3-22-2018

# Quality of Service Impacts of a Moving Target Defense with Software-defined Networking

Samuel A. Mayer


# QUALITY OF SERVICE IMPACTS OF A MOVING TARGET DEFENSE WITH SOFTWARE-DEFINED NETWORKING

THESIS

Samuel A. Mayer, 2d Lt, USAF

AFIT-ENG-MS-18-M-045

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

***AIR FORCE INSTITUTE OF TECHNOLOGY***

**Wright-Patterson Air Force Base, Ohio**

AFIT-ENG-MS-18-M-045

# QUALITY OF SERVICE IMPACTS OF A MOVING TARGET DEFENSE WITH SOFTWARE-DEFINED NETWORKING

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Samuel A. Mayer, B.S. C.S.

2d Lt, USAF

March 2017

AFIT-ENG-MS-18-M-045

# QUALITY OF SERVICE IMPACTS OF A MOVING TARGET DEFENSE WITH SOFTWARE-DEFINED NETWORKING

THESIS

Samuel A. Mayer, B.S. C.S.
2d Lt, USAF

Committee Membership:

Barry E. Mullins, Ph.D., P.E.
Chair

Timothy H. Lacey, Ph.D., CISSP
Member

Michael R. Grimaila, Ph.D., CISM, CISSP
Member

## Abstract

Computer networks face continual attacks from adversaries that devise innovative ways to achieve their goals. An adversary often conducts careful reconnaissance and scanning of the target network to amass actionable information before they launch an attack. IP addresses are one form of information sought by adversaries. A defensive method that uses a Moving Target Defense (MTD) to change the perceived IP addresses of hosts on a network attaches an expiration date to information from the intelligence-gathering phases of an attack. To counter this, an adversary must act fast and introduce a higher chance of committing an error, or scan more often which increases their visibility.

Technical challenges with MTD pose obstacles for those who wish to use this technique on their networks. Software-Defined Networking (SDN) provides network engineers with a flexible way to determine network behavior, which overcomes some of these technical challenges. One form of a SDN MTD is Random Host Mutation (RHM), which assigns hosts a Virtual IP address (vIP) to pair with their Real IP address (rIP). At a given interval, these rIP:vIP mappings "mutate" and link a different vIP to the same rIP.

RHM is an offshoot of a proof-of-concept implementation in a simulation network from researchers at the University of North Carolina (UNC). While this research did not include statistical analysis, research by Aust in 2017 at the Air Force Institute of Technology (AFIT) confirms the defensive benefits of this technique through experiments on actual hardware with statistically-significant research. With a proven defensive technique in hand, the impact of RHM on Quality of Service (QoS) for legitimate network users is the next area of interest for network engineers. This research confirms previous work in the research area with validation experiments. QoS experiments use a test network similar to the one in past AFIT research, which supports a simulated adversary and servers for several protocols in common use. Test scripts generate packet captures of network traffic for later analysis of legitimate user and adversary actions. Conclusions about the efficacy of RHM on adversary

actions and QoS stem from this data.

Results confirm the defensive benefits of RHM against scanning actions by the comparison of total *perceived hosts* with both quick and intense network scans by a simulated adversary. T-tests compare scan times and total *perceived hosts* versus total *actual hosts* for quick and intense scans. These tests, done at the 99% confidence level ($p < 0.01$), reveal a statistically-significant difference in both scan time and number of hosts found. As a result, Aust's claims of the defensive benefits of a SDN-based MTD are valid.

Of the seven protocols under test in QoS trials (File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), Internet Message Access Protocol (IMAP), Post Office Protocol (POP), Real-time Transport Protocol (RTP), Simple Mail Transfer Protocol (SMTP), and Secure Shell (SSH)), FTP does not function in a RHM-enabled network and HTTP displays anomalous behavior that creates up to 14.2 times as much network traffic during mutation connections versus control trials. RTP shows an increase in jitter of 128 ms that, depending upon the requirements of applications using this protocol, may not be acceptable. IMAP, POP, SMTP, and SSH display some differences from control studies with an average overhead latency increase of no more than four milliseconds after accounting for outliers. While RHM introduces a meaningful impact on QoS, the scale of this impact may be small enough that the defensive benefit justifies the cost on a case-by-case basis. The results of this thesis serve as a next-step in application of RHM to real-world networks such as enterprise settings.

## Table of Contents

Page

Page

Page

## List of Figures

Figure Page

## List of Tables

## List of Acronyms

**AFIT** Air Force Institute of Technology

**API** Application Programming Interface

**ARP** Address Resolution Protocol

**ASIC** Application-Specific Integrated Circuit

**bps** Bits per second

**C2** Command and Control

**CPU** Central Processing Unit

**DCAN** Decentralized Control of ATM Networks

**DDoS** Distributed Denial of Service

**DNS** Domain Name System

**DoS** Denial of Service

**FQDN** Fully Qualified Domain Name

**FTP** File Transfer Protocol

**GB** Gigabyte

**Gbps** Gigabits per second

**GRE** Generic Routing Encapsulation

**HTTP** Hypertext Transfer Protocol

**ICMP** Internet Control Message Protocol

**ICS** Industrial Control System

**IDS** Intrusion Detection System

**IMAP** Internet Message Access Protocol

**IoT** Internet of Things

**Kbps** Kilobytes per second

**MAC** Media Access Control

**MB** Megabyte

**Mbps** Megabits per second

**MoE** Margin of Error

**MTD** Moving Target Defense

**NAT** Network Address Translation

**NETCONF** Network Configuration Protocol

**NIC** Network Interface Card

**OPEX** Operation Expenses

**OS** Operating System

**OTSDN** Operational Technology SDN

**OWASP** Open Web Application Security Project

**PHM** Proactive Host Mutation

**POP** Post Office Protocol

**QoS** Quality of Service

**RAM** Random Access Memory

**RHM** Random Host Mutation

**rIP** Real IP address

**RTP** Real-time Transport Protocol

**RTT** Round Trip Time

**SCADA** Supervisory Control and Data Acquisition

**SDN** Software-Defined Networking

**SHA-1** Secure Hash Algorithm 1

**SMTP** Simple Mail Transfer Protocol

**SSH** Secure Shell

**TCP/IP** Transmission Control Protocol/Internet Protocol

**TCP** Transmission Control Protocol

**UDP** User Datagram Protocol

**UNC** University of North Carolina

**vCPU** Virtual CPU

**vIP** Virtual IP address

**VoIP** Voice over IP

**WAN** Wide-Area Network

QUALITY OF SERVICE IMPACTS OF A MOVING TARGET DEFENSE WITH SOFTWARE-DEFINED NETWORKING

# I. Introduction

This thesis refines a security measure against network scans with a Software-Defined Networking (SDN) controller and an OpenFlow-capable switch. This chapter gives context for the research problem of interest and an approach to gather meaningful data. Finally, this chapter states research goals along with assumptions and limitations for this line of research.

## 1.1 Background

Computer networks are a vital part of modern society. Their applications for economic, personal, and industrial purposes revolutionize several aspects of life. Networked systems have a similar effect and are a staple of modern life in most nations. Those who wish to inflict harm on companies, individuals, or nations recognize this new means of attack and develop exploits to achieve destruction, degradation, or Denial of Service (DoS) of these systems. The attack surface these systems present results in efforts by security professionals to slow down or deny adversary actions. Adversaries counter these efforts with new techniques to get around these defensive measures, which creates a cat-and-mouse game between attacker and defender.

SDN is an emerging form of network operations for computer networks. In this architectural paradigm, the separation of control and data planes allows a central network intelligence and an abstraction of the underlying infrastructure (e.g., switches or routers) for applications. The OpenFlow protocol is one instance of SDN that allows for a vast number of uses on a network. In this configuration, the use of a centralized controller directs the flow of data across the network in a manner that is difficult or impossible in traditional

networking approaches.

Research efforts by the University of North Carolina (UNC), and later on by the Air Force Institute of Technology (AFIT), examine an implementation of a defensive countermeasure that uses SDN to thwart adversaries as they conduct the first steps of their attacks. This technique uses Random Host Mutation (RHM) to shuffle the perception of network asset locations to impede adversary action with a link between a Real IP address (rIP) and a Virtual IP address (vIP). Results from these research efforts are promising, but do not consider the Quality of Service (QoS) implications of this defensive technique.

## 1.2 Problem Statement

There exists proven value of RHMs as a means of Moving Target Defense (MTD) in SDN. Unfortunately, this evidence does not consider the network performance impact on legitimate users; it only examines the challenges adversaries experience. This thesis attempts to confirm results of the defensive benefits of RHM and provide statistically-sound information that addresses the QoS implications of this MTD on a network. Tests focus on a stable network that provides an effective means of defense while enabling experiments that report the QoS legitimate users experience.

## 1.3 Goals and Hypothesis

This thesis builds upon previous work in the SDN-enabled MTD topic area by confirming the benefit of mutations as a MTD and their impact on legitimate users with a test network running services in common use. The framework used to design these tests focuses on stability, effectiveness, and the measurement of QoS metrics. The severity of these QoS impacts is unknown and their assessment is one of the primary motivations for this thesis. The hypothesis for this thesis is that a RHM-network shows statistically-significant differences in network scan times and scan accuracy at the cost of a decrease in QoS of an unknown magnitude as experienced by legitimate users.

## 1.4 Approach

A test network of thirty hosts, one SDN-enabled switch, one adversary machine, and one SDN controller provide the basis for experiments. Experiments measure the duration of adversary and legitimate actions with internal clocks on the hosts. Adversary measurements focus on the duration of network scans and number of *perceived hosts* that result from network scans. Legitimate user actions create traffic with protocols in common use and report information about the throughput, reliability, and performance of those connections. Experiments are run with and without RHMs on the network and record packet captures for later analysis. Comparison of the data from packet captures indicate if the impact on adversary or legitimate users is statistically significant.

## 1.5 Assumption and Limitations

Assumptions and limitations enable proper interpretation of the results and keep experimentation focused on the research goals. This thesis applies the following assumptions:

1. To replicate Aust's study, the adversary scans from inside the network.
2. All target hosts run Windows XP Service Pack 2 to ensure they are vulnerable to the adversary.
3. The adversary does not attempt to exploit network infrastructure, nor the servers that host the protocols under test.
4. Legitimate users know the vIP address of the network assets that support the protocols under test.
5. Due to identical target hosts, the adversary can exploit a random machine from the list of scan results instead of the same host every time.
6. While dedicated adversaries can identify hosts based upon Media Access Control (MAC) addresses, the simulated adversary does not. Some exploits launched through Metasploit require an IP address and do not function without this information [1].

Since a rescan of the network must occur to supply the exploit with a current vIP address of the target host, the system still provides a layer of shifting obfuscation.

Research in this thesis has four main limitations:

1. Identification of hosts may occur via characteristics other than IP address (e.g., open ports, MAC addresses, Operating System (OS) version).

2. RHMs sacrifice the impact on duration of adversary access after a successful attack from Aust's work to ensure broader network usability that maintains connections across mutations. This trade-off allows for lengthier connections, which overcomes a limitation of previous research and increases usability of the network.

3. The vIP of a given host is known by the client that attempts to open a connection to it. In experiments, the tester examines the mutation table on the SDN controller and supplies the correct vIP to the test script which creates connections between the client and server. A real-world system must automatically update clients with current rIP:vIP mappings.

4. Wireshark collects QoS with built-in tools. Other tools to measure QoS may exist that provide better data.

Chapter VI discusses these limitations in greater detail as areas of future work.

## 1.6 Thesis Overview

This thesis addresses the defined research area in six chapters. Chapter II defines Software-Defined Networking, network attack, the concept of a moving target defense, presents a case study where MTD provides a clear benefit, defines random host mutation as a form of MTD, and reviews previous research efforts. Chapter III details the framework used in tests and provides a detailed description of how each component fits together. Chapter IV discusses the experimental process, and the results of experiments are in Chapter V. Chapter VI summarizes the research, and explains the significance of this research and future work contribution to the SDN body of knowledge.

# II. Background and Related Research

This chapter discusses the information necessary to understand Random Host Mutation (RHM) in Software-Defined Networking (SDN). Section 2.1 discusses the key characteristics and developmental history of SDN. Section 2.2 reviews network attack methodology to provide context for defensive countermeasures. Section 2.3 covers the topic of Moving Target Defense (MTD) and how RHM provides this capability; this concept is the basis of the thesis. Section 2.4 provides an example of how MTD reduces the threat posed by the Mirai botnet. RHMs are described in detail in Section 2.5. Previous research into MTD through SDN is reviewed in Section 2.6.

## 2.1 Software-Defined Networking

As shown in Figure 1, traditional network infrastructure combines the processing logic for network behavior (control plane) and processing of network traffic (data plane) in each network appliance (e.g., switches). Management of larger, more complex network topologies in this way, produce complicated administrative challenges. Software-Defined Networking separates the control and data planes through the use of SDN-capable devices and a centralized controller which defines network behavior for the SDN devices. SDN controllers maintain network state information and interact with control applications and switches via an Application Programming Interface (API). In effect, this becomes a distributed system the emphasizes performance, scalability, fault-tolerance, and robustness. Centralized control allows for easier installation and removal of extra hardware as the logic that determines behavior is located at the controller and not the device that are installed or removed. In this configuration, the centrally-located controller uses a secure channel (shown as a green dotted line) to define switch behavior.

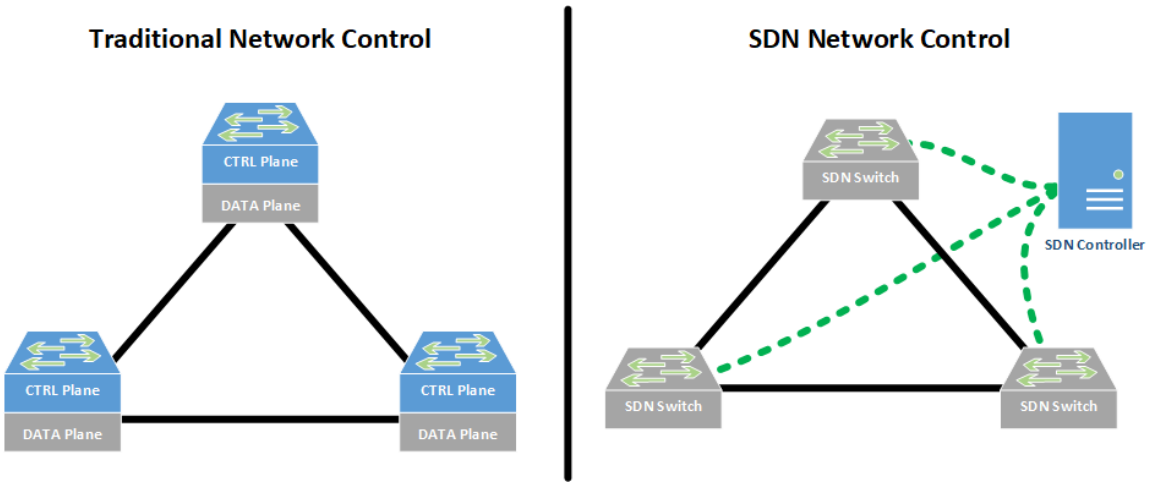



**Figure 1. Separation of Control and Data Planes in Traditional Networks Versus a SDN [2]**

### 2.1.1 SDN Basics.

Four main concepts characterize SDN [2]:

- *Plane Separation*: The control plane determines data plane actions and establishes the rules that govern what *flow table* entries get installed on devices that interact with the data plane. Data kept in these *flow tables* define how data plane devices (e.g., switches) process traffic as it traverses their ingress and egress ports [3]. Data plane devices forward, drop, consume, or replicate incoming traffic based upon rules stored in their *flow tables*. Section 2.1.3 provides examples of these four actions in an example SDN.

- *Simple Devices and Centralized Control*: Complex software that determines network function is stored on the centralized controller, not on switches that handle the data plane.

- *Network Automation and Visualization*: SDN simplifies network operations through abstraction as well as *Northbound* and *Southbound* APIs. As shown in Figure 2, northbound APIs interface with software applications that interact with the controller. Southbound APIs govern the interface between controllers and network devices. This concept is similar to how high-level programming languages make programming more

accessible through layers of abstraction.

- *Openness*: Software engineering approaches such as agile development have shown their value in rapid creation of prototypes and a mindset that embraces change. The flexible nature of SDN enables the development of networks that facilitate research, experimentation, and vendor interoperability to lower consumer cost and support rapid innovation. This design goal allows for faster development of and changes to network behavior.

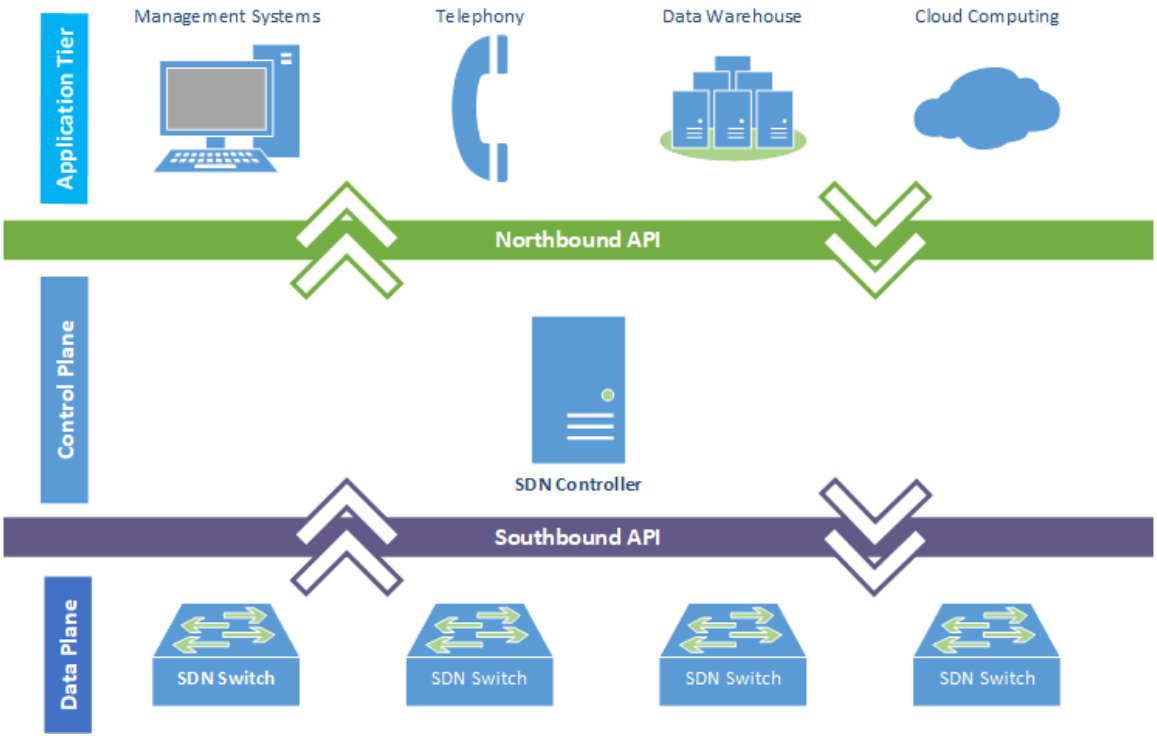



**Figure 2. Northbound and Southbound APIs in a SDN [4]**

The SDN controller provides the conduit between the network programmer and the nuances of network functionality. Controllers manage topologies, flows, discover devices, and gather locally-stored statistics. The modular nature of the Northbound and Southbound APIs minimizes differences between SDN devices due to the standardized interface. SDN devices use the Southbound API to receive updates to their flow tables. The highest priority match between the packet and a given rule in the flow table dictates how to process network traffic. The concept of processing network traffic through examination of its characteristics

is not a novel concept. For example, firewalls have allowed network operators to reject packets through criteria such as IP addresses and source or destination ports. However, the generic flow tables and flexible logic in SDN are what make this technology versatile for packet processing.

### 2.1.2 SDN versus Traditional networking.

To understand the benefits of SDN, the key differences between SDN and traditional network management deserve an overview. The Open Networking Foundation notes that current networking technology without SDN suffers from four main limitations [2]:

- *Stasis*: Increases in complexity from highly-interconnected control and data plane hardware present obstacles to ensure stability at scale and inhibit innovation.
- *Inconsistent Policies*: Uniform adoption of new control plane policies does not always occur since each control element need individual configuration.
- *Poor Scalability*: Growing networks require more time from administrators to properly configure. Automation from SDN due to vendor-independence reduces Operation Expenses (OPEX).
- *Vendor Dependence*: Vendor-specific interfaces and configurations create dependencies on third parties and complicate the role of administrators, especially if hardware from multiple vendors is in use. Interoperability challenges can lead to vendor lock-in due to past purchasing decisions.

