Streaming - VLC media player
Media Playback Audio Video Subtitle Tools View Help
H.264 HTML5
00:31
01:00
0%

Figure 4.8.5: Video stream with jitter | LEFT: Client, RIGHT: Server

4.9 Summary

This chapter proved that Mininet networks can easily be created and tested. From the tests conducted in this chapter, it is clear that Mininet can emulate network conditions like network delay and packet loss, very accurately over a software interface. However, the bandwidth and jitter tests did not yield accurate results. As it is known that by running Mininet on a virtual machine the network becomes very limited, these inaccurate test results did not cause too much concern. The last section of this chapter, proved that Mininet's virtual hosts can run real software (VLC) and provided a further quality test of the emulated network. A summary of all test results is available in Chapter 7.2.

Chapter 5

Hardware Interface Experiment

This chapter focuses on Mininet's performance across a hardware interface. Tests were conducted to help test Mininet's capability of emulating paramaterized networks over a hardware interface, by testing Mininet's bandwidth-, delay-, packet loss- and jitter consistency with a varying amounts of hosts. These tests would also prove whether different Mininet networks can communicate over a hardware interface. The data captured from these tests helped in plotting performance graphs. In section 1, an in depth overview of the test setup is discussed - this setup was used for all tests. Section 2 provides us with a motivation, to why it is necessary to test Mininet's performance over a hardware interface. In sections 3 to 5 the different network effects are tested for their consistency. Each test is discussed in terms of; motivation to perform the test, setup, results and why or why not measured values differ from theoretical values.

5.1 Setup Overview

Figure 5.1.1 is a visual representation of the experimental setup for testing Mininet across a hardware interface. In this test a physical switch and two Raspberry Pi's were used to set up a small network - these hosts will be used to run Mininet. To allow full control of the network, a DHCP server was used on Windows (Host PC), to assign custom IP's, in the range 10.0.0.101-255 for the two Raspberry Pi's and VirtualBox. Table 5.1.1 shows the IP address of each machine.
To enable the Raspberry Pi's to run Mininet, the appropriate software had to be installed - this was easily achieved by using the **install.sh** command. To enable performance testing across a hardware interface, a remote controller can be used to connect Mininet networks on different machines. A POX controller was started in VirtualBox, to be used as a remote controller. Rouse (2015*a*) gives a good definition of a POX controller, as an open source development interface for python based SDN control applications, such as OpenFlow SDN controllers.
On each Raspberry Pi, a Mininet network was created, with two hosts and a remote controller, found at 10.0.0.101 (VirtualBox). This remote controller, allows the use of a GRE tunnel between the Mininet switch found on Raspberry Pi1 and the Mininet switch found on Raspberry Pi2. The GRE tunnel between the switches connects the two emulated networks, and thus allows the virtual hosts from the two Mininet networks to exchange data.
One problem that came to our attention, was that Mininet's standard switches and the remote controller did not use the same OpenFlow protocol. This meant that Iperf tests

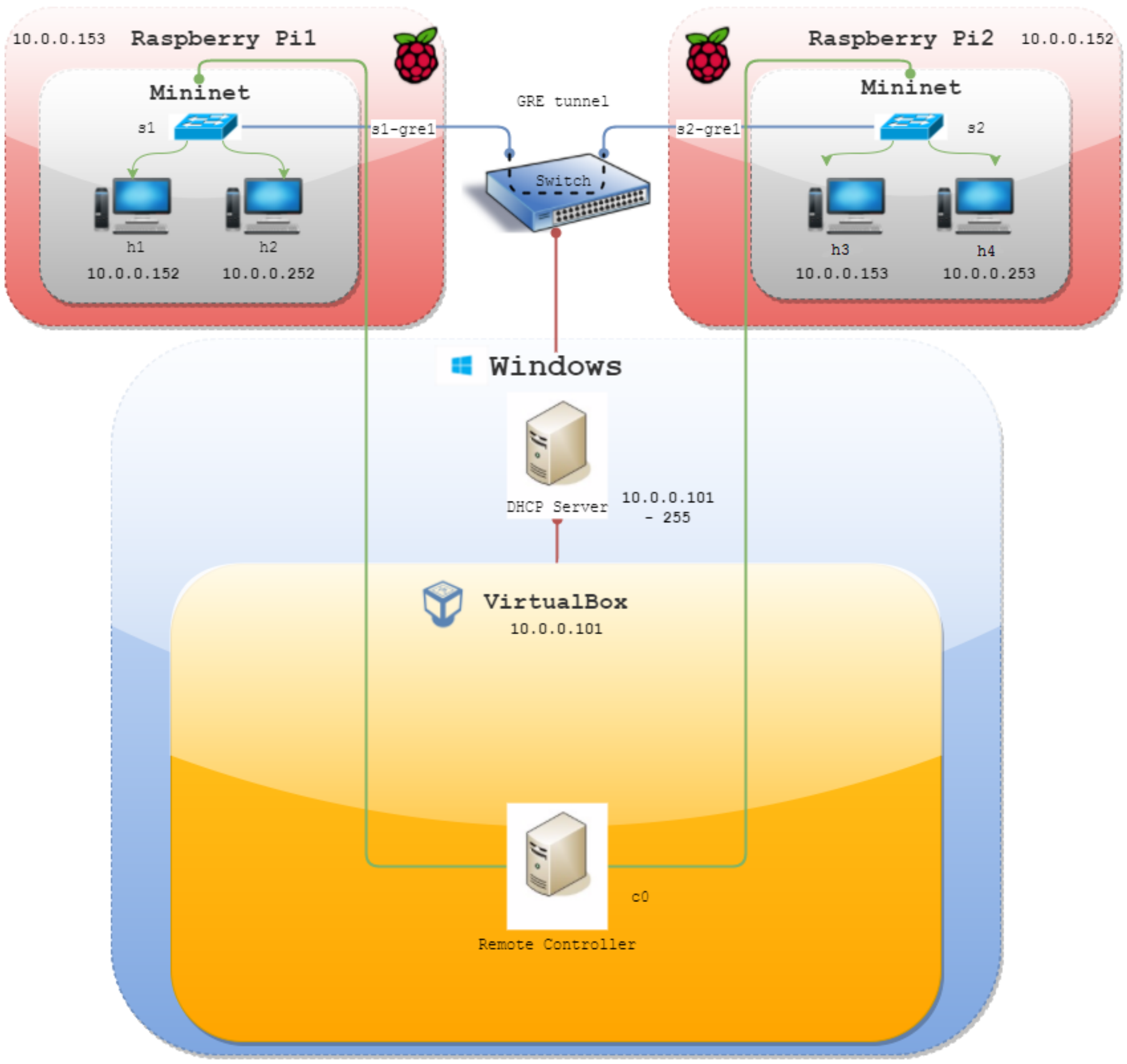


Figure 5.1.1: Hardware experiment setup

could not be performed, as machines could not properly exchange data. To allow testing of UDP and TCP data streams, OVSSwitches were used instead of Mininet's standard switches. OVSSwitches allowed us to match the controller and switch OpenFlow protocols.

Machine	Use	IP
Windows	DHCP server	10.0.0.7
VirtualBox	Remote Controller	10.0.0.101
Raspberry pi1	Mininet Network 1	10.0.0.153
Raspberry pi2	Mininet Network 2	10.0.0.154
h1	Host on Network 1	10.0.0.153
h2	Host on Network 1	10.0.0.253
h3	Host on Network 2	10.0.0.152
h4	Host on Network 2	10.0.0.252

Table 5.1.1: Addresses for hardware interface experiment

5.2 Motivation

After successfully setting up a network, consisting of two Mininet networks from separate devices, it was necessary to test Mininet's performance across the hardware interface. Tests would prove whether network effects are still present, from an external host's (outside of the Mininet network) point of view. To achieve this, we tested Mininet's bandwidth-,delay-,packet loss- and jitter consistency across a hardware interface, using Iperf and ping tests.

5.3 Bandwidth Test

5.3.1 Motivation

The objective of this test is to see how consistent, measured- vs user-defined, bandwidth values are, over a hardware interface. See chapter 4 section 4.4.1 for more information.

5.3.2 Test Setup

To test the bandwidth consistency over a hardware interface, Iperf TCP data stream tests were used to measure the bandwidth between two hosts, one Mininet host from each Raspberry Pi, increasing the link bandwidth after each run. To ensure that no other network effects, affect the bandwidth, a delay of 0 ms, packet loss of 0% and jitter of 0 ms was used. The following bandwidths were tested:

Run no.	Bandwidth
1	1 Mbps
2	2 Mbps
3	4 Mbps
4	8 Mbps
5	12 Mbps
6	20 Mbps

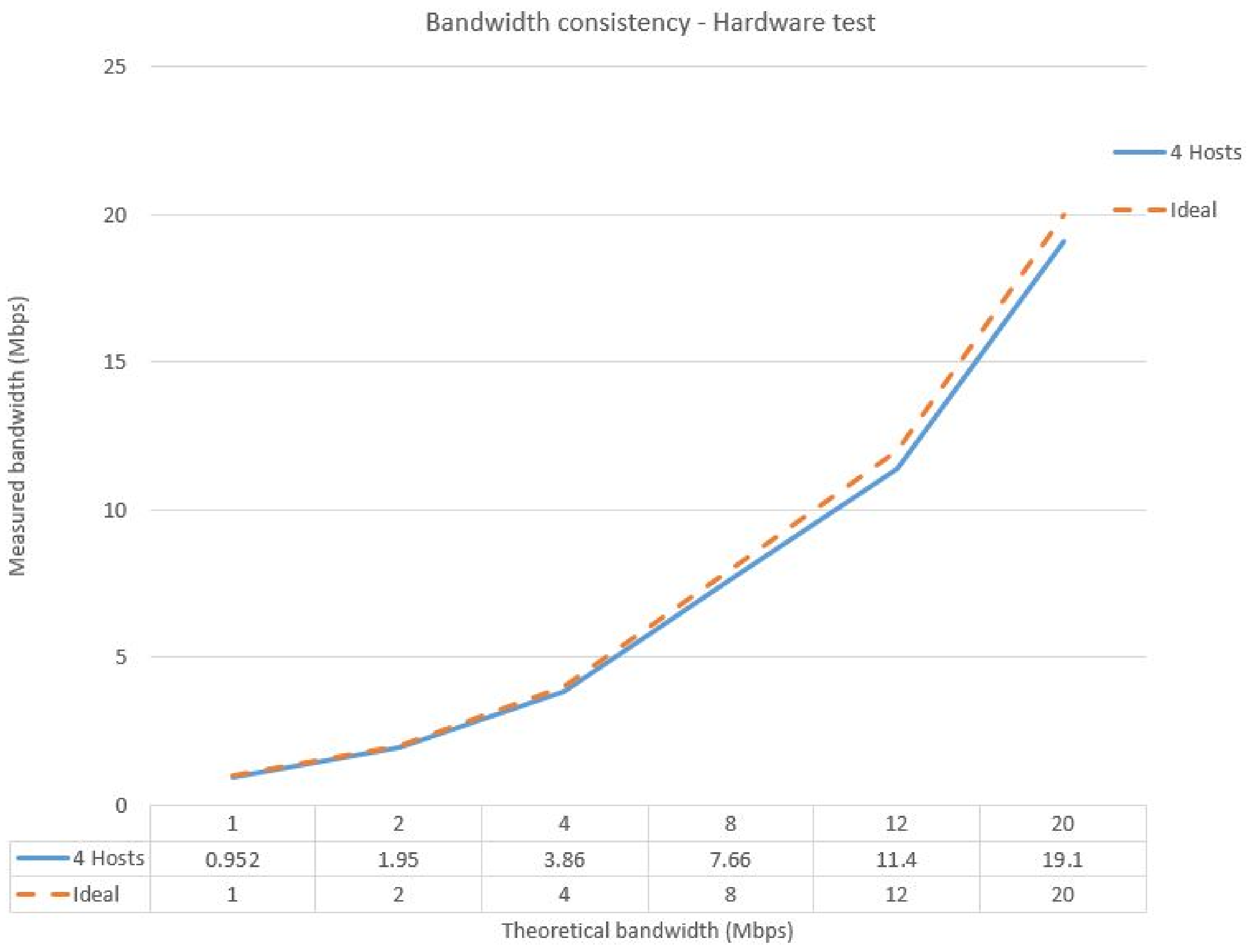
Table 5.3.1: Bandwidth Test

Parameter	Theoretical	Measured
Packet Loss	0 %	0 %
Delay	0 ms	0.952 ms
Jitter	0 ms	3.52 ms

Table 5.3.2: Average of measured ideal network conditions

To ensure that the values of delay, packet loss and jitter were in fact zero, The two emulated networks were used to run a preliminary Iperf test, to measure the actual delay, packet loss and jitter values, between two hosts from different Mininet Networks. Table 4.4.2 shows the measured values for each of the parameters. It is clear that the measured values of delay and jitter are not zero - and is something that should be considered in further discussions. These non-zero conditions were assumed for all tests.

5.3.3 Results



	1	2	4	8	12	20
4 Hosts	0.952	1.95	3.86	7.66	11.4	19.1
Ideal	1	2	4	8	12	20

Figure 5.3.1: Bandwidth Consistency results

5.3.4 Discussion

From Figure 5.3.1, we can see test results for Mininet's bandwidth consistency. From this figure it is clear that Mininet can simulate various bandwidths very accurately over a hardware interface. As we can see, there are some small deviations from the theoretical values, this is most likely a result of the non-zero values for delay and jitter. Even though, measured values are not identical to theoretical values, the differences are small enough to ignore. It can therefore be accepted that Mininet can emulate bandwidth over a hardware interface very accurately. These results clearly differ from the results generated in chapter 4.4. This is a result of using a real hardware device, rather than a virtual machine to emulate the Mininet network.

5.4 Delay Test

5.4.1 Motivation

The objective of this test is to see how consistent, measured- vs user-defined, network delay values are, over a hardware interface. See chapter 4 section 4.5.1 for more information.

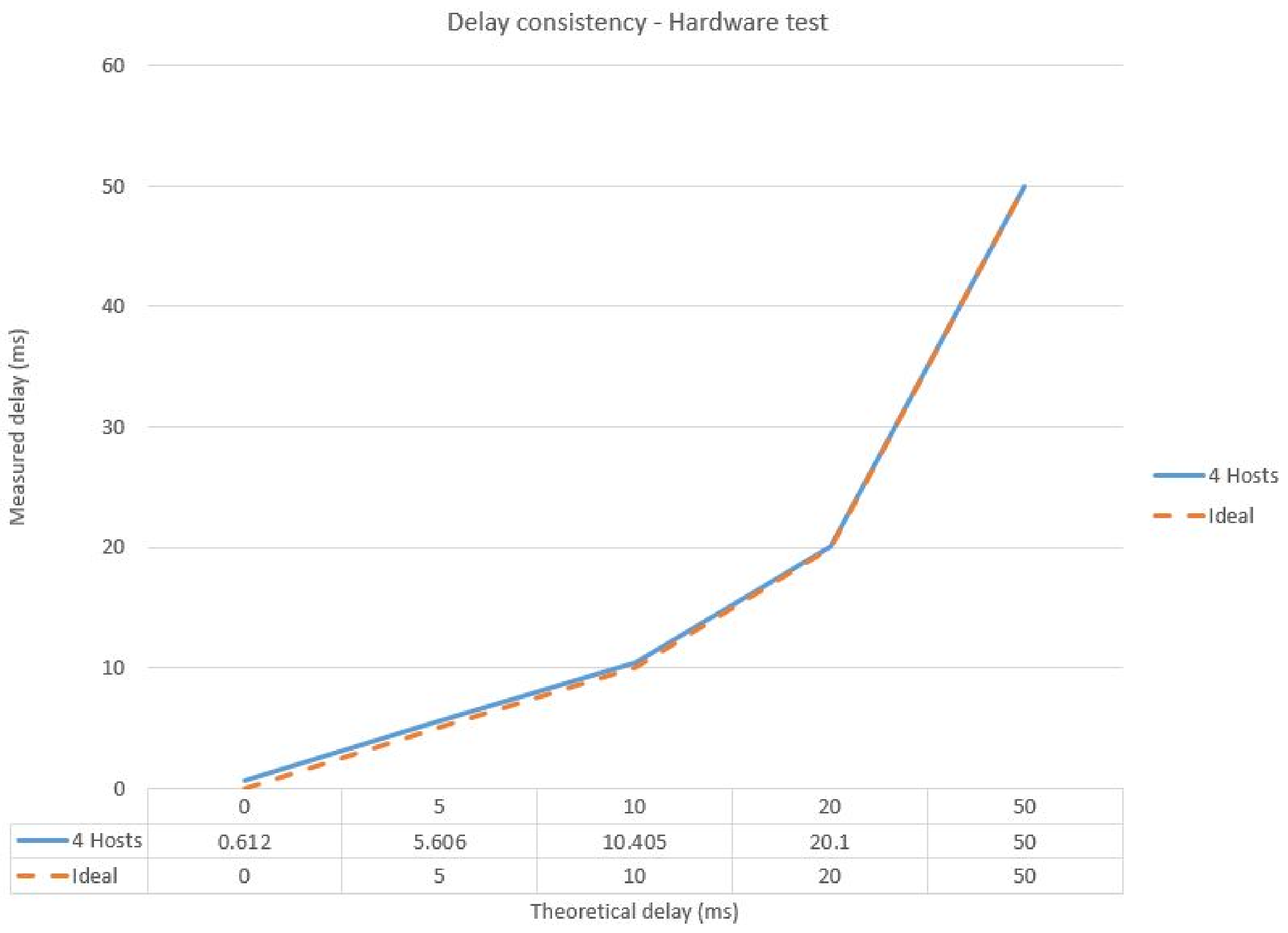
5.4.2 Test Setup

To test the delay over a hardware interface, a ping function was used, increasing the Delay after each run. To ensure that no other network effects, affect the delay, a bandwidth of 2 Mbps, packet loss of 0% and jitter of 0 ms was used (non-ideal zero values were assumed). The ping test was performed between two hosts, one Mininet host from each Raspberry Pi, sending out a ping every second for 50 seconds. It is important to know that the measured delay is the travel time to-and-from the destination, thus the actual delay time to one host is the measured value, divided by 2. The average of the 50 pings was accepted as the delay for each network. Table 5.4.1 shows the delay times that were tested:

Run no.	Delay
1	0 ms
2	5 ms
3	10 ms
4	20 ms
5	50 ms

Table 5.4.1: Delay Test

5.4.3 Results



	0	5	10	20	50
4 Hosts	0.612	5.606	10.405	20.1	50
Ideal	0	5	10	20	50

Figure 5.4.1: Delay consistency between two hosts from different Mininet networks

5.4.4 Discussion

From Figure 5.4.1, we can see tests results for Mininet's delay consistency over a hardware interface. It is clear that Mininet can simulate delay very accurately over a hardware interface, despite a very small offset delay of about 0.6 ms in each measurement. This offset delay is most likely caused by the non-zero values for delay and jitter. It can therefore be accepted that Mininet can simulate network delay/latency over a hardware interface very accurately. It is noted that the offset delay is a lot smaller over the hardware interface than it is over the software interface (0.6 ms $<$ 2 ms), which was tested in chapter 4.5. This is a result of using a real hardware device, rather than a virtual machine to emulate the Mininet network.

5.5 Packet Loss Test

5.5.1 Motivation

The objective of this test is to see how consistent, measured- vs user-defined, packet loss values are, over a hardware interface. See chapter 4 section 4.6.1 for more information.

