AN ANALYSIS OF NETWORK INTERFACE CARD PROMISCUOUS MODE DETECTION METHODS IN A VIRTUAL NETWORK

A Thesis
Presented in Partial Fulfillment of the Requirements for the
Degree of Master of Science
with a
Major in Computer Science
in the
College of Graduate Studies at
University of Idaho

by

Lee D. VanGundy

July 2014

Major Professor: Jim Alves-Foss, Ph.D.

Authorization to Submit Thesis

This thesis of Lee VanGundy, submitted for the degree of Master of Science with a major in Computer Science and titled **"An Analysis of Network Interface Card Promiscuous Mode Detection Methods in a Virtual Network,"** has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor	______________________ Dr. Jim Alves-Foss	Date	________
Committee members	______________________ Dr. Paul Oman	Date	________
	______________________ Dr. Mark Nielsen	Date	________
Computer Science Department Administrator	______________________ Dr. Gregory Donohoe	Date	________
Discipline's College Dean, College of Engineering	______________________ Dr. Larry Stauffer	Date	________

Final Approval and Acceptance by the College of Graduate Studies

	______________________ Dr. Jie Chen	Date	________

Abstract

Promiscuous mode is a powerful tool for both attackers and system administrators. The detection of a network interface card running in promiscuous mode is also a powerful tool, which has broad implications in terms of usage and legal application. Previous research has discussed the use of several techniques for the detection of network interface cards running in promiscuous mode. This thesis analyzes the potential use of several of these methods in a virtualized environment. Two tests in particular, Round Trip Time and Load Detection Technique, are tested to see if their statistical approach is suited to a virtualized system and if the methods will work, given the somewhat centralized resources of a virtualized environment.

Acknowledgements

I would like to thank my advisor, Dr. Jim Alves-Foss for his support, encouragement, and wonderful edits.

I would also like to thank my committee members, Dr. Paul Oman and Dr. Mark Nielsen for their valuable input on my thesis.

I would like to thank all of my professors and teachers who have helped to impart the knowledge which has helped me along in my academic career.

I would also like to thank my friends and family who have been supportive of my endeavors.

Dedication

To my dear parents .....

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Network sniffing is a useful technique for many purposes; some of which are legal and others not. Sniffing is the process of capturing packets of data while they are in transmission [23] over some communication medium (e.g., ethernet network, wireless networks, USB connections). Sniffing over an ethernet network involves special use of the network interface card (NIC) in a device. A NIC has several different operational modes that can be used for a variety of tasks, such as: network sniffing, specialized routing, and normal network traffic communication. In order to sniff traffic on a network, a machine's NIC should be running in either promiscuous mode or monitor mode. The ability to see all of the traffic on a network has a variety of uses for both malicious and legitimate users.

For both malicious and legitimate uses of network sniffing it is optimal for the sniffing to be passive. A device that is passively sniffing is, in theory, not possible to detect. At present this is true of devices using monitor mode, but devices running promiscuous mode are able to be detected.

There are several existing methods for the detection of promiscuous mode. These detection methods take advantage of different issues related to using or implementing promiscuous mode drivers for NICs. Previous work has been done with two of these techniques in particular, Round Trip Time and Load Detection Technique, on a physical network [23]. These methods potentially behave differently when used in a virtual environment; since the hardware and software requirements are changed. This thesis focuses on the detection of promiscuous mode in a virtual environment, and looks at the performance of Round Trip Time and Load Detection Technique.

Some may say that the detection of promiscuous mode is not all that important in either physical or virtual networks because anything of value that could be sniffed out should be sent over encrypted connections, so let an attacker sniff all they want. However, users and user applications are not always mindful of this and either do not use or do not properly use network encryption or authentication protocols. Therefore, being able to detect promiscuous mode allows network administrators to help protect the information of their users and employers on the network as well as protect the network itself.

With the increasing number of virtual networks and virtual machines in use in the working environment [13], VMs are becoming a crucial part of modern networks. VMs do not all have the same amount of resources to function that a physical machine has. With this increase of VMs and VM networks system administrators and attackers should be aware whether or not techniques to detect machines sniffing traffic on their network will work in their environment, and how they will work differently than in a virtual environment.

Just having knowledge that a device was using promiscuous mode on network can also be useful for network forensics investigators and lawyers involved in related legal cases. The gathered information would provide more knowledge about how a network was compromised and who to prosecute. It would also help show which information on a network could have been compromised, which would be useful for both legal action as well as knowledge of what must be disclosed about the breach. Even though it does not, in most cases, give specifics as to what was actually compromised, it does help give insight as to the how and what was compromised.

The legal implications that result from the misuse of promiscuous mode are discussed, when dealing with physical and virtual machines, as the issues are applicable to both. When promiscuous mode is not used properly by system administrators and forensic investigators leads to unsuccessful court cases. Also, the successful use promiscuous mode can lead to successful prosecution as in the New Zealand hacking incident discussed later in this thesis.

1.1 Goals of This Thesis

This thesis focuses on the detection of promiscuous mode in a virtual environment. Detecting the use of promiscuous mode in a virtual environment is important because of the increased use of virtual machines and virtual networks. This thesis fills the need for evaluating whether promiscuous mode detection mechanisms work in a virtual environment, and determining any changes that may need to be made to those detection mechanisms to allow them to work in a virtual environment.

1.2 Thesis Organization

This thesis is organized as follows. Chapter 2 focuses on the background of how promiscuous mode functions, how it is used, and details several detection methods. The set up of the experiment in this thesis is discussed in chapter 3, with results and analysis of what those results mean in chapter 4. Lastly, chapter 5 presents the conclusion of this thesis and lists possible avenues for future work stemming from the results.

Chapter 2
Background

2.1 Network Interface Cards

Network interface cards (NIC) are the physical devices that allow network capable devices to communicate over a specified network medium. Each NIC has a Media Access Code (MAC), which is meant to be unique to that specific NIC. This, along with an Internet Protocol (IP) address, allows for network devices to identify where to route network traffic for proper delivery. On the receiving end a NIC normally only looks for network traffic directed to it, so that processing software on the device is not flooded with all of the network traffic from the communication medium. If NICs did not do this, the device's network communication speed would slow down.

NICs have several different execution modes; though only one mode can be active at a time. These modes allow for different operation from the perspective of traffic handled and the administration of the device. Two of these that are of interest in this thesis are promiscuous and monitor mode, which allow for the capture of packets not destined for the host NIC. The other modes are *Master, Managed, Ad-Hoc, Mesh,* and *Repeater* [25]. A NIC can be placed in these other modes, but they do not allow for packet capture in a manner that is of interest to this thesis.

2.1.1 NIC Execution Modes

This section gives a brief explanation of the execution modes of a NIC. This is so their individual uses can be understood, and specifically why they are or are not of interest to the rest of this thesis.

2.1.1.1 Master

Master mode allows a device to run as a core networking device such as a switch, hub, or wireless base station. When a device wants to connect to the network it is done through a device running in master mode.

2.1.1.2 Managed

Most user machines run in managed mode. It is called managed mode because it is a constrained mode meant just for use among users, and not network devices. This allows them to connect to a network under normal circumstances. This can be through a wired network or through a wireless base station depending on the NIC being used.

2.1.1.3 Ad-Hoc

Ad-Hoc mode for a wireless NIC allows for the connection or creation of an Ad-Hoc network. These allow for the easy creation of a network for non-permanent use without a dedicated base station [22]. An Ad-Hoc network can be created using any NIC by changing the execution mode without using a base station which executes in Master mode.

2.1.1.4 Mesh

A wireless NIC running in mesh mode allows the device to connect to and interact with other devices in a mesh network. This allows for the device to function as both a member of the network and be part of the network infrastructure [4]. Network sniffing on such networks is a potential issue, but not a part of this thesis since the methods for detecting the device that is sniffing are different and not as applicable to the virtual environment.

2.1.1.5 Repeater

A device running in repeater mode has been set up for the purpose of extending the range of a wireless network. This is done through receiving and then retransmitting a signal [4]. Devices set up as repeaters are considered to be part of the network infrastructure, and are typically only used for wireless NICs.

2.1.1.6 Promiscuous Mode

Normally a NIC runs in managed mode. In managed mode a NIC will only handle traffic with the correct MAC and IP addresses for the device that the NIC is attached to. Running a NIC in promiscuous mode allows for the capture of all traffic on the network that can be seen by the machine. This means that any traffic that a NIC can see is no longer ignored by the NIC. While capturing packets in promiscuous mode a NIC passes all of the captured packets to the operating system to handle. These are then handled by a software device driver used by an application running on the device [23].

In theory a device in promiscuous mode is undetectable; since the network device should only reply to packets directed to the network device. However, this is not the case in practice, partially due to the way promiscuous mode has been implemented, and due to the temporal realities of processing time. The specifics will be discussed later in this chapter. Promiscuous mode has various uses depending on the network type and can be used both by malicious and non-malicious parties [23].

2.1.1.7 Monitor Mode

Monitor mode is similar to promiscuous mode, but it is only available on wireless NICs. It allows a user to sniff all network traffic in the wireless spectrum without first being associated with a wireless network, which cannot be done on a wired network. In general, a typical association would be where a computer that is authenticated with an access point has an assigned IP and can bi-directionally communicate with other devices on the network. With no association to a network there is no standard way to detect this type of wireless sniffing, as the machine doing the sniffing is completely passive. As a result the access point has no way of eliciting a response [25].

2.2 Network Types

This section describes the different network types and how promiscuous mode can be used and to what effectiveness. This will give an insight into how some of the detection methods discussed in this thesis work with each type of network. Of particular interest are different wired networks, wireless networks and virtual networks, since each of these has different implications with respect to network sniffing.

2.2.1 Wired Networks

This section discusses two different types of wired networks topologies: hub/bus and switched network topologies. Also, it should be noted that the wired networks of interest for this thesis use the IEEE 802.3 (ethernet) standard, though they are not restricted to this standard. Along with looking at how these wired networks are structured, this thesis also discusses how the different network types can be sniffed through the use of promiscuous mode and various other techniques [21].

2.2.1.1 Hub/Bus Network

A hub/bus based network is a broadcast network where anyone on the network has the potential to see the network traffic of others (Fig. 2.1). For the purpose of this thesis, the only difference that should be noted between a hub and a bus is purely the implementation; they still both

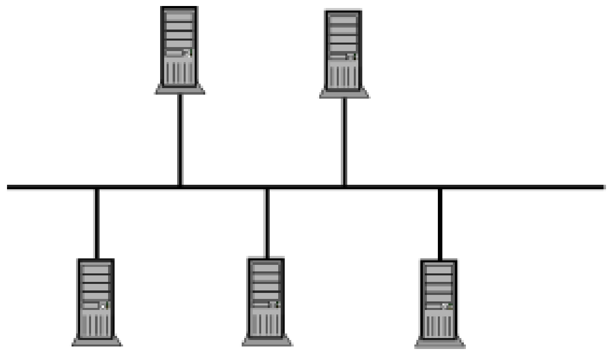


Figure 2.1: Example of a Bus Network

perform in the manner of a broadcast network. A NIC that is running in normal mode will not see all of the traffic that is broadcast on the network; any excess packets, which do not match the device's IP and MAC addresses will be dropped. However, a computer with a NIC running in promiscuous mode will pass all of the network traffic through to be processed by the operating system kernel instead. This allows a user, malicious or otherwise, to sniff all of the traffic that goes over the network through the use of software on that machine [21].

2.2.1.2 Switched Network

On a switched network, unlike a hub or bus, network traffic is isolated between connections (Fig. 2.2). This means that if machine A and machine B are communicating with each other over a switched network then machine C cannot see this network traffic. The isolated communication means that an individual computer cannot openly sniff the network traffic [21].

This, however, does not mean that network traffic cannot be sniffed on a switched network. One way that network traffic can be sniffed is to set up tap-port [12]. With this setup, the switch will forward all of the network traffic to the tap-port as well as the originally intended recipient. This is typically done by a network administrator.

A machine, which is not inside the network, can sniff all of the external network communication at the gateway (connection from the switch to the larger network cloud). Any network communication that does not pass through the gateway, however, will be missed by this method.