The separation of control and data plane is not a groundbreaking concept, as modern switches that do not support SDN still have Application-Specific Integrated Circuits (ASICs) that communicate with general-purpose Central Processing Units (CPUs) to handle control plane messages or traffic without defined rules [3]. The distinction lies in the fact that traditional hardware represents these two planes as tightly-woven entities within the same physical hardware. This in turn led to more complex machines which increase development costs and contribute to stagnant network technologies.

The architecture of SDN adapts to large networks due to the separation of control and data planes. Separation allows the determination of fast routes without taxing the same hardware that must forward network traffic. Once new optimal routes are calculated by the controller's processing logic, the rules that govern the SDN devices propagate from the controller to the SDN devices. This process is done through a live update to SDN devices without service interruptions. The global view of a controller also allows for link state algorithms such as Dijkstra's algorithm to compute optimal routes. Since SDN uses well-defined API calls, the limitation of vendor independence becomes less of a factor as SDN-capable hardware must perform to a standard specification such as OpenFlow. Changes in network policy conducted at the controller-level propagate out to infrastructure devices. This overcomes the traditional network limitation of inconsistent policies.

### 2.1.3 Example SDN Configuration.

This section provides an example of how all the components in a SDN come together to process network traffic. Figure 3 illustrates a network consisting of one SDN controller and three SDN switches along with the basic actions that SDN devices can take. The controller has a global view of the network in its memory and can interface with the switches using calls across the southbound API to update the flow information kept on each switch through a communications protocol such as OpenFlow (Section 2.1.5). As switches receive network data, they perform one of four actions: Forward, drop, consume, or replicate [2]. Forwarded packets are processed based upon a matching flow in the *flow table*. Dropped packets can result from specific filtering. Packets are consumed when they require additional processing by the control plane. For example, if a packet did not have a matching filter, the data plane device uses its isolated channel with the controller to send the packet for further processing [2]. Replication of packets is a special case of forwarding where packets are sent out across multiple egress ports.

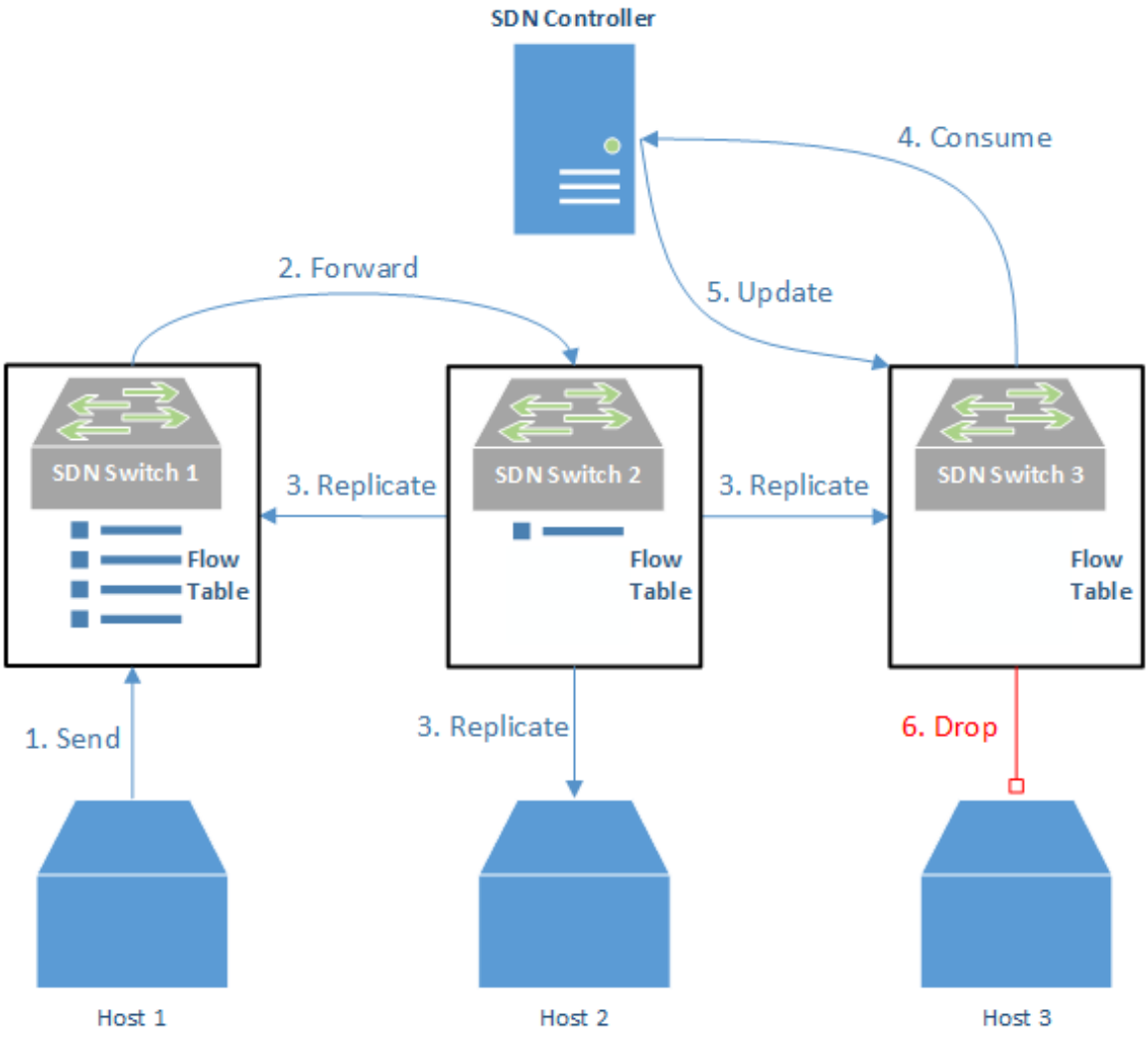



**Figure 3. Demonstration of SDN Actions**

Messages that pass between hosts and SDN devices in Figure 3 demonstrate all actions that a SDN switch can take and an update from the controller in six steps. As a start condition, SDN Switch 1 has several flows installed on it to process traffic from Hosts 1, 2, and 3. SDN Switch 2 has only one flow which states that it must replicate any traffic received to all its egress ports. SDN Switch 3 does not have any active flows and must consume any packets to receive instruction from the controller before it takes any actions. The six steps taken contain these actions:

1. *Send* - Host 1 sends a message to Host 2. This message is representative of any network traffic that a traditional network would handle, such as a Transmission Control Protocol (TCP) SYN packet or a Hypertext Transfer Protocol (HTTP) GET request.

2. *Forward* - As the packet reaches SDN Switch 1, the switch examines its flow table and finds a match based upon the characteristics of the message from Host 1 (e.g., source IP address, destination IP address, TCP or User Datagram Protocol (UDP), destination and source ports). The rule in the flow table says that it must forward the message to the egress port associated with SDN Switch 2.

3. *Replicate* - When Host 1's message reaches SDN Switch 2, it finds an single flow table entry which replicates a message of the type that Host 1 sent. A replicate rule is a special type of forward rule so the switch sends Host 1's message out on all of its egress ports (i.e., to SDN switches 1 and 2 as well as Host 2). Host 2 receives Host 1's message. When the replicated message reaches SDN Switch 1, it drops the message according to a rule in the switch's flow table.

4. *Consume* - When the replicated message reaches SDN Switch 3, it does not find a matching entry in the flow table. It now consumes this packet and forwards it to the controller for instruction on how to handle future packets of this type.

5. *Update* - The controller examines Host 1's message and determines that it must be dropped. A rule to drop further packets of this type is installed in the switch's flow table.

6. *Drop* - The packet consumed by SDN Switch 3 is dropped and does not reach Host 3.

With knowledge of desired network behavior and protocol specifics, these core actions allow network engineers to define the behavior of a SDN. By using actions broken into their most basic components, SDN switches can be designed with simplicity in mind. This separation allows network behavior to become a property of the network as defined in the controller. As network requirements change, the core actions taken by switches do not change as they adapt to instructions from the controller through modified flow tables.

### 2.1.4 Developmental History.

SDN has its beginnings in the 1990s with four different attempts at network control: Open Signaling, Active Networking, Decentralized Control of ATM Networks (DCAN), and Network Configuration Protocol (NETCONF). In each case, efforts prioritized ways to make networks more programmable and scalable. Open Signaling produced the idea of programmable interfaces similar to the North and Southbound APIs discussed in Section 2.1.1 [4]. APIs for computer networks enable customization of network infrastructure with greater ease. Active Networking used separate channels for data and control planes [5][6]. The ability of SDN controllers to update flow tables to influence network behavior drew from this idea. DCAN separated the control and data planes entirely [6]. This separation technique later became one of the hallmarks of SDNs. In 2006, NETCONF provided a management protocol that allowed network devices to expose an API to exchange configuration data although it did not separate control and data planes [6]. This feature enabled the ease-of-use associated with network management through a SDN controller. Along the developmental timeline of SDN, predecessors such as DCAN, NETCONF, and Ethane provided elements that influenced SDN in its current form [6][7]. A direct predecessor to OpenFlow, Ethane focused on the use of a centralized controller to manage policy and security on a network. Table 1 lists several SDN controllers and provides a brief description for each.

**Table 1. Comparison of Different SDN Controllers [6]**

| **Controller Name** | **Controller Description** |
|---|---|
| NOX | Multi-threaded C++ on top of Boost library [8] |
| POX | Single-threaded Python for rapid prototyping |
| Beacon | Multi-threaded Java using OSGi and Spring frameworks [9][10] |
| Floodlight | Multi-threaded Java using Netty framework [11] |
| MUL | Multi-threaded C based on top of libevent and glib [12][13] |
| Maestro | Multi-threaded Java |
| Ryu | Python based using gevent wrapper from libevent [14] [12] |

These research efforts culminated in the creation of a communication protocol called OpenFlow. In 2007, NOX and POX controllers became publicly-available OpenFlow controllers. Since their release, several other controllers have been released to include: Floodlight, Ryu, Beacon, Mul, and Maestro [7]. These OpenFlow controllers can implement services such as Domain Name System (DNS), firewalls, and Intrusion Detection Systems (IDSs). SDN controllers act as one logical unit so modifications to the network became easier to make. By comparison, a traditional network infrastructure needed individual configuration of disparate components to achieve the same effect.

### 2.1.5 OpenFlow Communications Protocol.

SDN only conceptualizes network behavior. To actually implement these concepts requires a specific communications protocol. A prominent communications protocol in use for SDN is called OpenFlow [15]. This protocol meets the need of researchers to experiment with new network protocols without risking campus network outages. For ease of access, OpenFlow adopts the idea that switches and routers contain *flow tables* used for activities such as Network Address Translation (NAT), Quality of Service (QoS), statistics collection, etc. Three characteristics usually comprise a flow: match fields process incoming traffic, counters collect statistics, and actions define how to handle a packet caught by the match fields [6]. A flow handles network traffic with predefined network rules (i.e., match: IP address x, action: forward to port y). McKeown, Parulkar, et al. highlight the use of OpenFlow on computer networks through six examples that demonstrate the ability to modify the individual flow tables on these switches and routers [2][5][15]. Under OpenFlow, these rules apply on a per-rule basis which allows greater granularity for data prioritization, processing, and transmission across a network. The OpenFlow protocol API made this manual process much faster.

### 2.1.6 Major Users.

SDN is no longer just a novel technology for use by researchers; several large companies use SDN for aspects of their network management. Google, Verizon Wireless, and Dell

all use SDN for tasks ranging from isolated product testing to data relocation across data-centers [16]. Google and IBM recognize the value of SDN and support the Open Networking Foundation [16]. IBM and Cisco also produce OpenFlow-enabled switches, indicating that some of the major players in networking hardware wish to capitalize on the developing market [17][18]. Visualization efforts are underway by Nicira to decouple network information from physical switch hardware in a move similar to how VMware shapes virtual machine management [16]. Each of these major players remain subject to similar security concerns that any enterprise network would face. In addition to current defensive techniques common to computer networks, users of SDN can benefit from the addition of a MTD to their networks.

## 2.2 Network Attack

Regardless of if a network is configured through traditional means or via SDN, the actions of an adversary tend to fall into five sequential steps to achieve their objective. In the case of the Mirai Botnet discussed in Section 2.4, an adversary used poorly-secured devices to launch a crippling Distributed Denial of Service (DDoS) attack at an opportune moment. An understanding of how network attacks are conducted is necessary to see how the flexibility of SDN can enable new defensive techniques that foil adversaries in these phases. Adversaries typically launch their attacks by following the process described in Figure 4: reconnaissance, scanning, gaining access, maintaining access, and covering tracks [19].


**Figure 4. Five Stages of Network Attack [19]**

Reconnaissance and scanning are distinct activities used to gather intelligence about a target network. Reconnaissance focuses on the acquisition of target information. Adversaries can detect all sorts of target information from a plethora of sources. With this

information, targeted attempts to gather intelligence about the target network can occur, often with a higher chance of success. The pervasive nature of the online world has made it difficult to reduce the attack surface that organizations and individuals present to adversaries. Scanning searches for computers and openings in networks for later exploitation by an adversary. In combination with proper reconnaissance, selective examination of weakpoints across the network attack surface occurs. Critical network components such as servers and network infrastructure present high-value targets to adversaries. Poor configuration or use of default settings offers little guarantee to rebuff even script-kiddie attacks.

Having found a target, gaining access through software exploits or social engineering allows an adversary to act with greater freedom. Any level of access, not just that of an administrator, has the potential for great harm. At this point, adversaries establish footholds to fulfill their broader objectives such as data exfiltration. Now that an adversary has access, creation of other processes or programs that allow them to maintain access with less effort if their current form of access fails is common. Throughout this entire process, adversaries attempt to remain stealthy and cover their tracks. IDSs apply a broad range of strategies to detect malicious activity within a network. Statistical analysis of network traffic for sudden high levels of traffic that links to network mapping or connections to strange external locations may indicate malicious activity. As described in Section 2.3, a moving target defense forces adversaries to increase certain types of network traffic during the scanning and maintaining access phases.

The addition of countermeasures after a successful attack offers little solace for the victims, and no single solution resists all attacks by adversaries. A robust strategy must rely upon a defense-in-depth to complement the weaknesses posed by other means of countermeasures. The Open Web Application Security Project (OWASP) advocates for layered security mechanisms to manage risk in a situation where one control could suffice stating, "Controls, when used in depth, can make severe vulnerabilities extraordinarily difficult to exploit and this unlikely to occur" [20]. For example, firewalls must be used in tandem with endpoint anti-virus, network segmentation, and security awareness training for improved security.

The centralized nature of the controller presents a high-value target to adversaries and a single point-of-failure. Therefore, multiple controllers provide redundancy [21]. The inherent flexibility of SDN opens up a spectrum of new defensive measures to protect networks. MTD with SDN increases the challenges adversaries face when they try to launch an attack.

## 2.3 Moving Target Defense (MTD)

From the perspective of an adversary, static network configurations make the task of gaining and maintaining access more tenable. A compromised system with shifting characteristics is harder to successfully exploit than a compromised system with static characteristics. Much like how a randomly zig-zagging target is harder to shoot with a rifle, an equivalent approach to network defense presents additional challenges to an adversary. Even if systems on a network are kept "up-to-date," public vulnerability disclosures occur regularly and advanced threats may have access to zero-day exploits on supposedly robust systems. MTD, by definition, seeks to "create, evaluate, and deploy mechanisms which are diverse, continually shift and change over time to increase complexity and costs for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency" [22]. Anything that adds chaos to the adversary's efforts serves as an effective defensive countermeasure if deemed worth the cost.

Valuable defensive countermeasures must have costs proportional to the cost of the asset they defend in addition to the operational impact of the countermeasure [23]. Research discussed in Section 2.6 concluded that MTD is a useful defensive countermeasure from broad-spectrum attacks [24]. Opportunistic attacks that adversaries launch to establish access on a network may not have the benefit of honed phishing attempts as compared to more targeted attacks. In either case, a defensive measure that constantly morphs the topology an adversary observes constitutes a valuable defense as long as the QoS the end user experiences does not suffer. Figure 5 illustrates the network equivalent of zig-zagging that a MTD provides for a network to impede the scanning efforts of an adversary. Scans that show a printer at IP address 2 would then resolve to a computer after the network topology has shifted. Thus, to target the printer after a mutation, the adversary must

conduct another scan to see the current IP address of its target.

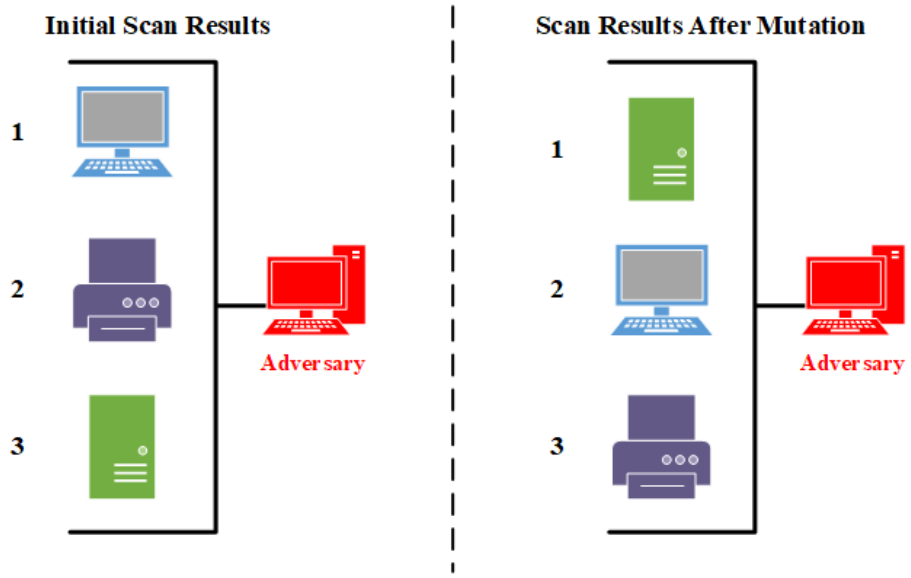



**Figure 5. Concept of a Moving Target Defense**

With respect to networked computers, standard components include MAC addresses, open ports, active services, and IP addresses. Each of these characteristics uniquely identifies hosts on a network. IP addresses are a prime candidate for MTD as they are commonly used to make assets visible on a network. In conjunction with IP addresses, ports (Transport Layer) create a unique 4-tuple between a client and server to establish connections. Media Access Control (MAC) addresses (Data Link Layer) are also used, but protocols such as TCP and UDP ride above their layer of abstraction. For a corporate network, a MTD that changes IP addresses is more practical than changing MAC addresses even though it may not constitute a perfect defense. In a network dealing with sensitive Industrial Control Systems (ICSs), a MTD that focuses on multiple aspects, such as IPs and/or MAC addresses may be justified despite an increase configuration cost. The specifics of a network influence what constitutes a valuable defensive countermeasure.

## 2.4 Mirai Botnet Case Study

One example of an attack with far-reaching consequences that could have been impeded by a MTD was the Mirai botnet attack in October of 2016. Botnets are swarms of networked

computers which leverage their mass to achieve some goal Denial of Service (DoS)) [25]. Technical specifics have evolved over the years, but the result is that botnets are hard to detect and defend against once created. The Mirai attacks carried out a DDoS against Dyn, a major DNS provider, by sending approximately 1.1 terabits of malicious traffic per second. Several attack variants exist to include Generic Routing Encapsulation (GRE), TCP, and HTTP flooding [26]. The service outages resulting from this botnet caused services such as Twitter, Netflix, and Facebook to go offline for several hours.

Figure 6 outlines the 7-step attack process for this botnet [26]. First, a bot searches an addresses range for improperly configured IoT devices and attempts to gain access via brute force. Upon successful discovery of a vulnerable host, the bot reports this information to a report server that keeps track of vulnerable devices. A Command and Control (C2) server then checks for potential victims through the information stored on the report server. Once the C2 server chooses a group of devices to attack, an infection command with the requisite details (i.e., IP addresses and hardware configurations) is configured on a loader which delivers the malicious binary. Once this exploit is installed on the new victim, it can communicate with the C2 server for further instruction. Finally, the C2 server issues attack commands to the newly-formed botnet. This command then causes the bots to perform their DoS.

