NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

VIRTUAL MACHINE DETECTION IN SOFTWARE-DEFINED NETWORKS

by

Timothy D. Bihl

December 2018

Thesis Advisor:	John C. McEachen
Co-Advisor:	Murali Tummala

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE VIRTUAL MACHINE DETECTION IN SOFTWARE-DEFINED NETWORKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Timothy D. Bihl	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

In this thesis, we evaluate a means of determining whether or not a network host is a virtual machine from the perspective of a network administrator using software-defined networking infrastructure. Virtualization presents a user with a desktop and computing environment identical to what is normally expected while also permitting them to be unwittingly controlled from outside the desktop environment. The added complexity of virtual environments causes extra computing delays, which may be observed in traffic round-trip times. In this thesis, we demonstrate how the observed round-trip times may be used to determine which machines were virtualized and which were running natively directly atop the hardware. Two versions of the experiment were performed. The first substantiated that the approach was feasible. The second, using a more realistic software-defined networking infrastructure, showed that delay measurement must be done by methods that minimize unnecessary hops before measurement, though the experiment still succeeded in differentiating virtual machines in the majority of cases.

14. SUBJECT TERMS SDN, virtualization, virtual machine	**15. NUMBER OF PAGES** 57
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

VIRTUAL MACHINE DETECTION IN SOFTWARE-DEFINED NETWORKS

Timothy D. Bihl
Lieutenant Junior Grade, United States Navy
BS, United States Naval Academy, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2018

Approved by: John C. McEachen
Advisor

Murali Tummala
Co-Advisor

Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In this thesis, we evaluate a means of determining whether or not a network host is a virtual machine from the perspective of a network administrator using software-defined networking infrastructure. Virtualization presents a user with a desktop and computing environment identical to what is normally expected while also permitting them to be unwittingly controlled from outside the desktop environment. The added complexity of virtual environments causes extra computing delays, which may be observed in traffic round-trip times. In this thesis, we demonstrate how the observed round-trip times may be used to determine which machines were virtualized and which were running natively directly atop the hardware. Two versions of the experiment were performed. The first substantiated that the approach was feasible. The second, using a more realistic software-defined networking infrastructure, showed that delay measurement must be done by methods that minimize unnecessary hops before measurement, though the experiment still succeeded in differentiating virtual machines in the majority of cases.

TABLE OF CONTENTS

I. INTRODUCTION....................1
  A. BACKGROUND....................1
  B. OBJECTIVES....................1
  C. RELATED WORK....................3
  D. THESIS ORGANIZATION....................4

II. BACKGROUND AND PRIOR WORK....................5
  A. VIRTUAL MACHINES....................5
    1. Implementations....................5
    2. Security Concerns....................7
  B. SOFTWARE–DEFINED NETWORKS....................7
    1. Traditional Networks....................8
    2. Software-Defined Networking....................8
  C. VIRTUAL MACHINE DETECTION....................9

III. APPROACH....................13
  A. ADVANTAGE OF TIMING ANALYSIS....................13
  B. UTILITY OF SDN FOR TIMING ANALYSIS....................14
  C. EXPERIMENTAL METHOD....................15

IV. EXPERIMENTATION AND RESULTS....................21
  A. SDN TESTBED....................21
  B. SDN RESULTS....................28

V. CONCLUSIONS....................33
  A. SIGNIFICANT RESULTS....................33
  B. FUTURE WORK....................34

LIST OF REFERENCES....................37

INITIAL DISTRIBUTION LIST....................39

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Protection rings of Intel architecture. Source: [9]..6

Figure 2. Network topography for technique validation on bare metal. 16

Figure 3. Network topography for technique validation on VM..............................16

Figure 4. Network topography for experimental setup to bare-metal host. 17

Figure 5. Network topography for experimental setup to VM.................................18

Figure 6. SDN testbed equipment. 21

Figure 7. Histogram of RTTs for bare-metal host...24

Figure 8. Histogram of RTTs for bare-metal host, extreme outliers removed..........24

Figure 9. Histogram of RTTs for VM pings. 25

Figure 10. Histogram of RTTs for VM pings, extreme outliers removed..................25

Figure 11. Histograms of RTTs for bare metal and VM..26

Figure 12. Confidence interval for bare-metal RTTs, with 100-ping samples plotted for VM and bare metal...28

Figure 13. Histograms of bare-metal and VM RTTs on SDN, extreme outliers removed...30

Figure 14. Length of all RTTs for ICMP echoes in SDN. 30

Figure 15. Confidence interval for bare-metal RTTs, with 100 ping samples plotted. 32

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Directly connected ICMP echoes, summary. 23

Table 2. Summary of SDN ICMP echo timestamps 29

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	application programming interface
ARP	address resolution protocol
BGP	border gateway protocol
CPU	central processing unit
ICMP	internet control message protocol
IDT	interrupt descriptor table
IP	internet protocol
NIC	network interface card
OS	operating system
QoS	quality-of-service
RTT	round trip time
SIDT	store interrupt descriptor table
SDN	software-defined networking
TCP	transmission control protocol
VM	virtual machine
VMM	virtual machine monitor

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisors for their guidance throughout the writing process. In addition, I would like to thank my wonderful wife for her endless patience and encouragement.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A fundamental tenet of network security is knowing one's network, with all its hardware, software, and connections, better than an adversary [1]. Part of knowing the network is having lowest-level access to the machines on the network. Where virtualization is unwittingly allowed to persist on machines, this access to the machines can be interrupted; thus, blind spots may be present in the network, leaving network administrators with no ability to detect the anomalies.

A. BACKGROUND

Virtualization has a wide range of productive uses that justify its continued presence on networks by adding flexibility to the employment of hardware resources. On the other hand, the presence of virtualization presents a unique set of challenges in terms of maintaining the security of a network. The same features supporting misdirection and limited direct access to the central processing unit (CPU) instruction set can make virtual environments more difficult to authenticate, adding to the difficulty of maintaining a trust relationship to hosts utilizing virtualization [2]. Attributes used to fingerprint hosts can be affected by virtual machine hypervisors, whether inadvertently in the process of virtualization or by intentional spoofing.

If virtual environments are not tightly controlled and limited in use, such a relaxed policy raises the chance that an attacker can put an unwitting user within a virtual environment while having complete visibility of that user's actions. If such a situation were permitted, it would afford persistent access to the network. In the case that the compromised user had elevated permissions on the network, this access could also serve as a source of credentials that would facilitate lateral movement to other portions of the organization's network.

B. OBJECTIVES

Many virtual machine monitors (VMMs) leave small telltale signs of their intervention underneath the operating system [3]. Often, these signs manifest as uncommon values in packet headers originating from the virtual machines. Most networking protocols contain enough infrequently used fields in their headers that the treatment of those fields constitutes a fingerprint and afford the recipient some information about the sender. VMMs tend to check or switch several header fields to match the VMM's default values while leaving other fields unchanged. The resulting packet header often contains a fingerprint that does not match common configurations for either the virtual machine's operating system or the application being run.

While this fingerprinting approach can detect some virtual machines, it relies on a blacklist approach that utilizes a set of known anomalies. This list could presumably also be known, and consequently defeated, by an informed adversary. A whitelist model that indicates when each individual host is not virtualized, based on slower round-trip times (RTTs) for virtual machines (VMs), would be more robust. This is done by leveraging the additional complexity that is inherent in virtualization when compared to an identical configuration that is being run natively on hardware. The delays from processing packets should be distributed in a way that can be modelled statistically. A single round-trip time measurement can be analyzed as the sum of several component delays, commonly separated into processing, queueing, transmission, and propagation delays for each hop of the trip.

For VM traffic, a component of the processing delay results from the overhead associated with virtualization. If the distribution of extra delays from virtualization has a sufficiently large mean and small variance, the added delay should be detectable, even in the presence of statistical noise from the variation in other component delays in the round trip. Reducing jitter in RTTs to keep variance small also minimizes the need for large samples. Short trips, as well as other means of reducing jitter in RTTs, can be supported by software-defined networking (SDN).

SDN is a paradigm of networking in which the de-centralized, best-effort protocols from the early years of the internet, which are still in wide use, can be selectively augmented, clarified, or overridden by communication with a controller having a more complete understanding of the performance and topology [4]. This better understanding also puts the controller in a position to arbitrate amongst the competing demands for service on the network. By leveraging SDNs, it should be possible to measure RTTs within just a few hops of any device on the network, raising the precision of the measurement of processing delay and allowing detection of higher-performance VMMs on the network.

The goal of this thesis is to achieve a reliable whitelist model for VM detection based on the inherent delay of virtualization. SDN supports this goal by affording shorter and more consistent network delays to the monitoring infrastructure. The delay from virtualization is expected to provide a sufficient signal to be detected over network traffic when the precision afforded by large sample sizes and SDN is utilized.

C. RELATED WORK

Most work on the topic of detecting virtualization has taken place on the device itself rather than from the perspective of the network. There has been some discussion of ‘red pill’ instructions, which cause a CPU to respond in one of several ways depending upon which VMM, if any, is present and interfering with normal execution of the instructions [5]. This method has the same disadvantage of fingerprinting in that it is specific to particular implementations of virtualization and would be pursuing the doubly-weak tactic of attempting to play defense by utilizing zero-day vulnerabilities in the implementation of the VMM. Zero days, so named because they are not known until being exploited (on the zeroth day), are expensive, elusive, and cannot be expected to come along regularly. Additionally, the use of zero-day exploits demands an understanding of CPU behavior, which requires more in-depth analysis than simply finding uncommon combinations of header values; thus, ‘red pill’ instructions and zero days are not good candidates for a comprehensive VM detection strategy.