Other than through the use of a tap-port a machine inside a network is not able to sniff network traffic over a switched network without some manipulation of the switch. A switched network can be manipulated by a malicious user in order to sniff traffic over the switch through

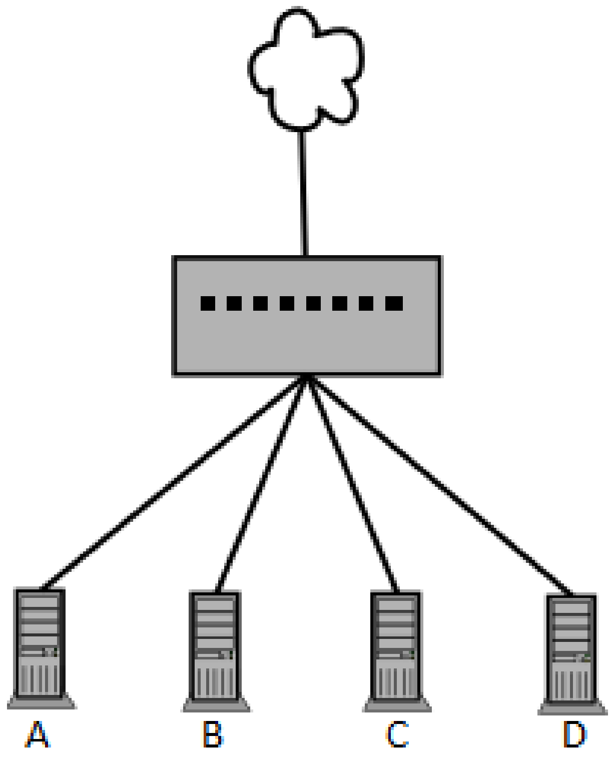


Figure 2.2: Example of a Switched Network

the poisoning of the Address Resolution Protocol (ARP) cache. This will be discussed in more detail later in this chapter, but essentially allows for the rerouting of network traffic through the router. This rerouting then allows for the attacking device to create its version of a tap-port [17]. ARP poisoning is often used for the man-in-the-middle style attacks, but if the machine can still be detected it does not affect the dection methods discussed later in this thesis.

2.2.2 Wireless Networks

Wireless networks can be based off of a number of protocols in the IEEE 802.11 specification and can include a variety of devices. Each of these different types of network traffic can be sniffed, as wireless communication protocols are a broadcast type network just like hubs and buses. The proper hardware is still required to be able to sniff traffic using a specific wireless network protocol, but it is doable. Wireless network traffic can also be sniffed using monitor mode. This allows a device to not be associated with a network and still be able to sniff all wireless traffic within the spectrum of the card (802.11x) [21].

2.2.2.1 Base Station

A wireless NIC operating as a base station fulfills the role analagous to a hub on a wired network, and not a switch because of the way that network traffic is just broadcast into the air. An administrator, or an attacker who has compromised the device, can reroute network traffic to be sniffed by another machine. This is the standard setup of most wireless networks.

2.2.2.2 Ad-Hoc and Mesh

An Ad-Hoc network is a wireless network which can be created using a normal wireless NIC and running it in Ad-Hoc mode so that it can act as the base station. The overall network structure can be less rigid and it can be created anywhere a simple network connection is desired. A mesh network is flexible in shape and uses the nodes of the network to form the connections, which can change over time if the nodes are physical devices which move around. As a result the network topography of a mesh network can change drastically over time.

Sniffing can also take place on Ad-Hoc and Mesh networks. This happens with the node running in promiscuous mode, but instead of being completely passive it interacts with the other nodes to pass their traffic along when appropriate [22]. The information sniffed over a mesh network can vary in which information from which node is gathered due to the topology changes.

2.2.3 Virtual Networks

In this thesis a virtual network is a network that is set up for the purpose of a connecting virtual machines, so that all of the machines as well as the network are virtualized. Virtual networks are modeled off of physical networks. Each virtual machine has a virtual NIC, and there is virtual networking hardware, specifically switches, to enable network communication. Because the implementation is modeled off of physically networks, networks in virtual environments can be set up to model any physical wired network. Besides being useful for communication between virtual machines, the ability to model physically wired networks allows for research when physical resources are limited [6].

With the current virtualization capabilities that are in use today, a true virtual model of a wireless network and protocols is not currently possible because the protocols have yet to be defined in a virtual environment. This is something that could be done in the virtualization software or potentially as an implementation within a virtual environment of the protocols [6]. Although a wireless network behaves similar to a hub, each machine on the network is not directly linked by a networking cable. This is an added level of complexity that has yet to be investigated [6].

The interest in this thesis is not only the virtualization of the physical network but also examination of the implementation of the virtual NIC. This is because this is one of the areas where the software implementation of the hardware will not always perform the same. This can then lead to different response times in some of the detection methods that will be discussed and then analyzed later. The longer response time could be due to several aspects of a virtual environment. This thesis aims to see if the detection methods produce better or worse results when used in the virtual environment.

2.3 Network Packets

There are many types of network packets depending on the protocol used. Of interest in this thesis are protocols built on top of the Internet Protocol Version 4 (IPv4) [15]. Using the concepts of the Open Systems Interconnection (OSI) Model, IPv4 works at the network layer (or layer 3 of the OSI stack) [26]. The OSI model allows for the easy wrapping of protocols upon a base layer, which for the purposes of the work in this thesis is IPv4. Internet Protocol Version 6 (IPv6) could be used instead of IPv4, and the work in this thesis would still be consistent.

Each layer within the OSI model is devoted to a different aspect of transmission of data over a network. The layers of most interest to us are the application layer and the network layer. This is because of the protocols contained within these layers. The application layer is used with protocols such as Domain Name System (DNS), and Hyper Text Transfer Protocol (HTTP). The network layer contains IPv4, Internet Protocol Version 6 (IPv6), and Internet Control Message Protocol (ICMP) [15] [26].

2.4 Uses of Promiscuous Mode

There are many different uses for promiscuous mode as well as the detection of promiscuous mode. Some of these uses are useful for system administrators and forensics investigators, and some are useful for malicious users with malicious intent. This section discuss these uses. Understanding these applications allows for a better understanding of our need to detect promiscuous mode.

2.4.1 Malicious Uses

The main malicious use of network sniffers is for the purposes of gathering information. Often this means that a malicious user is sniffing for usernames and passwords. With this information they can go on to compromise more systems [24]. A malicious user could also be using promiscuous mode for the purpose of some type of man in the middle or replay attacks. A man-in-th-middle attack is where a user's computer is able to either intercept all communication in the middle of a communication stream. A replay attack involves the use of previously intercepted messages at a later time [21].

A malicious user can also attempt to detect machines on a network that are using promiscuous mode. From an attacker's perspective this is a device that has possibly been compromised by another attacker or is a machine that has been set up by a network administrator to log network traffic. This gives information to the attacker to help them choose which attacks they could employ as well as how to prioritize their attacks.

2.4.2 Non-malicious Uses

For an authentic user there are a couple of valid uses of promiscuous mode to sniff traffic. First is for the purpose of logging network traffic with an Intrusion Detection System (IDS) or an archiving system. Penetration testing is another use, which is the sanctioned attempt of a user to successfully attack a system or network. For forensics purposes, promiscuous mode primarily supports using IDS logs as a way of gathering information for what happened on the network. This is often done through the use of a tap-port [12]. Also, a device can be set up specifically for the purpose of network forensics gathering; a topic discussed in Section 2.5.

For a network IDS or another archiving system to be effective, it must have access to all network traffic. On a switched network, as many are today, a tap-port is needed. This allows all network traffic to be forwarded to a monitoring device. This device can then log various information, including the network traffic, if its NIC is running in promiscuous mode. That device can log information for a predetermined time based on physical storage limits or policy. This information can then be given to forensic examiners by providing a snapshot of the system at the time of interest; this is assuming that the network data is still located on the device due to time frame [12].

Using an IDS may prove more appealing to some users than a separate dedicated archiving system because it allows the system to serve more than one purpose. For some applications, space, time, or money (etc.) may be a concern, but also of interest is the potential load on the computer running both the IDS and logging software. That is, overloading network hardware is a major concern [12].

Note that just because a tap-port has been set up, this does not mean that the switch will be able to log all of the network traffic. There is the possibility that the network device can be oversubscribed and as a result packets will be dropped. Under these circumstances the device is not comprehensive in its monitoring, and therefore may not hold up in court [12]. Failure to understand where the threshold is for these devices is potentially detrimental to the validity of evidence gathering in any legal proceedings, and therefore lessens the usefulness of setting up a device in promiscuous mode for the purpose of forensics, especially if the end goal is prosecution. This is further discussed in Section 2.5.2.

2.4.2.1 Penetration testing

Penetration testing is the sanctioned attempt to break into a network. A penetration tester acts like an attacker, using promiscuous mode first for surveillance and then to enable replay attacks. The results of a penetration test give knowledge of what improvements can be made so that a future attacker cannot replicate the attack, part of which could have involved the use of sniffing software. Also, knowledge of previous penetration testing results can be used to aid a network forensics investigator by giving knowledge of previous successful intrusion scenarios.

2.5 Legal Applications

This section discusses some of the preparation of network devices for the purpose of forensics and how they relate to court cases. It is specifically discussed how these devices can be useful for court, what kinds of information they should collect, and what needs to be done for network devices so that the information that they collect will hold up in court. The section also provides and example of how network forensic information gathered over promiscuous mode was either misused or properly used in court cases.

2.5.1 Legal Precedence

The laws surrounding the use of promiscuous mode for wireless networks are not entirely well founded, and much of it is based on case law. The dispute that people are arguing is whether or not sniffing wireless networks is considered wiretapping. Two recent cases have more clearly found it on the side of wiretapping instead of being viewed as open electronic communication.

The cases referenced for the purposes of this thesis are the cases involving Google and their street car mapping program [1]. Along with taking pictures for their Street View purposes for Google Maps, they were also capturing wireless network traffic data. This was for the purpose of mapping out wireless network locations. Their argument was wireless communication is available to everyone just like radio, since it too is a radio based communication [8]. However, the courts have ruled twice that although people might be able to connect to their neighbor's wireless it does not mean that they are considered free and open to everyone. This is because "data transmitted over a Wi-Fi network is not an 'electronic communication' that is 'readily accessible to the general public' [1]." As a result, the sniffing of wireless networks is, at present, considered to be wiretapping and illegal [1].

These rulings have helped to clearly lay out that the use of promiscuous mode to gather information on a wireless network without permission from the network administrator is not considered legal in the eyes of the courts. This includes, but is not limited to, internet cafes, workplace internet, and neighborhood wireless access points.

2.5.2 Network Devices

Network devices, used strictly for the purpose of gathering information about the network, need to gather meaningful data in a way that is verifiable. When purposefully logging information for forensic analysis it is useful to have a device that is specifically designed for that purpose. Therefore, the forensic data gathering will not comprise system production functionality [11].

The gathering of information is conducted through the use of promiscuous mode, which as mentioned before, allows for the capture of all traffic on the network. This can mean, depending on the network, that there is a lot of data being logged, especially if everything being logged is just a raw dump of the network traffic. This could amount to gigabytes of data in a rather short amount of time and quickly becomes impractical.

To make long-term storage practical, the solution is to be selective about which information will be saved. This can be accomplished through the careful selection of certain kinds of traffic, as well as by ignoring traffic that can be deemed as noise. Whichever method is chosen, both will require the use of tools for the purpose of pattern matching. Tools such as the UNIX utilities `strings` and `grep` can be used for this purpose [11]. A separate IDS could also be used for this purpose, through selection of special pattern matching rules.

In addition to determining which data to store, there is also a concern about keeping that data secure. The most common malicious use of network sniffing, as mentioned before, is for the purpose of finding username and password information. If an attacker were to discover the data that is being stored on a forensics device, their job has just been made easier. There are several steps that can be taken to avoid this. The first step is to make sure that the monitoring device is not discovered. This can best be done through having the device only passively listen to network traffic. Also, if the device is listening via a tap-port on a switch the switch can be used to hide the existence of the device. In the event of detection and/or compromise of the device, the stored data should be encrypted so that only certain people can use it [12] [11].