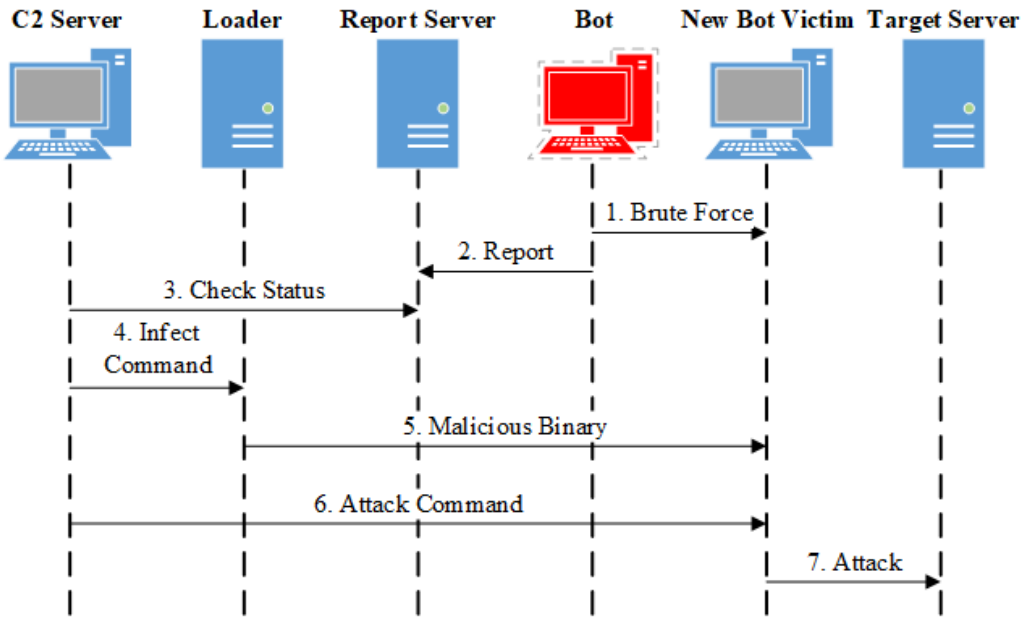



**Figure 6. Mirai Botnet Attack Overview [26]**

In a network using RHMs as a MTD, the Mirai botnet may be able to complete steps 1 and 2 in Figure 6, but when the C2 server initiates step 3 the new bot victim could have a new IP address and would show up as inactive. Next, the C2 server would skip step 4 and move on to a new target in its list. Alternatively, the Report server may indicate an active target in step 3 but a mutation could occur between steps 3 and 4. In either scenario, the reliability of this attack methodology is no longer guaranteed to work due to RHMs which cause the intelligence gathered in the reconnaissance phase to have a shortened lifespan.

In the case of the Mirai botnet, DoS attacks were made possible by poorly secured Internet of Things (IoT) devices and demonstrates the massive attack surface posed by a line of technology so willingly embraced with general disregard toward security implications [25]. Many users of IoT devices are unaware or apathetic to security implications that may cause these devices to be repurposed by adversaries [26]. The anticipated existence of over 100 billion IoT devices by 2025 presents serious security concerns for network operators [25]. Use of a MTD reduces risk posed to this attack surface by disrupting the ability of an adversary to reliably discover and exploit vulnerable devices.

## 2.5 Random Host Mutation

Having defined the value of a MTD, this section addresses the specifics of how a theoretical MTD exists on a computer network. Identification of services is one of the main parts of the scanning phase of the cyberattack methodology. Much like how company-wide password policies introduce a level of variability in user credentials, the concept of a shifting field of "cyber terrain" through RHM at the network level challenges attackers to act within a constrained window of opportunity [27]. Researchers at University of North Carolina (UNC) provide several proof-of-concept proposals that inspire the general application of RHMs [28][29][30]. These research efforts produce systems that are unpredictable, fast, operationally safe, and transparent.

Figure 7 describes a SDN that supports RHM. This network architecture consists of SDN switches, a SDN controller, and hosts connected to the switch. For connectivity to the Internet, standard pieces of infrastructure such as routers may exist. Without a certain level of interoperability, SDN technology does not easily integrate with the well-established networking techniques of at a cost-effective level. Each SDN switch contains a flow table that defines how each particular switch processes network traffic. The flow tables are updated according to information sent by the SDN controller.

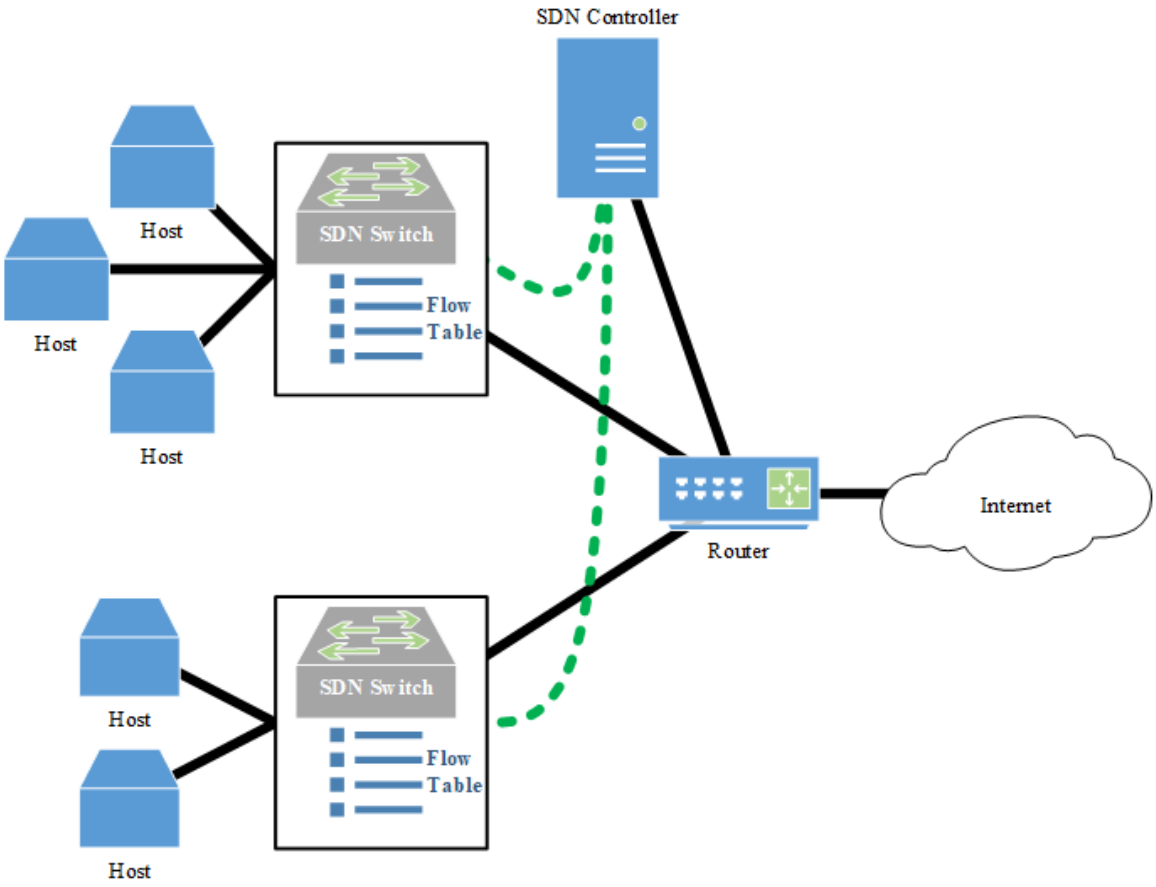



**Figure 7. OpenFlow RHM Architecture [31]**

A scanning adversary may find a node and see that it is at IP address a.b.c.d running services x, y, and z. After a RHM, that same IP address may or may not map to an actual node on the network. If it does, an entirely different set of services could be running on a.b.c.d which forces an additional scan to account for inaccurate network information. Further compounding the complexity for adversaries could be the existence of honeypots that run the same set of services as a legitimate target. Honeypots are targets designed to attract adversaries so that defenders can better understand or entrap them [32]. Intentionally placing weak targets on a RHM network that appear similar to legitimate hosts require adversaries to acquire more identifying information about targets (e.g., MAC addresses or Operating System (OS) Versions) to reacquire them after an IP address mutation.

At its core, the controller to leverages the power of SDN to map the Real IP address (rIP)

of each host to a Virtual IP address (vIP) in an RHM. If end hosts perform this mutation process, numerous synchronization problems would emerge and inconsistent mutation calculations done by each host may render themselves isolated from the network. When a host without a rIP:vIP mapping initiates a connection, the switch examines its flow table for an existing flow and forwards the packet if one exists. In the event that a flow does not exist, the controller creates a new flow for the connection and that information propagates to the switch.

RHM shares similarities with NAT but has a key distinction. While NAT allows multiple devices to share one Internet-routable address, once an adversary gets beyond a NAT there is no longer a defensive benefit to a NAT. RHMs allow the use of a similar construct to add another layer of obfuscation to internal networks which slow down an attacker and force increased scanning activity. This behavior can make an adversary more visible to an IDS, especially since SDN flows are designed to facilitate the collection of statistics related to individual flows. Furthermore, the limited window-of-opportunity constrains adversary actions and may force careless actions.

Figure 8 provides a sequence diagram of how a host sends network traffic to a host on a different subnet in a SDN using RHM. First, Host A wants to send a message to the vIP address of Host B. Additional network management must occur to inform clients of vIPs. One potential way to share this information with the clients is through DNS cache updates that occur in sync with mutations (this idea is discussed in Section 6.4). Second, the SDN switch receives the request and discovers that there is no rule in the flow table to handle this type of traffic. Third, the switch asks the controller how to handle this request. The controller determines that traffic coming from Host A's rIP (10.13.1.5) destined for Host B's vIP (10.13.2.42) must have the source/destination headers modified. Host A's rIP must be replaced with its vIP (10.13.1.86). The destination of Host B's vIP then translates to its rIP (10.13.2.10). After this calculation, the fourth step is to update the flow table entries on the SDN switch. Fifth, the switch conducts address translation between the rIPs and vIPs for Hosts A and B. Finally, the switch sends the message to Host B. Note how the SDN switch sends a request to the SDN controller when predefined behaviors are not present at

the switch. The network policies set by the network engineers are instantiated as rules at the controller and transmitted across the Southbound API to the data plane (as described in Section 2.1.1).

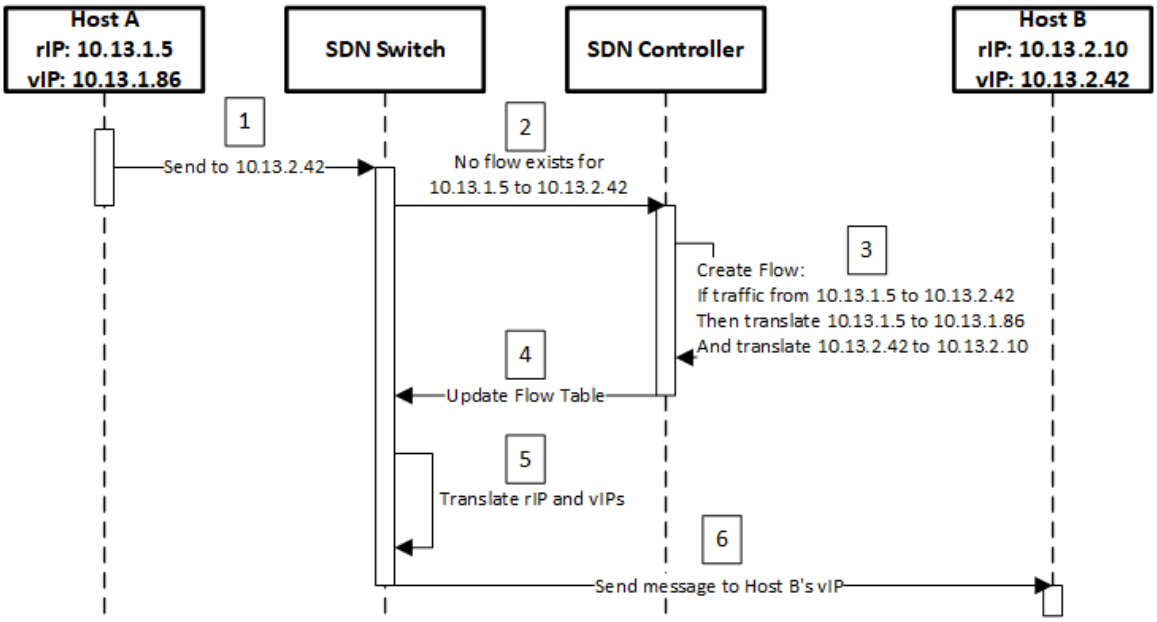



**Figure 8. Sequence Diagram for Flow Creation [24]**

In the example described in Figure 9, a host attempts to communicate with another host on a different subnet and the SDN switch already has a defined flow. Now that a flow has been established between the two hosts, traffic flows freely hosts much like in a traditional network as long as the flow table is not modified. First, Host A wants to send a message to the vIP of Host B. Second, the switch checks its flow table for a rule on how to handle this traffic. There exists a rule on the switch stating that traffic from Host A's rIP destined for Host B's vIP must modify the source/destination headers. Host A's vIP replaces its rIP and the switch translates the destination of Host B's vIP to its rIP much like in Figure 8. Finally, the Host B receives the message. The difference between this example and the one in Figure 8 is that a flow already exists on the switch so the controller does not supply the switch with instructions. If RHMs were used in this example, the rule on the SDN switch that governs this translation process must update to a different set of translations based upon source IP addresses.

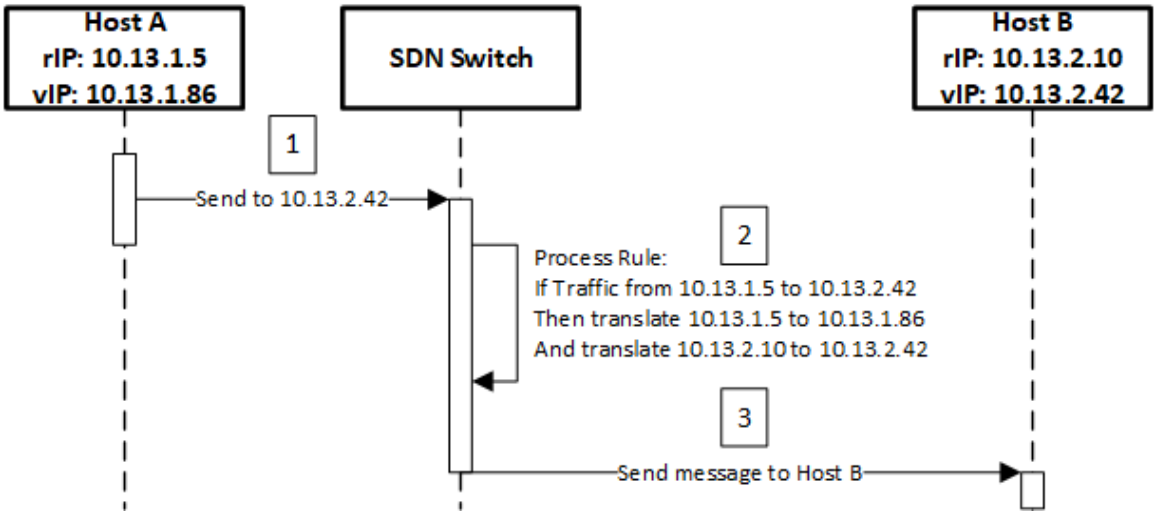



**Figure 9. Sequence Diagram for an Existing Flow [24]**

A variety of specific details may govern RHMs based upon the desires of network engineers. Hosts can mutate in a purely random way, they can be weighted to have mappings that are not near their current vIP in the address pool, or could mutate based on adversary actions. The optimal means of calculating the mutation rate is subject to numerous elements of the network but is beyond the scope of this thesis which measures the performance impacts of RHMs on legitimate users.

After a certain interval has passed, the mappings between rIPs and vIP change such that no mapping from the previous mutation table exists in the current mutation table. If rIP:vIP mappings did not always change with each mutation and the host were already compromised, an adversary would have even more time to execute attack scripts and create persistence. Imagine a vIP pool of four with three hosts. In this scenario, host mutations resemble those illustrated in Figure 10 if a flow table on one of the switches were examined. Examination of host A in the figure shows it has a vIP of 1 at the start. After the first mutation, its rIP:vIP mapping becomes A:2. After another mutation host A has a vIP of 3. This process continues indefinitely with the restriction that each host does not keep the same vIP across mutations. In this example, it means that host A cannot have a vIP of 3 after a theoretical third mutation. The specifics of how mutations are handled (i.e., frequency, available address ranges) can vary depending upon the level of complexity that

the network programmers wish to undertake.

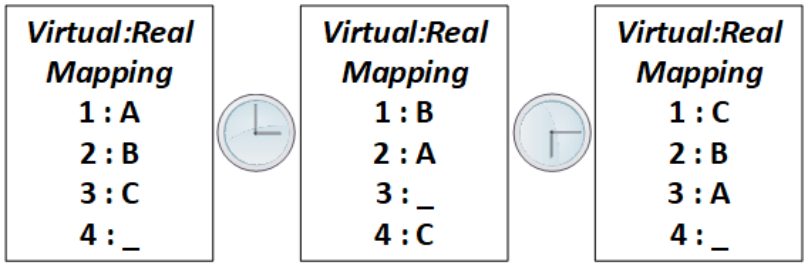



**Figure 10. rIP:vIP Mutations Example**

One possible means of RHM in an OpenFlow network is described in Algorithm 1:

**Algorithm 1:** OpenFlow Random Host Mutation [31]

**Data:** Unused address ranges, Range-to-Subnet assignments
**Result:** Mutation of rIP:vIP mappings

```
1  Unused ranges
2  Range-to-subnet assignments
3  forall packets p from OF-Switches do
4      if p is a Type-A DNS response for host h_i then
5          set DNS address to current vIP(h_i), TTL ≃ 0
6      else if p is a TCP-SYN or UDP from h_i to h_j then
7          if p.src is internal then
8              install in flow in source OF-Switch with action srcIP(p) := vIP(h_i)
9              install out flow in source OF-Switch with action dstIP(p) := rIP(h_i)
10         if p.dst is rIP then
11             if h_i access to h_j is authorized then
12                 install in and out flows in destination OF-Switch
13         else p.dst is vIP
14             install in flow in destination OF-Switch with action dstIP(p) := rIP(h_j)
15             install out flow in destination OF-Switch with action srcIP(p) := vIP(h_j)
16         end
17     forall mutation of each host h_i do
18         set vIP(h_i) to a new vIP
19     end
20 end
```

Algorithm 1 allows devices on the same subnet to communicate with each other using only their rIPs and establishes rIP:vIP mappings. Unused address ranges are determined to define the vIP address pool which are then assigned to subnets in lines 1 and 2. At this point, each subnet on the network receives a subset of the vIP address pool for its hosts. Translation/flow creation then happens as required based upon network traffic. Lines 4 and

5 show the assignment of a vIP for a packet that contains a DNS response for a particular host on the network. When information reaches the host, the DNS cache now contains a link between the Fully Qualified Domain Name (FQDN) and the vIP of the host in question. The else block that spans lines 6-16 shows how other network traffic could be handled. The if block from lines 7-9 conducts address translation between two hosts on the mutation network by changing the IP addresses in the packet through a new flows on the switch. The if block that starts on line 10 handles a scenario where hosts are allowed to communicate with the rIP of a host. In this case, address translation does not necessarily occur. Such an action may be practical for gateways or DNS servers that new devices must contact when they join a network. The else block starting on line 13 conducts the translation of a packet bound for a vIP to the rIP of the intended server. This block also changes the source IP to the vIP of the client. The forall block that starts on line 17 defines what occurs during a mutation. In a broad sense, each vIP of a host on the network receives a new vIP. The specifics of this process are an area that can be custom-tailored to the requirements of the network.

## 2.6 Previous Work in RHM Topic Area

Much of the spearheading into the application of SDN for MTD and the concept of RHM stemmed from work by researchers at UNC starting in 2012 [31]. This research provided mathematical models with which to define RHMs as well as tests in Mininet, a network simulator. Preliminary results indicated that adversaries were not able to generate as clear of a picture of a network topology compared to a static network due to shifting rIP:vIP mappings. With a proof-of-concept established, this idea was further refined by a previous research project from Air Force Institute of Technology (AFIT) [24].

To build upon the work done by the team at UNC, a study to test the concept on enterprise-quality hardware instead of a simulated environment was conducted by Aust at AFIT [24]. Aust refers to this work as Proactive Host Mutation (PHM). The study showed that an adversary in a "PHM network is significantly worse at finding and maintaining connections to hosts" [24]. T-tests conducted with a 90% confidence level showed differences

in network scan performance with p-values equal to or less than 0.1. The results supported Aust's hypothesis that PHM made scans by an adversary less effective [24]. Aust also noted the impact that PHM has on legitimate users as an area of future work stating, "given that servers host critical network functionality and data, they are highly valuable targets for attackers. The limitation seen in this research is that some network functions require lengthy connections, such as File Transfer Protocol (FTP). If these connections were interrupted by the mutation, it would interfere with user connectivity, which is unacceptable in most enterprise networks. An addition to PHM to allow for extended connections, without allowing for a new attack vector would significantly improve the functionality of PHM" [24]. The code that calculates PHMs does not update the rIP:vIP mapping for connections after a mutation occurs. As a result, network assets are no longer at the location in use by a client in an active connection. This design flaw requires a solution before a SDN-enabled MTD can see use in real-world networks. This thesis creates a version of Aust's work, RHM, that overcomes this critical issue. The limitation of interrupted connections drives the main focus of this thesis to assess the QoS impact that mutations have for a SDN-enabled MTD.