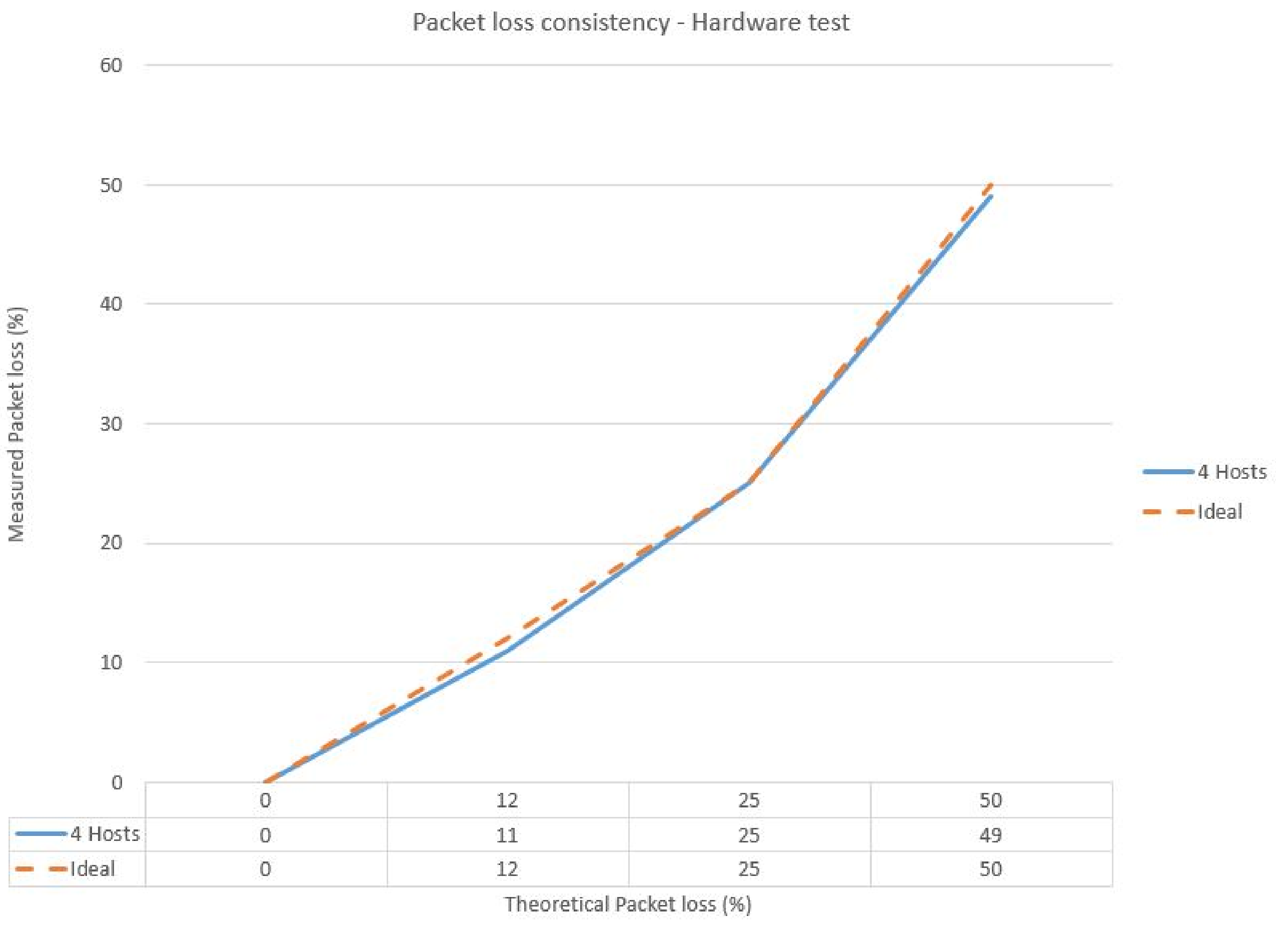
5.5.2 Test Setup

To test the packet loss for each network, Iperf UDP data stream tests were used to measure the packet loss between two hosts, one Mininet host from each Raspberry Pi, increasing the packet loss after each run. To ensure that no other network effects, affect the packet loss, a bandwidth of 2 Mbps, delay of 0 ms and jitter of 0 ms was used (non-ideal zero values were assumed). Packet loss up to 50% was tested, as any packet loss higher than 50% would cause Iperf tests to fail. The follow packet loss values were tested:

Run no.	Loss
1	0 %
2	12 %
3	25 %
4	50 %

Table 5.5.1: Packet Loss Test

5.5.3 Results



	0	12	25	50
4 Hosts	0	11	25	49
Ideal	0	12	25	50

Figure 5.5.1: Packet Loss Consistency results

5.5.4 Discussion

From Figure 5.5.1, we can see test results for Mininet's packet loss consistency. It is clear that Mininet can simulate packet loss very accurately over a hardware interface. This result was expected to be accurate, as in theory the emulator would simply discard a certain percentage of packets, to simulate packet loss.

5.6 Jitter Test

5.6.1 Motivation

The objective of this test is to see how consistent, measured- vs user-defined, jitter values are, over a hardware interface. See chapter 4 section 4.7.1 for more information.

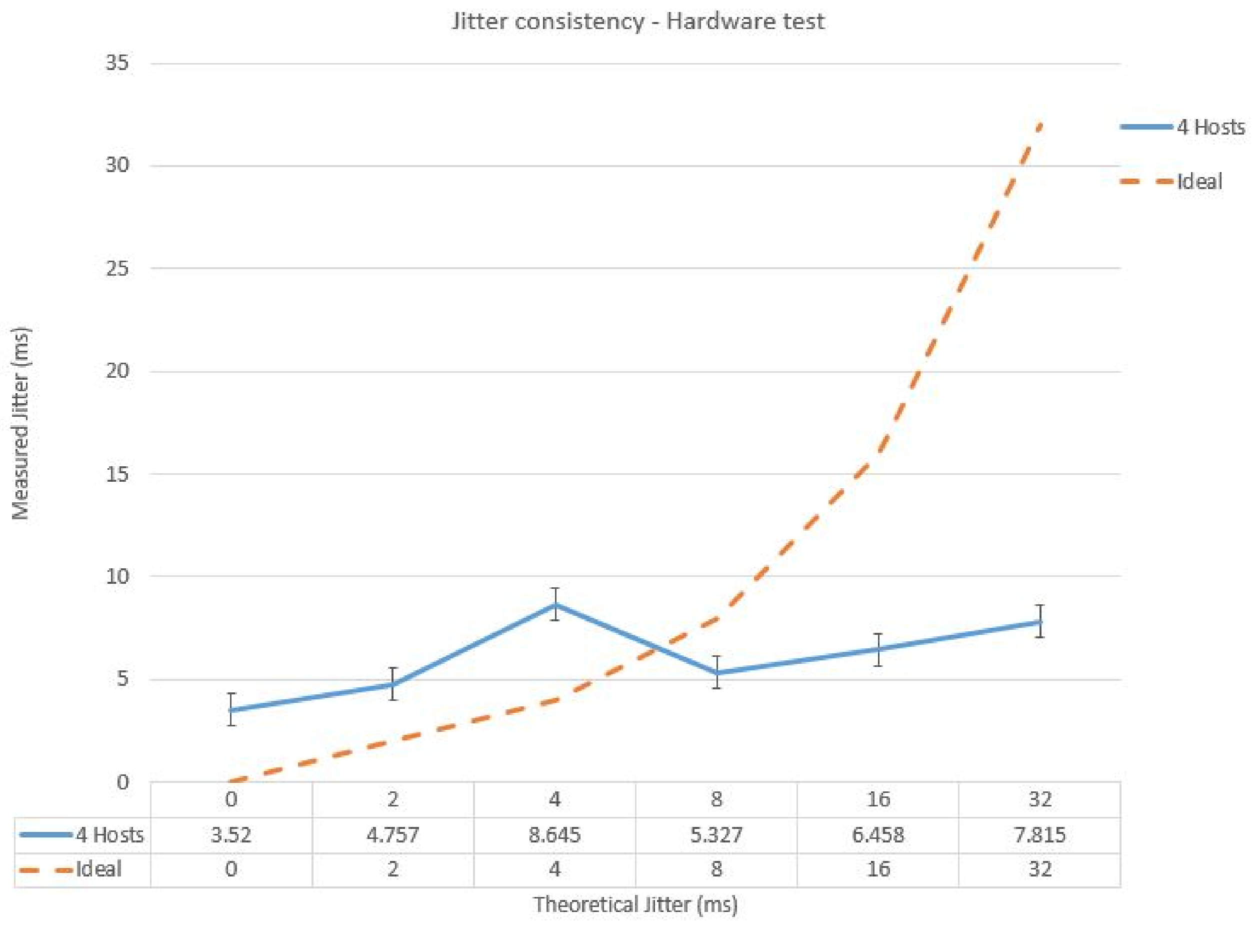
5.6.2 Test Setup

To test the jitter for each network, Iperf UDP data stream tests were used to measure the jitter between two hosts, one Mininet host from each Raspberry Pi, increasing the jitter after each run. Each test ran for approximately 300 seconds to help improve accuracy of the measured results. To ensure that no other network effects, affect the packet loss, a bandwidth of 1 Mbps, delay of 0 ms and jitter of 0 ms was used (non-ideal zero values were assumed). A jitter up to 32 ms was tested, as a jitter larger than 32 ms can cause Iperf tests to fail.