There has also been some work to compare the delays caused by various virtualization products. The conclusions suggest that VMMs running directly on the hardware tend to be faster than hosted VMMs running within operating systems. Network interface card (NIC) throughput was compared, and VMWare’s ESXi hypervisor achieved nearly twice the bandwidth of Virtualbox in a *netperf* test [6]. Source [6] also found significant disparities amongst various VMMs from several vendors. VMWare released a white paper quantifying the delay of their virtual server product, vSphere; they found that pinging from another host to a vSphere VM added approximately 20 µs compared to sending the same pings between two physical machines over several packet sizes ranging from 64 to 1024 bytes [7]. It was also shown in [7] that, as CPU utilization rose, the mean time for the pings from the VM rose by a factor of approximately three. Jitter rose dramatically, as the 99th percentile RTT rose by a factor of five, whereas it had been nearly equal to the mean RTT in low utilization cases. Added delay for a particular VMWare product was demonstrated and quantified in [7] in a small network specifically set up for monitoring delay from a minimum distance across the network. In this thesis, we aim to establish the efficacy of monitoring similar delays from a data or analysis plane of a software-defined network so that a similar result can be achieved even in a complex production environment. A virtual server's characteristics may be used as an approximation of minimal delay from virtualization since the customers for such a product would be very interested in minimizing latency and jitter and maximizing bandwidth; thus, if a particular detection technique were effective at detecting the delay from a server VMM, it would probably suffice for a lightweight VMM built to avoid detection.

D. THESIS ORGANIZATION

This thesis is organized as follows. In Chapter II, the risks of virtual machines, the capabilities of software-defined networking, and a discussion of several possible methods for detecting virtual machines through network traffic monitoring are discussed. The utility of monitoring delays for detecting virtual machines, the value added by software-defined networking for measuring those delays, and the experimental method that is used to validate the concept are all discussed in Chapter III. The collection of data is further clarified, and the results of the experiment are given in Chapter IV. Finally, significant outcomes of the experiment and recommendations for future work are contained in Chapter V.

II. BACKGROUND AND PRIOR WORK

Virtual machines pose unique threats to network security that can be mitigated by awareness and management of virtual environments where necessary. Software-defined networking presents the opportunity for network monitoring schemes that have greater flexibility and less dependence on physical topology. Both virtual machines and software-defined networking, as well as several methods for detection of virtual machines, are discussed in Chapter II.

A. VIRTUAL MACHINES

Virtualization is a method of managing computing resources by splitting or joining the resources of one or many machines into abstract allotments of computing resources known as virtual machines [8]. Virtualization offers many benefits. It allows for containerization of software suites (e.g., Docker), secure testing of files and software via sandboxing, workspace flexibility, and dynamic allocation of computing resources to name just a few ways that organizations benefit from virtualization.

1. Implementations

Computers consist of many layers of abstractions: applications abstract the underlying functionality of an operating system's (OS) application programming interface (API) libraries for a specific purpose; operating systems abstract the functions of the system's hardware drivers to coordinate amongst programs and facilitate user interaction; programming languages abstract the instructions given to processors; and instruction sets for processors are themselves abstractions of complex circuits [8]. At any of these layers, a virtual machine monitor can be interposed between the layers above and below it, creating a virtual machine. In many ways, virtualization methods could be said to exist in all systems. For example, processes on a modern computer, rather than being allowed to consume all resources, receive rationed processing time from the operating system scheduler to achieve various goals of per-process quality-of-service (QoS) or fairness [8]. Large loads on processors, rather than being allowed to run at full speed, are throttled down by processors to control temperature spikes, denying them full functionality of the underlying technology (though for a good reason [8]).

In the same way that the processor holds itself back to keep from overheating and destroying itself, VMMs at higher abstraction levels have to maintain control of the way VMs are addressing the underlying hardware; this behavior is seen with privileged instructions. These instructions allow programs to take priority on, or entirely control, underlying hardware. In Intel processors, this tiered strategy of managing permission is depicted as a four ring model in which applications only get into the outer-most ring, and only operating systems can function in the inner ring, as shown in Figure 1. VMs can have difficulty adhering to this paradigm since, for example, there may be operating systems running unwittingly within an application; whereas the OS may demand special privileges, granting those permissions would not work when many VMs may be competing for the same resources [6]. This interception allows VMs to be used for sandboxing in which a VM can be considered a safe environment to run malicious code because its reach can be tightly controlled.

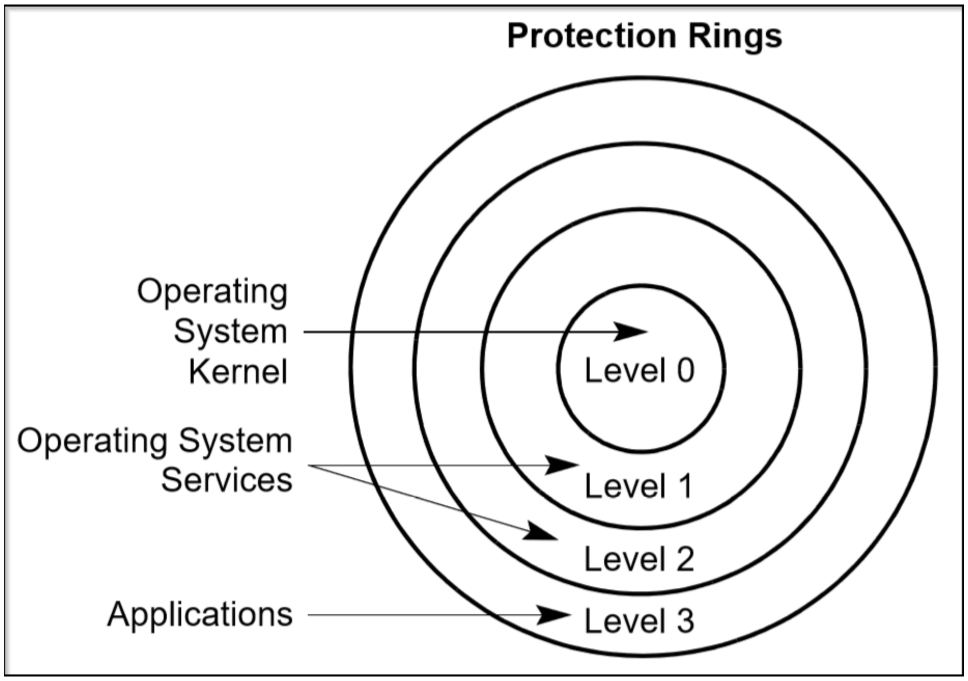


Figure 1. Protection rings of Intel architecture. Source: [9].

2. Security Concerns

The use of virtualization for sandboxing is noteworthy because it takes direct advantage of the paradigm, in concept, if perhaps not always in truth, that the virtual machine sits fully atop the layer of the computing stack where it is built. The virtual machine should not be able to penetrate through the abstractions of this layer, whether that be escaping the browser into the operating system (as in attacks on the Tor browser) or achieving direct execution on the CPU (as might happen on a workstation if an escape attack were performed from a virtual operating system). If a virtual environment is used, it is probably safe to say that its contents are intended to remain within the control of the VMM, whether for stability or security reasons. So, while there are some virtual environments that communicate somewhat openly with underlying software and hardware to achieve performance boosts or support greater functionality, those connections from virtual machine to the outside environment are the exceptions to the rule of virtualization.

By this separation and quarantining of tasks, virtualized environments can offer protection against various threats that require low-level access to system resources by stopping software from achieving malicious goals. On the other hand, virtualization makes possible the disastrous scenario of being unknowingly caught within someone else's virtual environment. In computer security it is commonly accepted that physical security necessarily underlies computer security, but having ultimate physical control of the machine is made meaningless if malware can be made to intervene between the software and hardware without a user's knowing it [10]. If an organization cannot definitively say that its users are not unwittingly sandboxed, then a malicious actor controlling a system with a lightweight VMM can effectively have more control of the machine than the organization that purports to own it. The thinner that layer of virtualization, the more unobtrusive the attack may be, allowing it the further advantage of persistence.

B. SOFTWARE–DEFINED NETWORKS

Software-defined networking augments the performance of traditional networks and affords a richer feature set for optimizing security and performance.

1. Traditional Networks

Many networking protocols used widely today—transmission control protocol (TCP), internet protocol (IP), address resolution protocol (ARP), border gateway protocol (BGP)—have, since their inception, relied on decentralized control [11], [12], [13], [14]. When many of these protocols were designed in the 1980s, bandwidth and processing were expensive, and the number of devices connected to networks was relatively small. Decentralized protocols accommodate unpredictable growth and configuration; as long as packets have correct headers and are destined for properly configured computers, they can be reasonably assured of eventually arriving at the correct machine. There are some means of optimization of path length and other variables to achieve a higher level of performance than can be achieved by random propagation across all available connections. In many protocols, such as BGP, optimization is done by sharing inter-nodal link distances amongst the various nodes making decisions, with the goal of all nodes converging to a common understanding and routing algorithm for the network. The iterative nature of such processes requires time to develop full network pictures; in some cases, networks change too rapidly for convergence ever to be reached [15].

2. Software-Defined Networking

Software-defined networking is an answer to that problem of sub-optimal network performance from decentralized control (as well as an answer to many other problems). In SDN, the routing devices use traditional protocols as their basis for directing traffic but can also be customized to achieve various goals by communicating with a central controller. The controller, which can communicate with all routing devices, is able to view network traffic holistically and make smarter decisions as a result. SDN can also be used to facilitate network virtualization in which an abstraction of the physical network can be provided to give simpler network topologies or provide multiple separate virtual networks on the same physical infrastructure [16]. More generally, SDN is a toolset, rather than a feature in itself, which can support advanced network functionality.

C. VIRTUAL MACHINE DETECTION

There are two promising methods of virtual machine detection that were considered for this research; detection of virtualization through the use of anomalous headers and detection through timing variations. There are many header fields in various networking protocols in which default values are heavily used, and in many of these fields the particular value is of minimal importance (typically because the creators envisioned use cases that never materialized). In these cases, implementations for default values are commonly determined by the operating system, and applications seldom find cause to change those default values. These header values are consistent enough that they are used in web applications to allow the server to guess attributes of the hosts making requests [17]. This process is called fingerprinting.

When the host operating system is different from the operating system on the virtual machine, this may result in a set of default headers that do not fit an expected set of header values. If this set of header values is fingerprinted, the host sending the packets may stand out from hosts using more common combinations of default headers. Several of these anomalies have been discovered and documented and can be used as telltales for virtual machines [18].