SDN has also been applied to ICSs by Jason Dearien from Schweitzer Engineering Laboratories. In a 2017 presentation, he described the use of Operational Technology SDN (OTSDN) to meet the strict requirements for ICSs. This line of research leveraged the "unprecedented network diagnostic visibility and ability to visualize packet flow..." [33]. Traditional networking is subject to topology-dependent performance and it is difficult to achieve 100% test coverage. Completely predefined network behavior is impractical for networks that exist in a normal IT environment. In a high-reliability network, performance, security, and resiliency are critical characteristics. In SDNs, reactive flows are used to respond and adapt to traffic since loads are variable and devices may join and leave the network often. The research conducted into OTSDN used static flows to define network performance with greater certainty and reliance on a known network configuration [33]. The OTSDN research showcased the flexibility of SDN for specific organizational requirements and to provide a better network service when compared to traditional networking strategies.

## 2.7 Chapter Summary

SDN is an emerging technology that stands poised to redefine large-scale network management. The decoupling of control and data planes solves many problems faced in current networks (stasis, inconsistent policies, poor scalability, vendor dependence). Several major companies (Google, Verizon, IBM, et al.) recognize the power of this paradigm shift for network operations. If a network is SDN-enabled or not, it remains subject to attack from adversaries. Adversaries must scan the network to identify targets and potential weaknesses that can be exploited through the network topology. A defensive countermeasure to this is MTD, which can take many forms.

One form of MTD is RHM, which creates a mapping between rIP and vIP addresses in the network and mutates the mappings over time. The specifics of mutation frequency may vary, but at its core, this defensive strategy increases the challenge for adversaries to scan a network based upon work by researchers at UNC in simulators and by an AFIT project on enterprise-quality hardware in a more realistic validation of that work. The remainder of this thesis analyzes the performance implications of SDN and the effectiveness of RHM as a MTD in network configurations that are indicative of a small enterprise.

# III. Framework Design

## 3.1 Overview

Protocols commonly used on computer networks are either elastic and can adapt to service-interruptions, or inelastic and cannot. Elastic protocols include Hypertext Transfer Protocol (HTTP), Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), and Secure Shell (SSH). Inelastic protocols include Domain Name System (DNS), Voice over IP (VoIP), and Real-time Transport Protocol (RTP). The business case for Random Host Mutations (RHMs) must also account for any performance impacts on legitimate users. For example, if RHMs negatively impacts the throughput of a critical service it may not be appropriate for that network and its specific needs. Conversely, if the candidate network consists of devices that are difficult to upgrade (i.e., legacy systems) that do not suffer a performance hit, then RHM may prove a worthwhile addition to the network architecture. Tests of elastic and inelastic protocols in addition to their impact on an adversary inform network engineers about how this defensive technique influences normal operations.

## 3.2 Design Goals

Experiments are conducted using virtual machines to imitate activities of legitimate users, attackers, and Software-Defined Networking (SDN) controllers. A SDN switch is considered part of the network design. Three goals drive the development of the test network:

1. Stability
   (a) All hosts are reachable from inside their own subnets.
   (b) All hosts maintain network access as indicated by successful ICMP pings and DNS requests.
   (c) The adversary machine can exploit any given machine on the test network.

2. Effectiveness

   (a) RHMs reduce the adversary's ability to discover hosts.

   (b) The adversary cannot reach the Hosts through their Real IP address (rIP), only through their Virtual IP address (vIP).

3. Quality of Service (QoS)

   (a) The framework in Section 3.5 allows for isolated assessment of various protocols without interference from other confounding factors. For example, test for a single protocol minimize all other network traffic. This enables conclusions about performance to focus strictly on the SDN configuration and the protocol under test. Protocols under test include: File Transfer Protocol (FTP), HTTP, Internet Message Access Protocol (IMAP), POP, RTP, SMTP, and RTP.

   (b) All network assets (i.e., hosts, routers, and switches) have the ability to generate detailed logs of network traffic. This can be through Wireshark or some other traffic monitoring software.

   (c) The framework provides a controlled, static environment between trials to produce verifiable conclusions about QoS.

### 3.3 Network Design

The network under test consists of two subnets. Tests with and without RHMs on the network provide data for later analysis. The topology is shown in Section 3.3. Five virtual servers running services that use the protocols under test and 23 virtual Windows XP hosts are on PlebeNet1. A simulated adversary running Kali Linux is also present. This subnet allows validation of Aust's experiments. QoS trials also occur under this configuration. A single pfsense firewall/router, with a Network Interface Card (NIC) for PlebeNet1 and a separate NIC for connectivity to the Internet, acts as the gateway for the network. The network services and XP hosts are all kept on separate port groups. Each port group has a separate RJ45 port in use on the ESXi hosts. This configuration forces servers and hosts on

the network to send messages through the Pica8 switch if they wish to communicate with hosts outside of their port group on the ESXi host. The SDN controller communicates with the Pica8 switch on a separate channel from the rest of the network traffic (control plane). The wiring diagram shown in Appendix E details the connections between ESXi hosts and the Pica8 switch.

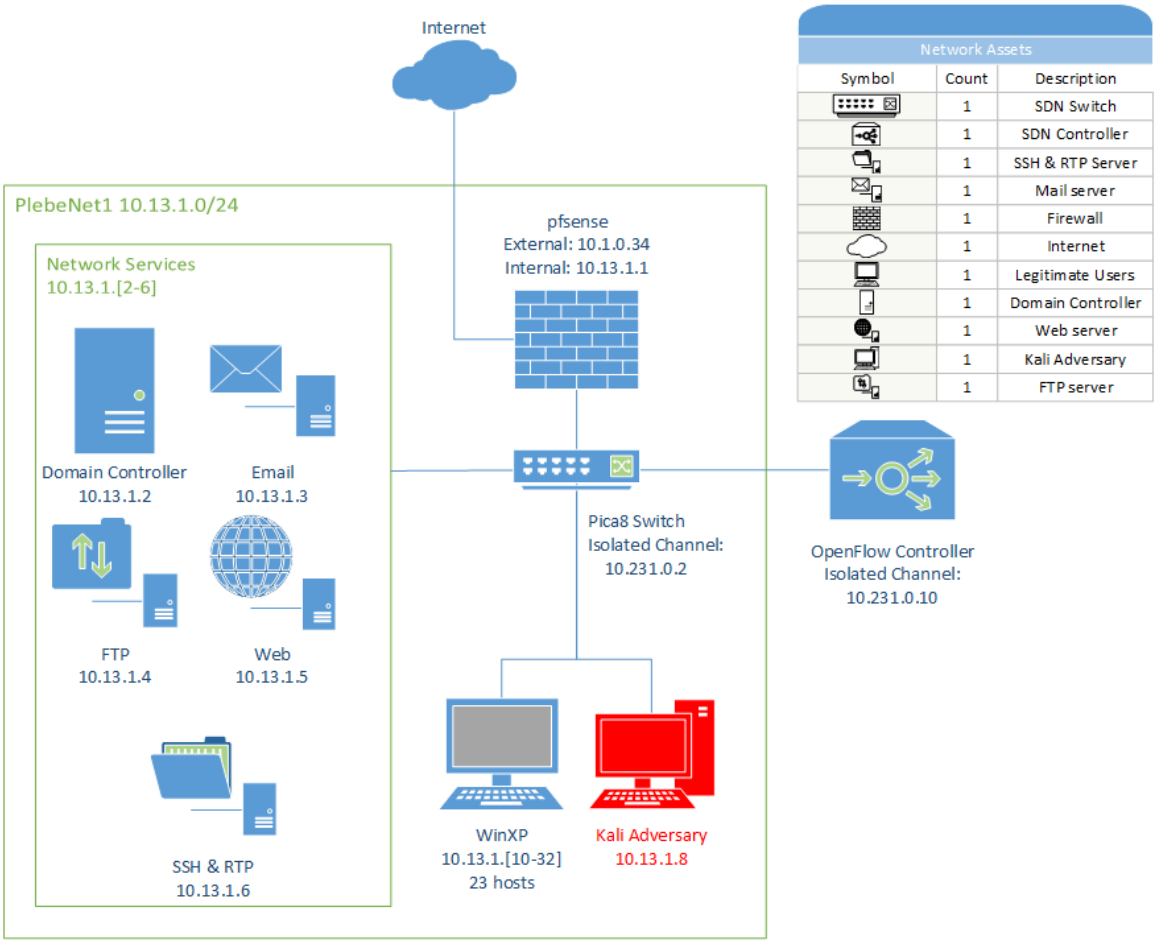



**Figure 11. Network Diagram for Experiments**

## 3.4 Network Assets

### 3.4.1 Legitimate Users.

Legitimate users are hosts that generate traffic used to evaluate the protocols under test. Clients, servers, and a firewall are present on the network. Clients and servers exist on separate physical NICs, otherwise the ESXi hypervisor would process network traffic

internally and it would not reach the Pica8 switch running mutations. The adversary machine can exploit the clients, which are the only network assets targeted in validation experiments since Aust did not attack other components of the network. The client and server ends of any given connection measure QoS metrics. Hosts are unaware of whether or not RHMs are active on a network. In order to access network services, the host knows the current vIP of the desired service for a given mutation. In real-world applications, the clients may receive Address Resolution Protocol (ARP) or DNS cache updates in lockstep with mutation updates to eliminate this assumption. More details about this idea are available in Section 6.4.

While Aust's trials have different numbers of active hosts, the validation of those results as part of this experiment focus on the trials done with 30 active hosts to provide more time for QoS tests. Aust's results indicate a consistent difference of at least five hosts in the *discovered amount* of hosts and *total number* of active hosts at this treatment level [24]. The main focus of this experiment is on the QoS for connections between clients and servers. A decrease in adversary effectiveness with shorter mutation windows must occur for successful validation of Aust's work.

Virtual machines running Windows Server 2012, Ubuntu 14.04 LTS, and Windows XP represent legitimate users. Additionally, a pfsense firewall is active on the network. Section 3.4.3.2 lists system specifications.

#### 3.4.1.1 Windows XP.

Virtual machines running XP represent users of the network. They open connections with the servers to send messages that use the seven protocols under test. XP is chosen since reliable exploits (e.g., MS08_067) are readily available in Metasploit. This research focuses on how RHMs impede the actions of an Adversary, not the resistance of targets.

#### 3.4.1.2 Windows Server 2012.

Win2k12 virtual machines provide a domain controller, web, email, and FTP server each compartmentalized to separate virtual machines. The domain controller which provides

DNS services to PlebeNet1 is an instance of Win2k12.

#### 3.4.1.3 Ubuntu 14.04 LTS.

The server which receives SSH and RTP communications from the client uses this Operating System (OS). A client to generate traffic for QoS trials also uses this OS.

#### 3.4.1.4 Pfsense Firewall/Router.

Pfsense is a firewall and router that also features unified threat management, load balancing, and multi Wide-Area Network (WAN).

### 3.4.2 Adversary.

The adversary serves to validate the results from Aust's thesis. This machine scans and attempts to exploit discovered clients from a separate physical NIC so that its traffic must traverse the Pica8 switch. As with the hosts, the adversary machine maintains detailed logs about perceived network traffic. The adversary machine runs Kali Linux (v2017.2) and uses current network scanning and attack tools such as Wireshark and Metasploit. Section 3.4.3.2 lists system specifications.

#### 3.4.2.1 Kali Linux.

This operating system is a penetration testing platform in common use [34]. Featuring several toolkits and frameworks for network attack, it is used to launch scans and exploits against targets on the testbed network.

### 3.4.3 SDN Devices.

#### 3.4.3.1 Ryu Controller (version 4.10).

Running on an Ubuntu 14.04 LTS virtual machine, the controller calculates RHMs and defines switch behavior through updates to the switch's flow table. Flow table updates happen on an out-of-band channel imperceptible to the adversary and hosts. If the switch is

unsure of how to process a packet, the switch sends it to the controller which determines how to process the packet (i.e., forward, drop, consume, and replicate). Section 2.1.3 contains more details about flow behavior. As with Aust's experiments, the controller uses a Ryu controller. In this experiment, network traffic to test RHMs and exploit hosts occurs on an isolated bridge defined on the SDN switch. The out of band channel in this experiment is an Ethernet port kept separate from the test network bridge. The adversary does not target the out of band channel or the controller. Section 3.4.3.2 lists system specification.

#### 3.4.3.2 Pica8 P-3290 Switch.

The SDN switch directs all traffic generated by adversary and hosts according to flow table entries that the SDN controller creates. In the event that a flow does not exist for a received packet, the switch consumes the packet and allows the controller to take further action. After a time specified by the controller (four minutes, in this experiment), flows timeout and disappear from the flow table [2]. The four minute expiration exists since QoS trials are done with a 30 second mutation interval and none of the connections in these trials last longer than four minutes in an attempt to keep packet captures to a manageable size for aggregate processing. In this experiment the switch used is a Pica8 P-3290 switch running PicOS v2.6.32.69 with OpenvSwitch v2.3.0 and has 1 $\mu$s latency with 64 byte frames. The switch fabric capacity is 176 Gigabits per second (Gbps). The adversary does not target the switch. Section 3.4.3.2 lists system specification.

**Table 2. Overview of System Specification for PlebeNet1**

| OS | Central Processing Unit (CPU) | Memory | Hard Drive |
|---|---|---|---|
| Windows XP | 1 Virtual CPU (vCPU) | 1 Gigabyte (GB) | 20 GB |
| Windows Server 2012 | 2 vCPU | 2 GB | 90 GB |
| Ubuntu 14.04 LTS | 1 vCPU | 1 GB | 16 GB |
| Pfsense Firewall/Router | 8 vCPU | 4 GB | 64 GB |
| Kali Linux | 2 vCPU | 2 GB | 50 GB |
| Ryu Controller | 4 vCPU | 4 GB | 16 GB |
| Pica8 P-3290 Switch | MPC8541 / Firebolt-3 | 512 Megabyte (MB) | 2 GB |

### 3.5 Test Framework

To validate Aust's work, the following sequence determines the effectiveness of RHMs at disrupting adversary scanning and exploitation actions. Each trial starts without influence from flows that may have been created in a previous trial. The procedures initially described by Aust influences the steps taken to execute validation trials.

1. Delete any preexisting flows on the SDN switch
2. Establish connection between controller and switch with the specified mutation interval.
3. Assign each rIP a vIP.
4. Initiate a network scan of the PlebeNet1 subnet with Nmap [35].
5. Attempt to exploit a machine discovered during the scan.
6. Send traffic between the adversary and exploited hosts such that a RHM occurs during transmission.

Given the goals from Section 3.2, the following sequence assesses the QoS impact of RHM. As with the previous list, each trial is independent from each other. Semi-automated

scripts allow the client and server machines to generate and record network traffic for later analysis.

1. Delete any preexisting flows on the SDN switch
2. Establish connection between controller and switch with a 30 second mutation interval. A 30-second interval was used by Aust and shown to reduce adversary effectiveness. This interval also enables faster trials which allow the tester to create more data to support statistical conclusions.
3. Assign all rIPs a vIP.
4. Begin packet captures on client and server machines.
5. Initiate a connection with one of the protocols under test.
6. Create flow between rIPs and vIPs, then mutate mappings at a given interval.
7. Send network traffic between hosts, such that an RHM occurs during transmission.

In the case of baseline data without RHMs active, the mutator script is inactive during the control experiments; the network functions as one would expect in a network that does not use Moving Target Defense (MTD). Network traffic using FTP, HTTP, IMAP, POP, RTP, SMTP, and SSH is then sent over the network and QoS metrics (latency, Round Trip Time (RTT), jitter, throughput, and dropped packets) are measured. In the instances where the mutator script is active, simulated network traffic is exposed to RHMs to see how the mutator script and protocols under test react to new rIP:vIP mappings. Ideal results would indicate no change in performance and remain transparent to the hosts. Figure 12 describes this measurement process. Wireshark to assess the performance of protocols under test from the client and server.

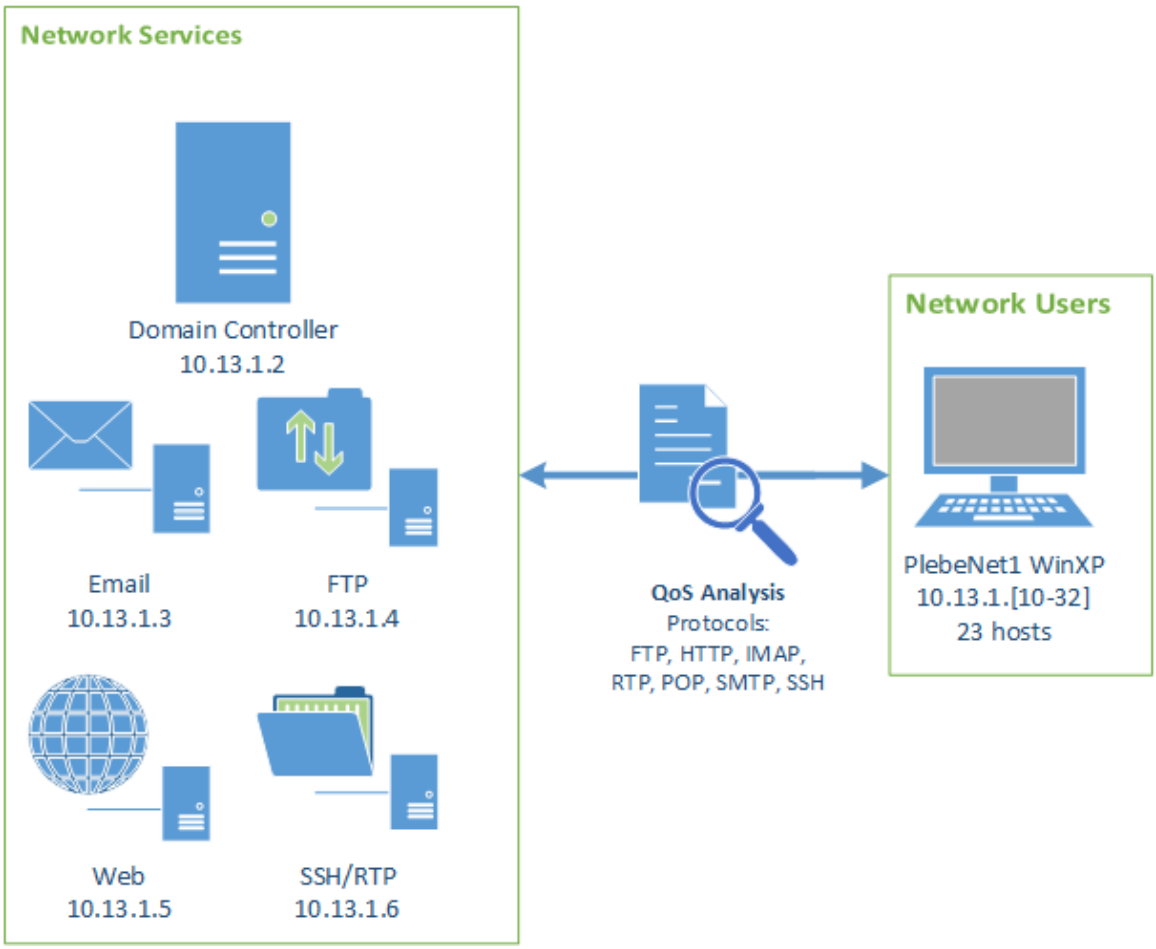



**Figure 12. QoS Measurement Concept**

## 3.6 Design Summary

This chapter describes how a test network is designed to be stable, effective, and facilitate the analysis of QoS metrics. The RHM framework validates work done by Aust while also collecting data on how various protocols react to mutations. Section 3.5 describes a framework for data collection on a network designed to approximate network traffic that exists in a small enterprise setting.

# IV. Research Methodology

## 4.1 Problem Statement

Random Host Mutations (RHMs) are a proven-effective means to interfere with scanning activity of an adversary. This chapter outlines experiments to verify RHMs as a means of disrupting adversary action in addition to examining the impact this framework has on legitimate users. The assessment of seven protocols quantifies the impact of RHMs. Section 4.1 shows the hypotheses under test for the validation experiments. Each hypothesis expects no statistically-significantly difference between a normal network and one that uses RHMs. The alternative hypotheses state that a difference does exist, but does not predict whether or not a RHM network performs better or worse than a standard one.