Run no.	Jitter
1	0 ms
2	2 ms
3	4 ms
4	8 ms
5	16 ms
6	32 ms

Table 5.6.1: Jitter Test

5.6.3 Results



	0	2	4	8	16	32
4 Hosts	3.52	4.757	8.645	5.327	6.458	7.815
Ideal	0	2	4	8	16	32

Figure 5.6.1: Jitter Consistency results

5.6.4 Discussion

From Figure 5.6.1, we can see test results for Mininet's jitter consistency. From this figure it is clear that Mininet's jitter is very consistent with an average jitter of 6.51 ms over the hardware platform, but unfortunately not accurate at all. Even though the jitter is not accurate, the delay jitter does not cause packets to be dropped like in previous tests performed. This could be a result of using an OVSSwitch rather than a standard Mininet switch. According to a previous report by Handigol *et al.* (2012), there are some problems with UDP Synchronization in Mininet networks.

5.7 Summary

This chapter proved that Mininet can easily communicate with real hosts, outside of the emulated network. From the tests performed in this chapter, it is clear that Mininet can emulate network conditions like bandwidth, network delay and packet loss, very accurately over a hardware interface. The jitter tests however did not yield accurate results and has lead to some concerns toward Mininet's ability to accurately emulate jitter. Two Mininet networks, from separate devices were successfully connected and tested. Accurate Iperf test results, therefore proved that Mininet is a scalable Network emulator. A summary of all test results is available in Chapter 7.1.

Chapter 6

Mininet Network vs. A Real Network

For a final test of Mininet's performance, a real network that was previously configured for the hardware interface tests (Chapter 5), was used to tests Mininet's ability to accurately emulate a real network. This chapter gives an overview of all test that were conducted on both the software and hardware platforms. In section 1 a brief overview is given on the real network that was used. Sections 2 and 3 presents and discusses and compare the measured results for the hardware- and software platforms.

6.1 Setup Overview

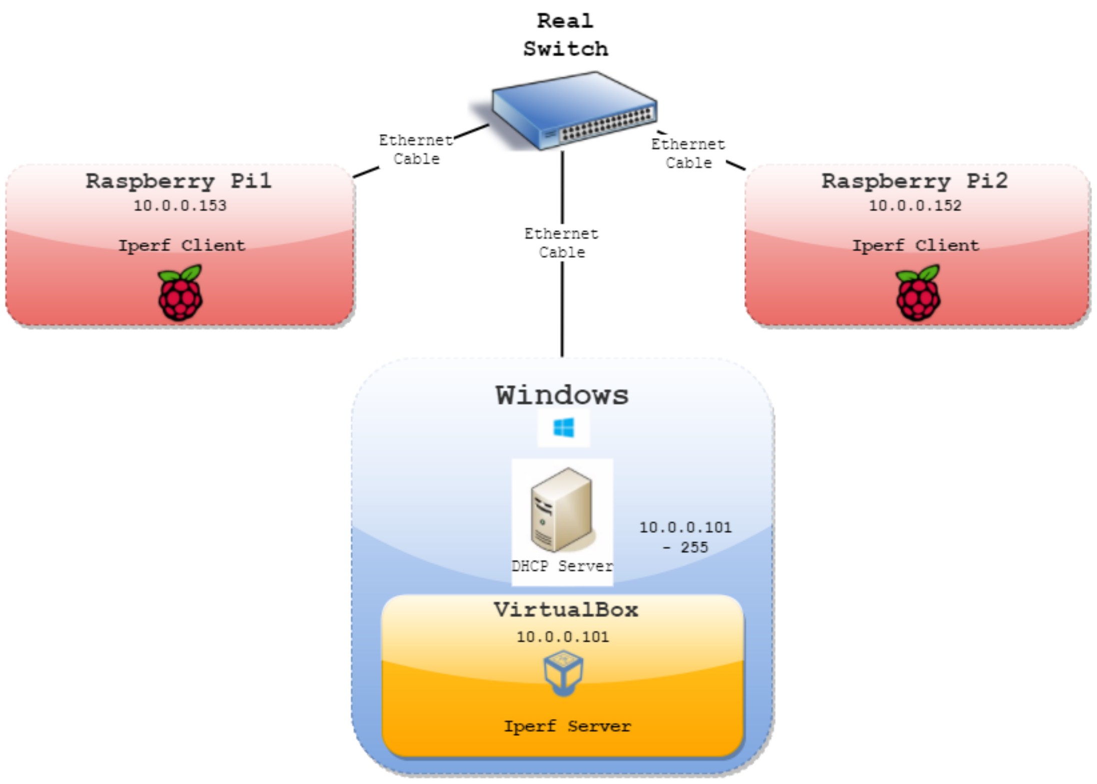


Figure 6.1.1: Real Network used to test Mininet's performance

Figure 6.1.1 is a graphical representation of the physical network. The network consists of a real switch, two Raspberry Pi's and the host PC for VirtualBox. A DHCP server was used on the host PC to assign custom IP addresses in the range 10.0.0.101-255 to the devices within the network. A list of all device IP's can be found in Table 6.1.1.

Machine	Use	IP
Windows	DHCP server	10.0.0.7
VirtualBox	Iperf Server	10.0.0.101
Raspberry pi1	Iperf Client	10.0.0.153
Raspberry pi2	Iperf Client	10.0.0.154

Table 6.1.1: Addresses for Real Network

6.2 Motivation

This test would give us a very good indication of how accurate the emulator can emulate a real network - thus we want to test whether Mininet can produce the desired network effects to emulate a very simple network accurately. This would serve as a final qualitative test of Mininet's abilities and will help us to form an informed conclusion on Mininet as a network emulator.

6.3 The Real Network

To measure the network conditions, Iperf tests were used. VirtualBox was used as the Iperf server, allowing both Raspberry Pi's to connect to the server as clients. Measured Results are shown in Table 6.3.1.

Parameter	**Raspberry Pi1**	**Raspberry Pi2**
Bandwidth	94.3 Mbps	94.3 Mbps
Packet Loss	0 %	0 %
Delay	0.889 ms	1.173 ms
Jitter	0.064 ms	0.201 ms

Table 6.3.1: Real network - Measured results

6.4 Mininet Network

To create an identical Mininet network, the results from Table 6.3.1 was used. The GUI's output can be seen in Figure 6.4.1. To test Mininet's performance, Iperf tests were used. One host was named VBox - this host would be used as the Iperf server. The other two hosts were called Pi1 and Pi2 and would be used as Iperf clients, to measure network conditions.

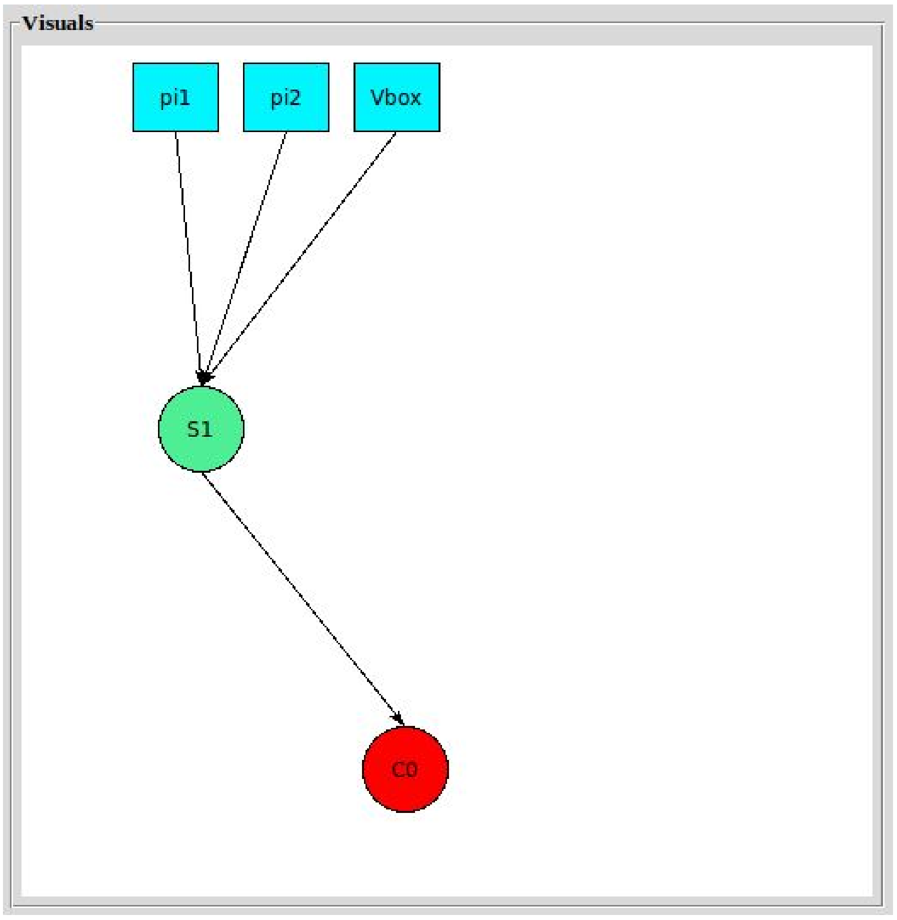


Figure 6.4.1: Jitter Consistency results

Parameter	Raspberry Pi1	Raspberry Pi2
Bandwidth	63.10 Mbps	60.56 Mbps
Packet Loss	0 %	0 %
Delay	2.46 ms	2.35 ms
Jitter	0.25 ms	0.26 ms

Table 6.4.1: Mininet network - Measured results

6.4.1 Bandwidth accuracy

From Table 6.4.1 it is evident that Mininet could not accurately emulate the systems bandwidth of 94.3 Mbps, but rather emulated a much lower bandwidth. Additional measurements of the Mininet network's bandwidth can be found in Table D.3.1 in appendix D. The lower bandwidth value was expected, as it is known that Mininet's ability to emulate bandwidth accurately is throttled when running Mininet on a VM.

6.4.2 Packet Loss accuracy

From Table 6.4.1 we can see that there is no packet loss in the network, therefore Mininet accurately emulated the absence of packet loss.