One similar method (utilizing lower-level CPU instructions) is the use of ‘red pill’ instructions, named after the red pill used to escape simulated consciousness in *The Matrix*. Red pill instructions rely on flawed implementations (usually unintentional) of virtualization software to find a particular instruction or set of instructions that causes a computer to behave differently in the virtualized case from how it would if the environment were running natively. For example, the Conficker worm was known to use a store interrupt descriptor table (SIDT) instruction to check for virtual environments, going to sleep indefinitely upon detecting one in order to resist attempts at analysis and mitigation [19]. The interrupt descriptor table (IDT) is used to locate the code that handles various interrupts; SIDT, or store IDT, stores the IDT register, giving the address of the IDT. In virtual environments, this table is stored at different locations than in native environments. Use of the SIDT instruction could thus tip off malware that it was in a virtual environment. When red pill instructions work, they are ideal because they work reliably. From a security perspective, however, they are not very helpful because they function as a blacklist model in which it is only possible to defend against known problems. While these one-off exploits are powerful when used in malware, they lose almost all potency in defense.

The second approach considered was to inspect for differences in round-trip times that set virtual machines apart from their natively-installed equivalents. Virtualization has several sources of delay inherent in the technology as compared to the same operating system or application running on a bare-metal platform. There are several processes of virtualization that result in added delays, such as the actual added processing involved in handling the packets through virtual network interface cards [7]. Another delay comes from scheduling by the VMM that accumulates some minimum number of packets before getting CPU time or network interface card (NIC) time, causing uneven delays for some packets [7]. Also, competition for bandwidth with other applications, especially if those other applications have been given higher priority, increases RTTs [7]. Manipulation of processes by VMMs to handle privileged instructions slows communication [7]. Finally, a disconnect may be present between the VMM scheduler and the CPU scheduler, causing time-sensitive packets to be unnecessarily delayed based on suboptimal coordination [7]. The sum of these effects results in a measurable rise in network latency.

Each operating system tends to have its network stack implemented slightly differently, leading to different delays from when packets are formed to when they leave the machine to begin transiting to their destinations. The same is true of hypervisor software; this can be seen especially when comparing multi-purpose software like VirtualBox to enterprise server virtualization products. In the first case, the development effort is likely pulled in many different directions at once, sacrificing specific performance for wide compatibility and functionality, thus compromising network delay time. In the case of enterprise server virtualization, the solutions are more finely tuned to provide short delays, as shown in NIC throughput comparisons of [6].

Regardless of this variation among virtualization products, the hypervisor should always introduce some delay to the network stack, even for high-performance enterprise versions. There is always some amount of redundancy as packets are created in the virtual machine's networking stack and then propagated outward to the real hardware and finally transmitted. Another delay is added upon the arrival of a response packet, when the process of moving into the virtual space is repeated in reverse. If there is little enough variation in round trip-time for a particular protocol (internet control message protocol (ICMP) echo performs a simple and consistent task, for example) and the added time cost of virtualization is sufficiently large, it may be possible to observe a round trip or sequence of round trips and determine whether the traffic is originating from a virtualized or native host. Such an approach may be applicable widely over slightly differing configurations of machines (different machines running a common operating system) but may also be far more limited, requiring ground truth data for each specific combination of hardware and software or perhaps even every specific host.

In addition to the need for simple traffic to keep processing times in the host to a minimum, networking delays must be minimized in order to also minimize their variations. The delays inherent in virtualization can readily be obscured by wireless communication or queueing, where the variation in delay can be substantial. A relatively rough-spun implementation of virtualization, such as VirtualBox, should be easily detectable in noisy networking environments, whereas server virtualization products may require very predictable networks with minimal traffic and very few hops to avoid obscuring the underlying difference.

In this chapter, virtual machines were described, including the risks associated with them and the consequent need to prevent unexpected virtual machine use. Then, traditional and software-defined networks were contrasted to highlight the added capability of SDN. Finally, a brief discussion was given of possible methods for detecting VMs. This information is used to support the choice of method in Chapter III and to explain the presence of the virtualization seen in the results of Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

III. APPROACH

In this chapter, we describe the advantages offered by using timing analysis for detecting virtual machines as well as explaining how such a setup is aided by SDN infrastructure. We then lay out the format of the experimentation that produced the results to be discussed in the next chapter.

A. ADVANTAGE OF TIMING ANALYSIS

The significant problem with the header anomaly method is that it offers very limited coverage of the set of all 3-tuples of host OS, VM OS, and hypervisor program along which the anomalies would be expected to exist. Anomalies may only be relevant for particular versions of an operating system or may appear only in particular hypervisors (whereas other hypervisors handle the transition from guest to host more gracefully, without changing header fields). The testing of even the few most common operating systems and hypervisors would be tedious, and a comprehensive solution is impossible. In this setup, packet captures from the device being inspected might be run through a fingerprinting application to check for known combinations, but falling through the cracks of this blacklist would be common even in the case of normal users, to say nothing of malicious actors with knowledge of the detection system.

Analysis of round-trip times, on the other hand, has the potential to achieve VM detection in a more comprehensive manner by avoiding the pitfalls of searching for header anomalies. Unlike mismatched header fields, which are little more than serendipitous and (presumably) rare tip offs of virtual machines, round-trip delay from virtualization is a fact of the underlying technology that is probably immutable; even the fastest enterprise and server virtualization solutions have some delay from virtualization. By keeping track of the systems on a network, and their associated round-trip time statistics for a sample packet, it should be possible to create a whitelist against which network hosts can be compared. With this whitelist, it should be possible to disprove the possibility of virtualization for any given host by taking a sample packet or series of packets and measuring delay at that host.

B. UTILITY OF SDN FOR TIMING ANALYSIS

The test of round-trip times, as presented above, could theoretically be implemented on a traditional network. Variation in networking delays, however, might obscure any virtualization delay by requiring an impractically long series of packets to make a decision about whether a given host is virtualized. As more hops are added to the round trip being measured, the potential for variable delays increases; while the virtualization delay should always be present with a virtualized host, it could become prohibitive to send enough traffic to reliably ascertain the status of a machine due to use of both computing and networking resources. In fact, a similar method was proposed in 2006 and showed promise for wired connections [20]. The technique was largely written off by [20] because the timing analysis was deemed too dependent upon network topology.

SDN, on the other hand, affords network administrators the ability to reduce the noise in the measurement of round-trip times by inspecting the traffic from the controller. The control plane (of which the controller is a part) can have a connection to each switch in the network and can issue on-the-fly commands for those switches to forward any packets to it. By inspection of the traffic sent to, and returning from, the host of concern as it receives an ICMP echo request (or other traffic), it is possible to observe the response time of that host at a consistent distance of only two hops (host to switch, then switch to controller). This should reduce the timing variation to manageable levels and also give consistent networking delays from host to the point of observation, regardless of where the host exists in a network's topology. That consistent delay should allow for a shorter whitelist, including only each type of host (hardware and software configuration) on the network rather than each individual host.

If the variation of delays within the network can be kept to low levels, then this monitoring scheme for virtual machines in a network may be viable. When connected hosts are periodically tested, it would be possible to have positive confirmation that no user has been unwittingly virtualized.

C. EXPERIMENTAL METHOD

To test the efficacy of the approach described in the previous chapter, two basic networks were used: one, with no SDN component; the other, having an OpenFlow switch and controller sitting atop a similarly simple network topography.

In the first experimental setup, two computers were directly connected to a router to validate the approach. Each computer was running the same operating system, Ubuntu 14.04. One computer sent ICMP echoes to a receiving computer, which in one case was running Ubuntu (see Figure 2) and in the other was running Ubuntu inside a VirtualBox virtual machine which was running atop Ubuntu (see Figure 3). The virtual machine was operating in bridged network mode. In both cases, the data were collected immediately after restarting the machines as a means to limit the impact that other processes might have on the results. A total of 40,000 ICMP echo messages were sent using the *ping* command (with an option set to reduce the interval between pings to 0.1 s, down from the default one second). While the *ping* command was run, a Wireshark capture was performed on the machine originating the traffic. This capture would be analogous to the monitoring that would be performed by a machine on an analysis plane on an SDN. Because the network topography was ideally simple and there was little or no demand on either the network infrastructure or the processors on the two machines, these trials were expected to show an ideal case for how detectable the virtualization delay could be. Virtualbox is also a fairly slow virtualization tool, so the underlying delay was expected to be significant.

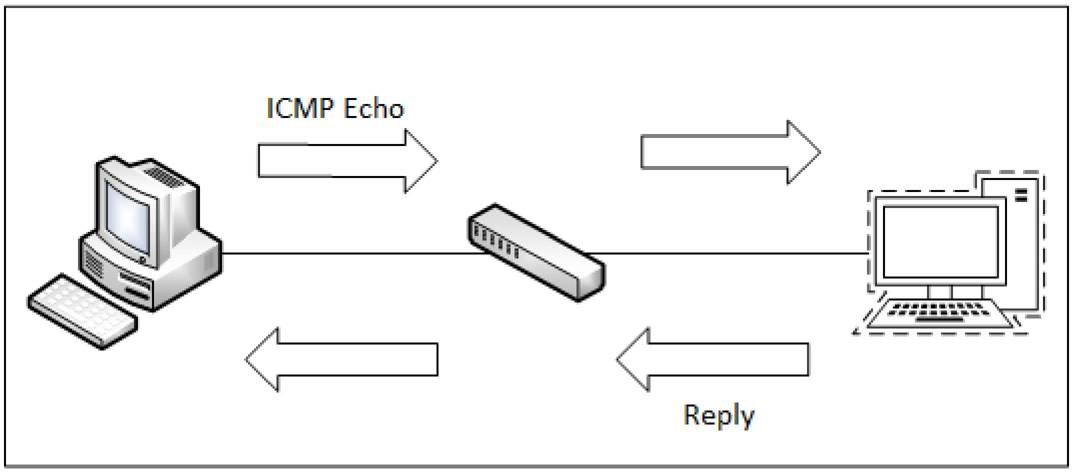


Figure 2. Network topography for technique validation on bare metal.