The hypotheses for scan time and hosts found have a null hypothesis which states there does not exist a difference between control and mutator experiments. The alternative hypotheses for these two metrics state that a difference does exist between scan time and hosts found for control and mutator experiments.

$$H_{O1} : Scan_C - Scan_M = 0, H_{A1} : Scan_C - Scan_M \neq 0 \tag{1}$$

$$H_{O2} : Hosts_C - Hosts_M = 0, H_{A2} : Hosts_C - Hosts_M \neq 0 \tag{2}$$

For the Transmission Control Protocol (TCP) protocols, latency and Round Trip Time (RTT) are two unique metrics under test. The null hypotheses for these metrics claim that there is not a difference in latency or RTT as experienced by the server between control and mutator experiments. The alternative hypotheses for these two metrics assert that a difference does exist between latency and RTT in control and mutator experiments.

$$H_{O3} : Latency_C - Latency_M = 0, H_{A3} : Latency_C - Latency_M \neq 0 \tag{3}$$

$$H_{O4} : RTT_C - RTT_M = 0, H_{A4} : RTT_C - RTT_M \neq 0 \tag{4}$$

For the User Datagram Protocol (UDP) protocol (i.e., Real-time Transport Protocol (RTP)), jitter is a unique metric under test. The null hypothesis for this metric asserts that no difference in jitter exists between control and mutator experiments. The alterna-

tive hypothesis claims that a difference in jitter does exist between control and mutator experiments.

$$H_{O5} : Jitter_C - Jitter_M = 0, H_{A5} : Jitter_C - Jitter_M \neq 0 \quad (5)$$

For all protocols (TCP and UDP), throughput and packet drop rate are measured. The null hypotheses for these metrics state that there is not a difference in either metric between control and mutator experiments. The alternative hypotheses claim that there is a difference in throughput and packet drop rate.

$$H_{O6} : Throughput_C - Throughput_M = 0, H_{A6} : Throughput_C - Throughput_M \neq 0 \quad (6)$$

$$H_{O7} : PktDrop_C - PktDrop_M = 0, H_{A7} : PktDrop_C - PktDrop_M \neq 0 \quad (7)$$

### 4.2 Experimental Design

Figure 13 illustrates the design of the system under test. The figure lists factors, parameters, and metrics involved with the experiment.

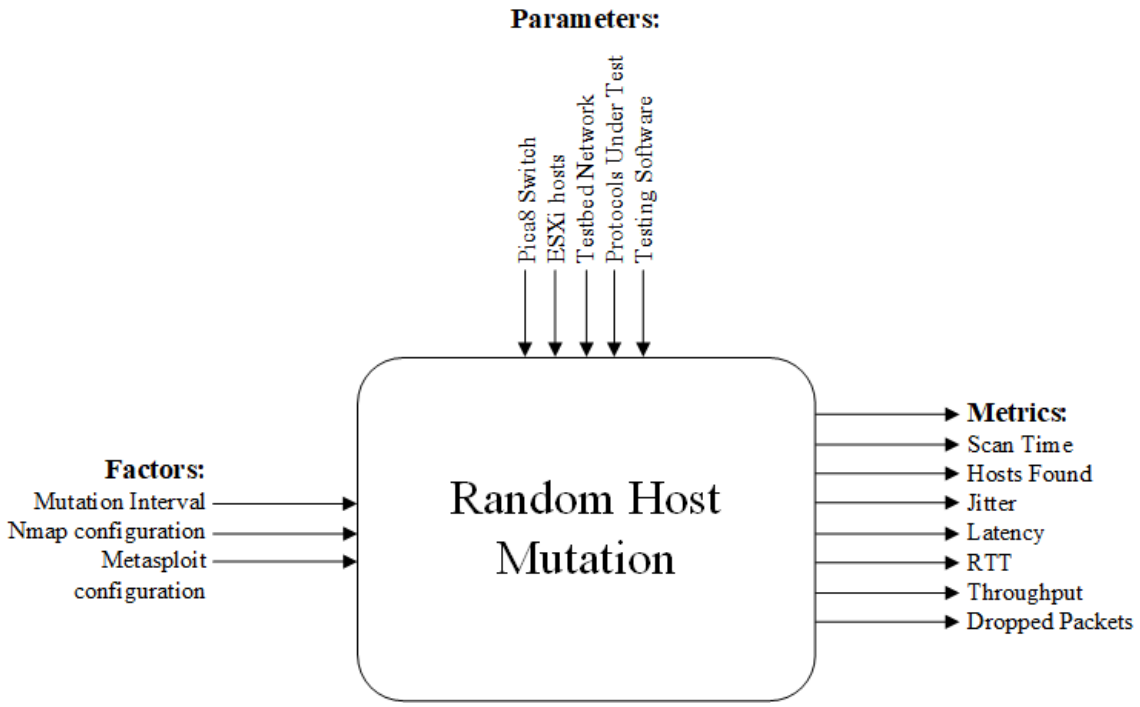



**Figure 13. System Under Test Diagram for RHM Framework**

### 4.2.1 Metrics.

This experiment aims to validate the results by Aust and measure the Quality of Service (QoS) impact of RHM. Therefore, trials measure the original metrics of scan time and number of hosts found. Trials do not measure the exploit length metric tested by Aust. Since RHMs require a means to continue connections across mutations for the sake of greater usability, that ability was sacrificed in lieu of usability. As a result, a connection associated with an active exploit cannot be differentiated without additional work to examine characteristics of that connection. Objective measures of latency, RTT, jitter, and dropped packets determine the QoS for each protocol. Any measurements of time rely upon system clocks determined by the operating system. The following list provides a detailed description of each response variable used as a metric. Table 3 offers a summary of the five QoS metrics gathered in experiments. Appendix A contains the results for scan time and found hosts from Aust's experiments for reference [24].

1. *Scan Time*: This measures, in seconds, the amount of time it takes for an attacker to scan the network and find all hosts.

2. *Number of Hosts Found*: This measures the number of hosts that Nmap is able to successful find during a scan.

3. *Jitter*: RFC 4689 defines jitter as "the absolute value of the difference between the forwarding delay of two consecutive received packets belonging to the same stream" [36]. Jitter is measured in milliseconds. Generally, the maximum amount of jitter for acceptable service is 50 ms [37].

4. *Latency*: Latency refers to the amount of time, in milliseconds, it takes information to reach its destination [38]. There are four key causes of latency: propagation delay, serialization, data protocols, and routing and switching overhead [39]. Since the network configuration remains static aside from the presence of RHMs, the only source of latency that needs to be accounted for is associated with routing and switching.

5. *Round-trip Time*: RTT is "the elapsed time for transit of a signal over a closed circuit"

[40]. This metric influences throughput of telephony and ACK-based systems such as TCP.

6. *Throughput*: Throughput refers to the amount of data a system can process in a given amount of time. In the case of network communications, it refers to the amount of information crossing the channel at a set interval (e.g., Megabits per second (Mbps))

7. *Dropped Packets*: Dropped packets (or packet loss) refers to information never reaching its destination. Internet Control Message Protocol (ICMP) pings can diagnose loss-rates of packets sent and received. This method has limitations in production environments where the network cannot be fully controlled since loss rates are not always constant [41]. A controlled environment allows later analysis to ignore this limitation since experiments are conducted in a controlled environment where the same network traffic is sent in each trial. For inelastic connections, a drop rate of 0-5% is viewed as acceptable [42].

**Table 3. Metrics**

| **Metric** | **Normal Operating Level and Range** | **Measurement Precision, Accuracy** | **Relationship of Metric to Objective** |
|---|---|---|---|
| *Scan Time* | Depends upon number of live hosts | 1-sec steps, measured on attacker's clock | Shows any increase RHMs add to network scan time |
| *Number of Hosts Found* | Number of Live Hosts | Counted, not measured | Shows accuracy of network scans |
| *Jitter* | Normal: 0-50 milliseconds, Range: 0-Infinite milliseconds | 5-millisecond steps, measured on Wireshark .pcapng file | Measures service quality of transmission |
| *Latency* | 0-4000 milliseconds (default timeout for 'ping' command) | 10-millisecond steps, measured on XP host's system clock | Describes efficiency of network configuration (i.e., with or without active mutations) |
| *RTT* | Normal: 0-50 milliseconds, Range: 0-Infinite milliseconds | 5-millisecond steps, measured on Wireshark .pcapng file | Measures service quality of transmission |
| *Throughput* | Normal: Protocol dependent, Range: 0-Infinite bps | 0.1 bps steps, measured on Wireshark .pcapng file | Measures service quality of transmission |
| *Dropped Packets* | Normal: 0-5%, Range: 0-100% | Percentage of packets lost with respect to packets sent, measured on Wireshark .pcap file | Describes reliability of transmissions |

### 4.2.2 Factors, Parameters, and Range.

#### 4.2.2.1 Factors.

A controlled environment that emulates a production environment supports the gathering of test data. Experiments modify three factors for the sake of simplicity. The network configuration (i.e., number of active hosts) does not change over the course of experiments which assess the impact of RHM. Table 5 summarizes factors used in experiments.

1. **Mutation Time:** This is the amount of time in minutes that a specific Real IP address (rIP):Virtual IP address (vIP) mapping remains active. At the end of this time period, hosts receive a new rIP:vIP mapping. This categorical variable remains unchanged across all rounds of testing with four possible values (0.5, 1, 5, 15). This variable does not require monitoring as it does not change once an experiment begins.

2. **Nmap Configuration:** This refers to the active options used during an Nmap scan. Scan types are held constant during a test. Therefore, it is a categorical variable. Two scans are be used, Intense and Quick. Their options are presented below:

   *Intense scan:*

```
db nmap -min-hostgroup 96 -T4 -A -v -n 10.13.1.0/24

     db nmap = The command to run Namp

     -min-hostgroup 96 = Sets the parallel host scan group size to 96

     -T4 = Sets timing to 4

     -A = Enable OS detection, version detection, script scanning, and traceroute

     -v = Tell Nmap to be verbose

     -n = Never do Domain Name System (DNS) resolution/Always resolve [default:
sometimes]

     10.13.1.0/24 = The IP Address ranges to conduct the scan in
```

*Quick scan:*

```
db nmap -min-hostgroup 96 -T4 -n -F 10.13.1.0/24
```

```
db nmap = The command to run Namp

-min-hostgroup 96 = Sets the parallel host scan group size to 96

-T4 = Sets timing to 4

-n = Never do DNS resolution/Always resolve [default:  sometimes]

-F = Fast mode (Scans fewer ports than the default scan)

10.13.1.0/24 = The IP Address ranges to conduct the scan in
```

3. **Metasploit Configuration:** Similar to the Nmap configuration, this value represents the specific Metasploit settings used to compromise a host. In each round of tests the value is remains constant, making this a categorical variable. Only one factor of SMB exploit is used as the configuration parameter. This value does not change as measurements are taken to ensure Metasploit is working correctly before attempted exploits.

**Table 5. Factor Summary**

| Factor | Normal Operating Level and Range | Settings | Predicted Effects |
|---|---|---|---|
| *Mutation Time* | 0-15 minutes | 0.5, 1, 5, and 15 | Lower values reduce the time an attacker has to access an exploited machine. |
| *Nmap configuration* | N/A | Intense and Quick | Difference in time for scan completion time. Possible decrease in host discovery. |
| *Metasploit Configuration* | N/A | SMB | Difference in time to exploit machine and maintain access. |

#### 4.2.2.2 Parameters.

Two items remain constant in the experiment: The Software-Defined Networking (SDN) switch and servers running virtual machines.

1. **SDN Switch:** One Pica8 P-3290 switch running OpenFlow 1.3 and controlled by a SDN controller (Ryu version 4.10).

2. **Servers:** Two SuperMicro SuperServers 8027R-TRF+ with Xeon E5-4600 v2 processors, eight 1000BASE-T Network Interface Cards (NICs), 12 cores, 1000Gigabyte (GB) of Random Access Memory (RAM), running ESXi hosts the SDN controller and all hosts.

### 4.2.3 Determining Sample Size.

Pilot study data from Aust gathered results for quick and intense scanning configurations of five hosts with and without a 30-second mutation interval. The pilot study consisted of 30 repetitions to enable assumptions of normality. In the control network, all 5 hosts were enumerated but only an average of 2.6 were found in the network with active mutations [24]. T-tests conducted by Aust revealed a difference between networks that was not due to chance with a p-value of 2.028e-12. Based upon those results, Aust calculated a minimum

sample size for trials with 5, 10, 20, 30, 40, and 50 hosts with the formula:

$$n \geq (\frac{z \cdot \sigma}{MoE})^2 \tag{8}$$

The result of Aust's calculations showed that five samples or more were required to report the mean within a five-second Margin of Error (MoE) with 95% confidence [24].

Review of Aust's pilot study data shows a sample size of 9 or larger is required to achieve results with a margin-of-error of 2.25 seconds. This ensures results that offer a smaller margin-of-error than Aust's initial data. A z-value of 1.96 which corresponds to a 95% confidence level determines the minimum sample size for these experiments. The largest standard deviation from Aust's pilot study (3.362 seconds for intense scan time in a control network) was also used along with a 2.25 second MoE [24]. With this information, Equation (9) produces a minimum sample size of 8.577.

$$n \geq (\frac{1.96 \cdot 3.362}{2.25})^2 \tag{9}$$

This rounds up to a minimum sample size of 9. For additional data to support the results, experiments to support validation analysis shall consist of 10 trials when analyzing the impact of mutations on adversary actions.

For QoS data, no changes happen to the system under test aside from whether or not mutations are active. Since the computer hardware is deterministic based upon the set of instructions given to it, the same inputs yield the same outputs. Variation in the time required for certain tasks may be accounted for by human involvement with the semi-automated trials and small differences in how traffic is routed between hosts. This implies a data distribution that is approximately normal clustered around the mean time taken by the hardware to perform the same set of actions in each trial. The Central Limit Theorem guides the determination of sample size for QoS data to be 30 trials [43]. Appendix B contains the results from each trial and displays histograms for each protocol under test, which describe the overall distribution of the trial data.

### 4.2.4 Experimental Process.

This section details the specific steps taken to collect test data. One of the assumptions for these trials states that the client knows the vIP address of the host it wishes to open a connection with. The script files are located in Appendix C for further details. Trials use bridge br1 on the SDN switch. This bridge consists of physical ports 10-15 on the Pica8 switch. In all cases, two actions are taken at the start and end of each trial. At the start, the SDN controller initializes in verbose mode with the following command:

```
ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
```

params.conf defines the subnet ranges for mutation and the mutation interval. Arguments are stored in the OpenStack Oslo Config format [44].

At the end of each trial, the flows installed on the switch are deleted with the following command:

```
ovs-ofctl del-flows br1
```

For further debugging information, installed flows and traffic received by the switch may be displayed with the following commands, respectively:

```
ovs-ofctl dump-flows br1
ovs-ofctl snoop br1
```

#### 4.2.4.1 Validation Study.

The validation study performs the scanning and exploitation activities used in Aust's research. Scanning of PlebeNet1 is done with either quick or intense options. Section 4.2.2.1 lists specific options available in trials. This phase occurs before an adversary attempts to exploit one of the machines revealed in the scan. Once a target is found, an exploit (e.g., MS08_067) is launched against it. If a scan persists beyond the amount of time required for two mutations to occur, it shall be manually terminated and recorded as 0 seconds. Any results from a scan that took longer than two mutations are out of date and therefore useless to an adversary.

Validation trials consist of a 9-step process and issue commands to the adversary, con-

troller, and switch. Target hosts are turned on before the trial begins. The sequence of events is as follows. Commands 1-5 occur before the mutator starts as they do not generate network traffic. The only requirement for this sequence of commands is that steps 7 and 8 occur while the mutator is active.

1. From the adversary's root prompt, start the PostgreSQL database. This is the scanning prompt.

   ```
   service postgresql start
   ```

2. In a separate root prompt on the adversary, start the PostgreSQL database. This is the exploit prompt.

   ```
   service postgresql start
   ```

3. Initialize the Metasploit framework database in both prompts.

   ```
   msfdb init
   ```

4. Start Metasploit in both prompts.

   ```
   msfconsole
   ```

5. From both prompts on the adversary, ensure connectivity to the Metasploit database for each prompt.

   ```
   db_status
   ```

6. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

7. From the scanning prompt, launch the scanning script (Quick or Intense).

   ```
   resource [quickPN1.rc/intensePN1.rc]
   ```

8. Wait until scan completion. Then, from the exploit prompt, launch the exploit script against a target found in the scan if one exists. Otherwise, note that the scan did not find any targets.

   ```
   resource exploit.rc
   ```

9. Wait for the exploit to succeed or fail. From the switch's root prompt, delete any flows stored in the flow table.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.2 HTTP.

A 50 MB file is stored on the Hypertext Transfer Protocol (HTTP) server for retrieval by the client. To reduce the size of the packet capture file for later analysis but still have a connection that persists across multiple mutations, the transfer speed is limited by the –limit-rate option. Upon completion of the transfer (or a broken connection), the Secure Hash Algorithm 1 (SHA-1) hash of data transferred to the server is compared to the SHA-1 hash of the file on the HTTP server to verify successful transmission. QoS trials for HTTP consist of a 6-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

2. Start Wireshark on HTTP server.

3. Start Wireshark on client.

4. From the client's command prompt, initiate HTTP from the client traffic with curl [45].

   ```
   curl -O 10.13.1.[ip]/Data50.txt --limit-rate 1k
   ```

5. After the transfer, stop calculate SHA-1 sum of transferred data from the server on the client's command prompt.

   ```
   sha1sum Data50.txt > HTTP_[M/C]_[Trial#]
   ```

6. From the switch's root prompt, delete the flows created during the trial.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.3 SSH.

To test Secure Shell (SSH) a client opens a connection to a server and periodically sends commands at an interval that lasts longer than the mutation window to simulate a human interaction with the command prompt. Output of the SSH session is monitored for broken connections. QoS trials for SSH consist of a 6-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

2. Start Wireshark on SSH server.

3. Start Wireshark on client.

4. Open a SSH connection to server from client. Supply password when prompted.

   ```
   ssh root@10.13.1.[ip]
   ```

5. Activate SSH script stored on the server from the client.

   ```
   ~/Send_ssh.sh
   ```

6. Wait for the script to terminate. From the switch's root prompt, delete the flows created during the trial.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.4 RTP.

RTP is tested through the client opening a connection to the server and sending a message at 1-second intervals for a duration that lasts longer than the mutation window with `rtpgen` [46]. Output of the RTP session is monitored via Wireshark and the `rtpdump` utility to measure QoS information [47]. QoS trials for RTP consist of a 7-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

2. Start Wireshark on server.

3. Start Wireshark on client.

4. Start a RTP listener on server [47].

   ```
   ./rtpdump 10.13.1.6/9000
   ```

5. Send RTP traffic from the client with rtpgen [46].

   ```
   ./rtpgen -a 10.13.1.6 -p 9000 -c message.txt
   ```

6. After the client transmits for more than at least two mutations, stop the transmission by the client.

   ```
   Ctrl + C
   ```

7. From the switch's root prompt, delete the flows created during the trial.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.5 POP.

A Powershell script executed through PowerCLI automatically starts Wireshark captures and sends Post Office Protocol (POP) data. Two scripts are activated in sequence to generate POP traffic. QoS trials for POP consist of a 7-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. From PowerCLI, log on to the vCenter server. Supply credentials when prompted.

   ```
   Connect-VIServer 10.1.0.85
   ```

2. Navigate to the directory holding powershell scripts (for example, `C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Powershell_Scripts`)

3. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

4. From PowerCLI, start transmission script on client.

   ```
   .\Capture_Mutation_Kali.ps1 -Duration 120 -Trials 10 -Interface "eth0" -SVC_Name "Aust_Kali" -Protocol "POP" -C:[$True/$False]
   ```

5. When prompted, enter the vIP of the POP server and press enter.

6. Start transmission script on email server.

   ```
   .\Capture_Mutation_Svcs.ps1 -Duration 120 -Trials 10 -Interface "eth0" -SVC_Name "Clio" -Protocol "POP" -C:[$True/$False]
   ```

7. From the switch's root prompt, delete the flows created during the trial.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.6 IMAP.

A Powershell script executed through PowerCLI automatically starts Wireshark captures and sends Internet Message Access Protocol (IMAP) data. Two scripts are activated in sequence to generate IMAP traffic. QoS trials for IMAP consist of a 7-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. From PowerCLI, log on to the vCenter server. Supply credentials when prompted.

   ```
   Connect-VIServer 10.1.0.85
   ```

2. Navigate to directory holding powershell scripts (for example, `C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Powershell_Scripts`)

3. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

4. From PowerCLI, Start transmission script on the client

   ```
   \Capture_Mutation_Kali.ps1 -Duration 120 -Trials 10 -Interface "eth0" -SVC_Name "Aust_Kali" -Protocol "IMAP" -C:[$True/$False]
   ```

5. When prompted, enter the vIP of the IMAP server and press enter.

6. Start transmission script on email server.

   ```
   .\Capture_Mutation_Svcs.ps1 -Duration 120 -Trials 10 -Interface "eth0" -SVC_Name "Clio" -Protocol "IMAP" -C:[$True/$False]
   ```

7. From the switch's root prompt, delete the flows created during the trial.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.7 SMTP.

A Powershell script executed through PowerCLI automatically starts Wireshark captures and sends Simple Mail Transfer Protocol (SMTP) data. Two scripts are activated in sequence to generate SMTP traffic. QoS trials for SMTP consist of a 7-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. From PowerCLI, log on to the vCenter server. Supply credentials when prompted.