6.4.3 Delay accuracy

From Table 6.4.1 it is evident that the delay measured for each device is very small, however they differ from the desired values. We clearly see the small 2 ms delay mentioned earlier in Chapter 4, for the software interface experiment. Additional measurements of the Mininet network's delay can be found in Table D.3.2 in appendix D. This slightly larger network delay was most likely caused by the non-ideal zero values present in the virtual environment.

6.4.4 Jitter accuracy

From Table 6.4.1 it is evident that a delay jitter is present, however the measured values differ slightly from the desired values. Additional measurements of the Mininet network's delay can be found in Table D.3.3 in appendix D. This slightly larger packet jitter was most likely caused by the non-ideal zero values present in the virtual environment.

6.5 Summary

From the tests performed in this chapter, Mininet yielded similar results to those of the real network. Test results proved that Mininet can emulate packet loss very accurately, other network effects like delay and jitter can also be emulated fairly accurately. The non-ideal conditions in the virtual environment, caused small offset values to be present in the measured delay and jitter values. Bandwidth however isn't emulated accurately, due to the resource limitation caused by working on a virtual machine. A summary of all test results is available in Chapter 7.3.

Chapter 7

Summary of test results

7.1 Mininet on a Hardware Platform

Parameter	**Results**	**Discussion**	**Graphs**
Bandwidth	It was found that Mininet can emulate bandwidth accurately over a hardware interface. A bandwidth of 20 Mbps could be emulated accurately	Non-ideal zero values may have caused the bandwidth to be slightly lower than expected	Figure 5.3.1
Packet Loss	It was found that Mininet can emulate packet loss very accurately over a hardware interface	Non-ideal zero values may have caused the packet loss to be slightly higher than expected	Figure 5.5.1
Delay	It was found that Mininet can emulate delay very accurately over a hardware interface	Non-ideal zero values may have caused the delay to be slightly larger than expected	Figure 5.4.1
Jitter	It was found that Mininet's jitter emulation was fairly accurate for small jitter values, but as the jitter's magnitude increased, it was evident that the jitter was not emulated accurately. Tests results prove that Mininet cannot emulate jitter accurately over a hardware interface	Non-ideal zero values may have caused the jitter to become inconsistent and inaccurate	Figure 5.6.1

Table 7.1.1: Summary of Mininets performance over a Hardware Interface

7.2 Mininet on a Software Platform

Parameter	**Results**	**Discussion**	**Graphs**
Bandwidth	It was found that Mininet can emulate bandwidth accurately up to 5 Mbps - for a bandwidth larger than 5 Mbps, the emulated bandwidth will be much lower than expected	The network was severely limited by the virtual machine's resources, limiting accurate bandwidth emulation to 5 Mbps	Figure 4.4.1
Packet Loss	It was found that Mininet can emulate packet loss accurately for networks with less than 20 hosts For networks with more than 20 hosts it was found that packet loss was almost double the expected value	The Network was limited by the virtual machine's resources, limiting accurate packet loss emulation to networks with less than 20 Hosts	Figures 4.6.1, 4.8.3
Delay	It was found that Mininet can emulate delay accurately for networks with less than 20 hosts. For networks with more than 20 hosts it was found that the delay was almost double the expected value	The network was limited by the virtual machine's resources, limiting accurate delay emulation to networks with less than 20 Hosts.	Figure 4.5.1
Jitter	It was found that Mininet's jitter emulation was fairly accurate for small jitter values, but became very inconsistent and inaccurate as the magnitude of jitter increased. Tests results prove that Mininet cannot emulate jitter accurately	Jitter may have limited by the virtual machine's resources, limiting accurate jitter emulation. It was also found that for jitter with a magnitude larger than 4 ms, packets were dropped, causing packet loss of up to 7%	Figures 4.7.1, 4.7.2

Table 7.2.1: Summary of Mininets performance over a Software Interface

7.3 Mininet Network vs. A Real Network

Parameter	**Results**	**Discussion**
Bandwidth	It was found that Mininet could not accurately emulate the real networks bandwidth	The network was severely limited by the virtual machine's resources, resulting in inaccurate bandwidth emulation
Packet Loss	It was found that Mininet could emulate the real network's packet loss very accurately	Packet loss was not affected by the resource limitations within the virtual environment
Delay	It was found that Mininet could emulate the real network's delay very accurately	A small offset in the delay value was not surprising, as previous tests have proven that by working in a virtual environment a small offset delay is added
Jitter	It was found that Mininet could emulate the real network's jitter very accurately. As jitter resluts in previous tests have proven to be inconsistent and inaccurate, it was surprising to see accurate jitter results.	Previous tests have proven that Mininet can emulate small jitter accurately

Table 7.3.1: Summary of Mininet's performance compared to a real network

Chapter 8

Conclusion

As network emulation plays such a prominent role in the testing of software and Software-Defined Networks, it is very important to inspect whether an emulator can emulate networks accurately and efficiently.

This report has discussed the accuracy and scalability of Mininet as a network emulator and has highlighted some of the emulators limitations. The objectives of this project was to evaluate Mininet's performance by; (1) Developing a GUI for Mininet, to assist in creating test networks, (2) by conducting performance tests on both a software and hardware interface and (3) by comparing the performance of a Mininet network to an identical physical network. Test results have proven that when running Mininet on a software platform, small networks (networks with less than 20 hosts) can be emulated very accurately in terms of bandwidth, packet loss and delay. However, for large networks (networks with more than 30 hosts) Mininet's performance is throttled by the virtual machine, causing emulated effects to be inaccurate. When running Mininet on a hardware platform, it was evident that the accuracy of the emulated values were considerably better than those on a virtual machine, due to better resource availability. For both software- and hardware platforms it was found that emulated values of jitter is very inconsistent and inaccurate. By connecting Mininet networks from separate devices over a hardware interface, proved that very large Mininet networks can be created. Qualitative tests also proved that emulated network effects, affect data streams in a similar manner to how real-life network effects would. Finally, by testing Mininet's ability to emulate a real network, proved that when resources are limited, Mininet can accurately emulate packet loss, delay and a small jitter, but struggles to emulate a large bandwidth.

From these results we can conclude that Mininet is an accurate and scalable network emulator. However, Mininet's performance relies heavily on the host device's available resources.

[Contributions] This project has lead to the development of a user-friendly GUI, which can be used to create and save custom Mininet networks. Performance graphs generated in this project can be used to predict how accurately a network will be emulated by Mininet. Qualitative tests proved that video can be streamed across the emulated network and that network effects, affect data streams in an expected manner. This proved that Mininet is qualitatively an accurate network emulator. Lastly, a hardware interface test proved that Mininet networks from separate devices can be connected to one another. This proved

that Mininet is a scalable network emulator.

[Recommendations] As it known that Mininet relies heavily on the host device's available resources, it will be recommended that Mininet is not used in a virtual environment - This will ensure accurate emulation of network effects, regardless of network size. If it is necessary to run Mininet in a virtual environment, the user should overcompensate for network effects by setting; bandwidth 50% higher than the desired value, network delay 2 ms faster than the desired delay and to keep jitter smaller than 4 ms.

[Further work] The GUI allows a user to build custom parameterized networks of any size, however many improvements can be made, in order to allow; more custom networks topologies, customizable host resources (CPU limited hosts), customization of switch- and controller type and further customization of controller protocol.
As jitter measurements were very inconsistent and inaccurate, there are some concerns that jitter tests perfomed in this project were not adequate. Further tests should therefore be conducted to accurately test Mininet's ability to emulate jitter.

Appendix A

Outcome compliance

Outcome	Outcome fulfilled
1. Problem Solving (identify, assess, formulate and solve convergent and divergent engineering problems)	Chapter 1, 3, 4, 5, 6
2. Application of Scientific and Engineering Knowledge	Chapter 2, 3, 4, 5, 6
3. Engineering Design (procedural and non-procedural design and synthesis of components, works, products and processes)	Chapter 1, 2, 3, 4, 5, 6
4. Investigations, experiments and data analysis (design and conduct investigations and experiments)	Chapter 2, 4, 5, 6, 8
5. Engineering Methods, Skills and Tools, including Information Technology (methods, skills and tools, including those based on information technology)	Chapter 2, 3, 4, 5, 6
6. Professional and Technical Communication (effective oral and written communication)	Oral presentation & Report
9. Independent learning ability (independent learning through well developed learning skills)	Chapter 2, 3, 5, 6