Another concern about the network device is that it functions properly. The concern here is that we need to verify whether or not the device successfully gathers all of the network traffic which it was supposed to. This requires that we determine the threshold related to how much network traffic can be handled before the device starts dropping packets. If there is more network traffic than the device (e.g., a switch with a tap-port) can manage, it will drop packets, and as a result the analysis will be missing information. This then leaves doubt about the missing data that can be brought forward in legal proceedings. It can be claimed that any device, which can be shown to be functioning below its threshold of network traffic, for the purpose of legal proceedings, captured all of the information of concern [12].

2.5.3 Preparing for Court

Due to the still relative newness of computer forensics, more specifically network forensics, to the justice system, there still remain a large number of cases that either are dismissed or do not come to trial. This is partially due to developing standards, and also due to juries and judges who lack education in the field of computer forensics, making it easier for lawyers to cause doubt where there should not be cause for it [12].

There are two things that are needed in order to not have a case thrown out. First, if a device is specifically being used for monitoring, then the thresholds of the device should be known to help eliminate this as a source of potential doubt [12]. Second, forensic investigators should be experts in their field, and the attorneys should have some fundamental knowledge of the field so that they can discern whether or not an "expert" witness is really an expert [16]. Knowledgeable forensics investigators and attorneys allow for better preparation of information for trial and general examination of the evidence.

2.5.4 Example Case: New Zealand

The case that will be used as an example of good promiscuous mode usage, involves an incident in New Zealand in 1998 where script kiddies, (attackers who just use scripts or programs that are well known and were developed by someone else) were able to cause about $400,000 in damages. There was a machine on the network for the purpose of sniffing the network internally. By analyzing the sniffer log files, the investigators were able to find backdoors and sniffers that had been installed by the script kiddies, what files had been copied, and what scans of the systems had been performed [12].

With the information gathered, investigators were able to track the malicious hacker's origins to several locations, including New Zealand and the United States. The parties that were eventually named as the attackers were convicted in New Zealand and sentenced to community service [12].

At that time in New Zealand there were no cyber crime laws and the perpetrators were tried under a law that was related under legal theory [12]. This was an instance where technology had outpaced the law. The ruling of this case became case law, which served to influence the future passage of cyber crime laws [12]. Because the technology involved in a court case is not always understood by the judicial system as a whole, computer forensics investigations should be prepared with caution and in high detail.

The proper use of an archiving system on the network to log network traffic is what made this case a success. Also seen in this case was the use of sniffers by the attackers. Although there was no real time detection of the attacker using promiscuous mode, investigators were still able to prove that the attackers used sniffers, which helped narrow down how they got some of their information to further compromise the system. Promiscuous mode, though useful by both sides, is a powerful tool for forensics investigators, but only if the system has been properly set up ahead of time.

2.6 Detection Methods

Being able to detect promiscuous mode has advantages for both system administrators and attackers. Knowledge of a promiscuous machine on a network can lead to further investigation of a possible breach. Also, knowledge of a non-authorized machine running in promiscuous mode can be used in any legal proceedings that occurs related to an investigation of that network. An attacker can use such information to more subtly compromise machines on a network, and evade detection.

Prior to 1998 the primary method for trying to figure out if some entity was sniffing network traffic would be to create fake network traffic. This fake network traffic would contain bogus usernames and passwords. Then, through monitoring of login attempts, the use of the bogus usernames can be observed [24]. This method is still used, but it is no longer the only method. The rest of this section discusses the some of the newer methods, Round Trip Time (RTT) and Load Detection Technique (LDT), which are the detection methods of most concern in this thesis.

2.6.1 Forged ARP Requests

One detection mechanism uses the broadcast of forged Address Resolution Protocol (ARP) requests. The success of this technique depends on how different operating systems handle the network traffic received from a NIC running in promiscuous mode. The standard broadcast address is *FF:FF:FF:FF:FF:FF*. All devices should respond upon receipt of an ARP broadcast in order to be recognized by the network. Whether a device responds to a broadcast or not in promiscuous mode depends on the promiscuous mode driver and the operating system.

If a device is running in managed (normal) mode the NIC will forward the request with an address of *FF:FF:FF:FF:FF:FF* to the OS network driver. However, a NIC in promiscuous mode passes all of the packets to the OS. Operating system kernels and their drivers do not always compare all of the bits in the MAC address when checking for the broadcast address. Using an address like, *FF:FF:FF:FF:FF:00*, would illicit a response just like a true broadcast address *FF:FF:FF:FF:FF:FF* would, because only the first several bits are checked. This allows for the detection of promiscuous mode being used on a machine's NIC, since the targeted operating system will respond as it would for a broadcasted ARP request, when it should not [23].

2.6.2 Round Trip Time

Another method, Round Trip Time (RTT), can be used with several different types of network packets. The easiest way is to make use of an ICMP packet to initiate a ping. When the response is received, the time that the packet took to travel is set as the round trip time. To use RTT detection a control set of data must first be gathered. This allows for the gathering of a test set of data. The gathered test set of data can be used to compare to the control set in order to determine with the use of a Z-test whether or not the machine's NIC is running in promiscuous mode [23]. RTT works because a NIC in Promiscuous mode passes packets to the operating system kernel for processing. With a machine processing all of a network traffic and all of that traffic processed by the OS kernel, the round trip time will be longer than for a machine that is running in managed mode.

The difference in time is then measured by the use of the Z-test from statistics. This means finding both the average as well as the standard deviation of both data sets. Once this is done the Z-Statistic can be calculated by

$$Z = \frac{\bar{x}_2 - \bar{x}_1}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

where Z is the Z-Statistic, x_1 and x_2 are the averages of their respective data sets and S_1 and S_2 are their standard deviations. The value of the Z-Statistic is then used for the Z-test. When comparing the two data sets the Z-Statistic is used in order to determine the level of confidence that the data sets are two different data sets. For the purposes of this thesis, a Z-Statistic of 2.36 is the base point, which means that we can be 99% sure that a machine's NIC is running in promiscuous mode [23]. So, any Z-Statistic larger than 2.36 is sufficient to say that the machine is running in promiscuous mode.

Looking for a confidence level of 99% may be too high for many reasonable predictions. However, the use of this level helps to eliminate false positives that could occur at lower confidence levels. This does not prevent false positives, but reduces the likelihood of seeing one, when the possible population is smaller than it would be if a confidence level of 90% was used. The drawback is that a false negative is more likely to occur, but this can be reduced through knowledge of the system's load bearing capabilities as discussed in Section 2.4.2 and Section 4.2.

2.6.3 Load Detection Technique

The Load Detection Technique (LDT), is based off of RTT. The main difference is the amount of network traffic that is required. RTT works best when there is little to no other network traffic, in order to minimize the impact of network use between control and test sets. LDT aims to have a high network load for both control and test data sets. The best scenario is a network load that approaches the limits of the network, such that a DoS attack is just avoided internally [23]. The use of the Z-test is still the same for LDT as RTT.

Although pushing so much network traffic over the system with LDT is a tricky balance, it does have the potential to produce better results. Again, this is due to the NIC passing network packets to the OS kernel. On a physical network the main point of interest is the networking hardware such as routers and switches. If the networking hardware is dropping packets then communication over the network becomes unreliable [12]. Trying to measure response time when not all of the packets are received on both ends means that reliable results can not be gathered since there will be holes in any data gathered. In a virtualized environment the goal is to not overload the servers you are running on. If the server(s) that the virtual environment is running atop of are not able to handle all of the network traffic the control and test data sets will look too similar to draw any useful conclusions.

2.6.4 DNS Lookup

Another method of detection relies on the choices made by the developers and users of sniffing software. Original drivers, for the purposes of sniffing traffic, made use of DNS lookup to identify machines. Also, newer applications often give the option to use DNS to resolve host names [3]. This, however, can be used to detect the use of a sniffer in promiscuous mode.

If a machine is sniffing traffic and making use of DNS hostname resolution, it then sends out specific packets to the DNS server. This can be detected through one of two ways. The first involves being in control of network's domain controller. A system administrator can then craft packets to be sent out to machines which do not exist. As a result, only a machine running in promiscuous mode and sniffing traffic will attempt to perform DNS hostname resolution on the bogus machine addresses, and the domain controller can be configured to detect this. The second method also makes use of crafted packets to non-existent machines. This involves the sniffing of network traffic by the system administrator. Traffic is analyzed for packets that are performing DNS hostname resolution for those non-existent machines [24].

Of the two methods, determining which one is better is dependent on the scale of the network. For large networks it would be easier to do checking with use of the domain controller because it would be dealing with much less network traffic. On smaller scale networks it could be beneficial to use the second method, especially if there is no domain controller, and if there is less network traffic. Since there would be less DNS lookups, in general, it would be harder to try and piece a pattern together; a problem which is solved by using the second approach. The second method is also more useful for more aggressive searches in any sized network.

2.7 Prevention

Although this thesis focuses upon detection, it should also be noted that there are some ways to prevent an attacker from successfully using promiscuous mode to sniff over a network. These methods cannot provide absolute prevention of network sniffing, as that is not possible; however, these methods attempt to minimize the likelihood of malicious sniffing on a network. The first measure is to use a switched network. Although a switched network does not provide complete protection due to the potential of an ARP poisoning attack (See 2.6.2), it does aid in blocking sniffing attempts and ARP poisoning attacks can potentially be detected [17].

Another more extreme measure is to try and confuse the attacker by sending random information containing fake usernames and passwords. This serves to mislead an attacker with bad information as it gives more information for an attacker to sift through. Also with regards to detection, an added benefit is that when the network administrator detects the use of a bad username or password, they can then conclude that someone is sniffing network traffic. To successfully do this, a network administrator must know the limits of extraneous traffic that can be sent over the network without impacting performance, as it can negatively affect the network bandwidth, depending on the size and usage scenarios [22].

Adding extraneous network traffic also aids statistical detection methods. This is because the increased traffic will give more traffic for a computer running in promiscuous mode to handle. This then leads to increased processing time and then will lead to an increased response time to ICMP pings or ARP requests as either can be used by LDT, which results in being further outside of the allowable standard deviations before arising suspicion of sniffing.

Prevention is also possible in an Ad-Hoc network through the use of a method such as that proposed by Su and Yeh [22]. They proposed that the nodes of an Ad-Hoc network could be used to detect on of their nodes that was sniffing traffic on the network. Their protocol uses neighboring nodes to test each other using a method similar to RTT then through the use of the neighboring nodes, isolates the malicious node from the rest of the network. Such a detection method could also be used in a mesh network. From the proposed protocol a group of nodes in the network can detect the use of promiscuous mode on a node. Once a bad node is detected then the other nodes can route traffic in such a way that avoids the compromised node.

An attacker could attempt to get around many of these detection methods by disabling some of the network traffic that their machine responds to. A completely passive machine is an example that cannot be detected by these detection methods. However, this set up requires an attacker to have their own machine on the network. If an attacker compromises an existing machine on the network, then disabling portions or all of its networking functionality risks arousing suspicion from network administrators. In a hard-wired network a completely passive machine is only possible in a hub/bus set; where a device is not required to interact with the hardware, such as a switch in order gather network communications. On a wireless network, a completely passive machine can exist with the use of monitor mode. Outside of these exceptions it is difficult to get around the detection and prevention methods.