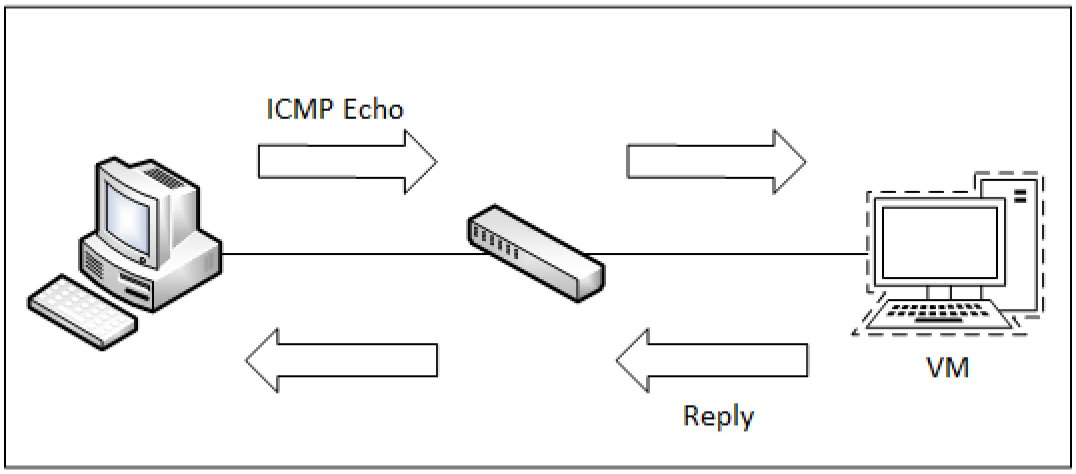


Figure 3. Network topography for technique validation on VM.

In the second set of trials, the same experimentation was performed using the SDN infrastructure of the testbed. The two computers were interfacing this time with one another through an Openflow-enabled switch, which coordinated the passing of packets between the two machines on a case-by-case basis. In this case, it was not precisely a round-trip time that was being calculated; but the basic idea, observing time for a packet to enter into and return from the machine, was preserved. In a sense, the monitoring infrastructure could be thought of as a permissive man-on-the-side interaction. This is done because the data plane, where the ping is occurring, is meant to be kept separate from the control plane, where decisions are made and passed about the flow of traffic through the network. In the case of this experiment, the packets were forced to go to the controller by repeatedly deleting flow information for the destination address. Faced with no knowledge of the correct forwarding action, the switch passed the packet to the controller, where a routing decision was made and passed back to the switch. On the way back to the originator, the same process occurred. In Figures 4 and 5, where all packet traversals are numbered, all eight labelled segments of the trips happened sequentially. The interval recorded was from the transmission of 3 until the receipt of 6.

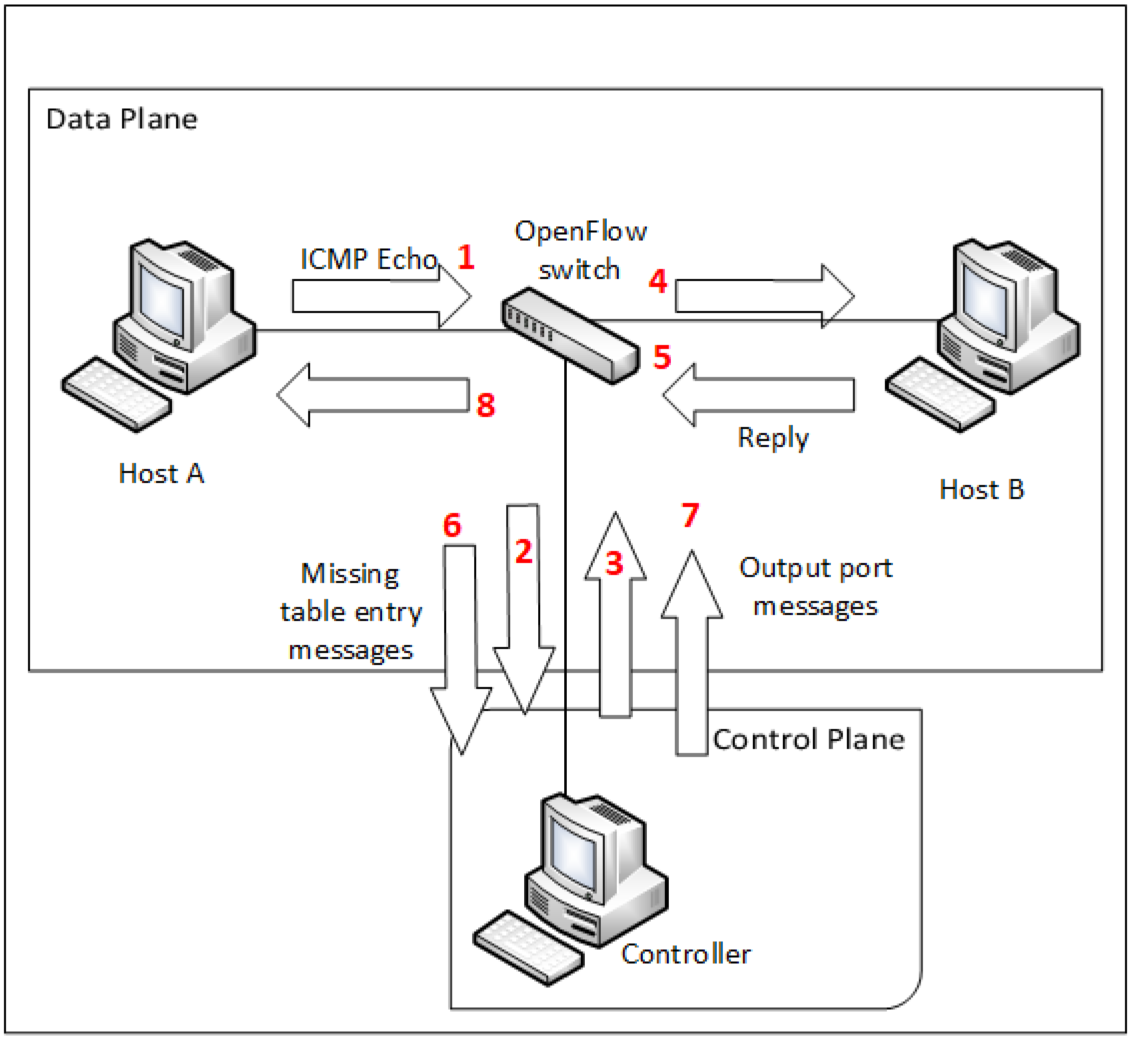


Figure 4. Network topography for experimental setup to bare-metal host.

In a sense, passing this information to the controller was not a realistic portrayal of how the collection would occur since the controller would quickly become bogged down by the extra traffic. On the other hand, the control plane was a sufficiently close stand-in for an analysis plane that would actually be used to achieve the same data collection in a network with loading concerns. The analysis plane, like the control plane, would be a mostly separate plane where timing and other characteristics could be collected in close proximity to the source of the signals, then sent to a central analysis machine or cluster of machines to analyze various aspects of the traffic for more effective load balancing, security, and implementation of other network priorities. Given that there was no regular traffic loading the network, timing with the control plane was considered a viable means of testing.

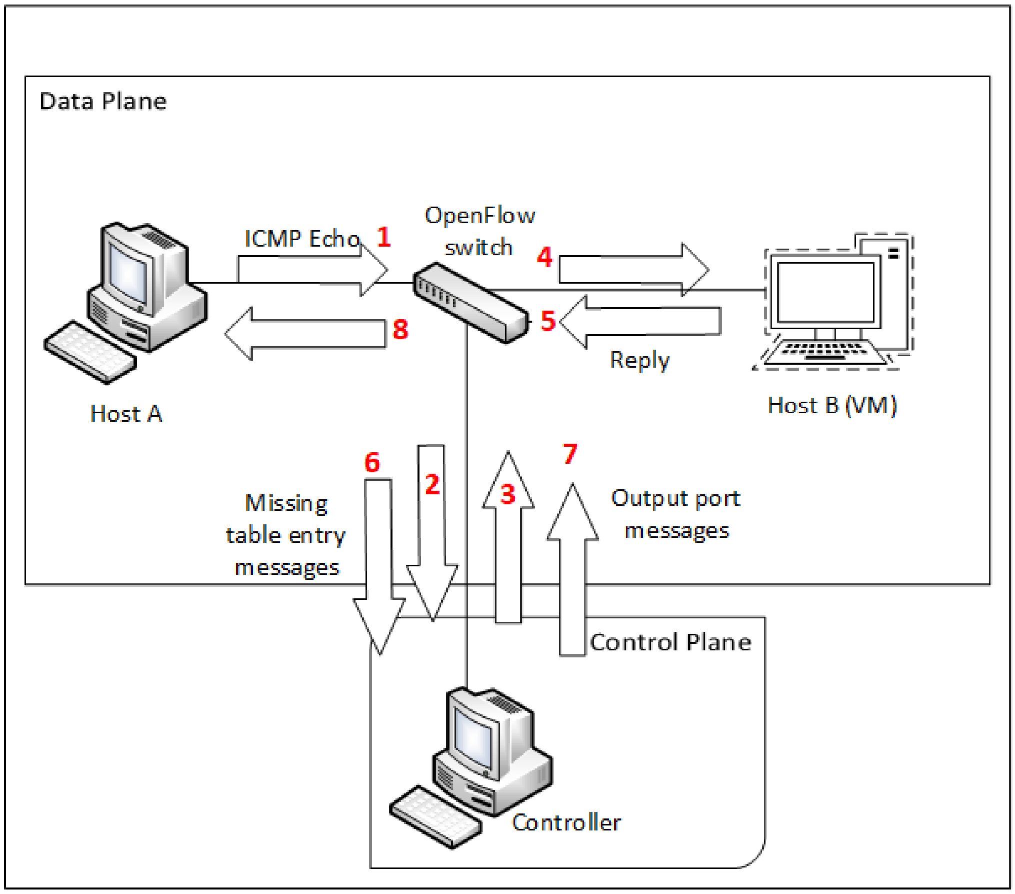


Figure 5. Network topography for experimental setup to VM.

Aside from the administrative difference in which plane was being used for timing, there was also an added, and possibly highly variable, delay from forcing the packets to all go to the controller for forwarding instructions.