   ```
   Connect-VIServer 10.1.0.85
   ```

2. Navigate to the directory holding powershell scripts (For example, `C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Powershell_Scripts`)

3. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

4. Start transmission script on the client.

   ```
   .\Capture_Mutation_Kali.ps1 -Duration 120 -Trials 10 -Interface "eth0" -SVC_Name "Aust_Kali" -Protocol "SMTP" -C:[$True/$False]
   ```

5. When prompted, enter the vIP of the SMTP server and press enter.

6. Start transmission script on the SMTP server.

   ```
   .\Capture_Mutation_Svcs.ps1 -Duration 120 -Trials 10 -Interface "eth0" -SVC_Name "Clio" -Protocol "SMTP" -C:[$True/$False]
   ```

7. From the switch's root prompt, delete the flows created during the trial.

   ```
   ovs-ofctl del-flows br1
   ```

#### 4.2.4.8 FTP.

The File Transfer Protocol (FTP) trials retrieve a 50 MB file from the FTP server. The transmission rate is limited to 500 Kbps to ensure a connection that lasts across several mutations. A passive FTP connection is used [48]. One use case for a passive FTP connection is a situation where the FTP server is unable to establish a data channel, such as a restrictive firewall rule on the client side [49]. Packet captures are initialized before the FTP connection is attempted. QoS trials for FTP consist of a 9-step process and issue commands to the client, server, controller, and switch. The sequence of events is as follows.

1. Start Wireshark on client and server.
2. From the controller's root prompt, initialize the mutator in verbose mode. Wait until the controller displays the list of rIPvIP mappings. These are the mappings for the current mutation interval.

   ```
   ryu-manager Mutator_PlebeNet.py --config-file params.conf --verbose
   ```

3. From the client, open a FTP connection via the `lftp` tool [50].

   ```
   lftp 10.13.1.[ip]
   ```

4. Set transmission rate to 500 Kbps.

   ```
   set net:limit-rate 500000:500000
   ```

5. Initiate download.

   ```
   get Data50.txt
   ```

6. Once the transfer completes, close the FTP connection.

   ```
   bye
   ```

7. Stop Wireshark on the client and server.
8. Calculate SHA-1 hash of transferred data.

   ```
   sha1sum Data50.txt > FTP_[M/C]_[Trial#]
   ```

9. From the switch's root prompt, delete the flows created during the trial.

```
ovs-ofctl del-flows br1
```

### 4.3 Methodology Summary

This chapter describes the methodology used to measure the stability and effectiveness of the RHM framework in addition to gathering QoS data for analysis. The validation experiments use the same levels of hosts and mutation times as Aust did in the 30-host experiments with and without active mutations. This information validates previous work and indicates that the test network is properly configured for QoS analysis. Furthermore, trials examine seven protocols under test for QoS performance. Controllable factors such as hardware and software configurations remain constant to reduce variability in the data and isolate the affect of RHM. The ability of the adversary to conduct a quick scan and detect all hosts with and without active mutations determines network stability in addition to ICMP pings and DNS requests.

# V. Results and Analysis

## 5.1 Summary of Results

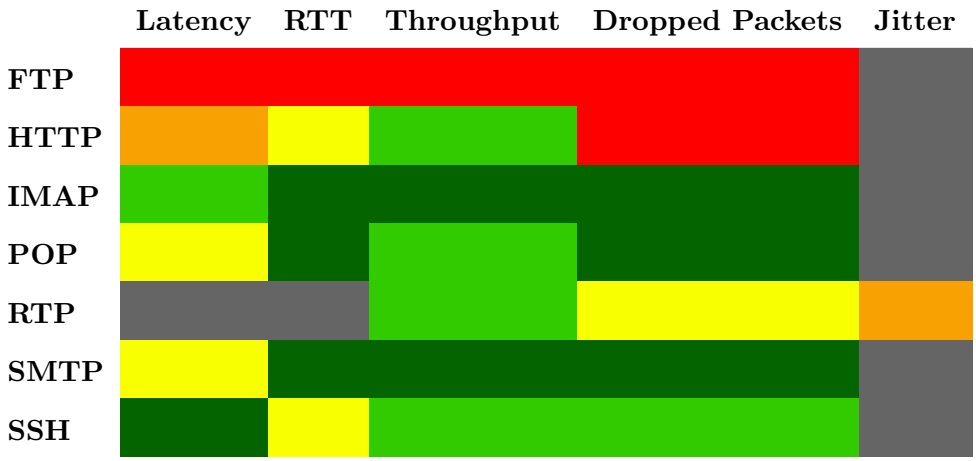
Experiments have two goals. First, gather more data for efficacy of Random Host Mutations (RHMs) as a way to impede adversary actions. Second, quantify the impact (if any) on the Quality of Service (QoS) of common protocols of an RHM implementation. Validation data on a network with 30 hosts confirms that scanning actions by an adversary are less effective than a network without RHMs. Of the seven protocols under test, one protocol (File Transfer Protocol (FTP)) fails and another (Hypertext Transfer Protocol (HTTP)) presents anomalous behavior. This chapter provides an overview of the results; test data can be found in Appendix B.

Experimental data gathered by Aust showed an increase in scan times and a decrease in the number of hosts discovered by an adversary in a RHM network. The validation study produced differences in the length of scan times when compared to Aust but supported the conclusion that RHMs impede the ability of an adversary to successfully gather intelligence about a target network.

T-tests at the 99% confidence level on QoS data found statistically-significant differences in varying aspects of the seven protocols under test ($p = 0.01$). Table 7 describes which protocols exhibited statistically-significant differences from control data for each metric. FTP connections initiated by IP address do not function when mutations are active. Connections through Fully Qualified Domain Name (FQDN) may still be possible, but require extra modification to the mutator script (see Section 6.4). HTTP connections successfully transfer data, but sent as much as 14.2 times the amount of packets when compared to control experiments. Such a dramatic increase in network traffic is unacceptable for large-scale applications as the network would be overwhelmed with extraneous traffic and degrade throughput. RTP exhibits an increase in jitter that may or may not be acceptable, depending upon the performance requirements of the application using the protocol. Internet Message Access Protocol (IMAP), Post Office Protocol (POP), Simple Mail Transfer Pro-

tocol (SMTP), and Secure Shell (SSH) connections function with minimal differences when compared to control experiments. In Table 7, cells colors correspond to level of difference between means for control and mutator trials, which are detailed in the legend. The table is colored in terms of the difference of the order of magnitude for the sample means since some control samples had extremely small standard deviations. Coloring the table according to the difference in standard deviation creates a misleading table where a reported difference of hundreds of standard deviations masks the fact that the true performance difference is only 1-4 milliseconds in most cases.

**Table 7. Overview of Protocol Performance**




| Legend | Description |
|---|---|
| | No difference |
| | Difference within one order of magnitude |
| | Difference of one order of magnitude |
| | Difference of two orders of magnitude, or application-specific judgment required |
| | Difference of at least three orders of magnitude, or broken protocol |
| | Not Applicable |

## 5.2 Validation Data

### 5.2.1 Quick Scan.

#### 5.2.1.1 Scan Time.

Time to complete a quick scan for Aust's study and the validation study are shown in Figure 14 along the y-axis in seconds. Of particular note is the large difference in scan times despite the use of the same Nmap command in Aust's experiments and the validation experiments. It is possible that several options were omitted in the original trial or that a smaller IP address range was scanned as this could account for some of the difference in scan time.

While Aust's data suggests that mutations increase scan time, the results from validation experiments show a decrease in scan time compared to control. Regardless of the difference in scan time, both experiments show a decrease in the number of hosts discovered by the adversary as shown in Figure 15.

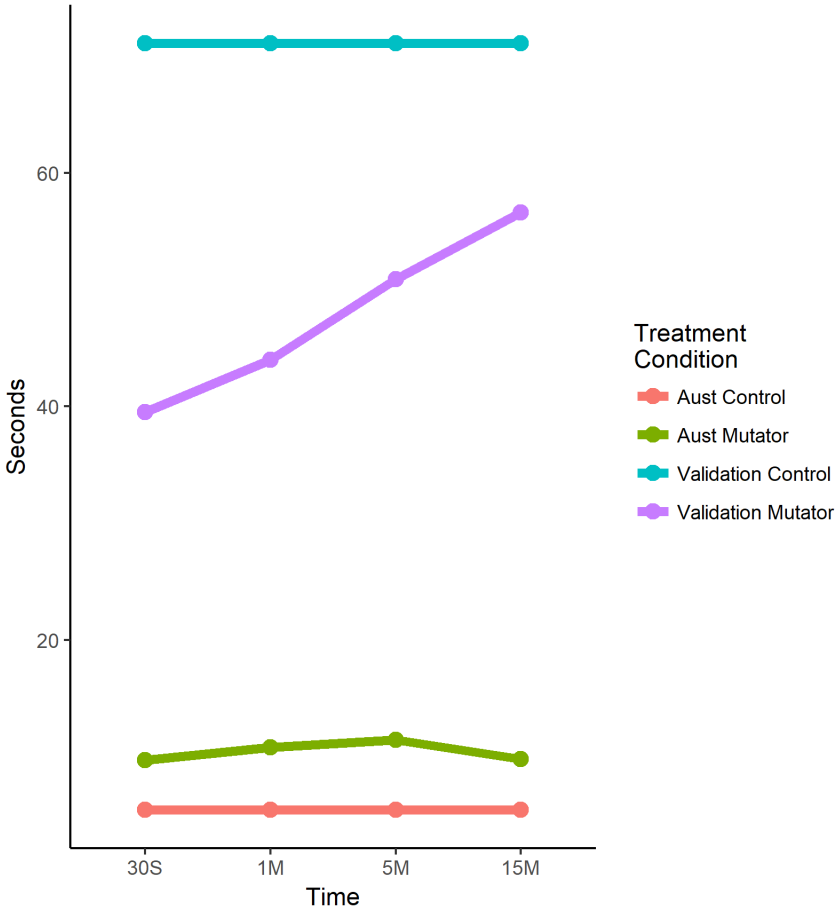



**Figure 14. Quick Scan Results**

#### 5.2.1.2 Hosts found.

Figure 15 reports the amount of hosts found by quick network scans in control and RHM trials. In both the original data and the validation trial, RHM scans consistently reported a number of detected hosts that was less than 74% of active hosts on the network at the time of scan. Aust's control data is not visible on the figure because it has the exact same values as the validation control data. Control experiments validate Aust's data. However, mutator scans took between 40 and 60 seconds to complete so the amount of discovered hosts plateaus at the maximum level of 22 hosts as mutation interval grew beyond the required scan time. This data suggests that mutation intervals are most effective when shorter than scanning activity by an adversary.

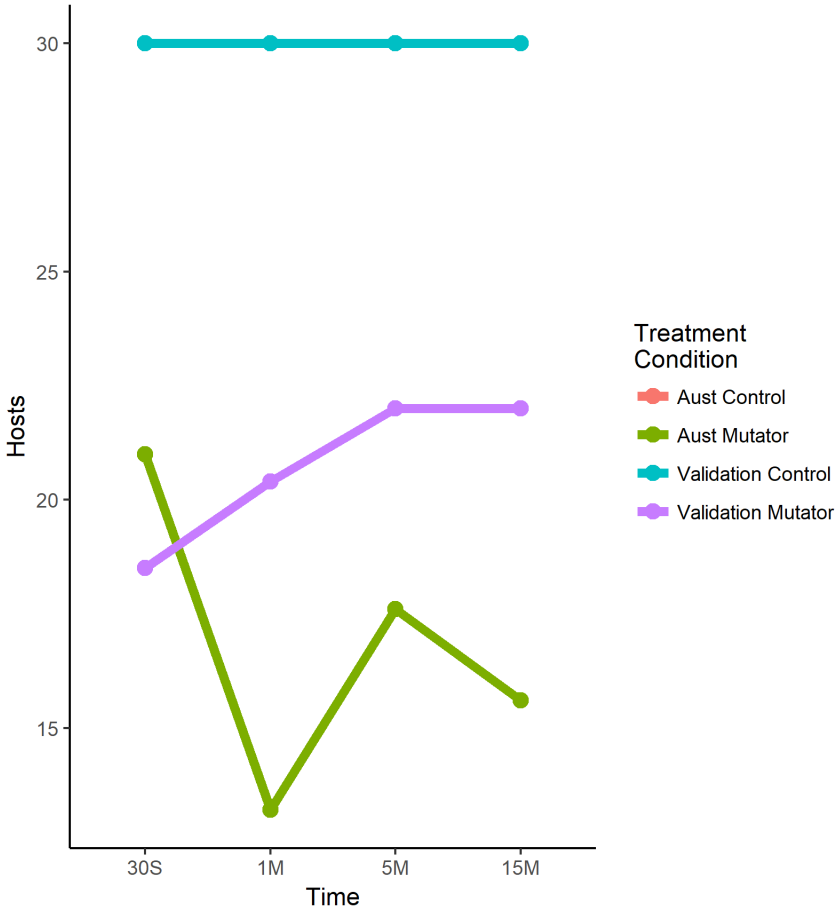



**Figure 15. Quick Scan Hosts Found**

### 5.2.2 Intense Scan.

#### 5.2.2.1 Scan Time.

Time to complete an intense scan for Aust's study and the validation study are shown in Figure 16 along the y-axis in seconds. As with the results in Section 5.2.1.1, there is a large difference in scan times despite the use of the same Nmap command in Aust's experiments and the validation experiments. It is possible that several options were omitted in the original trial or that a smaller IP address range was scanned as this could account for some of the difference in scan time. Regardless of the difference in scan time, both experiments showed a decrease in the number of hosts discovered by the adversary as shown in Figure 17. Unlike Aust's results, scans were manually terminated if they persisted beyond the amount

of time required for two mutations to occur. Any results from such a scan would be out of date and therefore useless to an adversary. This accounts for the mutator results of the 30-second, one-, and five-minute mutation windows. As with the results in Section 5.2.1.1, the mutator caused adversary scans to terminate faster than control experiments which provides inaccurate information.

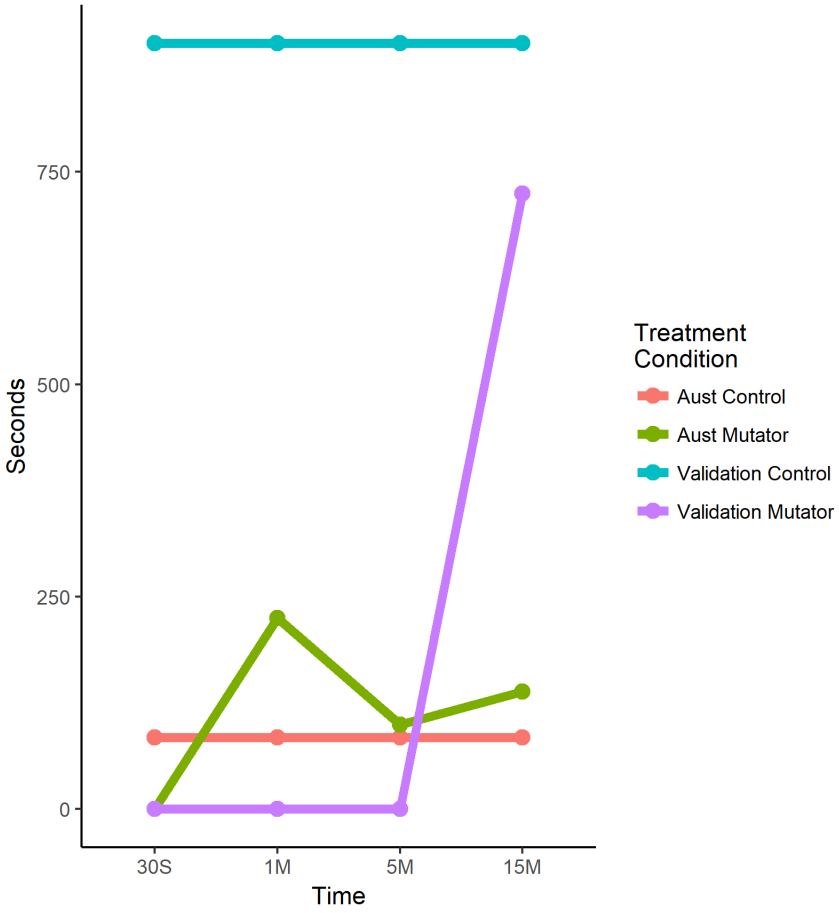



**Figure 16. Intense Scan Results**

#### 5.2.2.2 Hosts found.

Figure 17 reports the number of hosts found by intense network scans in control and RHM trials. In both the original data and the validation trial, RHM scans consistently report a number of hosts that was less than 74% of active hosts on the network at the time of scan. Aust's control data is not visible on the figure because it has the exact same values

as the validation control data. Unlike Aust's results, scans were manually terminated if they persist beyond the amount of time required for two mutations to occur. Any results from such a scan are out of date and therefore useless to an adversary. This accounts for the results of the 30-second, one-, and five-minute mutation windows where no hosts were found. The reason for results from intense scans in Aust's study at smaller mutation intervals is likely due to logging intermediate output of the Nmap scan in combination with the fact that intense scans by Aust finished in a shorter amount of time (Figure 16). Since this was not specified in the experimental procedure, intermediate output was not logged and therefore lost when scans were terminated after exceeding the two-mutation limit [24]. For scans at the 15 minute interval, a similar amount of hosts were discovered when compared to Aust's data. This data suggests that mutation intervals are most effective when shorter than scanning activity by an adversary.

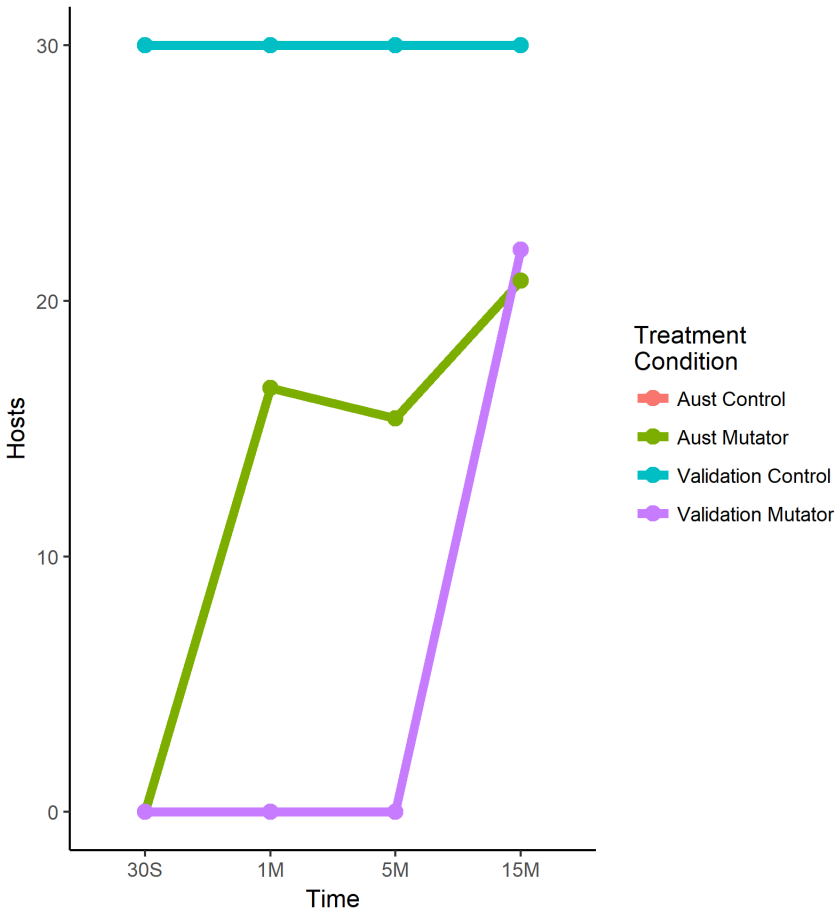



**Figure 17. Intense Scan Hosts Found**

## 5.3 Quality of Service Data

This section reports the QoS data for the protocols under test. Each applicable cell in Table 7 has a corresponding section for each protocol. The output of T-tests support claims of statistical significance. T-tests are reported using a standardized format of:

$$t([degrees\ of\ freedom])=[t\text{-}statistic],\ p = [p\text{-}value]$$

### 5.3.1 FTP.

FTP failed to function under a RHM network. A connection could not be established since server transmits its Real IP address (rIP) in payload of the FTP data to establish a passive connection. While it is possible to open a connection by supplying a FQDN

instead of an IP address, this would require modification to the mutator script to update the Domain Name System (DNS) cache on the client. This process must occur in sync with the frequency of mutations. Section 6.4 examines this concept in greater detail. Trials used the Virtual IP address (vIP) of the FTP server to initiate connections. As a result, no useful test data was collected to compare with control data as the client could not establish a data channel to to the rIP of the server. Since the entire point of the mutator framework is a focus on masking the rIP with a shifting vIP, the rIP was not accessible by the client.