2.8 Application to Computer Forensics

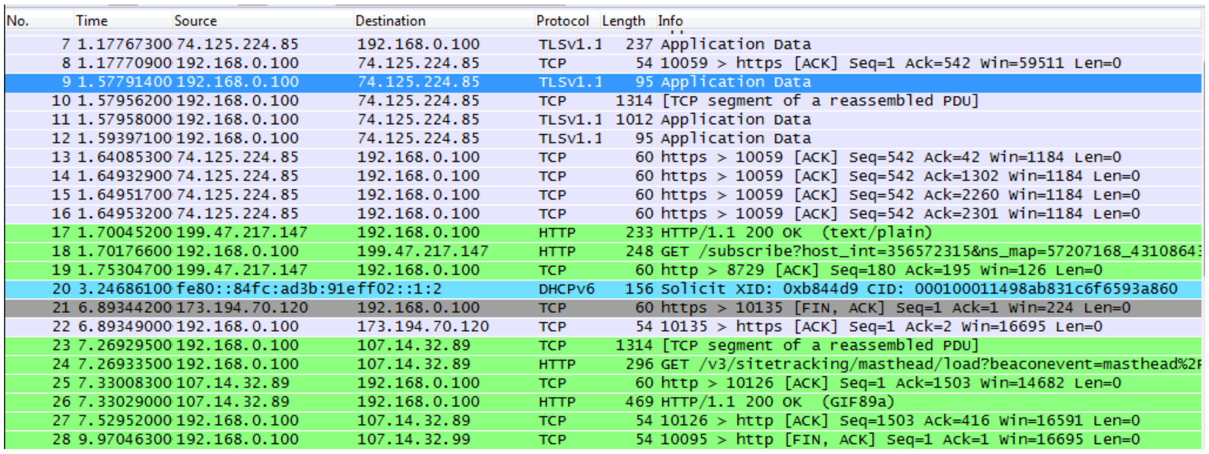
This section discusses the application of detecting promiscuous mode to the aid of a computer forensics investigator. This includes a discussion of what kinds of packets to look for, as well as what needs to be done to be properly prepared for legal cases. Application of what to do with the valid use of promiscuous mode as well as the use of detection results is an important concept.

2.8.1 Analyzing Packet Captures

Packet captures are one way that forensics investigators can glean information about what kind of traffic went over the network. These can be obtained by devices specifically for the purpose of sniffing traffic. They could be found on other machines, but to trust the packet captures, they should come from a device that has been certified, with a threshold, to capture that traffic.

When a forensics investigator obtains packet capture data, no matter how it is obtained, the next step is to analyze the information. This first involves determining which analysis software should be used. Most packet captures are saved in the pcap (packet capture) format, and most software for this purpose can read that format. The investigator must be aware of what to look for, which involves being familiar with all common protocols and some rare protocols that can be used for nefarious purposes. With knowledge of protocols a forensic investigator can then start looking for patterns in network traffic that are a match for known attack types, in order to figure out not only what kind of attack has occurred, but also from where, and what data was compromised [16].

An example capture using the application Wireshark is provided in Fig. 2.3 [3]. Wireshark breaks this information down into an easy to read format. From left to right the columns include the internal ID number of the packet received, the time at which the packet was received after



```
No.  Time       Source                     Destination      Protocol Length Info
   7 1.17767300 74.125.224.85              192.168.0.100    TLSv1.1   237 Application Data
   8 1.17770900 192.168.0.100              74.125.224.85    TCP        54 10059 > https [ACK] Seq=1 Ack=542 Win=59511 Len=0
   9 1.57791400 192.168.0.100              74.125.224.85    TLSv1.1    95 Application Data
  10 1.57956200 192.168.0.100              74.125.224.85    TCP      1314 [TCP segment of a reassembled PDU]
  11 1.57958000 192.168.0.100              74.125.224.85    TLSv1.1  1012 Application Data
  12 1.59397100 192.168.0.100              74.125.224.85    TLSv1.1    95 Application Data
  13 1.64085300 74.125.224.85              192.168.0.100    TCP        60 https > 10059 [ACK] Seq=542 Ack=42 Win=1184 Len=0
  14 1.64932900 74.125.224.85              192.168.0.100    TCP        60 https > 10059 [ACK] Seq=542 Ack=1302 Win=1184 Len=0
  15 1.64951700 74.125.224.85              192.168.0.100    TCP        60 https > 10059 [ACK] Seq=542 Ack=2260 Win=1184 Len=0
  16 1.64953200 74.125.224.85              192.168.0.100    TCP        60 https > 10059 [ACK] Seq=542 Ack=2301 Win=1184 Len=0
  17 1.70045200 199.47.217.147             192.168.0.100    HTTP      233 HTTP/1.1 200 OK  (text/plain)
  18 1.70176600 192.168.0.100              199.47.217.147   HTTP      248 GET /subscribe?host_int=356572315&ns_map=57207168_4310864
  19 1.75304700 199.47.217.147             192.168.0.100    TCP        60 http > 8729 [ACK] Seq=180 Ack=195 Win=126 Len=0
  20 3.24686100 fe80::84fc:ad3b:91eff02::1:2                DHCPv6    156 Solicit XID: 0xb844d9 CID: 000100011498ab831c6f6593a860
  21 6.89344200 173.194.70.120             192.168.0.100    TCP        60 https > 10135 [FIN, ACK] Seq=1 Ack=1 Win=224 Len=0
  22 6.89349000 192.168.0.100              173.194.70.120   TCP        54 10135 > https [ACK] Seq=1 Ack=2 Win=16695 Len=0
  23 7.26929500 192.168.0.100              107.14.32.89     TCP      1314 [TCP segment of a reassembled PDU]
  24 7.26933500 192.168.0.100              107.14.32.89     HTTP      296 GET /v3/sitetracking/masthead/load?beaconevent=masthead%2
  25 7.33008300 107.14.32.89               192.168.0.100    TCP        60 http > 10126 [ACK] Seq=1 Ack=1503 Win=14682 Len=0
  26 7.33029000 107.14.32.89               192.168.0.100    HTTP      469 HTTP/1.1 200 OK  (GIF89a)
  27 7.52952000 192.168.0.100              107.14.32.89     TCP        54 10126 > http [ACK] Seq=1503 Ack=416 Win=16591 Len=0
  28 9.97046300 192.168.0.100              107.14.32.99     TCP        54 10095 > http [FIN, ACK] Seq=1 Ack=1 Win=16695 Len=0
```

Figure 2.3: Sample Wireshark Network Traffic Capture pcap File

the start of the packet capture, the source IP address, the destination IP address, the protocol of the packet, the size/length of the packet, and any info which is known from the protocol of the packet. Wireshark also allows for the display of the raw packet data as well, but through the use of a program like Wireshark, a lot of the packet parsing is done for the user. This then allows for both search strings and visual recognition of desired packets, either by patterns or by protocol.

2.9 Chapter Summary

There are many situations in which promiscuous mode can be used and a variety of purposes for it. Likewise, there are many uses for promiscuous mode detection. In a complete virtual environment there are detection methods have not previously been investigated, which is the goal of this thesis.

Chapter 3
Approach

This chapter contains the methodology that was used, as well as a description of the environment and software used for the experiments in this thesis.

3.1 Methodology and hypothesis

The choice to analyze RTT and LDT was because these two methods are not reliant on an implementation or design flaw in how promiscuous mode is implemented. The detection of DNS calls by a sniffer is a design flaw by those who made sniffers to do that; yes, it grants the users some extra information, but it is at the risk of being detected. To quote Ram Dass "The quieter you become the more you can hear." Relying on the broadcast of faked ARP broadcasts is not reliable, as an attacker could filter out such traffic, and those bad OS drivers can be patched. In addition there is no guarantee that a future OS will continue to have this flaw or any other known flaw for that matter.

Aside from not relying on implementation or design flaw, RTT and LDT methods can be implemented using a number of protocols. Although RTT and LDT in this thesis are implemented through the use of ping response, this is not the only protocol for which the detection methods will work. Any communication type that can be used to illicit a response from a machine can be used. This means it is harder for some entity to create a machine that blocks potential detectable protocols. In the VMware virtual network set up, (which is used for this thesis), a VM cannot truly passively sniff since all machines connected to a virtual network communicate with the virtual switch through ARP [5]. This means that in a virtual environment, a completely passive sniffer cannot be created because the machine must respond to ARP broadcasts in order to be connected to the network. In this thesis, RTT and LDT are implemented using pings because the aim of this thesis is to show that the detection methods will work, and ICMP pings are what was used in Trabelsi et al. [23].

Because RTT and LDT are flexible in their implementation and the way they work, both of these tools could be part of a dedicated tool for the detection of promiscuous mode. The actual implementation of such a tool is discussed in Section 5.2.

3.1.1 Hypothesis

Since both RTT and LDT work on a physical network, and since virtual networks are supposed to be a software representation of hardware, it stands to reason that both RTT and LDT should work in a virtual environment. The experiment described here sets out to look at how well they perform in the virtual environment and if there are any special limitations specifically due to the virtual environment.

3.2 Experiment Environment and Setup

3.2.1 RADICL

The experiments for this thesis were all carried out in the Reconfigurable Attack Defend Instructional Computing Lab (RADICL) at the University of Idaho. This is a virtual computing laboratory set up for the purpose of investigating different offensive and defensive strategies in the field of information assurance. The virtual nature of RADICL made it a prime facility for carrying out this research.

The operating system virtualization hardware used in RADICL as of the writing and experimentation of this thesis, is VMware Virtual Center (vCenter) version 3. The implementation of virtual machine networks and virtual NICs have not changed significantly for newer versions of VMware [6]. This is also compatible with VMware Player 3.

RADICL itself can, at times, be limited due to processing power, and hard-drive read and write accesses. The experiments in this thesis have been designed so that such impacts did not influence the results. This was done by running the experiments while no other users were using RADICL, and by doing tasks that could all be handled in the VM RAM in order to limit reads and writes to the hard disk, which are previously known bottlenecks for performance. The LDT test run, however, sought to make use of these performance bottlenecks, by finding the point where the system starts to fail.

3.2.1.1 Virtual Machine Drivers

As mentioned in Section 5.2.1, there are VM implementations other than VMware's vCenter. The VMware implementation was chosen for the experiments in this thesis because of initial familiarity and availability. As a result of this choice the drivers used were for the VMware implementation. Other VM drivers could expand future work as well as other operating systems. The work in this thesis has been limited to one VM platform and five OS types.

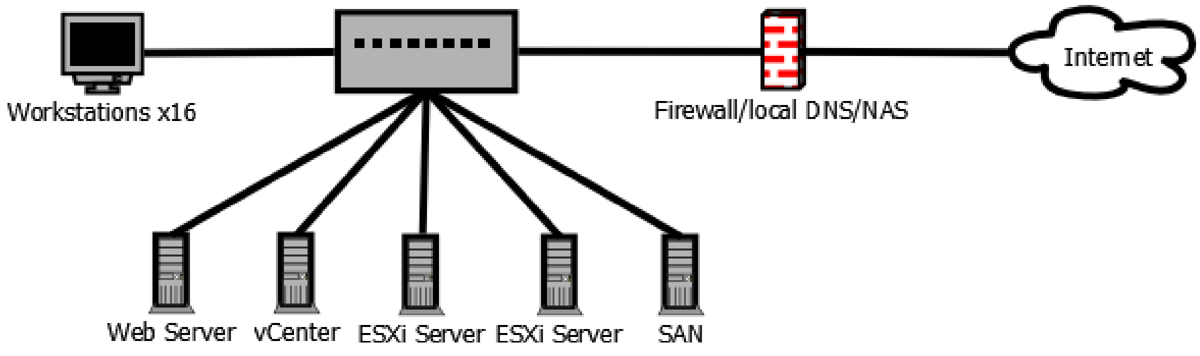


Figure 3.1: RADICL setup

```
#!/usr/bin/python

from scapy.all import *

packet = IP(dst='192.168.0.1')/ICMP()

while True:
  send(packet,verbose=0)
  sendp(IP(dst='192.168.0.1')/'I'm a network packet',
      iface = 'eth0',verbose=0)

```

Figure 3.2: Code to Generate Network Traffic

3.2.2 Code

The code used in this thesis was written in the programming language Python. The Python programming module that was specifically used is Scapy [2]. This module allows for both the sniffing of network traffic and the forging of network traffic. This allows for fine grain control of the creation of traffic on the network for the analysis of some of the methods in this thesis.

3.2.2.1 Traffic Generation

Scapy was specifically used for its ability to generate, send, and receive custom packets. The program/script listed in Figure 3.2 is an example of traffic generation rules. This generation of network traffic allows for testing of the detection methods with different amounts of network traffic, which is particularly useful when looking at the amount of time that a target machine takes to respond.