On the SDN, because the side trip to the controller added significant delay, pings were left at one-second increments, and samples of 500 pings were taken. Two samples were taken for each of the bare-metal and VM setups. The first sample of each type was used to create a model for the delays, whereas the second sets were used for testing the models created from the first sets. The same was done in the case of the conventionally networked computers, except in that case the two samples of each setup were 40,000 pings each. Aside from supporting larger testing sample sizes and improved confidence intervals, the sample sizes had no effect on the overall results of the testing.

In this chapter, the proposed experimental scheme was presented. Then, a description of the extra monitoring capability offered by SDN was given. Finally, the method for the experimentation was delineated. This method is used to yield the results found in Chapter IV, and its limitations are the basis for recommendations for future work in Chapter V.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENTATION AND RESULTS

The method for detecting virtual machines on the network was proposed in the previous chapter. Results collected from implementing this method are presented below.

A. SDN TESTBED

Experimentation was performed on a network with Raspberry Pi hosts as well as several desktop computers running Ubuntu 14.04 (see Figure 6). A subset of a single Raspberry Pi one desktop computer running Ubuntu 14.04 was used, with the Raspberry Pi sending ICMP echo requests and the desktop replying. In the bare metal experiment, the network traffic went directly to the Ubuntu OS on the desktop host. In the VM experiment, network traffic went to a VirtualBox instance within the desktop PC; the VM also ran Ubuntu 14.04, and networking on the VM was set to bridged mode. The hosts were connected to an HP 3800 OpenFlow-capable switch. This switch was connected to a controller: in this case, another desktop (not connected to the data plane of the network) running Ubuntu 14.04. This desktop used Ryu for a northbound SDN interface to communicate commands to the Openflow switches on the network. All network connections were made with 100-Mbps wired Ethernet connections.

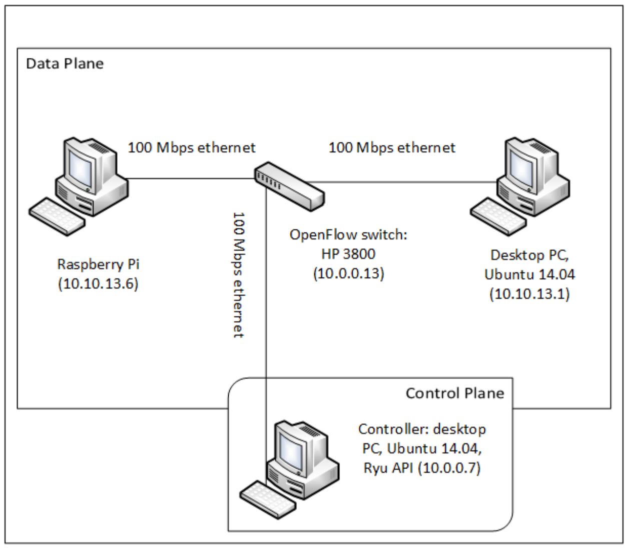

Data Plane
100 Mbps ethernet
100 Mbps ethernet
100 Mbps ethernet
Raspberry Pi (10.10.13.6)
OpenFlow switch: HP 3800 (10.0.0.13)
Desktop PC, Ubuntu 14.04 (10.10.13.1)
Control Plane
Controller: desktop PC, Ubuntu 14.04, Ryu API (10.0.0.7)

Figure 6. SDN testbed equipment.

First, a series of 40,000 pings was conducted twice with a single switch between them: once, to a non-virtualized host; second, to a virtualized host within VirtualBox running an identical operating system (on the same system as was used for the bare-metal host). These tests were performed to validate the timing discrepancy between virtual and bare-metal hosts that would subsequently be used to detect virtual machines in an SDN framework. The case of a single switch separating the two hosts was used because it is the best-case scenario in which noise from transmission and propagation delays is minimized.

The pings were performed with 0.1-s delays between each (default delay is one second) to significantly reduce time for data collection while still giving reasonable assurance that the times observed would not be affected by network congestion. Typical round-trip times were well below 0.001 s, suggesting that the reduced interval between pings would not contribute significantly to networking or processor congestion.

Timestamps were obtained at both the sending and receiving hosts, with Wireshark's timestamps being used to provide precision. Because there is no assurance that the two machines' clocks are synchronized, each machine's timestamps were only compared against other timestamps from that same machine. In this way, it was possible to obtain round-trip time (elapsed time between the sending the request and receiving the reply at the sender), processing time (elapsed time between receiving the request and sending the reply at the receiver), and travel time (difference between round trip and processing times).

For the virtual machine run, the timestamps at the receiving side were collected on the bare-metal OS outside the hypervisor in order to encompass virtualization delays within the other processing delays at the receiver.

In a production environment, a controller (or parallel device in an analysis plane) would be the single point for data collection since the host cannot be trusted. In this analysis, however, trying to separate the various components of the delay provides insight into how much delay occurs on the host as opposed to in transit.

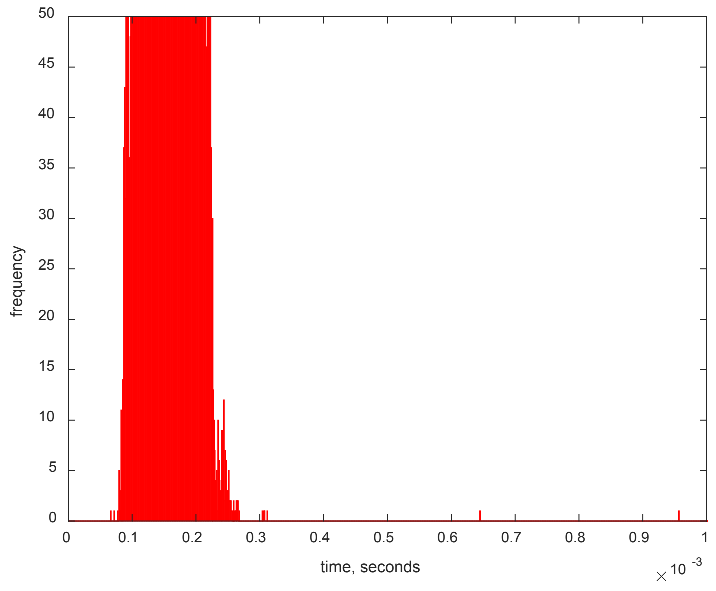
Extreme outliers (beyond five standard deviations) were removed from the data sets. These outliers represented less than 0.2% of both virtual and physical cases, and removing them allowed the data to be treated as a Gaussian distribution for simplicity of analysis. When the resulting data were analyzed, with extreme outliers removed, the summary statistics in Table 1 resulted.

Table 1. Directly connected ICMP echoes, summary.

	Bare Metal		Virtual Machine	
	Full data set	Extreme outliers removed	Full data set	Extreme outliers Removed
N	40000	39997	40000	39934
Mean (seconds)	1.5929×10^{-4}	1.5923×10^{-4}	5.2055×10^{-4}	5.1882×10^{-4}
Standard deviation (seconds)	3.5063×10^{-5}	3.4433×10^{-5}	1.0055×10^{-4}	8.8425×10^{-5}
Skewness	0.743	-0.100	3.514	-0.839
Kurtosis	22.211	1.815	102.617	3.173

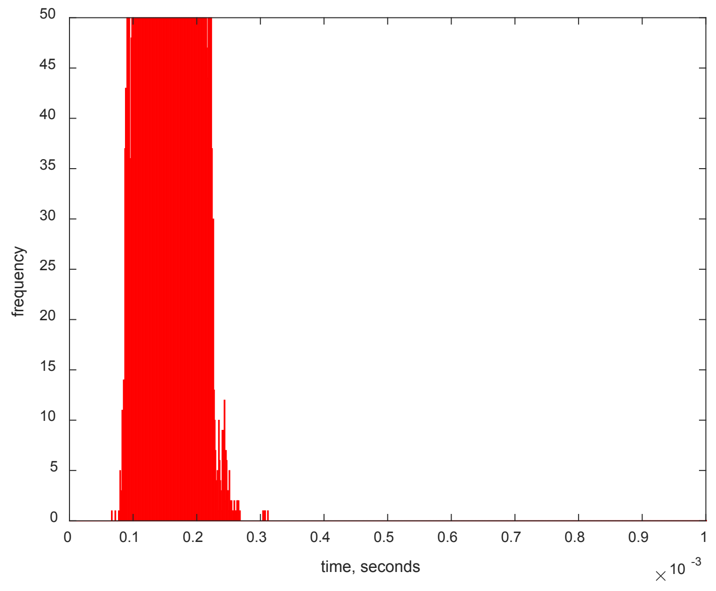
With extreme outliers removed, each set has kurtosis and skewness values that indicate a good fit for a Gaussian distribution. For each of bare metal and VM data sets, how the removal of extreme outliers changed the size of the data set, the mean, and the standard deviation estimate are seen in Table 1. Kurtosis and skewness, which provide a metric of how closely the data sets fit a Gaussian distribution, were also improved by the removal of extreme outliers.

Below, histograms of the data with extreme outliers included and omitted show how the data sets were altered by this alteration. The bare metal trial is shown in Figures 7 and 8, and the VM trials are shown in Figures 9 and 10. The vast majority of bare metal RTTs were 0.08 and 0.3ms, with several RTTs well above 0.3 ms, as shown in Figure 7. When extreme outliers were removed from this data set, only the three largest values were removed out of 40,000, as seen in Figure 8. In the VM case, a similar pattern was seen, with most RTTs between 0.3 and 1ms and several higher RTTs as high as 4.3 ms, shown in Figure 9. When the extreme outliers were removed, all RTTs above 1.0 ms were removed, which amounted to 66 of 40,000 RTTs. A truncated histogram resulted, as shown in Figure 10.