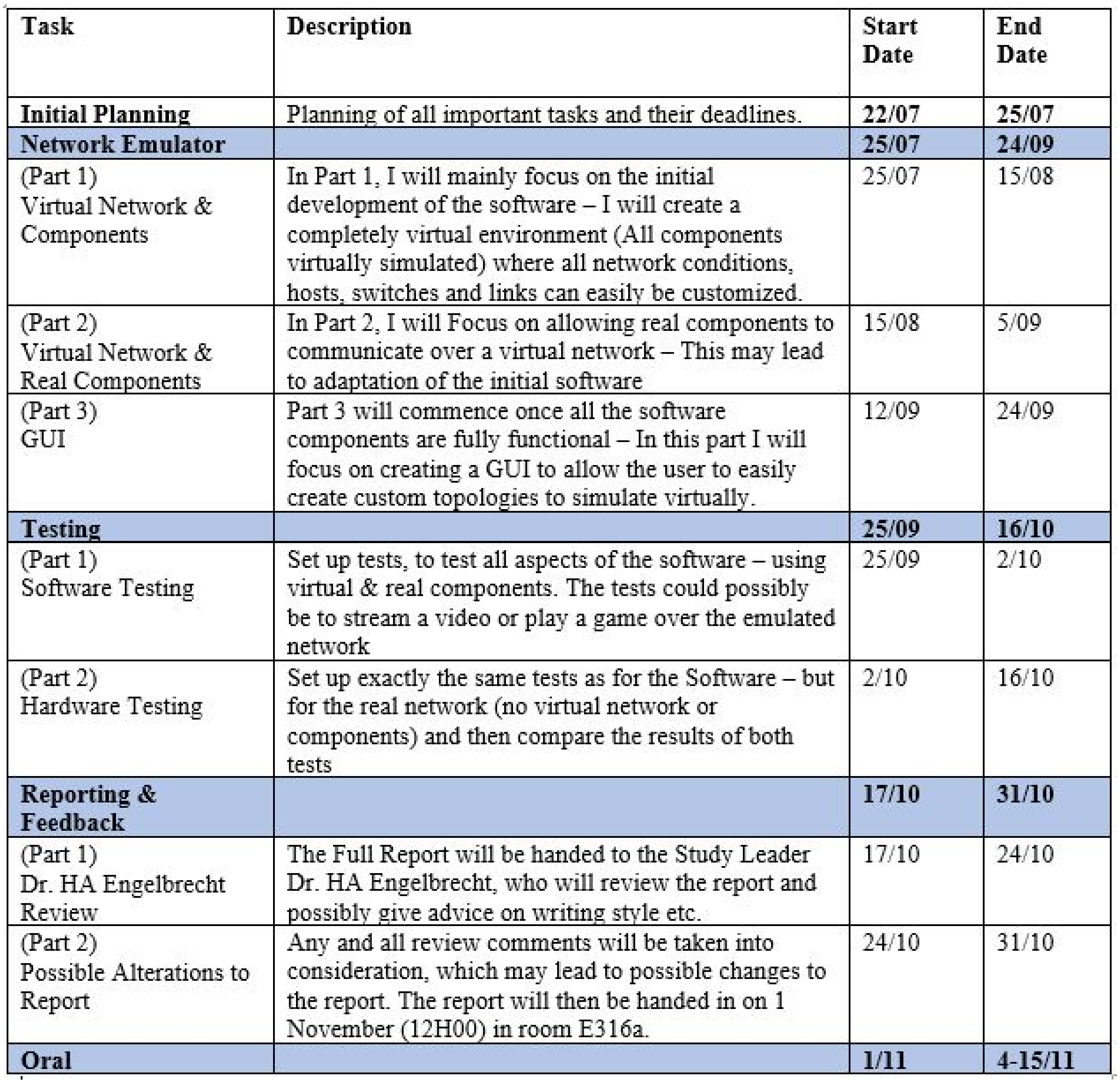
Appendix B

Project Planning Schedule

I will Split this project up into Five Main Categories:

1. Initial Planning
2. Network Emulator
    (a) Virtual Network & Virtual Hosts/Switches/etc. (Part 1)
    (b) Virtual Network & Real Hosts/Switches/etc. (Part 2)
    (c) GUI (Part 3)
3. Testing
    (a) Software Testing (Part 1)
    (b) Hardware Testing (Part 2)
4. Reporting & Feedback
    (a) Dr. H.A. Engelbrecht Review (Part 1)
    (b) Possible Alterations to Report (Part 2)
5. Oral

I will write the report section by section, Writing each sectional Report after completion of the specific Task.
All Tasks will be completed by me, IT de Villiers, close to - or on the indicated dates.
Scheduled Weekly Meeting time will be 09H00 on every Friday.



Task	**Description**	**Start Date**	**End Date**
Initial Planning	Planning of all important tasks and their deadlines.	**22/07**	**25/07**
Network Emulator		**25/07**	**24/09**
(Part 1) Virtual Network & Components	In Part 1, I will mainly focus on the initial development of the software – I will create a completely virtual environment (All components virtually simulated) where all network conditions, hosts, switches and links can easily be customized.	25/07	15/08
(Part 2) Virtual Network & Real Components	In Part 2, I will Focus on allowing real components to communicate over a virtual network – This may lead to adaptation of the initial software	15/08	5/09
(Part 3) GUI	Part 3 will commence once all the software components are fully functional – In this part I will focus on creating a GUI to allow the user to easily create custom topologies to simulate virtually.	12/09	24/09
Testing		**25/09**	**16/10**
(Part 1) Software Testing	Set up tests, to test all aspects of the software – using virtual & real components. The tests could possibly be to stream a video or play a game over the emulated network	25/09	2/10
(Part 2) Hardware Testing	Set up exactly the same tests as for the Software – but for the real network (no virtual network or components) and then compare the results of both tests	2/10	16/10
Reporting & Feedback		**17/10**	**31/10**
(Part 1) Dr. HA Engelbrecht Review	The Full Report will be handed to the Study Leader Dr. HA Engelbrecht, who will review the report and possibly give advice on writing style etc.	17/10	24/10
(Part 2) Possible Alterations to Report	Any and all review comments will be taken into consideration, which may lead to possible changes to the report. The report will then be handed in on 1 November (12H00) in room E316a.	24/10	31/10
Oral		**1/11**	**4-15/11**

Appendix C

GUI

C.1 GUI Error Table

Error no.	Description	Cause
Error 0	For a Single Switch Topology only one Switch is allowed	The Single Topology Radio Button is active, and the user tries to add more than one switch.
Error 1	Please add a Host(s)	User presses the Add Link button whilst no Hosts or Switches exist.
Error 2	Please add a Switch(es)	User presses the Add Link button whilst no Switches exist.
Error 3	Please select a Host(s) and switch to Connect	User Presses the Add Link button, whilst no Hosts or Switches are selected in the Listboxes.
Error 4	Host 'host' is unavailable	User tries to Create a link between a host and a switch, when the host is already connected to another switch.
Error 5	Link between 'host' and 'switch' already exists	User tries to make the same link twice
Error 6	Please Link all Hosts to a Switch	User presses the Complete Button, whilst not all hosts are connected to a switch.
Error 7	Please add a Link	User presses the Complete Button whilst no links between network entities exist.
Error 8	Host: 'host' already exists	User Tries to add a Host, with a Host Name that already exists (Add Host button pressed).

Table C.1.1: Table of possible GUI error massages

C.2 Parser.py function library

Function	**Description**
parse()	Create all Host, Switch python Objects
getLinks()	Create Links between hosts and switches
getHosts()	Return a list of all Host Objects
getSwitches()	Return a list of all Switch Objects
getTopo()	Return Topo type

Table C.2.1: Parser functions

C.3 Host.py function library

Function	**Description**
getBandwidth()	Return host's Bandwidth
getLoss()	Return host's Loss
getDelay()	Return host's Delay
getJitter()	Return host's Jitter
getHostName()	Return host's ID

Table C.3.1: Host Object functions

C.4 Switch.py function library

Function	**Description**
getBandwidth()	Return switch's Bandwidth
getLoss()	Return switch's Loss
getDelay()	Return switch's Delay
getJitter()	Return switch's Jitter
getswitchnamr()	Return switch's ID
getHost()	Return switch's Linked hosts

Table C.4.1: Switch Object functions

C.5 Link.py function library

Function	**Description**
getBandwidth()	Return Link's Bandwidth
getLoss()	Return Link's Loss
getDelay()	Return Link's Delay
getJitter()	Return Link's Jitter
getHost()	Return Host that need's to be linked
getSwitch()	Return Switch that need's to be linked