3.2.2.2 Ping Test

The code in Figure 3.3 shows the code used to generate pings over the network and to measure the subsequent response times. This was the same process used for testing both RTT and LDT.

```
#!/usr/bin/python

from scapy.all import *
import numpy
import sys

def send_ping(dest,count):
  time_array = []
  packet = IP(dst=dest)/ICMP()
  for x in range(count):
    ans,unans = sr(packet,verbose=0)
    rx = ans[0][1]
    tx = ans[0][0]
    time = rx.time - tx.sent_time
    time_array.append(time)

  print dest
  print '\tAverage =', numpy.mean(time_array), 's'
  print '\tStd =', numpy.std(time_array)
  write_data(time_array,count)

def write_data(time_array,size):
  count = 0
  for x in time_array:
    count += 1
    out_file.write(str(x))
    if count != size:
      out_file.write(', ')
    out_file.write('\n')

if len(sys.argv) == 4:
  out_file = open(sys.argv[3],'a')
  print int(sys.argv[2])
  send_ping(sys.argv[1],int(sys.argv[2]))
  outfile.close()
else:
  print 'Usage: pyping [IP dst] [run count] [output file]
    '
```

Figure 3.3: Code to Measure Ping Time

3.2.3 Network Sniffing

Wireshark was used for network sniffing [3]. Wireshark was used because it is the tool of choice of many users, and because it's available on many operating systems. There are many other programs which can be used for the purpose of sniffing network traffic, but their impact on

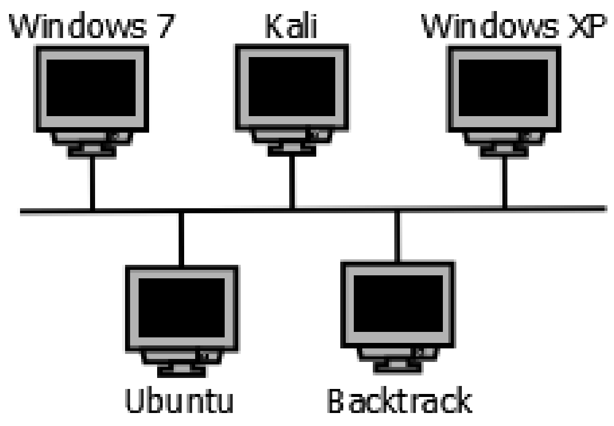


Figure 3.4: Network and Devices Used

performance of the detection methods are not of concern to this thesis. The issue of promiscuous mode drivers and a machine's response to forged ARP requests sent with faked MACs is not discussed in this thesis. Using Wireshark gave easy compatibility between operating system kernels, and a easy way to place the current machine's NIC in promiscuous mode.

3.2.4 Network Setup

The network used in this thesis was based on a simple virtual network that can be set up using VMware, configured to enable promiscuous mode. This network functions the same way that a hub/bus network functions in that all network traffic is visible to all of the devices on the network.

The VMs that were used for this experiment are Backtrack Linux 5 Release 3, Kali Linux 1.01, Ubuntu 12.04, Windows XP, and Windows 7. These were setup as a VM template within RADICL then deployed for the network to be set up. This serves to provide a variety of reactions with drivers as well as a diverse group of machines with different implementations of promiscuous mode drivers. Figure 3.4 shows the representation of the network.

With the VMs represented, four different operating system kernels were tested. These included Linux Kernel 3.2.6, Linux Kernel 3.8.8, Windows 7 and Windows XP. There is a slight potential for redundancy between the Ubuntu and Backtrack machines since they are both based on the base Linux Kernel version 3. The difference that potentially is introduced between the two OSs is that Backtrack built on top of a custom built kernel specifically for the purpose of penetration testing and other potentially malicious acts. This means that there could be a difference in the ability to detect promiscuous mode compared to that of a machine running

Ubuntu. Kali Linux was used as the detection machine because it takes less resources to run than Backtrack, which was deemed important due to the constrained resources of operating in a virtual environment.

3.3 Experiment

There were two main sets of runs that were carried out for this experiment. These were runs with and without other network traffic present on the network. Runs that did not involve other network traffic, (other than those necessary for a base layer), are used as an initial baseline. If RTT works on a virtual network it helps to verify that LDT could work; it then becomes a challenge to find the threshold for the system in question of the amount of network traffic.

For all runs, there were a total of 390 ICMP pings sent and measured for time sent and received, which is three times the number of pings used in Trabelsi et. al. (130 ICMP pings) [23]. This duplicates the previous work, but also allows for analysis of numbers of samples taken by looking at 130, 260, and 390 pings, (one, two, and three times the original, respectively). Each run was performed at least three times in order to assure that results were consistent.

For tests with network traffic there are a couple of runs performed with different amounts of network traffic. One set of runs was conducted with 600-1100 packets per second (pps). This was just an arbitrary number chosen as a starting point. Because LDT was not guaranteed to work depending on the number of network packets, several test runs were executed to find where the upper boundary of the server. This number was 200-700 pps. Using both the 600-1000 pps and the 200-700 pps, sets were run with both in promiscuous mode and managed/normal mode in order to compare results between NICs operating under the same network conditions. How many packets to use depends on the system in question. A larger system than RADICL would require a larger number of packets.

The collected data then analyzed with the Z-Test. The Z-Test determines if any confidence can be placed on whether or not the machine in question is running in promiscuous mode.

Chapter 4
Analysis

The results of the experiment carried out in this thesis were not entirely as expected, but have made for some interesting talking points and possible future work. It was expected that both RTT and LDT would both work in the virtual environment.

4.1 Round Trip Time

When there was no other network traffic on the system use of the Round Trip Time detection worked well in the virtual environment. For each of the operating systems in question the Z-test showed, with 99% certainty, that the machine in question was running in promiscuous mode, since the Z-Statistic was greater that 2.36 (Table 4.1). This serves to show that the RTT test works in a completely virtualized environment. The ability to detect a promiscuous sniffer during periods of low bandwidth implies that this method can be used during periods of low usage of a network.

Figures 4.1, 4.2, 4.3, and 4.4 are graphs of all the data points gathered while running experiments on each machine with the RTT detection method. Each data point represents the time for one ping to respond. Normal/managed mode and promiscuous modes are graphed in different colors.

	Windows XP	Windows 7	Ubuntu 12.04	Backtrack 5 R3
Normal Average	0.002359	0.002366	0.00238	0.002339
Promiscuous Average	0.002525	0.002665	0.0025528	0.002538
Normal Standard Deviation	0.000359	0.000341	0.00044	0.00363
Promiscuous Standard Deviation	0.000611	0.001273	0.00054	0.000782
Z-Statistic	3.783751	3.660855	3.71349	3.390316

Table 4.1: Round Trip Time (in seconds)

4.2 Load Detection Technique

When the system was under a load of 600-1000 pps , the results were slightly mixed with some results of the Z-test showing almost no real difference in the distributions and others showing little difference (See Table 4.2).