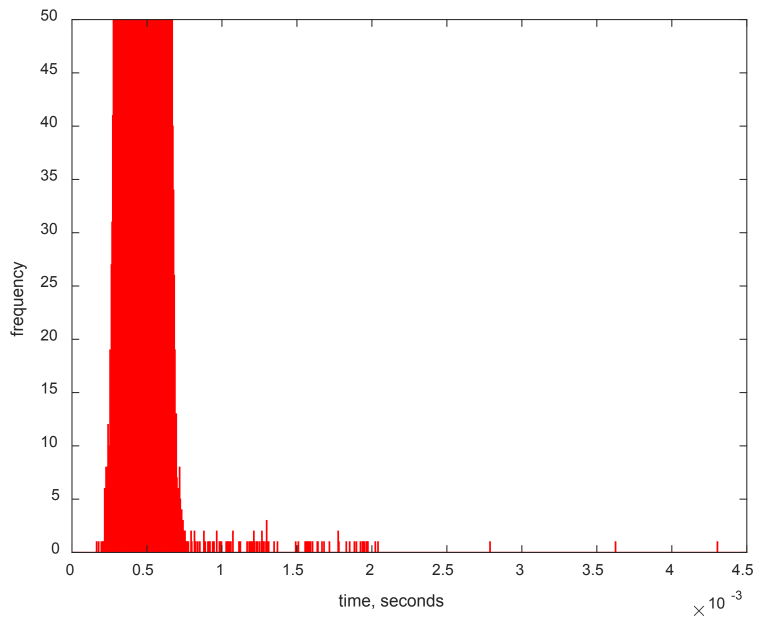
The y-axis is truncated to show the detail of outliers.

Figure 7. Histogram of RTTs for bare-metal host.



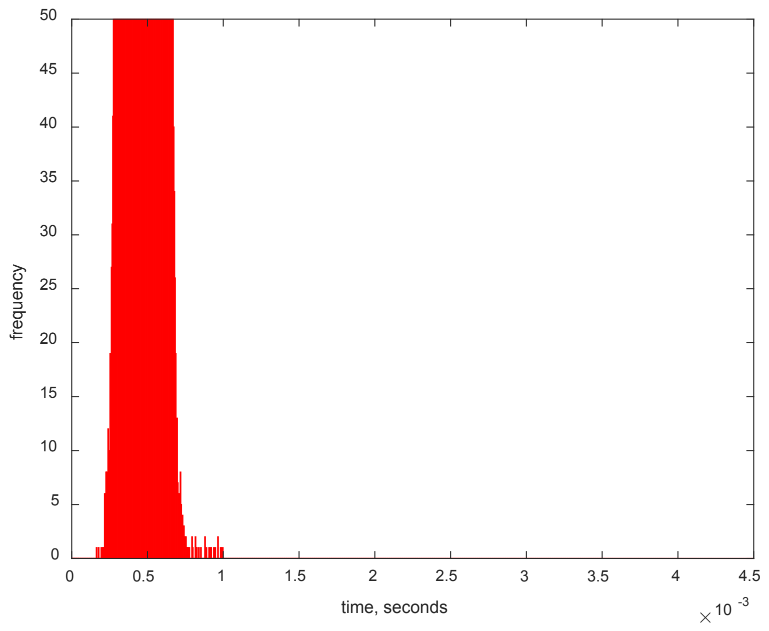
The y-axis is truncated to show the detail of outliers.

Figure 8. Histogram of RTTs for bare-metal host, extreme outliers removed.



The y-axis is truncated to show detail of outliers.

Figure 9. Histogram of RTTs for VM pings.



The y-axis is truncated to show the detail of outliers.

Figure 10. Histogram of RTTs for VM pings, extreme outliers removed.

In Figure 11, the histograms of RTTs give a better sense of the extent to which the round-trip times resemble Gaussian distributions while also showing how they deviate from a Gaussian distribution with strong non-central modes in both VM and bare-metal runs. The resemblance to Gaussian distributions means that a Gaussian distribution can be used as a model for the data. The separation of the two histograms, which have almost no overlap, indicates that correct sampling technique and statistical handling should yield a conclusive model for which traffic comes from VMs.

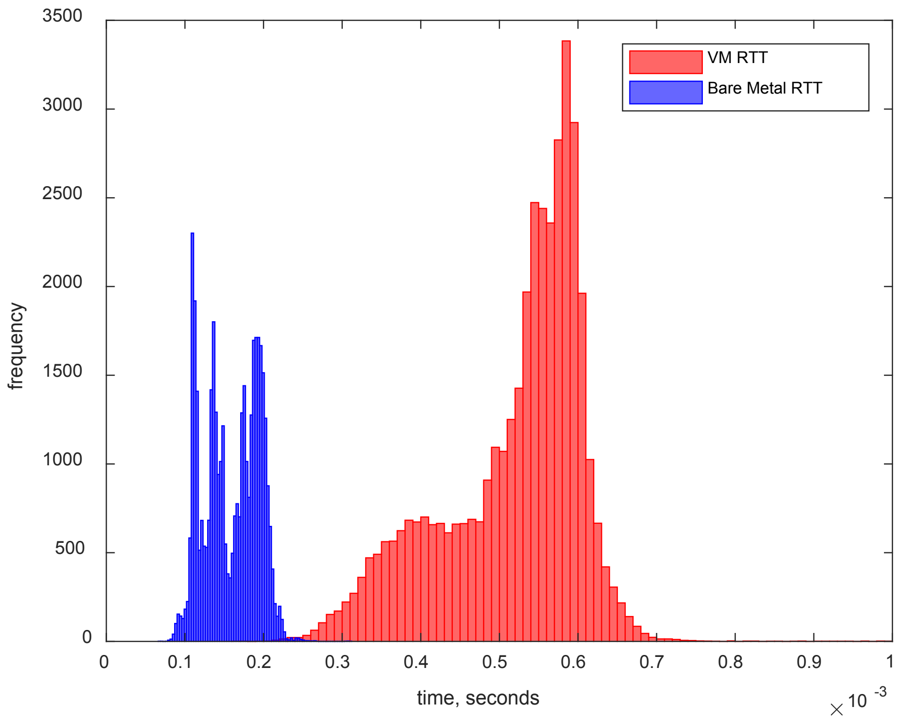


Figure 11. Histograms of RTTs for bare metal and VM.

Though the histograms show deviation from Gaussian distributions, confidence intervals for Gaussian distributions were nonetheless useful for the data with direct connection and measurement of RTTs. A 99.9% confidence solution with a single RTT gives the confidence intervals:

$$\text{Bare metal: } [0.0459 \text{ ms}, 0.273 \text{ ms}]$$

$$\text{Virtual machine: } [0.228 \text{ ms}, 0.810 \text{ ms}].$$

The overlap in these two ranges indicates that a single RTT from an ICMP echo will not suffice to determine whether a given host is virtualized at the intended confidence level. When the sample size is raised, the standard error of the sample is lowered in proportion to the square root of the sample size. When the lower bound of the VM confidence interval is set equal to the upper bound of the bare metal confidence interval at the 99.9% confidence level, a sample size of two is sufficient to provide non-overlapping confidence intervals:

$$\text{Bare metal: } [0.0791 \text{ ms}, 0.239 \text{ ms}]$$

$$\text{Virtual machine: } [0.313 \text{ ms}, 0.725 \text{ ms}].$$

At a conservative 0.1-s separation between pings, a 100-ping sample only takes 10.0 s. Non-overlapping confidence intervals result at the 99.9% confidence level:

$$\text{Bare metal: } [0.148 \text{ ms}, 0.171 \text{ ms}]$$

$$\text{Virtual machine: } [0.490 \text{ ms}, 0.548 \text{ ms}].$$

The lack of overlap in these two ranges means that a 100-ping sample should be able to say definitively whether a host is virtualized. Applying these confidence intervals to 10,000 more RTTs from the same testbed on each of bare-metal and VM hosts, shown in Figure 12, yields 100 sets of bare-metal pings correctly identified out of 100. Using the same confidence interval correctly identifies all 100 sets of pings from a VM. All samples of RTTs from both categories were correctly categorized, indicating that the detection method was successful. The minimal variance of bare metal RTTs results in a narrower confidence interval that includes no RTTs from VMs, and the VM confidence interval is also sufficiently narrow to preclude inclusion of a bare metal RTT.

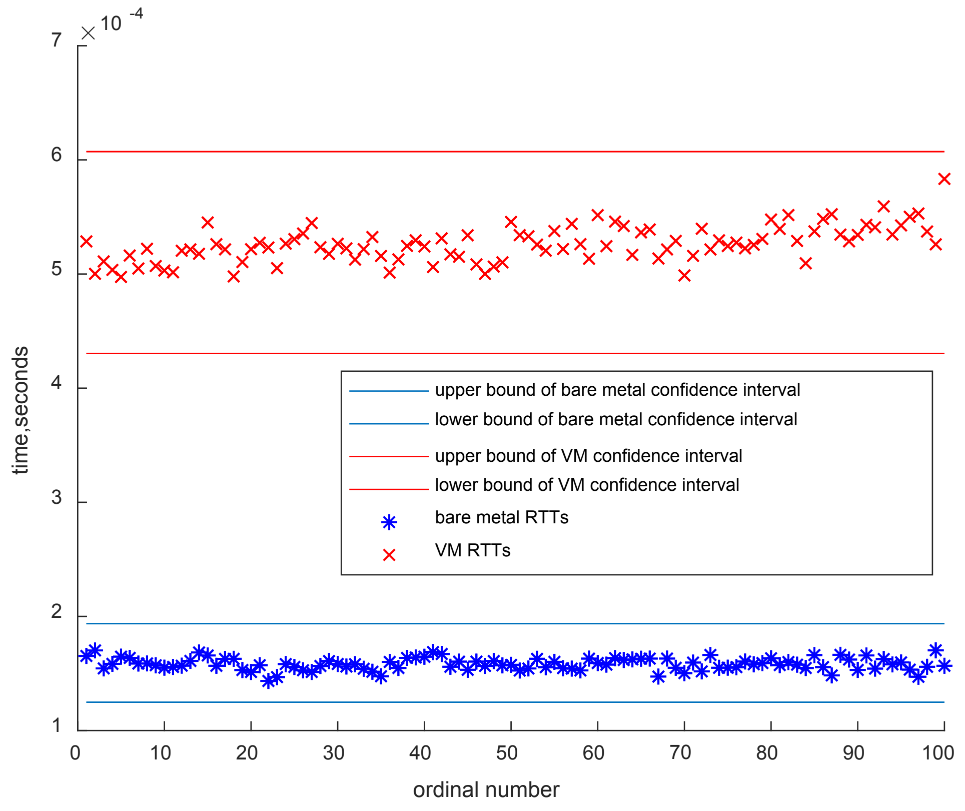


Figure 12. Confidence interval for bare-metal RTTs, with 100-ping samples plotted for VM and bare metal.