Table C.5.1: Link Object functions

Appendix D

Iperf test results

D.1 Bandwidth Tests

D.1.1 2 Host Network

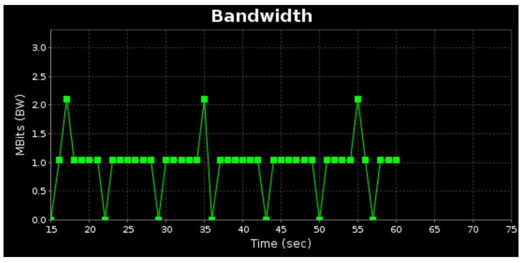


Figure D.1.1: Iperf results: 1 Mbps BW

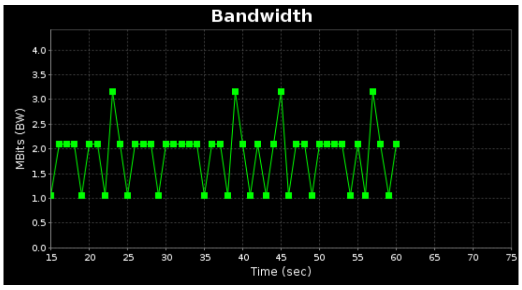


Figure D.1.2: Iperf results: 2 Mbps BW

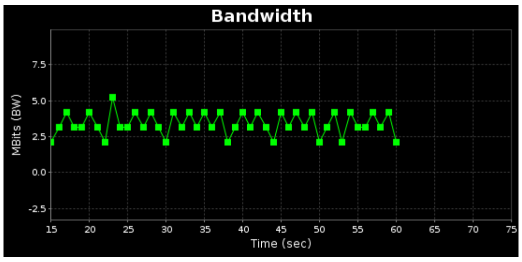


Figure D.1.3: Iperf results: 4 Mbps BW

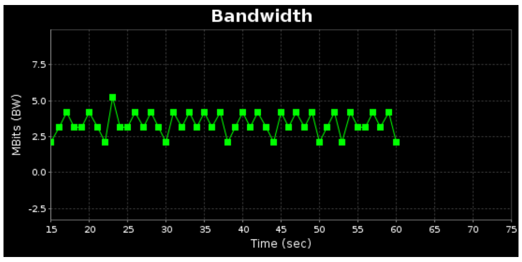


Figure D.1.4: Iperf results: 8 Mbps BW

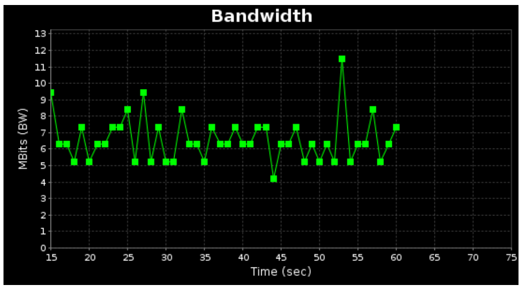


Figure D.1.5: Iperf results: 12 Mbps BW

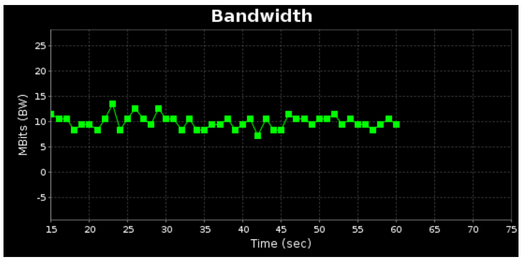


Figure D.1.6: Iperf results: 20 Mbps BW

D.1.2 4 Host Network

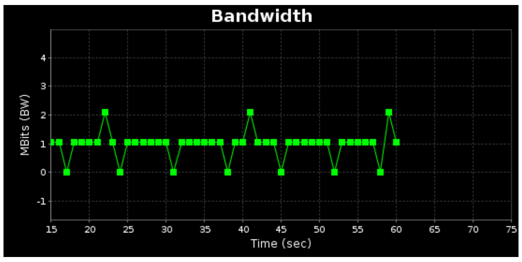


Figure D.1.7: Iperf results: 1 Mbps BW

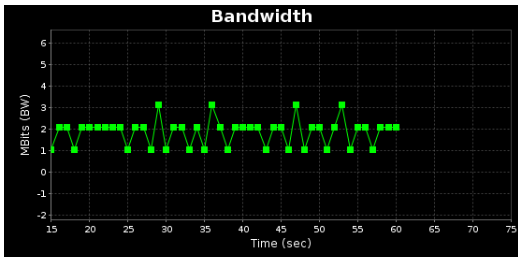


Figure D.1.8: Iperf results: 2 Mbps BW

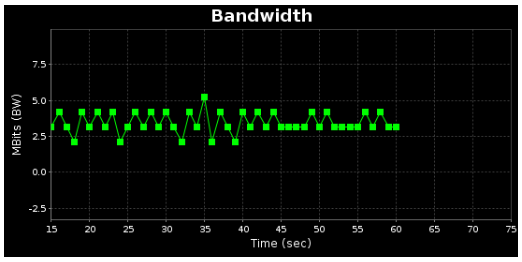


Figure D.1.9: Iperf results: 4 Mbps BW

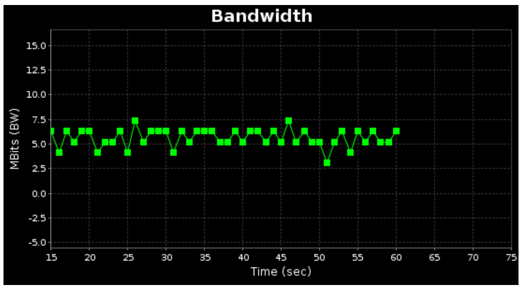


Figure D.1.10: Iperf results: 8 Mbps BW

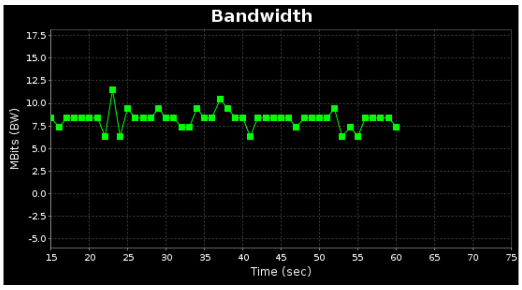


Figure D.1.11: Iperf results: 12 Mbps BW

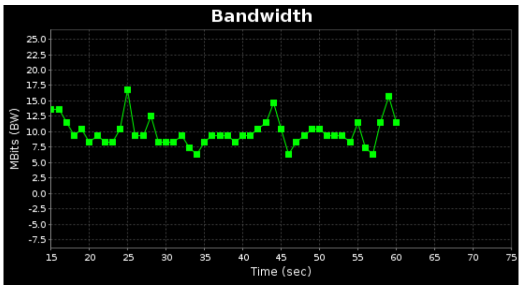


Figure D.1.12: Iperf results: 20 Mbps BW

D.1.3 8 Host Network

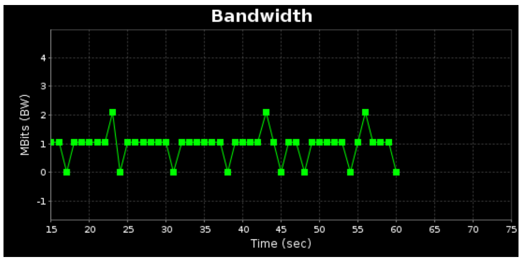


Figure D.1.13: Iperf results: 1 Mbps BW

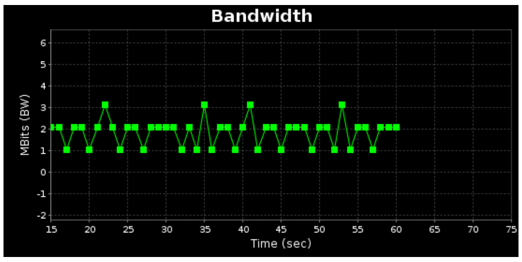


Figure D.1.14: Iperf results: 2 Mbps BW

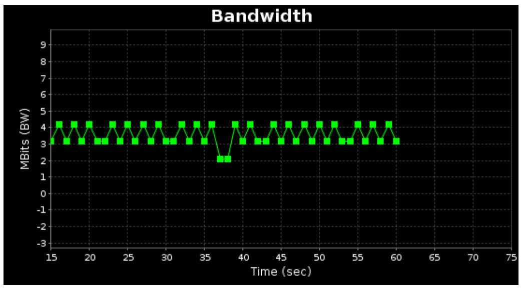


Figure D.1.15: Iperf results: 4 Mbps BW

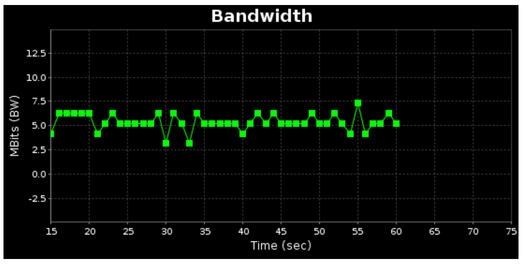


Figure D.1.16: Iperf results: 8 Mbps BW

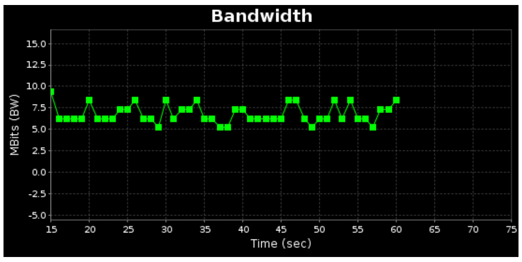


Figure D.1.17: Iperf results: 12 Mbps BW

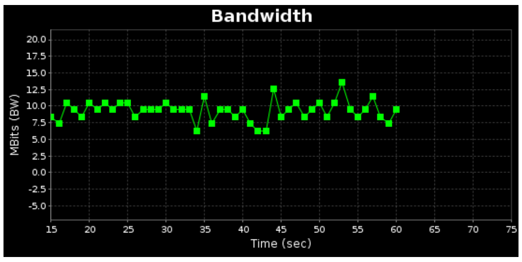


Figure D.1.18: Iperf results: 20 Mbps BW

D.1.4 15 Host Network

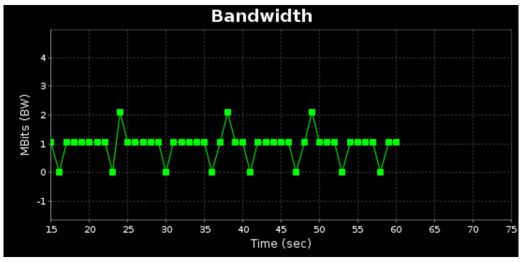


Figure D.1.19: Iperf results: 1 Mbps BW

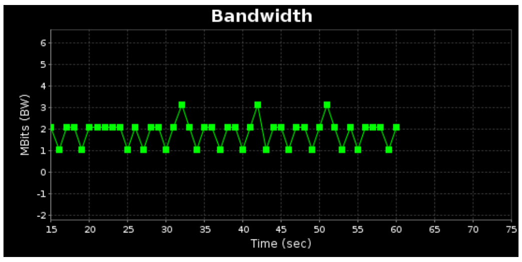


Figure D.1.20: Iperf results: 2 Mbps BW

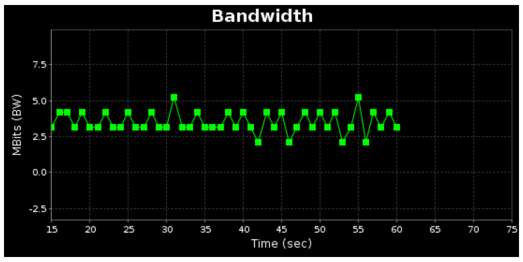


Figure D.1.21: Iperf results: 4 Mbps BW

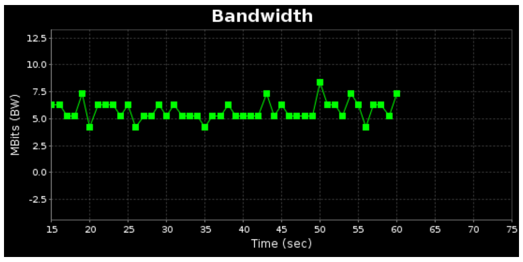


Figure D.1.22: Iperf results: 8 Mbps BW

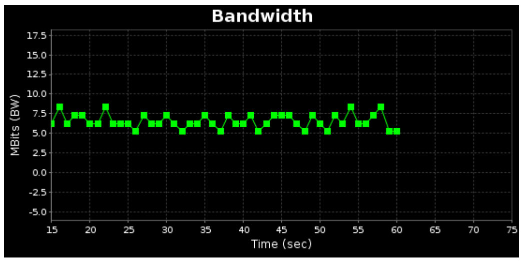


Figure D.1.23: Iperf results: 12 Mbps BW

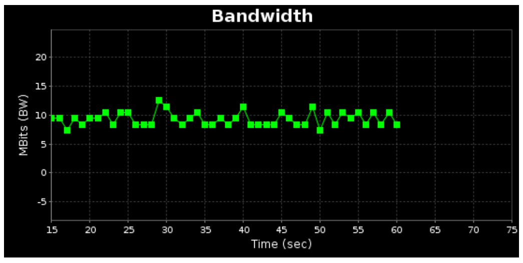
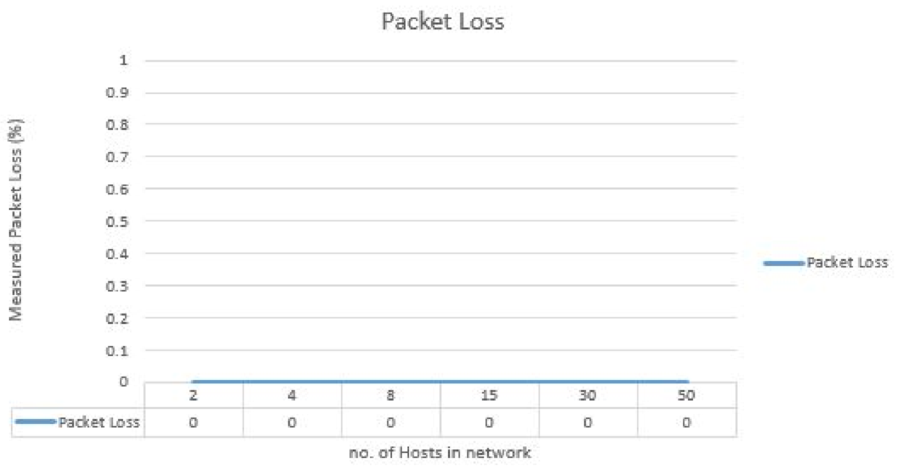


Figure D.1.24: Iperf results: 20 Mbps BW

D.2 Zero Value Tests

D.2.1 Packet Loss



	2	4	8	15	30	50
Packet Loss	0	0	0	0	0	0

Figure D.2.1: Packet Loss zero value test results

D.2.2 Delay

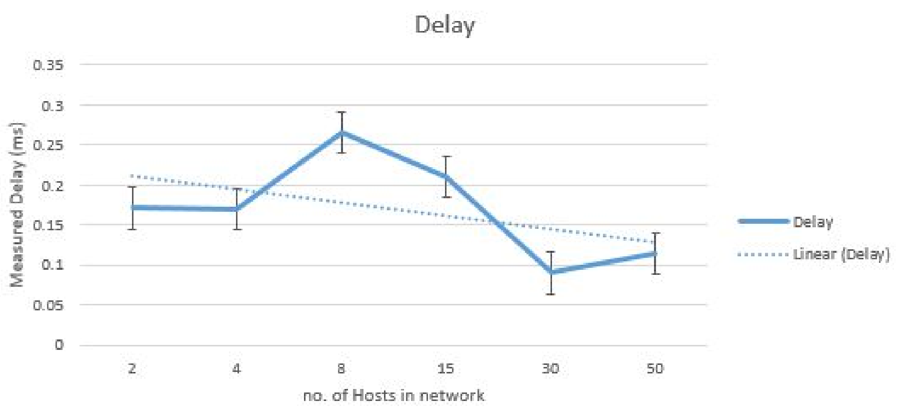
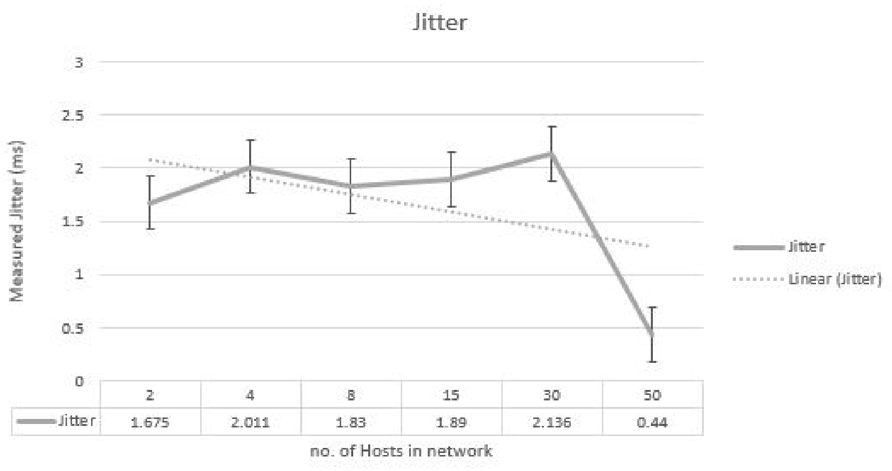


Figure D.2.2: Packet Loss zero value test results

D.2.3 Jitter



	2	4	8	15	30	50
Jitter	1.675	2.011	1.83	1.89	2.136	0.44

Figure D.2.3: Packet Loss zero value test results

D.3 Mininet Network vs. Real Network

Run no.	Bandwidth of Pi1	Bandwidth of Pi2
1	62.4 Mbps	55.5 Mbps
2	59 Mbps	65.4 Mbps
3	66.5 Mbps	65.3 Mbps
4	62.7 Mbps	64.8 Mbps
5	64.9 Mbps	51.8 Mbps
Average	63.1 Mbps	60.56 Mbps

Table D.3.1: Average of Bandwidth Test

Run no.	Delay of Pi1	Delay of Pi2
1	2.46 ms	2.35 ms
2	3.40 ms	1.74 ms
3	1.97 ms	3.40 ms
4	3.20 ms	1.52 ms
5	1.93 ms	1.81 ms
Average	2.60 ms	2.16 ms

Table D.3.2: Average of Delay Tests

Run no.	Jitter of Pi1	Jitter of Pi2
1	0.223 ms	0.22 ms
2	0.22 ms	0.25 ms
3	0.10 ms	0.15 ms
4	0.28 ms	0.21 ms
5	0.41 ms	0.47 ms
Average	0.25 ms	0.26 ms