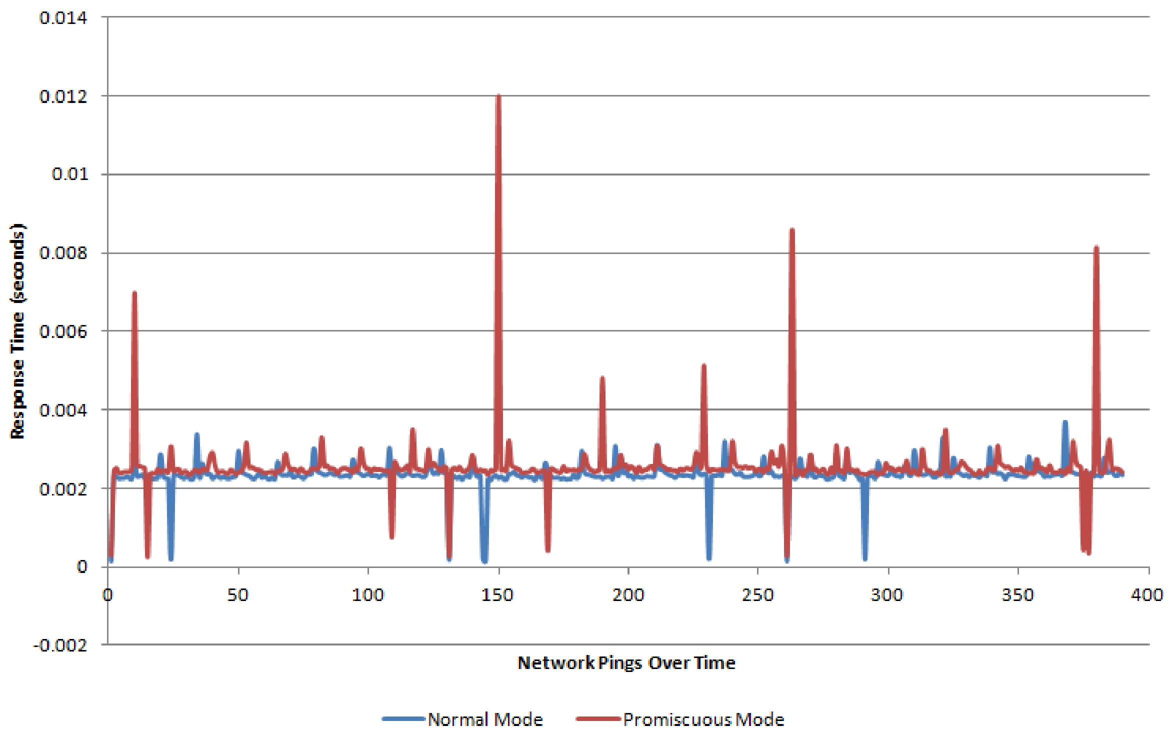


Figure 4.1: Round Trip Time Backtrack 5 Release 3

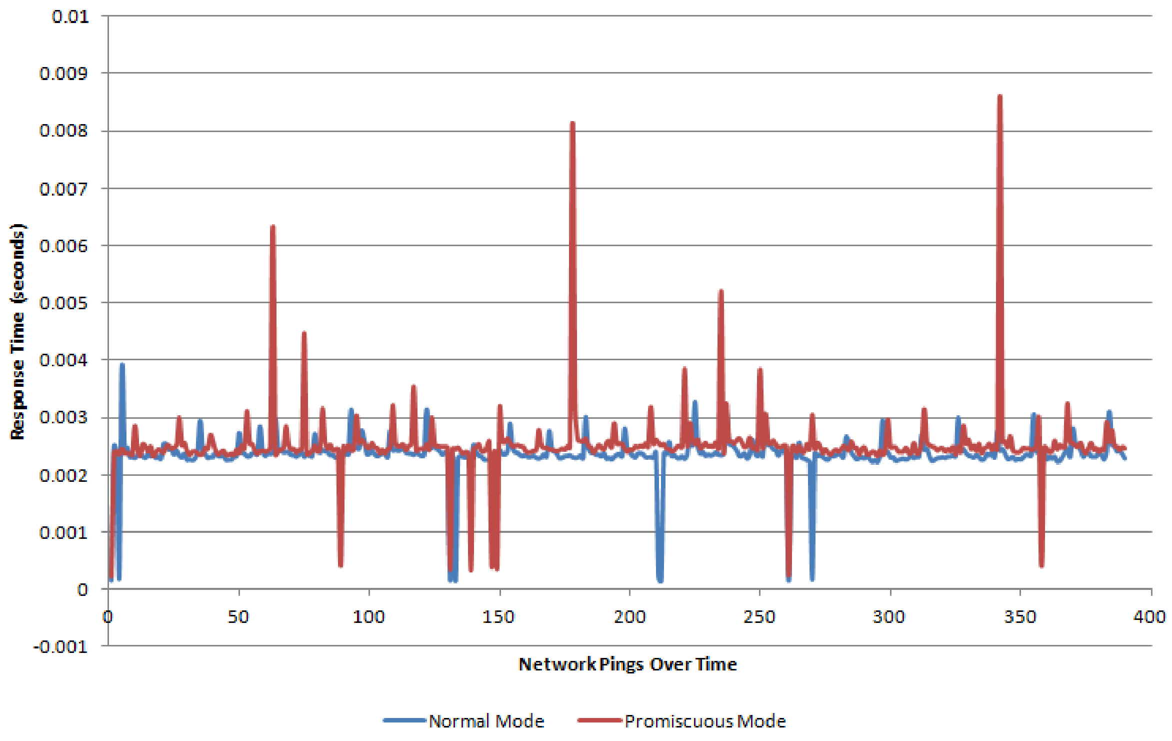


Figure 4.2: Round Trip Time Windows XP

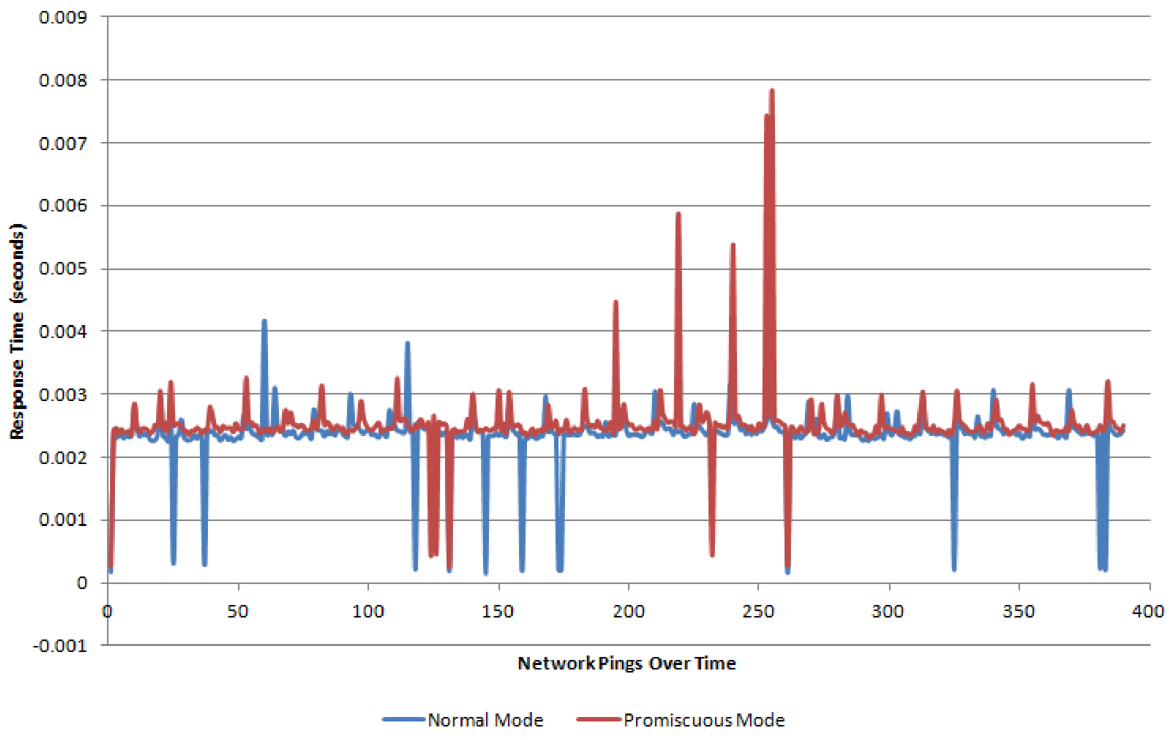


Figure 4.3: Round Trip Time Ubuntu 12.04

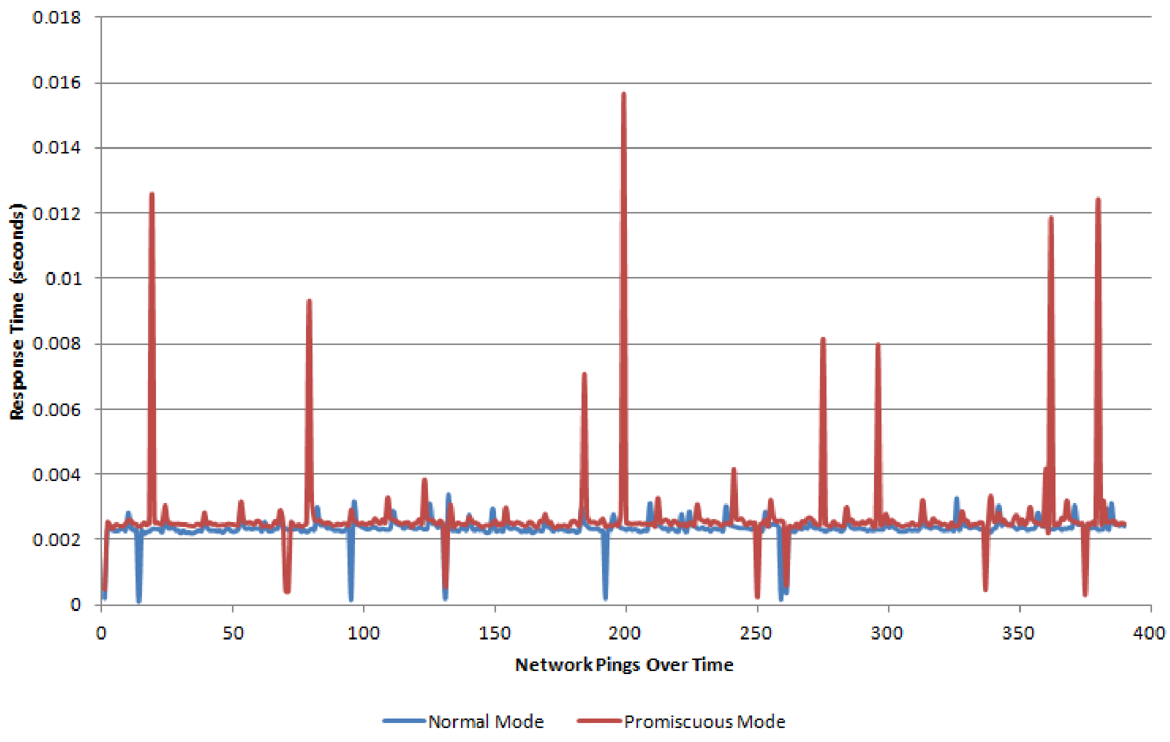


Figure 4.4: Round Trip Time Windows 7

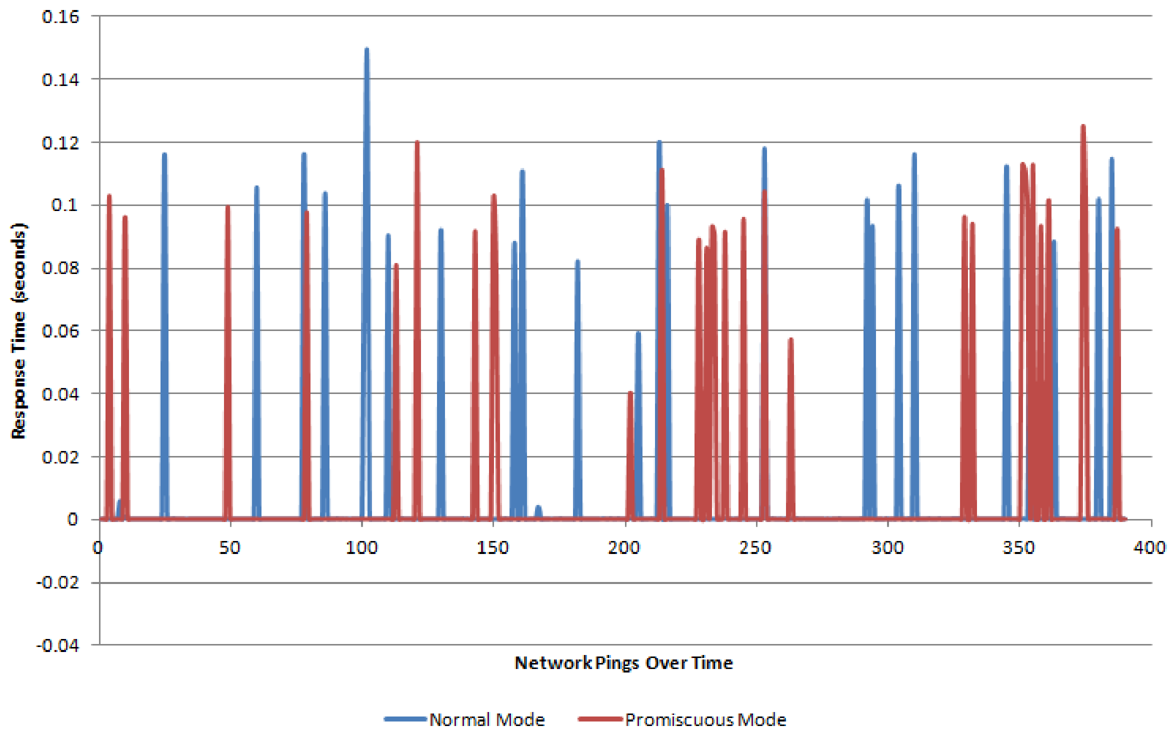


Figure 4.5: Load Detection Technique 600-1100 pps Backtrack 5 Release 3

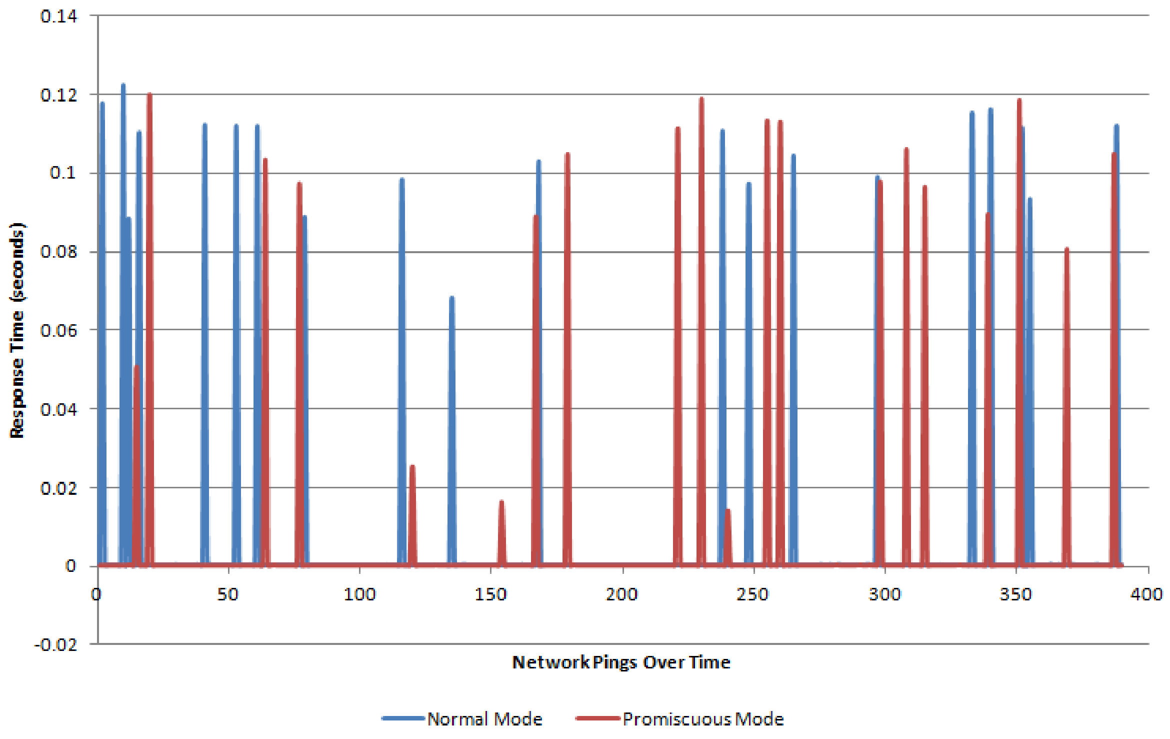


Figure 4.6: Load Detection Technique 600-1100 pps Windows XP

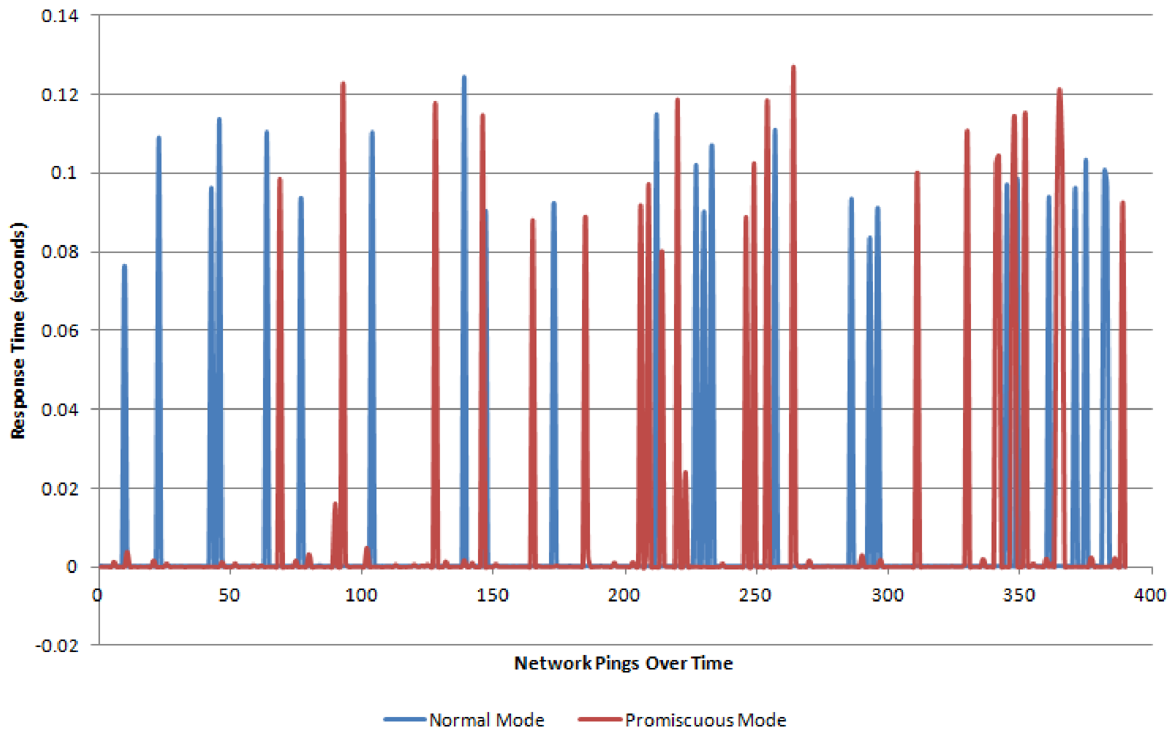


Figure 4.7: Load Detection Technique 600-1100 pps Ubuntu 12.04

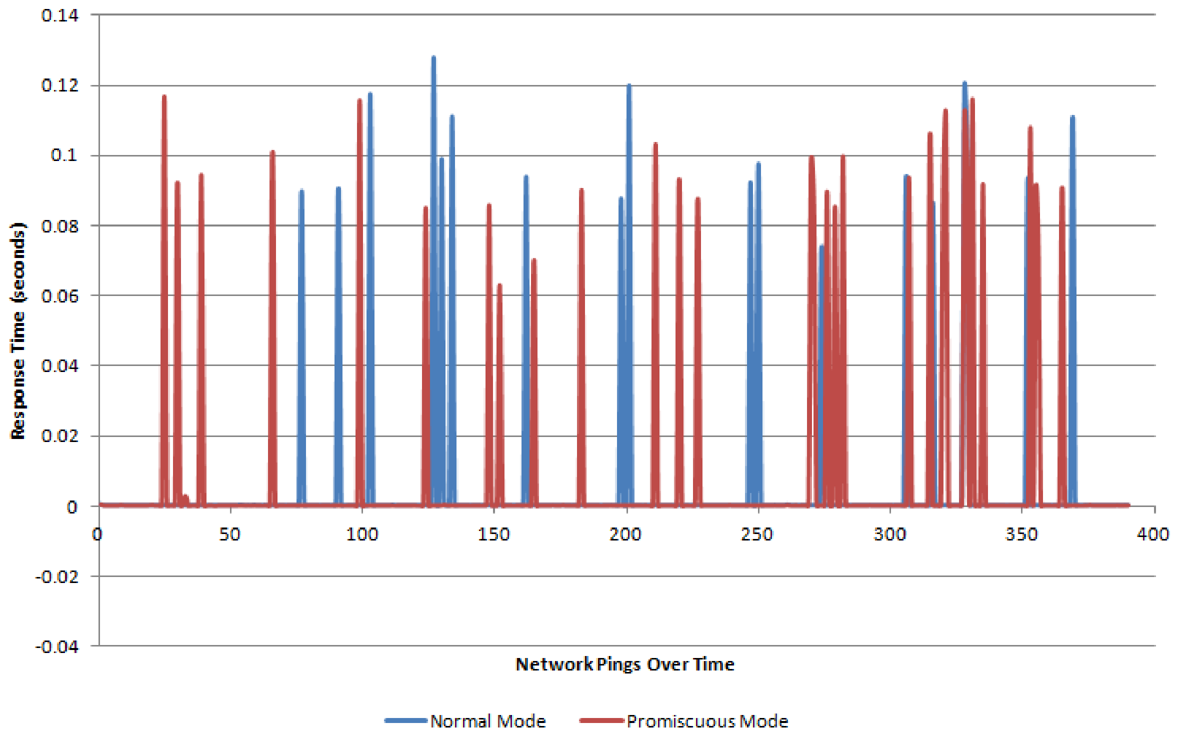


Figure 4.8: Load Detection Technique 600-1100 pps Windows 7

	Windows XP	Windows 7	Ubuntu 12.04	Backtrack 5 R3
Normal Average	0.005547	0.004804	0.006584	0.006525
Promiscuous Average	0.004724	0.007466	0.007102	0.007516
Normal Standard Deviation	0.23233	0.02128	0.024582	0.024888
Promiscuous Standard Deviation	0.020946	0.025359	0.025636	0.025788
Z-Statistic	-0.42439	1.29643	0.23479	0.445893

Table 4.2: Load Detection Technique 600-1100 packets/s

Figures 4.5, 4.6, 4.7, and 4.8 are graphs of all the data points gathered while running experiments on each machine with the LDT detection method. Each data point represents the time for one ping to respond. Normal mode and promiscuous modes are graphed in different colors.

A set of runs was also conducted with 200-700 pps. This was found to be the borderline between overloading the system and producing results. As can be seen in Table 4.3, the Z-Statistic on Windows XP and Windows 7 are 1.019 and 1.58 respectively, meaning that there is 34-44% probability that the machine was running in promiscuous mode. The low percentage is not enough to be prove the test true or not. The test conducted on the Backtrack machine generated results indicating a overloaded system. For the Ubuntu machine the Z-Statistic of 3.136879 indicates the desired 99% that the machine was running promiscuous mode. In Figure 4.9 a big spike can be seen. If this spike is removed from the data the Z-Statistic value is now 4.963086, which does improve the results of the test showing the amplified nature of LDT. The spike is the result of the number of packets running over the network being right at the threshold.