Figure 18 contains the important packet exchanges of the mutator FTP connection. The "No." column indicates the frame number. "Source" and "Destination" show the source IP address of the client and server. "Protocol" indicates the protocol associated with a given frame. "Length" states the length of a frame in bytes. "Info" provides an overview of the data transmitted in a frame. This packet capture occurred on the client (rIP of 10.13.1.8) when it communicated with the FTP server (vIP of 10.13.1.44). Frames 9-11 contain the three-way handshake for initialization of the Transmission Control Protocol (TCP) connection. Frames 12-16 show that FTP control channel data can travel between the client and server. Frames 17-39 contain similar information and are omitted for brevity. The entire conversation, including frames 17-39, is available in Appendix B.1.

The client requests a passive FTP connection in frame 40. The server responds with frame 41 and states that the IP address to use for the connection is its rIP of 10.13.1.4. The payload of this packet contains the rIP of the server in the payload, shown in Figure 19 and boxed in red. The client does not know how to contact 10.13.1.4, so it creates Address Resolution Protocol (ARP) requests such as those shown in frames 42 and 44-46. Since the rIP:vIP table only resolves vIPs, the passive IP address in the payload is inaccessible. Figure 20 displays the contents of an ARP request (boxed in red) created in this transmission. 10.13.1.4 is not in the mutations table so no ARP responses appear and the file transfer cannot happen. Frames 47-54 show the termination of the connection between client and server.

**Figure 18. Filtered FTP Stream**

| No. | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 9 | 10.13.1.8 | 10.13.1.44 | TCP | 74 | 41248 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=373596937 TSecr=0 WS=128 |
| 10 | 10.13.1.44 | 10.13.1.8 | TCP | 74 | 21 → 41248 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=614783254 TSecr=373596937 |
| 11 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=373596942 TSecr=614783254 |
| 12 | 10.13.1.44 | 10.13.1.8 | FTP | 93 | Response: 220 Microsoft FTP Service |
| 13 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=1 Ack=28 Win=29312 Len=0 TSval=373596943 TSecr=614783256 |
| 14 | 10.13.1.8 | 10.13.1.44 | FTP | 72 | Request: FEAT |
| 15 | 10.13.1.44 | 10.13.1.8 | FTP | 100 | Response: 211-Extended features supported: |
| 16 | 10.13.1.44 | 10.13.1.8 | FTP | 84 | Response: LANG EN* |
| 40 | 10.13.1.8 | 10.13.1.44 | FTP | 72 | Request: PASV |
| 41 | 10.13.1.44 | 10.13.1.8 | FTP | 113 | Response: 227 Entering Passive Mode (10,13,1,4,19,253). |
| 42 | Vmware_9d:... | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 43 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=133 Ack=542 Win=29312 Len=0 TSval=373596958 TSecr=614783258 |
| 44 | Vmware_9d:... | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 45 | Vmware_9d:... | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 46 | Vmware_9d:... | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 47 | 10.13.1.8 | 10.13.1.44 | FTP | 69 | Request: \377\364\377 |
| 48 | 10.13.1.8 | 10.13.1.44 | FTP | 67 | Request: \362 |
| 49 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [FIN, ACK] Seq=137 Ack=542 Win=29312 Len=0 TSval=373597698 TSecr=614783258 |
| 50 | 10.13.1.44 | 10.13.1.8 | TCP | 66 | 21 → 41248 [ACK] Seq=542 Ack=137 Win=66304 Len=0 TSval=614783558 TSecr=373597698 |
| 51 | 10.13.1.44 | 10.13.1.8 | TCP | 66 | 21 → 41248 [ACK] Seq=542 Ack=138 Win=66304 Len=0 TSval=614783558 TSecr=373597698 |
| 52 | 10.13.1.44 | 10.13.1.8 | TCP | 66 | 21 → 41248 [FIN, ACK] Seq=542 Ack=138 Win=66304 Len=0 TSval=614783558 TSecr=373597698 |
| 53 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=138 Ack=543 Win=29312 Len=0 TSval=373597699 TSecr=614783558 |
| 54 | 10.13.1.44 | 10.13.1.8 | TCP | 60 | 21 → 41248 [RST, ACK] Seq=543 Ack=138 Win=0 Len=0 |

**Figure 19. FTP Server Initiation of Passive Mode in Frame 41**

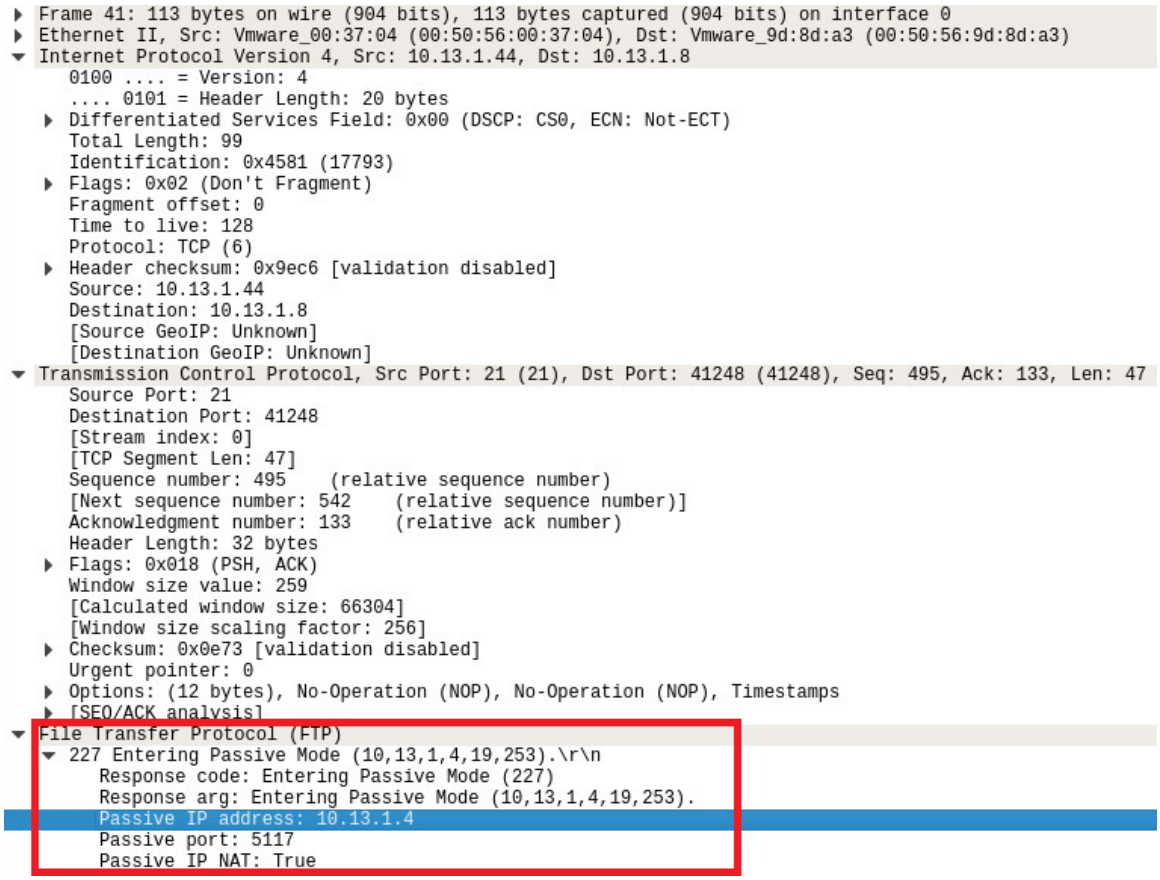
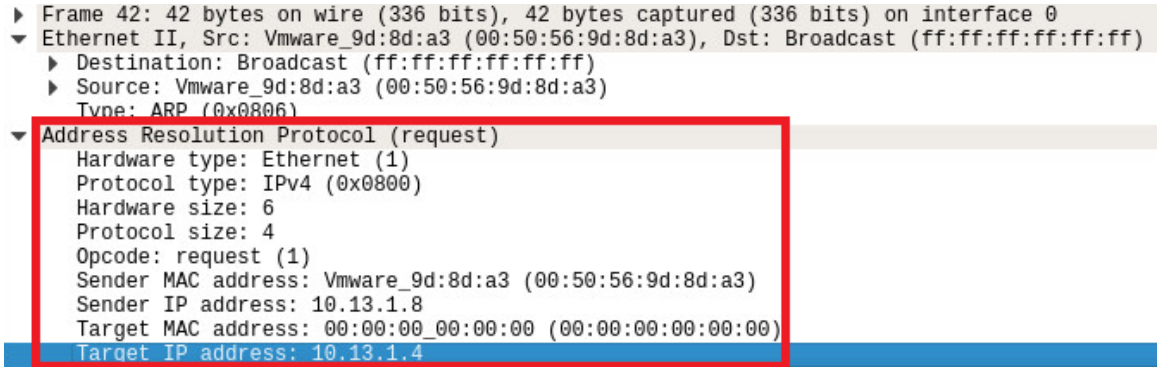



**Figure 20. ARP Request for rIP of FTP Server in Frame 42**




This data demonstrates the usability issue that RHM-enabled networks present for FTP connections. As long at FTP transmits the client or server rIP as in Figure 19, then this protocol will not function with RHM. Alternative means of file transfer should be examined if possible to use RHM on a network.

### 5.3.2 HTTP.

HTTP is a stateless application-level protocol designed for flexible interaction with network-based hypertext information systems [51]. This protocol presents a uniform interface to clients and has seen widespread adoption since its inception in 1989. TCP is a common choice of transport layer protocol for HTTP as HTTP assumes a reliable transport layer protocol. Four metrics were assessed: latency, Round Trip Time (RTT), throughput, and dropped packets. Of these four metrics, three indicated a statistically-significant difference between the control and mutator trials. The following sections provide box & whisker plots for tested metrics and an overview of t-test results. Unabridged output from t-tests and the R script used to generate them are available in Appendix B.2 and Appendix D, respectively. Due to anomalous behavior discussed in Section 5.3.2.4, HTTP should be approached with caution when used with RHM as described in this thesis.

#### 5.3.2.1 Latency.

Figure 21 depicts the latency perceived by the server that hosts the HTTP service used by a client. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation latency (1.875e-03 seconds) is greater than average latency in control experiments (3.249e-05 seconds). There was a statistically-significant difference in the latency for HTTP communications under a network using RHM; $t(58)=57.229$, $p < 2.2e\text{-}16$. While this difference was statistically significant, a difference of 2 milliseconds may still be an acceptable increase in latency for some networks.

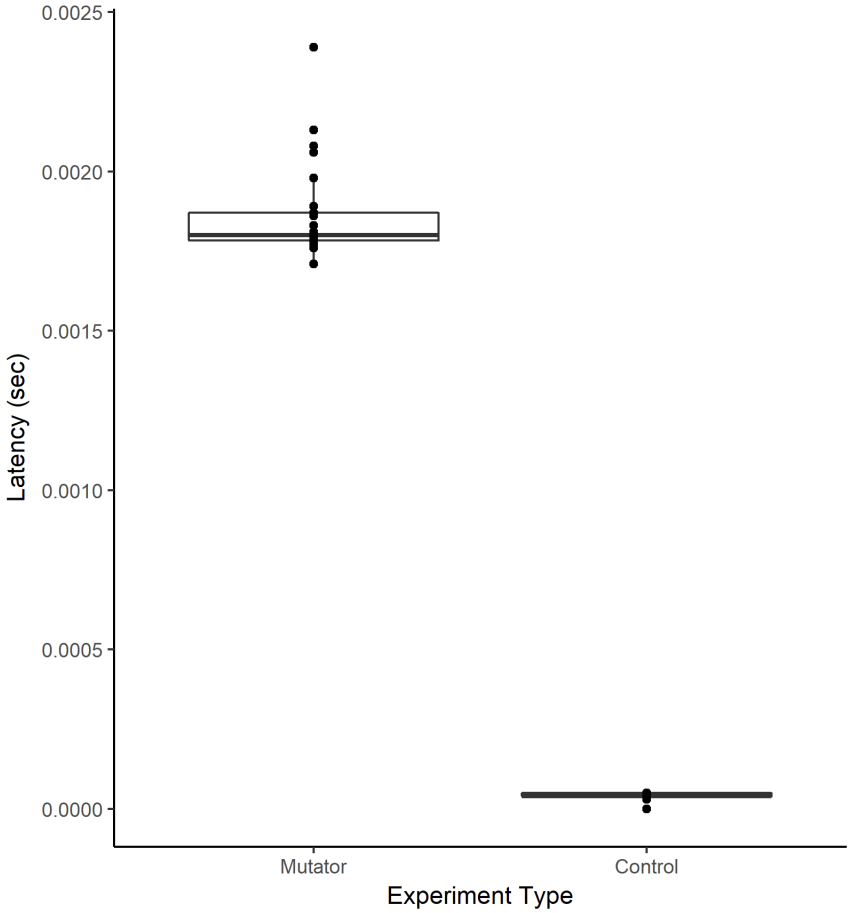



**Figure 21. Box & Whisker Plot of Latency Between Control and Mutator Trials for HTTP**

#### 5.3.2.2 RTT.

Figure 22 portrays the RTT for data sent by the client and subsequent server responses from the HTTP server. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation RTT (0.020 seconds) is greater than average RTT in control experiments (0.004 seconds). There was a statistically-significant difference in the RTT for IMAP communications under a network using RHM; t(58)=69.219, p <2.2e-16. The increase in RTT is due to the performance of HTTP under mutations, which sent massive amounts of traffic to convey the same amount of data when compared to the control, which was set to transmit at a limited rate. This restricted rate of transmission was chosen to decrease the size of packet capture files needed for analysis but still have an active connection across mutations. In other words, the mutator connection

needed to work "harder" (i.e., send more data at a faster rate) to transfer the same data as control trials.

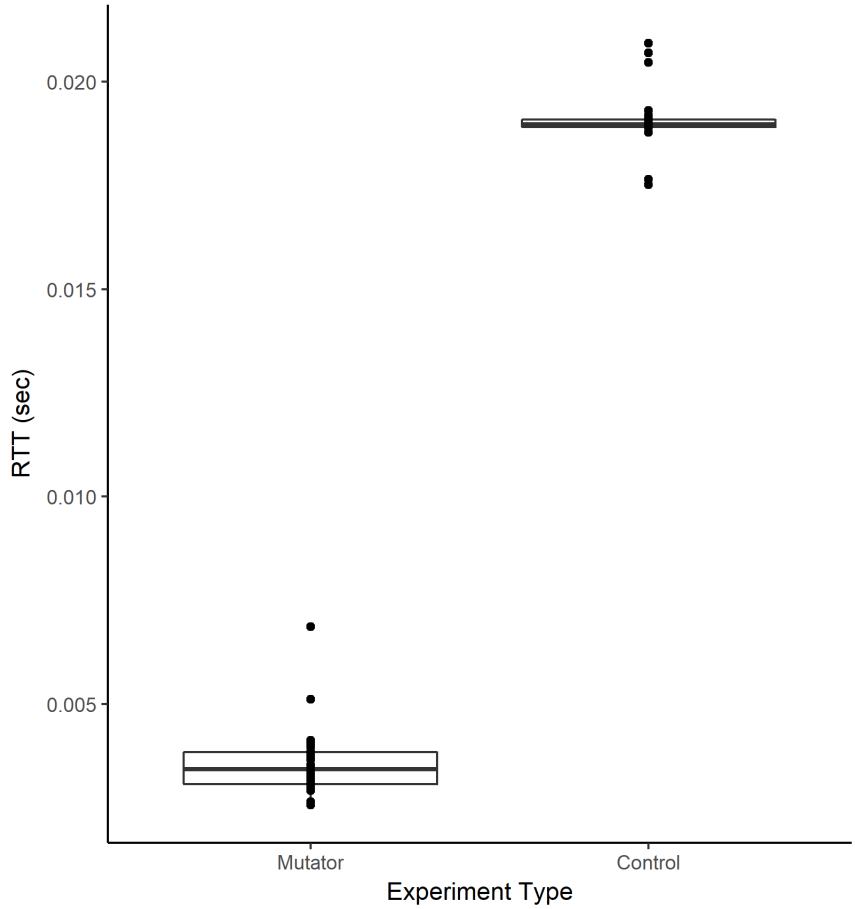



**Figure 22. Box & Whisker Plot of RTT Between Control and Mutator Trials for HTTP**

#### 5.3.2.3 Throughput.

Figure 23 details the throughput for data received by the server in terms of bits-per-second (bps). The y-axis represents bps. The x-axis separates trials with RHM or control trials without RHM. Average mutation throughput (7396.133 bps) is greater than average throughput in control experiments (6474.200 bps). There was a statistically-significant difference in the throughput for HTTP communications under a network using RHM; $t(58)=12.483$, $p < 2.2e\text{-}16$. HTTP mutation trials exhibited anomalous behavior that caused a large amount of retransmission packets to be sent during a trial. This accounts for the increased throughput even though the same file was transferred between hosts. As stated

in Section 5.3.2.2, the mutator connection needed to work "harder" (i.e., send more data at a faster rate) to transfer the same data as control trials.

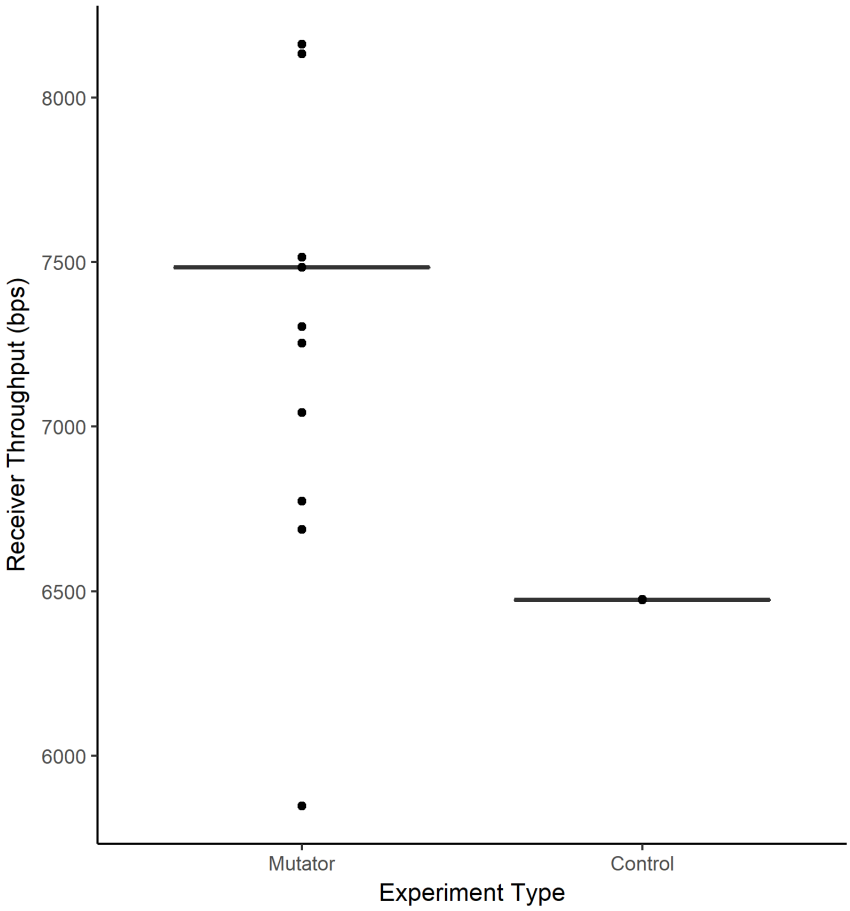



**Figure 23. Box & Whisker Plot of Throughput Between Control and Mutator Trials for HTTP**

#### 5.3.2.4 Dropped Packets.

HTTP traffic presented anomalous behavior when compared to control trials. While a standard control trial generated somewhere in the area of 5000 packets on the server side, mutator trials generated anywhere between 69,000 and 71,000 packets. Due to the underlying structure of TCP, all the HTTP data successfully transferred across. However, this comes at the cost of nearly 14.2 times as much traffic when compared to control trials. The balance of packet types seen across control and mutation trials is shown in Figure 24. It is hard to imagine a scenario where so much extraneous HTTP traffic is acceptable on

a real-world network. The root cause of this issue is puzzling since other TCP protocols did not exhibit similar behavior. Further research is required to determine the root cause, though there is a high chance the answer lies within the mutator script since no other aspect of the network or connection used was changed between trials. For this reason, RHM as presented in this thesis is unsuitable for HTTP traffic without further research.