Table D.3.3: Average of Delay Tests

List of References

Cisco (2006 February). Understanding jitter in packet voice networks (cisco ios platforms)@ONLINE.
Available at: http://www.cisco.com/c/en/us/support/docs/voice/voicequality/18902-jitterpacketvoice.html

Cisco.com (2011 June). Networking basics: What you need to know@ONLINE.
Available at: http://www.cisco.com/cisco/web/solutions/small_business/resource_center/articles/connect_employees_and_offices/networking_basics/index.html

Cranson, M. and Santay, D. (2003). Nist net a linux-based network emulation tool. pp. 1–2.

github.com (2012 October). Introduction to mininet by bob lantz, nikhil handigol, brandon heller and vimal jeyakumar@ONLINE.
Available at: https://github.com/mininet/mininet/wiki/IntroductiontoMininet

GNS3.com (2015 December). Frequently asked questions@ONLINE.
Available at: https://www.gns3.com/software

Handigol, N., Heller, B., Jeyakumar, V., Lantz, B. and McKeown, N. (2012 October). Mininet performance fidelity benchmarks@ONLINE.
Available at: http://hci.stanford.edu/cstr/reports/201202.pdf

Linkletter, B. (2016). List of network simulators and emulators@ONLINE.
Available at: http://www.brianlinkletter.com/opensourcenetworksimulators/

Marionnet.org (2007). Marionnet project@ONLINE.
Available at: https://www.marionnet.org/site/index.php/en/project

McNickle, M. (2014 August). Five sdn protocols other than openflow @ONLINE.
Available at: http://searchsdn.techtarget.com/news/2240227714/FiveSDNprotocols-otherthanOpenFlow

Mininet (2012 December). Mininet overview@ONLINE.
Available at: http://mininet.org/overview/

Openflow.org (2008 August). Openflow@ONLINE.
Available at: http://archive.openflow.org/wp/learnmore/

Opennetworking.org (2013). Software-defined networking sdn definition@ONLINE.
Available at: https://www.opennetworking.org/sdnresources/sdndefinition

packthub.com (2016 August). Gns3 limitations@ONLINE.
Available at: https://www.packtpub.com/mapt/book/Networkingand-Servers/9781782160809/7/ch07lvl1sec41/GNS3%20Limitations

Rouse, M. (2007 May). Packet loss@ONLINE.
Available at: `http://searchnetworking.techtarget.com/definition/packet-loss`

Rouse, M. (2012 June). Openflow@ONLINE.
Available at: `http://whatis.techtarget.com/definition/OpenFlow`

Rouse, M. (2014 August). Bandwidth@ONLINE.
Available at: `http://searchnetworking.techtarget.com/definition/packetloss`

Rouse, M. (2015 March*a*). Pox@ONLINE.
Available at: `http://searchsdn.techtarget.com/definition/softwaredefined-networkingSDN`

Rouse, M. (2015 August*b*). software-defined networking (sdn)@ONLINE.
Available at: `http://searchsdn.techtarget.com/definition/softwaredefined-networkingSDN`

Tyson, J. (). How network address translation works@ONLINE.
Available at: `http://computer.howstuffworks.com/nat.html`

WhatIsMyIPAddess.com (2006 July). What is network address translation@ONLINE.
Available at: `http://whatismyipaddress.com/nat`

www.juniper.net (2012 June). Understanding generic routing encapsulation@ONLINE.
Available at: `https://www.juniper.net/documentation/en`$_US/junos13.3/topics/concept/gre-tunnelservices.html$

www.nrl.navy.mil (2008 November). Common open research emulator (core)@ONLINE.
Available at: `http://www.nrl.navy.mil/itd/ncs/products/core`