However, even with the promising results of some of the VMs being detected by LDT the overall results are not enough to definitively say that LDT works in a virtual environment. This is because the data does not fit a normal distribution, which is required for a successful Z-Test. LDT could be looked into further on a more powerful VM host environment in order to gather better data.

At 200-700 pps, the Windows XP and Windows 7 machines are an example of the increase of false negatives by using a high confidence level. The VMs were running in promiscuous mode, but the measured Z-Statistic was not high enough to meet the desired confidence level for the expereiment's Z-test.

	Windows XP	Windows 7	Ubuntu 12.04	Backtrack 5 R3
Normal Average	0.00161	0.001728	0.001265	0.001008
Promiscuous Average	0.001717	0.002029	0.001775	0.00093
Normal Standard Deviation	0.001103	0.001054	0.002345	0.004887
Promiscuous Standard Deviation	0.001273	0.003606	0.001176	0.004787
Z-Statistic	1.019	1.578319	3.247542	-0.18185

Table 4.3: Load Detection Technique 200-700 packets/s

The greater chance of false negatives (Section 2.6.2) is slightly preventable with LDT, but not easy to do. Because LDT works best when it the network is close to being under a internal DoS attack, it is difficult to find that magic sweet spot. On a physical network the challenge is finding the limit of the physical hardware, such as a router or a switch. In a completely virtualized environment the challenge is finding the point where the server is not overloaded, since it is handling every machine, as well as all of the network communication which happens in the environment. Figure 4.10 shows the CPU usage of RADICL at and around the point where the system is overloaded due to the amount of network traffic. This was while 600-1100 pps were being transmitted over the network.

4.3 Summary

These mixed results show that for the RADICL setup, and the server in particular, that RTT works in a virtualized network, and that LDT is too variable for conclusion in this thesis. The trick for LDT is finding the upper limit of network traffic before the network/server is internally DoS. This point will be different for every system both physical and virtual. So LDT may or may not work in a virtualized network, it would require fine tuning in order to make greater use of it and prove whether or not it works. RTT works as expected.

In conclusion that RTT works in a virtualized environment, and LDT requires more study. Both methods require some initial setup in generating a test dataset. LDT requires system specific tuning in order to find the threshold of the system so that the base set and the test sets are run under the same network loads.