B. SDN RESULTS

When the simulation was run on an SDN, it used the setup of an Openflow switch, administered with Ryu as the Northbound API. ICMP pings were sent between two hosts on the network, both directly connected to the Openflow switch. The switch stored no flows so that it would be forced to send its packets to the controller for analysis and routing at each packet forwarding step. In this way, the packets were all sent to the Openflow controller, where timing data could be collected. Due to the significant delay produced by this method of collection, ICMP echo messages were separated by the normal one-second interval, and only two runs of 500 ICMP echoes were recorded. The first run of 500 pings, each of VM and physical cases, was used to define a model, with the second runs in each category being used to evaluate the model.

For the pings conducted through the SDN, 5-standard-deviation extreme outliers were once again removed. Summary statistics are shown below in Table 2.

Table 2. Summary of SDN ICMP echo timestamps

	Bare Metal		Virtual Machine	
	Full data set	Extreme outliers removed	Full data set	Extreme outliers Removed
N	500	499	500	498
Mean (seconds)	0.001120	0.001113	0.001319	0.001315
Standard deviation (seconds)	1.6906×10^{-4}	5.9323×10^{-5}	7.5513×10^{-5}	5.1284×10^{-5}
Skewness	18.5559	5.9910	7.2937	-0.4197
Kurtosis	384.115	68.297	110.109	3.743

Once again, removing the extreme outliers yields data sets much closer to Gaussian distributions and only slightly reduces the number of RTTs included. In the case of pings to the bare-metal host, there is significant skew, and the distribution is too narrow and high-peaked (indicated by high kurtosis value) to fit a Gaussian distribution. For simplicity of analysis, confidence intervals of Gaussian distributions were used again.

From Figure 13, a greater degree of overlap can be seen between the two versions' RTTs in the SDN testbed than was observed in the simpler setup. The overlapping region includes much more of the RTTs for both bare metal and VM cases, indicating that a random VM RTT could be harder to distinguish from a bare metal RTT than in the previous experimental setup. The similarity of the RTTs can be seen again in Figure 14; distinguishing between VM and bare metal RTT by a single RTT is clearly not possible in this instance. Nonetheless, the use of multiple-ping series allowed the distributions to be narrowed to the point where definitive results were possible.

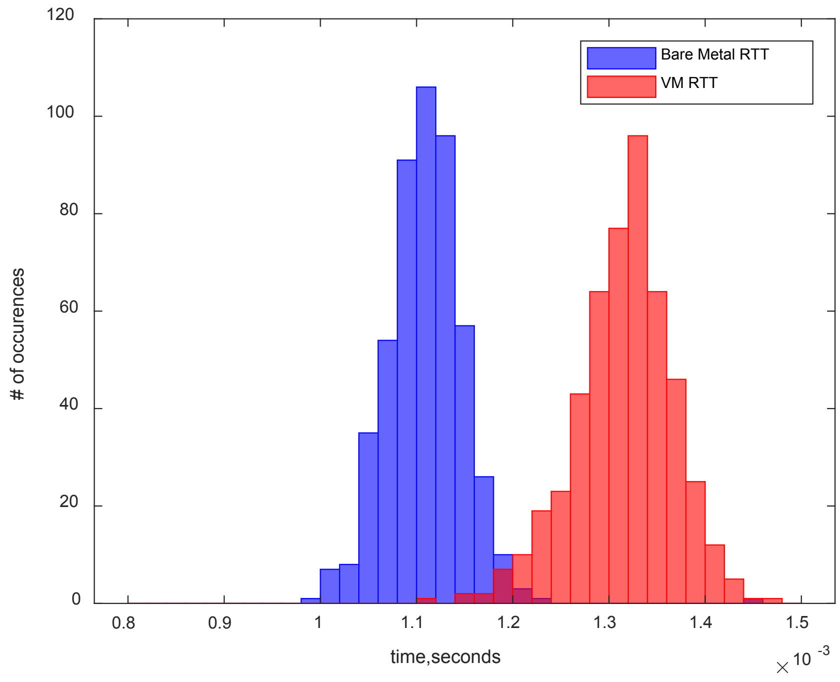


Figure 13. Histograms of bare-metal and VM RTTs on SDN, extreme outliers removed.

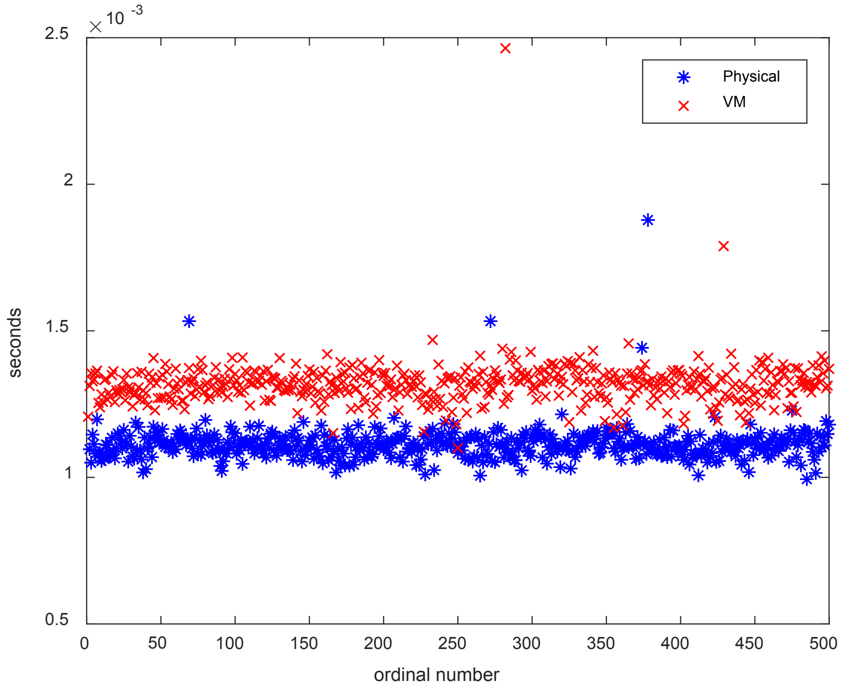


Figure 14. Length of all RTTs for ICMP echoes in SDN.

When the same strategy of producing non-overlapping confidence intervals for VM and bare-metal RTTs was revisited, this time in the SDN environment, 99.9% confidence intervals for a single RTT are

$$\text{Bare metal: } [0.918 \text{ ms}, 1.31 \text{ ms}]$$

$$\text{Virtual machine: } [1.15 \text{ ms}, 1.48 \text{ ms}].$$

The overlapping regions show that single-ping samples are ambiguous for determination of whether there was virtualization. When the lower bound of the VM confidence interval is set equal to the upper bound of the bare metal confidence interval at the 99.9% confidence level, a sample size of four is sufficient to provide non-overlapping confidence intervals:

$$\text{Bare metal: } [1.02 \text{ ms}, 1.21 \text{ ms}]$$

$$\text{Virtual machine: } [1.23 \text{ ms}, 1.40 \text{ ms}].$$

The use of 100-RTT samples again gives unambiguous 99.9% confidence intervals:

$$\text{Bare metal: } [1.09 \text{ ms}, 1.13 \text{ ms}]$$

$$\text{Virtual machine: } [1.30 \text{ ms}, 1.33 \text{ ms}].$$

Evaluation of the remaining 500 RTTs for each of the VM and bare-metal host, plotted in Figure 15, shows these confidence intervals to encapsulate all bare-metal samples correctly, while no VM samples are permitted to pass for a bare-metal run.

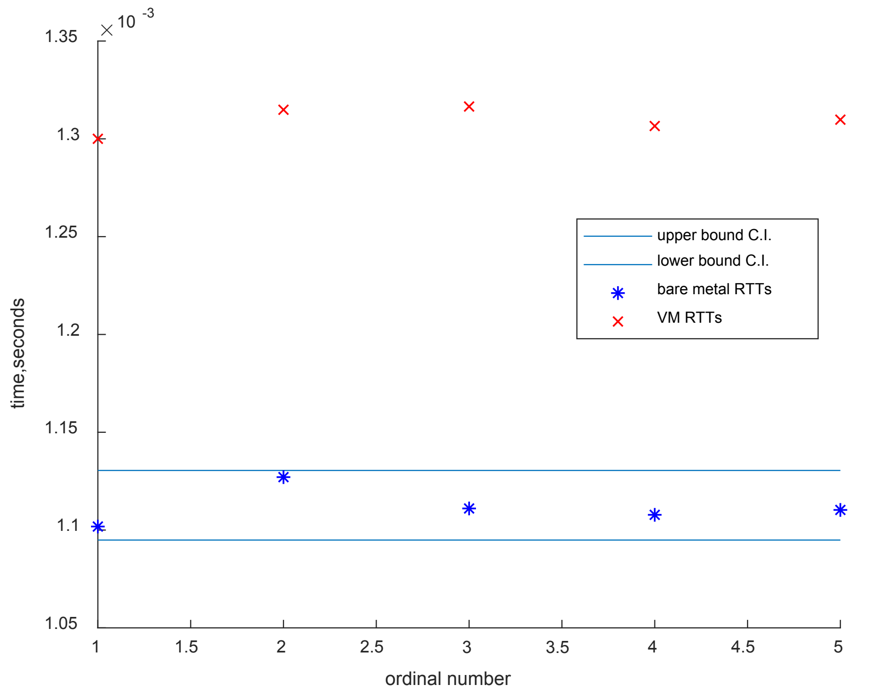


Figure 15. Confidence interval for bare-metal RTTs, with 100 ping samples plotted.

In both the simply connected and separately monitored SDN cases, it was possible to determine which samples were from the VM and which were from the bare-metal host when a Gaussian distribution is used. In both cases, the distinction was achieved with 100-ping samples, which could be considered an acceptable amount of additional traffic to generate for periodic verification of many machines on a network.