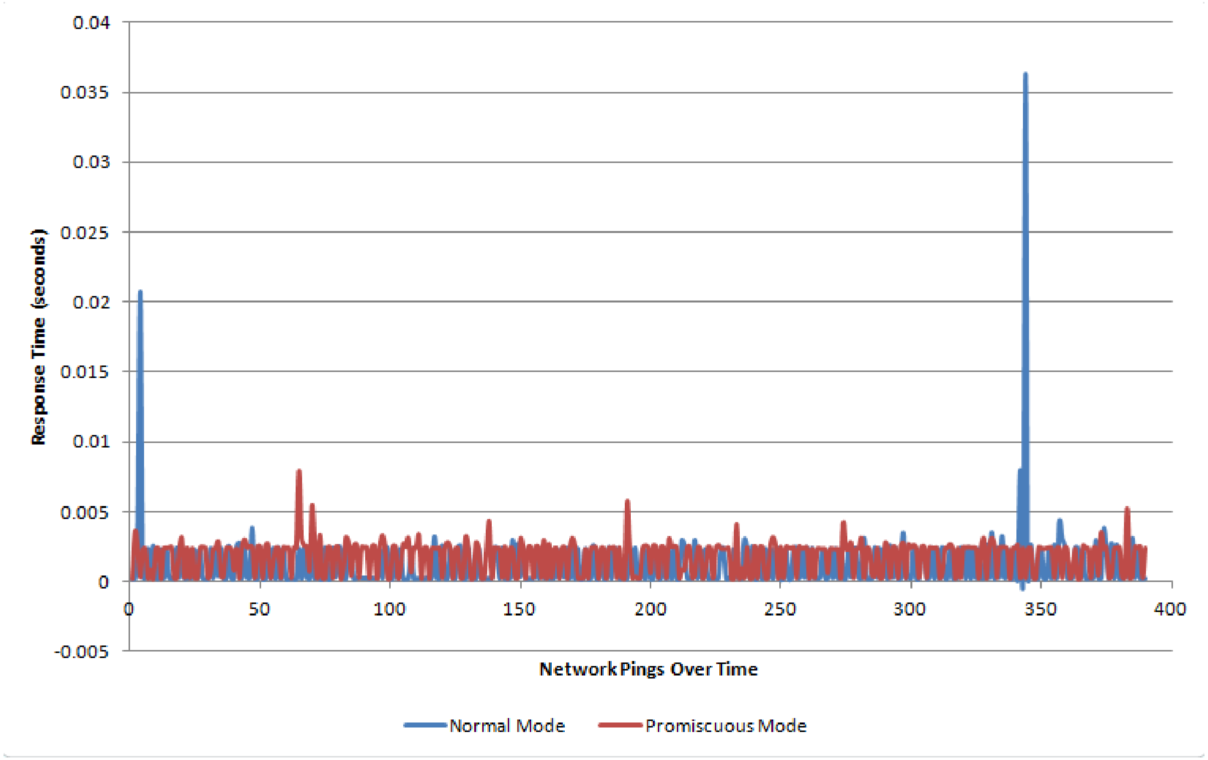


Figure 4.9: Load Detection Technique 200-700 pps Ubuntu 12.04

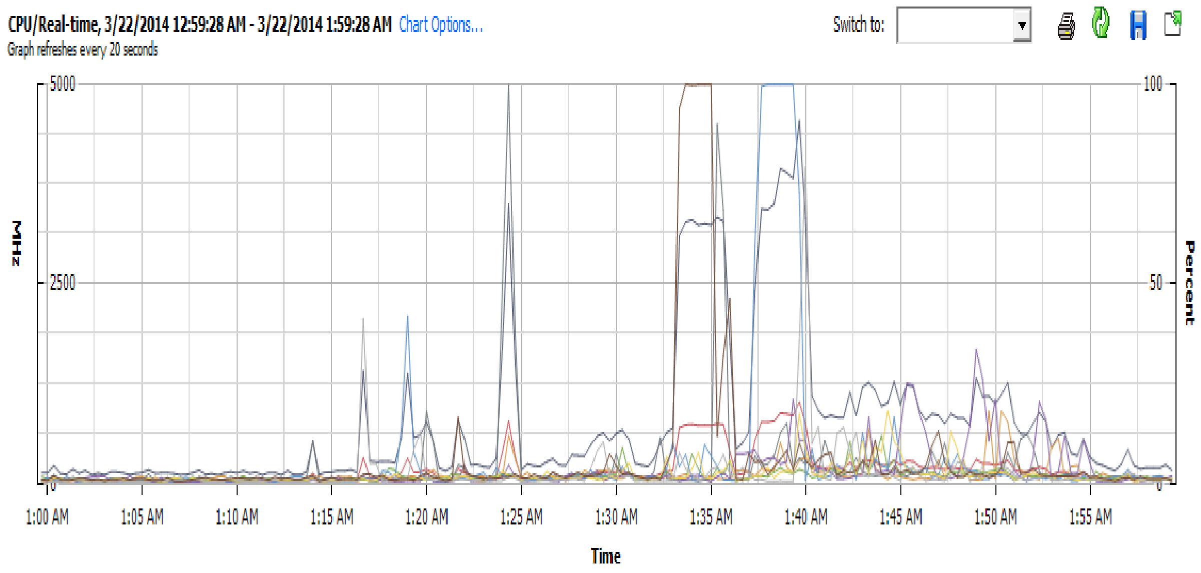

CPU/Real-time, 3/22/2014 12:59:28 AM - 3/22/2014 1:59:28 AM Chart Options...
Graph refreshes every 20 seconds
Switch to:

Figure 4.10: Server Utilization

Chapter 5
Conclusions and Future Work

5.1 Conclusions

Promiscuous mode is a very useful mode on a NIC for both malicious and non-malicious purposes. Proper use of it by the right people can help lead to fair verdicts in criminal investigations of various natures. It allows for the gathering of information and the big difference is what you do with it. The handling of it has legal issues on both sides. Handled by malicious people it allows the systems to be compromised. From the forensics standpoint it can aid investigations and lead to fair court decisions. Either way, promiscuous mode is useful for both purposes and something that will continue to be around. Understanding the uses and means of detection is just part the solution to dealing with it.

The knowledge that these detection methods work in both the physical and virtual environments is a great insight to the potential creation of a detection tool. Hopefully in the near future a tool like this might be created and used. With the increasing concern about privacy, and the increase ubiquitousness of computers, especially mobile computers, it will continue to remain in the best interest of system administrators to know what is happening on their network so that they can best protect their users.

5.2 Future Work

While the results of the experiments for this thesis did not completely match up with what was expected, they do serve the purpose of laying the ground work for a potential tool to be created. They showed that RTT does work in a virtual environment, but LDT may need fine tuned for the intended network it will be used on. A tool could use these detection techniques on both physical and virtual networks.

The detection of promiscuous mode on a network is no easy task, whether the network is physical or virtual. However, through the implementation of a tool using RTT or LDT, it is possible to make this task easier for system administrators. Any tool making use of either of these detection methods would require a couple of things. First would be a clean set of test data for the network in question such that it is known that each machine used in the test is not operating under promiscuous mode. Second is that whoever uses such a tool would need to be aware that in the absence of a controlled environment, a detection registered through the use of the Z-Statistic is not guaranteed accurate. Rather, a detection is just a means to say that a machine requires further inspection to determine whether or not it is running in promiscuous mode.

Also, in respect to using LDT, the biggest issue is knowing the limits of the network in question. As shown in this thesis the biggest issue in a virtual environment is not overloading the servers that the virtual network is running on. This requires testing to know how many packets per second can pass over the network so that the no part of the network is being placed under a DoS from within. This is by far the biggest challenge for using LDT.

Another potential avenue of future work is expanding the work to be verified for wireless networks. Though not virtualized wireless networks still have different constraints than wired networks and that would present potentially different functionality just like has been shown about virtual environments. Understanding those limits would lead to better knowing the limits of devices [12] with specific methods such as LDT.

Future work could also look into VMs of larger size. Working with a variation of sizes of virtual machine infrastructures would give more information as to the workings of LDT with different network sizes and limitations. Understanding more of the limitations of LDT varying on size could potentially lead to a more rigorous understanding of what approximate packets per second over a network which would work under any given network.

Finally, this thesis only looked at one, popular, VM implementation and type of environment. As discussed in this thesis there are other VM implementations, but the type of environment can be changed and tested further to better understand limitations. This could involve a physical network with single host VMs running or a hybrid network of VMs and a physical network. These pose the challenge of knowing the different limits of hardware and virtualized hardware.

5.2.1 Other Virtual Machines Implementation

There are several other implementations of virtual machines and networks. However, they were not all easily available for use in experimentation for this thesis for a comparison between implementations. Also, the focus of this thesis was whether to verify that RTT and LDT would work in a virtual environment. This section briefly discusses the VM implementations available and hypothesizes any major differences that could arise in detecting promiscuous mode on that particular implementation.

5.2.1.1 Single Host Virtual Machines

There are many single host virtual machines implementations such as VMware Player, XenDesktop, Parallels, and VirtualBox [13]. These products, while not operating with a virtual network, can still run a sniffer over non-virtual networks. These products offer hardware emulation in order to provide something that the guest operating system will expect to interact with [9]. These types of products are not of concern to this thesis since they do not make use of a virtualized network.

The detection methods, which make use of measuring the time between pings and responses, should still successfully connect, but variance in the standard deviation will vary if comparing between virtualized and non-virtualized machines. This is because of the increased processing time required to pass information to the guest operating system on the virtual machine. Although this route has been shortened by some virtual machine implementations there should still be enough to notice even if it is not emphasized by the virtualized nature.

5.2.1.2 Complete Virtual Environments

There are two completely virtualized environments that have a decent amount of popularity (other than VMware vSphere): XenServer and Hyper-V. These are made by Citrix and Microsoft respectively [7] [10]. Their handling of virtual networks is close enough to VMware to make some of the same hypothesises about detection of promiscuous mode, but the actual data to be gathered could vary because of the differences in implementation.

Bibliography

[1] Joffe v. google, no. 11-1748, 2013 (us district court for the northern district of cali.).

[2] Scapy. http://www.secdev.org/projects/scapy/.

[3] Wireshark project. http://www.wireshark.org/.

[4] U.s. department of commerce: Telecommunications: Glossary of telecommunications terms, August 1996. Federal Standard 1037C.

[5] Network throughput in a virtual infrastructure. http://www.vmware.com/pdf/esx_network_planning.pdf, 2005.

[6] Vmware virtual networking concepts. http://www.vmware.com/files/pdf/virtual_networking_concepts.pdf, 2007.

[7] Better virtualization of xenapp and xendesktop with xenserver. http://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/better-virtualization-of-xenapp-and-xendesktop-with-xenserver.pdf, 2010.

[8] Google loses appeal in street view privacy lawsuit. http://www.pcworld.com/article/2048541/google-loses-appeal-in-street-view-privacy-lawsuit.html, September 2013.

[9] Chapter 6. virtual networking. http://www.virtualbox.org/manual/ch06.html, March 2014.

[10] Hybervisor top level functional sepcification v4.0. http://download.microsoft.com/download/A/B/4/AB43A34E-BDD0-4FA6-BDEF-79EEF16E880B/Hypervisor%20Top%20Level%20Functional%20Specification%20v4.0.docx, Feb 2014.

[11] V. Corey, C. Peterman, S. Shearin, M. Greenberg, and J. Van Bokkelen. Network forensics analysis. *Internet Computing, IEEE*, 6(6):60 – 66, nov/dec 2002.

[12] B. Endicott-Popovsky. *A methodology for calibrating forensic ready, low layer network devices*. PhD thesis, University of Idaho, Moscow, ID, USA, 2007. AAI3281291.

[13] P. Ferrie. Attacks on virtual machine emulators. http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf, January 2006.

[14] S. Garfinkel. Network forensics: Tapping the internet. http://paulohm.com/classes/cc06/files/Week6%20Network%20Forensics.pdf, 2002.

[15] P. Johansson. Ipv4 over ieee 1394. http://tools.ietf.org/pdf/rfc2734.pdf, December 1999.

[16] G. C. Kessler. On teaching tcp/ip protocol analysis to computer forensics examiners. *Journal of Digital Forensic Practice*, 2(1):43–53, 2008.

[17] A. Khan, K. Qureshi, and S. Khan. Enhanced switched network sniffer detection technique based on ip packet routing. *Information Security Journal: A Global Perspective*, 18(4):153–162, 2009.

[18] T. King. Packet sniffing in a switched environment. http://www.sans.org/reading_room/whitepapers/networkdevs/packet-sniffing-switched-environment_244, 2006.

[19] J. Loui. Detection of session hijacking. http://uobrep.openrepository.com/uobrep/bitstream/10547/211810/1/louis2011.pdf, 2011.

[20] D. Sanai. Detection of promiscuous nodes using arp packets. http://www.securityfriday.com/promiscuous_detection_01.pdf, 2001.

[21] W. Stallings. *Network Security Essentials.* Prentice Hall, fourth edition, 2011.

[22] M.-Y. Su and S. Yeh. A study on the prevention of sniffing nodes in mobile *ad hoc* networks. *Security Comm. Networks*, 4(8):910–918, 2011.

[23] Z. Trabelsi, H. Rahmani, K. Kaouech, and M. Frikha. Malicious sniffing systems detection platform. In *Applications and the Internet, 2004. Proceedings. 2004 International Symposium on*, pages 201 – 207, 2004.

[24] D. Wu, F. Wong, et al. Remote sniffer detection. 1998.

[25] G. Zanetti and C. Palazzi. Non-invasive node detection in ieee 802.11 wireless networks. In *Wireless Days (WD), 2010 IFIP*, pages 1 –5, oct. 2010.

[26] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, Apr 1980.

Appendix A
Code

Code used in experiment.

```
#!/usr/bin/python

from scapy.all import *
import numpy
import sys

def send_ping(dest,count):
  time_array = []
  packet = IP(dst=dest)/ICMP()
  for x in range(count):
    ans,unans = sr(packet,verbose=0)
    rx = ans[0][1]
    tx = ans[0][0]
    time = rx.time - tx.sent_time
    time_array.append(time)

  print dest
  print '\tAverage =', numpy.mean(time_array), 's'
  print '\tStd =', numpy.std(time_array)
  write_data(time_array,count)

def write_data(time_array,size):
  count = 0
  for x in time_array:
    count += 1
    out_file.write(str(x))
    if count != size:
      out_file.write(', ')
    out_file.write('\n')

if len(sys.argv) == 4:
  out_file = open(sys.argv[3],'a')
  print int(sys.argv[2])
  send_ping(sys.argv[1],int(sys.argv[2]))
  outfile.close()
else:
  print 'Usage: pyping [IP dst] [run count] [output file]
    '
```

```
#!/usr/bin/python

from scapy.all import *

packet = IP(dst=‘192.168.0.1’)/ICMP()

while True:
  send(packet,verbose=0)
  sendp(IP(dst=‘192.168.0.1’)/‘I’m a network packet’,
      iface = ‘eth0’,verbose=0)

```