The experimentation in this chapter validated a concept in that it is possible to set confidence intervals for all data sets that encompassed all pings of their respective categories while being mutually exclusive. The data had several shortcomings that made the experiment unrealistic which are discussed in Chapter V. Some of those experimental limitations aided the task of categorization compared to a real-world scenario, whereas others made the experimentation unrealistically difficult.

V. CONCLUSIONS

Experimentation confirmed the ability of the proposed method to detect the delay imposed by a VM hypervisor in an SDN environment. The results also show how larger samples could be used to achieve this this goal even as the hypervisor delay is reduced.

Several approaches are suggested for future work. These approaches would expand the applicability of the conclusions reached in this thesis. The applicability could be expanded both by iterative future work exploring many possibilities of network and host configuration and use and by generalization and mathematical modelling of the delays observed.

A. SIGNIFICANT RESULTS

The results of this experimentation showed that VM traffic can be distinguished from traffic originating on a bare metal host. In an ideal measuring environment, where the originating system measures the RTTs, this distinction is made with no ambiguity in the results. This distinction was demonstrated in the first experimental setup. In an SDN environment where RTTs are measured by a controller inspecting traffic at a switch, the difference exists between bare metal and VM RTTs, but a greater number of RTT measurements is needed to conclusively distinguish between the two cases. Finally, the low sample sizes needed for the experimentation, coupled with the low cost of large samples, shows significant potential for detection in situations with less delay or more network jitter.

An ideal measurement was given by two hosts connected to a common switch with timing measurements taken at the host sending the ICMP echo messages and receiving the replies. The situation was further optimized for successful detection by the use of VirtualBox for the VM case; as discussed in Chapter I, VirtualBox introduces more delay than many other virtualization products. In this ideal case, a sample size of only two RTTs was needed to separate VM from bare metal RTTs at the 99.9% confidence level.

An SDN environment was given by two hosts connected to a common OpenFlow switch with a controller, directly connected to the same switch, interrogating the traffic at the switch to measure RTTs. In this environment, the mean RTTs were significantly slower due to processing being performed in the controller, and the standard deviation for bare metal RTTs rose by 72%, though the standard deviation for VM RTTs decreased by 42% compared to the ideal-measurement environment. In this SDN case, a sample size of four was needed to distinguish between VM and bare metal RTTs at the 99.9% confidence level.

The sample sizes needed in the two versions of the experiment were two and four for the ideal and SDN cases, respectively, to achieve separate confidence intervals between bare metal and VM cases. The low cost of simple traffic such as ICMP echo messages makes large sample sizes feasible on production networks, especially if testing for virtualization is infrequent. This potential for larger sample sizes means that detection would continue to be possible by the method in this thesis even for virtualization software that adds significantly less delay.

B. FUTURE WORK

In this thesis, we demonstrated a concept for VM detection, but many permutations of the problem remain unproven for this detection scheme. Future work could determine the extent to which this technique is applicable to varied systems. Particularly, it could be tested against faster virtualization software or generalized to provide an estimate for the fastest VM that could be detected with the proposed technique. More complex environments, both in terms of the connections and devices on the network and in terms of how heavily the devices are utilized, could be used for assessing this technique. Finally, refinement of the statistical model used to describe virtualization delay could result in more complete isolation and understanding of the delays.

First, the virtualization software used for the experiment, VirtualBox, is not strongly optimized for latency, and experimentation with VirtualBox might not align with the difficulties of detecting a lightweight virtualization layer, whether it be purpose-built or just faster software, such as latency-centric server virtualization clients [6]. VirtualBox is a hosted VMM, as opposed to a bare-metal VMM running directly atop the hardware; bare-metal VMs give consistently higher performance than hosted VMMs [6]. This finding is corroborated by [7], which also gives an example of a VMWare product with typical added latencies on the order of tens to hundreds of microseconds. The detection of such a short delay might require a more finely tuned process; nonetheless, it may be attainable when an analysis plane built atop OpenFlow is used. Future work could be done to generalize the variance of RTTs so that a limit could be hypothesized for the minimum virtualization delay detectable by this technique.

The simple network topography used for this experiment might be optimistically reductive. The experiment was performed on a network where no more than a single switch ever sat between the two hosts. Depending on the setup of a network, it may not always be feasible to collect timing data at a machine connected to the same switch. As the number of hops increases, the evaluation will tend to become less reliable at determination since the jitter from the extra hops will partially obscure the underlying timing difference. While proximity to the hosts being evaluated, in terms of network topography, is one of the advantages unlocked in this scenario by SDN, the network used for experimentation was an ideal case, and real-life versions would be more complex. Future work could be directed toward determining how well the detection technique performs when multiple switches lie between the two hosts.

Other processes running on computers could cause legitimate, bare-metal hosts to show delays comparable to those from virtualization, with a false positive as a result. In the rest of the discussion, the network is presumed to be the limiting factor in providing a clear signal for analysis. Processors can be subject to similar queueing in the face of high utilization, and this effect would obscure any virtualization delay. This processing delay could be even more difficult to work around than the network delays, since CPUs are so much more complex and varied than forwarding hardware in switches. Future work could focus on how RTTs and variance of RTTs vary with both CPU and network utilization.

Finally, identification of the correct distribution model would allow the model to more closely mirror the data. Though the Gaussian distribution provided an approximate fit after removing extreme outliers, [21] has indicated that other distributions exist which more accurately portray network and processing delays. Through the combination of high-accuracy modelling with smaller granularity of models (to encompass the variety of delays that comprise a single round trip), it could be possible to achieve reliable differentiation with small samples, even while robustness against lightweight virtualization software is shown. Future work could be done to assess for a better statistical model or pursue a machine learning model.

LIST OF REFERENCES

[1] R. E. Joyce, “Disrupting nation state hackers,” presented at USENIX Enigma 2016, San Francisco, CA, Jan. 27, 2016. [Online]. Available: https://www.usenix.org/node/194636

[2] N. A. Quynh, “Operating system fingerprinting for virtual machines,” presented at DEFCON 18, Las Vegas, NV, July 30, 2010.

[3] O. Bazhaniuk, Y. Bulygin, A. Furtak, M. Gorobets, J. Loucaides, and M. Shkatov, “Reaching the far corners of MATRIX: generic VMM fingerprinting,” presented at Source Security Conference, 2015. [Online]. Available: www.c7zero.info/stuff/source-seattle-2015-generic_vmm_fingerprinting.pdf

[4] D. Kreutz, P. E. Verissimo, and S. Azodolmolky, “Software-defined networking: a comprehensive survey,” in *Proceedings of the IEEE*, vol. 103, no. 1, Jan. 2015. [Online]. Available: https://ieeexplore.ieee.org/document/6994333/

[5] J. Rutkowska, “Red pill... or how to detect VMM using (almost) one CPU instruction,” Wayback Machine, November 2004. [Online]. Available: http://web.archive.org/web/20110726182809/http://invisiblethings.org/papers/redpill.html

[6] W. Graniszewski and A. Arciszewski, “Performance analysis of selected hypervisors (virtual machine monitors – VMMs),” *International Journal of Electronics and Telecommunications*, vol. 62, no. 3, pp. 231–236, Sept. 2016. [Online]. Available: https://content.sciendo.com/view/journals/eletel/62/3/article-p231.xml

[7] S. Agarwal. “Network I/O latency on VMWare Vsphere 5,” VMWare, Palo Alto, CA, 2012. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/network-io-latency-perf-vsphere5-white-paper.pdf

[8] S. Nanda and T. Chiueh, “A survey on virtualization technologies,” Department of Computer Science, SUNY Stony Brook, Stony Brook, NY, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.371&rep=rep1&type=pdf

[9] *Intel 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*, Intel, Santa Clara, CA, 2018. [Online] Available: https://software.intel.com/sites/default/files/managed/7c/f1/253668-sdm-vol-3a.pdf

[10] G. White, Security+ Certification All-in-One Exam Guide. New York: McGraw-Hill, 2003.

[11] *Transmission Control Protocol*, RFC 793, 1981. [Online]. Available: https://tools.ietf.org/html/rfc793

[12] *Internet Protocol*, RFC 791, 1981. [Online]. Available: https://tools.ietf.org/html/rfc791

[13] *A Border Gateway Protocol (BGP)*, RFC 1105, 1989. [Online]. Available: https://tools.ietf.org/html/rfc1105

[14] *An Ethernet Address Resolution Protocol*, RFC 826, 1982. [Online]. Available: https://tools.ietf.org/html/rfc826

[15] P. Lapukhov, "Understanding BGP convergence," INE Blog, 22 November 2010. [Online]. Available: http://blog.ine.com/2010/11/22/understanding-bgp-convergence/

[16] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *ACMQueue*, vol. 11, no. 12, Dec. 2013. [Online]. Available: https://queue.acm.org/detail.cfm?id=2560327

[17] M. Zalewski, "OS fingerprinting tool," Bugtraq Mailing List. 10 June 2000. [Online] Available: http://seclists.org/bugtraq/2000/Jun/141

[18] T. Liston, E. Skoudis, "On the cutting edge: thwarting virtual machine detection," presented at SANS@Night, 2006. [Online]. Available: https://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf

[19] "Conficker's virtual machine detection," Naked Security by Sophos, 27 March 2009. [Online]. Available: https://nakedsecurity.sophos.com/2009/03/27/confickers-virtual-machine-detection/

[20] P. Defibaugh-Chavez, R. Veeraghattam, M. Kannappa, S. Mukkamala, and A. H. Sung, "Network based detection of virtual environments and low interaction honeypots," in *Proceedings of the 2006 IEEE Workshop on Information Assurance*, West Point, NY, 2006. [Online]. Available: https://www.cs.nmt.edu/~rbasnet/research/DetectingHoneypots.pdf

[21] Y. Chen, "Mathematical modelling of end-to-end packet delay in multi-hop wireless networks and their applications to QoS provisioning," Ph.D. dissertation, Department of Electronic and Electrical Engineering, University College London, London, U.K., 2013. [Online]. Available: http://discovery.ucl.ac.uk/1415093/1/yuchen_phd_thesis.PDF

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California