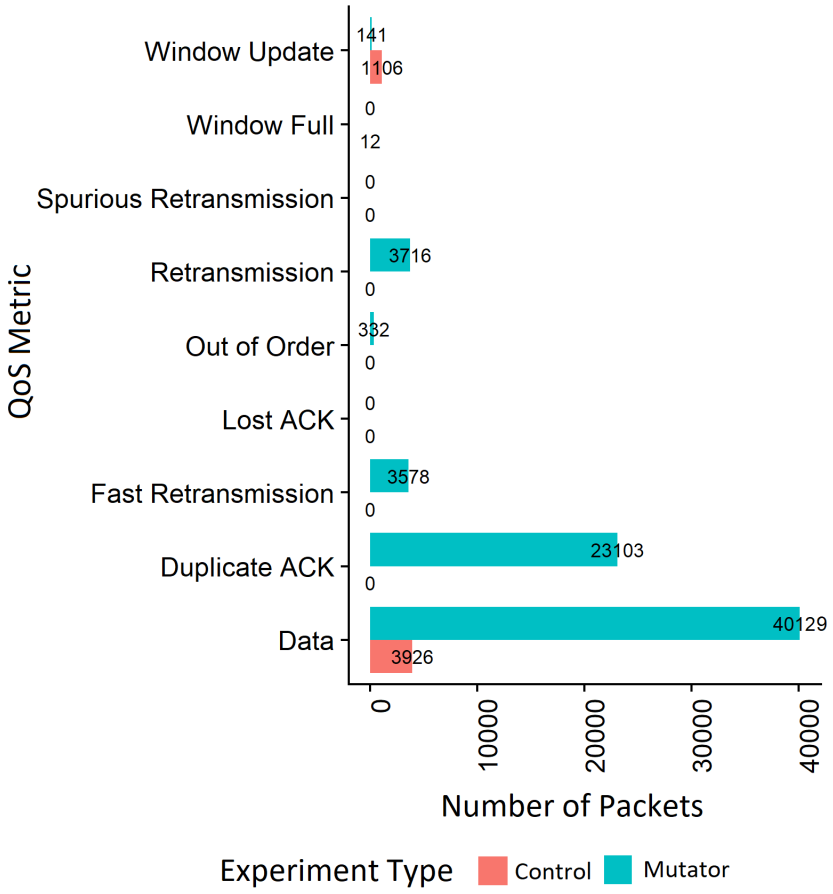



**Figure 24. Distribution of Packet Types Across All Control and Mutator Trials for HTTP.**

Several types of traffic are shown in Figure 24. They are defined in the following list:

- *Window Update*: Window Updates occur when the receiving application has created enough space in its TCP buffer that in can handle more data from the sender [52].
- *Window Full*: These packets indicate that the TCP window is full and cannot receive more data until the TCP buffer has emptied [52].

- *Spurious Retransmission*: These are the retransmissions in which the receiver acknowledged the packet after the Sender retransmits the packet due to a retransmission time out [53].

- *Retransmission*: Occurs when the sender retransmits a packet after the expiration of the acknowledgment [52].

- *Out of Order*: Occurs when a packet is seen with a sequence number lower than the previously received packet on that connection [52].

- *Lost ACK*: These are ACKs that Wireshark cannot match with a sent segment [52].

- *Fast Retransmission*: Occurs when the sender retransmits a packet before the expiration of the acknowledgment timer [52]. Senders should perform this action upon the receipt of three duplicate ACKs [52].

- *Duplicate ACK*: Occurs when the same ACK number is seen and it is lower than the last byte of data sent by the sender [52].

- *Data*: These packets contain the actual data requested by the client in a transmission.

### 5.3.3 IMAP.

IMAP is a standard email protocol defined by RFC 3501 that enables a client to store and manipulate messages on a server in a way that is similar to local folders [54]. IMAP uses a Transmission Control Protocol/Internet Protocol (TCP/IP) connection to transfer data. Four metrics were assessed: latency, RTT, throughput, and dropped packets. Of these four metrics, only latency indicated a statistically-significant difference between the control and mutator trials. The following sections provide box & whisker plots for tested metrics and an overview of t-test results. Unabridged output from t-tests and the R script used to generate them are available in Appendix B.3 and Appendix D, respectively. Overall, IMAP did not exhibit behavior that suggested it is unsuitable for use a RHM-enabled network.

#### 5.3.3.1 Latency.

Figure 25 depicts the latency perceived by the server that hosts the IMAP service used by a client. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation latency (0.004 seconds) is greater than average latency in control experiments (0.001 seconds). There was a statistically-significant difference in the latency for IMAP communications under a network using RHM; t(58)=4.4067, p= 4.593e-05. An extra 3 milliseconds of latency on average for IMAP connections is unlikely to have an adverse impact for users of a network.

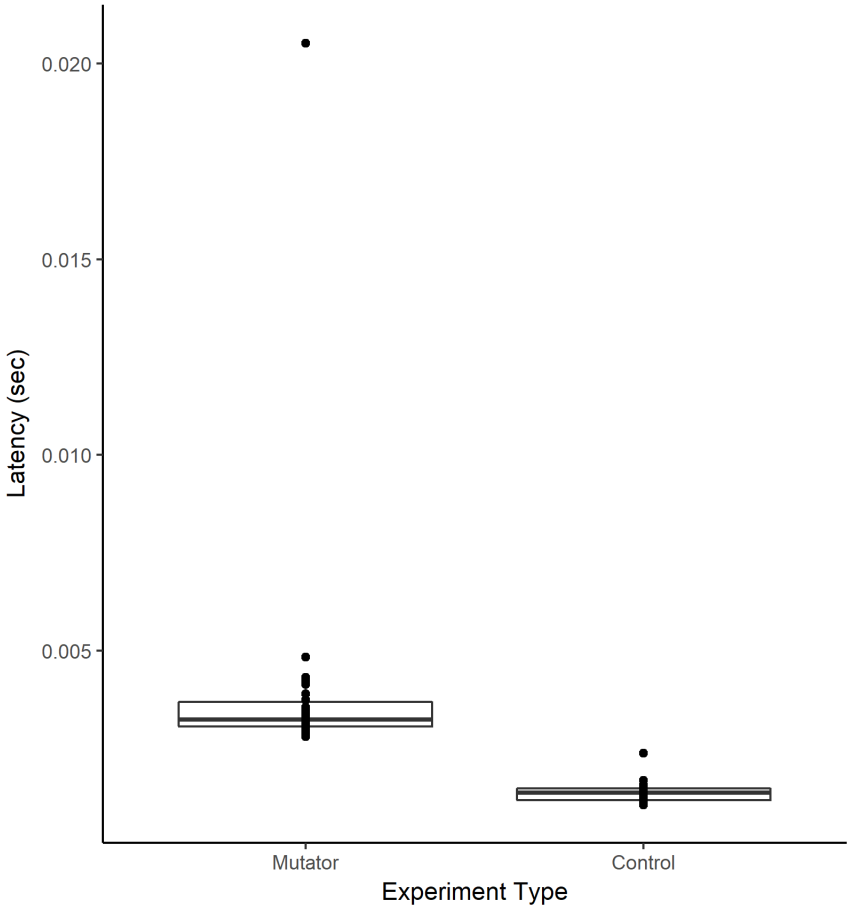



**Figure 25. Box & Whisker Plot of Latency Between Control and Mutator Trials for IMAP**

#### 5.3.3.2 RTT.

Figure 26 portrays the RTT for data sent by the client and subsequent server responses from the IMAP server. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation RTT (0.035 seconds) is greater than average RTT in control experiments (0.001 seconds). This difference across several orders of magnitude is due to outliers in the mutation trials. Without outliers, the average is 0.002 seconds. There was not a statistically-significant difference in the RTT for IMAP communications under a network using RHM; t(58)=1.4843, p= 0.1431. An additional millisecond added to RTT is a small price to pay for the benefits of RHM.

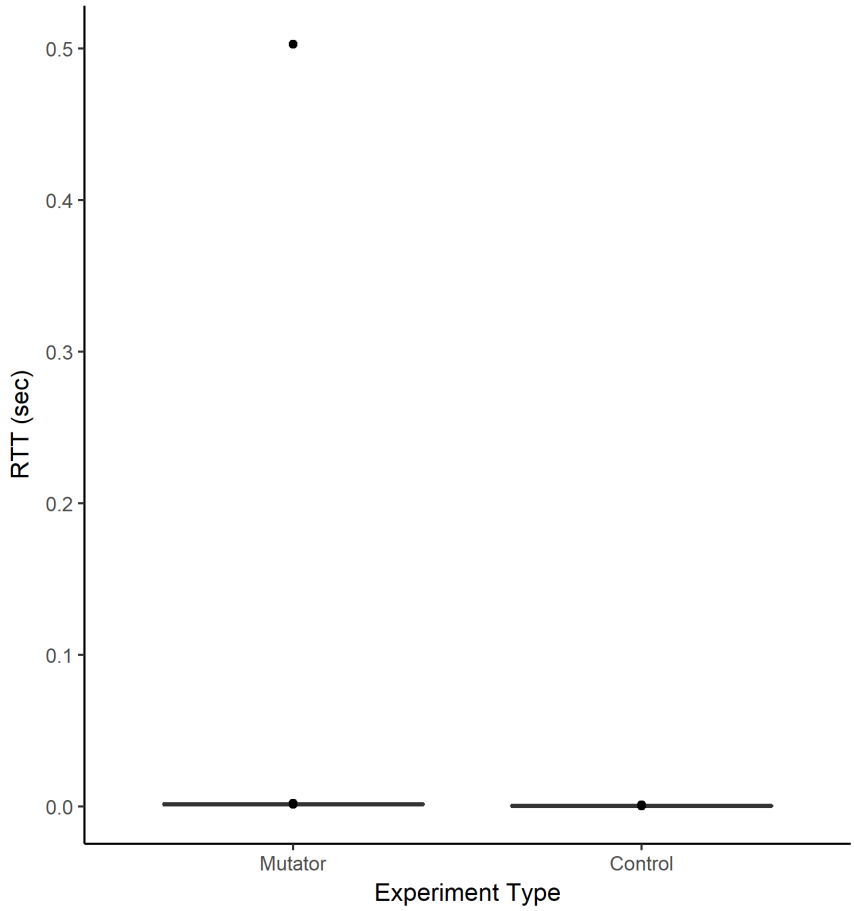



**Figure 26. Box & Whisker Plot of RTT Between Control and Mutator Trials for IMAP**

#### 5.3.3.3 Throughput.

Figure 27 details the throughput for data received by the server in terms of Bits per second (bps). The y-axis represents bps. The x-axis separates trials with RHM or control trials without RHM. Average mutation throughput (382.133 bps) is greater than average throughput in control experiments (380.000 bps). The slight increase in average throughput can be explained by the additional packets required to make up for dropped packets recored in Section 5.3.3.4. There was not a statistically-significant difference in the throughput for IMAP communications under a network using RHM; t(58)=1.3442, p= 0.1841. With respect to throughput, there is no meaningful difference and therefore no adverse impact to QoS.

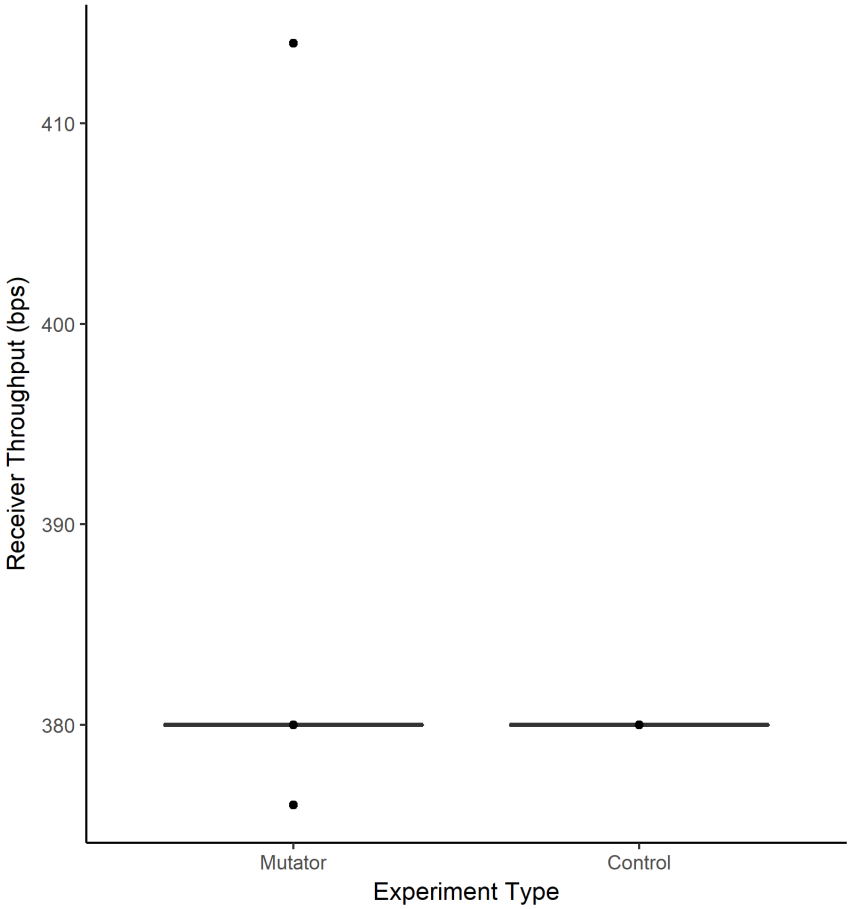



**Figure 27. Box & Whisker Plot of Throughput Between Control and Mutator Trials for IMAP**

#### 5.3.3.4 Dropped Packets.

Figure 28 illustrates the average amount of dropped packets for control and mutation trials. The y-axis represents the number of dropped packets, which is an integer value since packets cannot be partially dropped. The x-axis separates trials with RHM or control trials without RHM. The mutation drop rate of 0.300 packets is the calculated average across all trials and is greater than the average drop rate in control experiments (0.000 packets). The IMAP connection did not omit any data as TCP has built-in safeguards to handle dropped packets. There was not a statistically-significant difference in the amount of dropped packets for IMAP communications under a network using RHM; $t(58)=1.6075$, $p= 0.1134$.

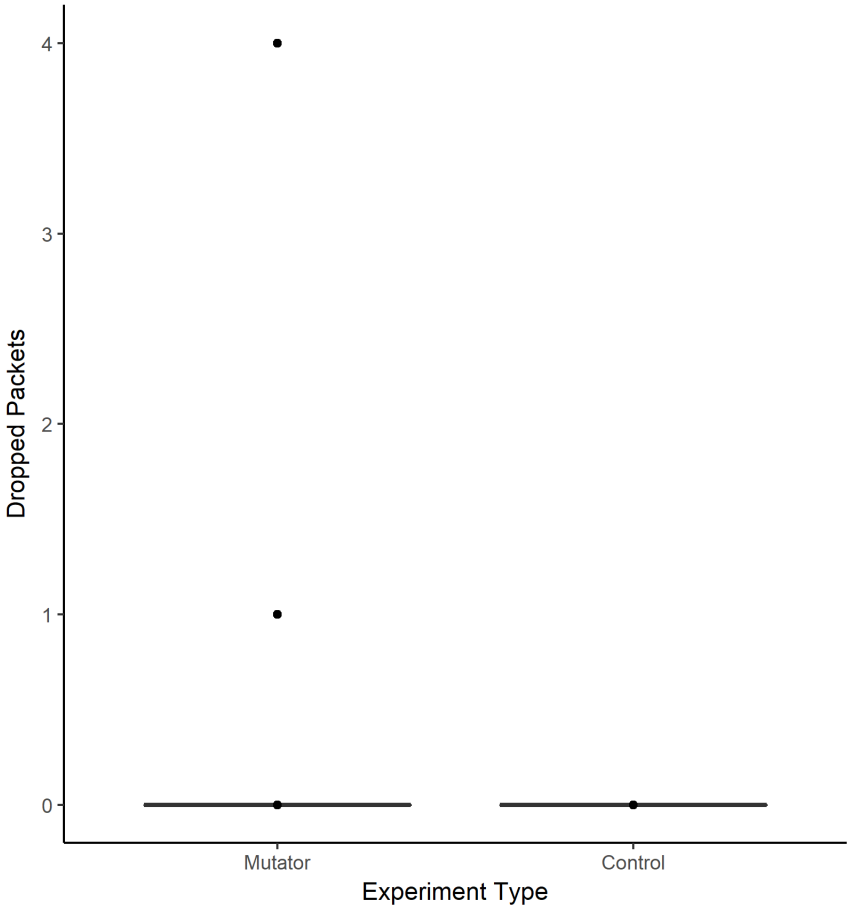



**Figure 28. Box & Whisker Plot for Number of Dropped Packets Between Control and Mutator Trials for IMAP**

### 5.3.4 POP.

POP is a protocol used by local a e-mail client to access mail stored on a server. Per RFC 1939, POP normally downloads and then deletes mail from the server [55]. Extensive manipulations of mail are handled by more complex protocols, such as IMAP or webmail [55]. POP uses a TCP/IP connection to transfer data. Four metrics were assessed: latency, RTT, throughput, and dropped packets. Of these four metrics, two indicated a statistically-significant difference between the control and mutator trials. The following sections provide box & whisker plots for tested metrics and an overview of t-test results. Unabridged output from t-tests and the R script used to generate them are available in Appendix B.4 and Appendix D, respectively.

#### 5.3.4.1 Latency.

Figure 29 depicts the latency perceived by the server that hosts the POP service used by a client. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation latency (0.028 seconds) is greater than average latency in control experiments (0.001 seconds). There was a statistically-significant difference in the latency for POP communications under a network using RHM; $t(57)=184.96$, $p < 2.2e\text{-}16$. Since users retrieve email in sporadic bursts rather than sustained queries to the mail server, an increase of 27 milliseconds when compared to control data is unlikely to have critical impacts to QoS. In the event that such an increase in latency in unacceptable, IMAP and webmail have established themselves as other means to handle email traffic.

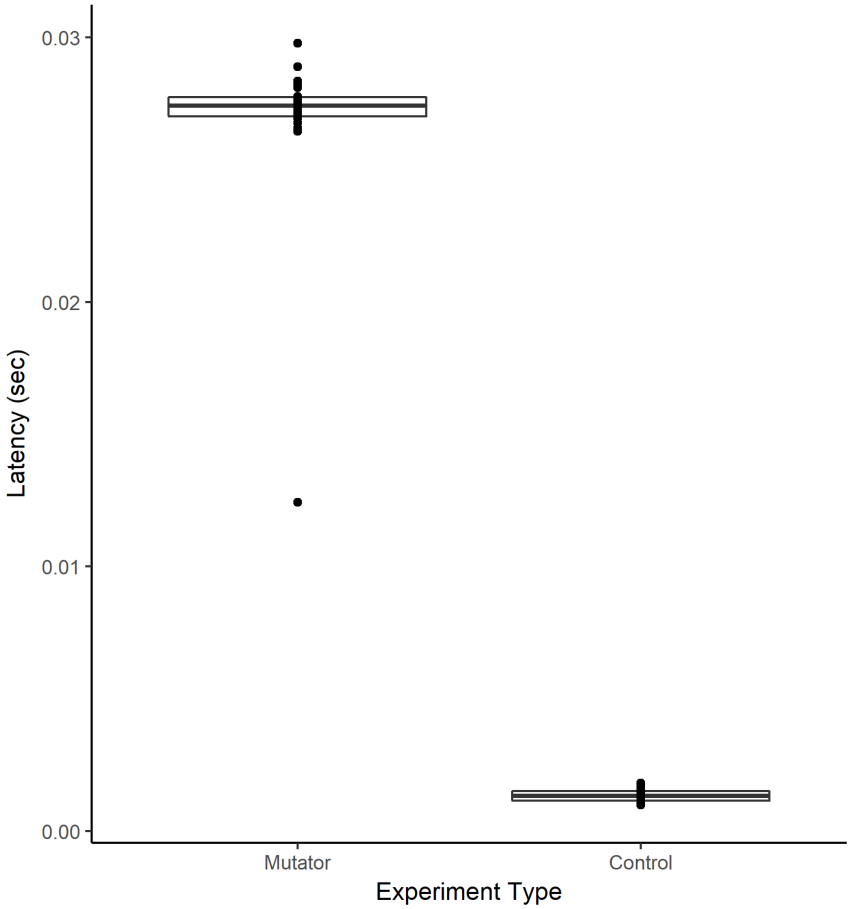



**Figure 29. Box & Whisker Plot of Latency Between Control and Mutator Trials for POP**

#### 5.3.4.2 RTT.

Figure 30 portrays the RTT for data sent by the client and subsequent server responses from the POP server. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation RTT (0.122 seconds) is greater than average RTT in control experiments (0.001 seconds). This difference across several orders of magnitude is due to two outliers that skews the data. Without these outliers, the average is 0.002 seconds. There was not a statistically-significant difference in the RTT for POP communications under a network using RHM; $t(57)=2.18$, $p= 0.0334$. When outliers are accounted for, an additional millisecond added to average RTT does not pose an adverse impact to QoS.

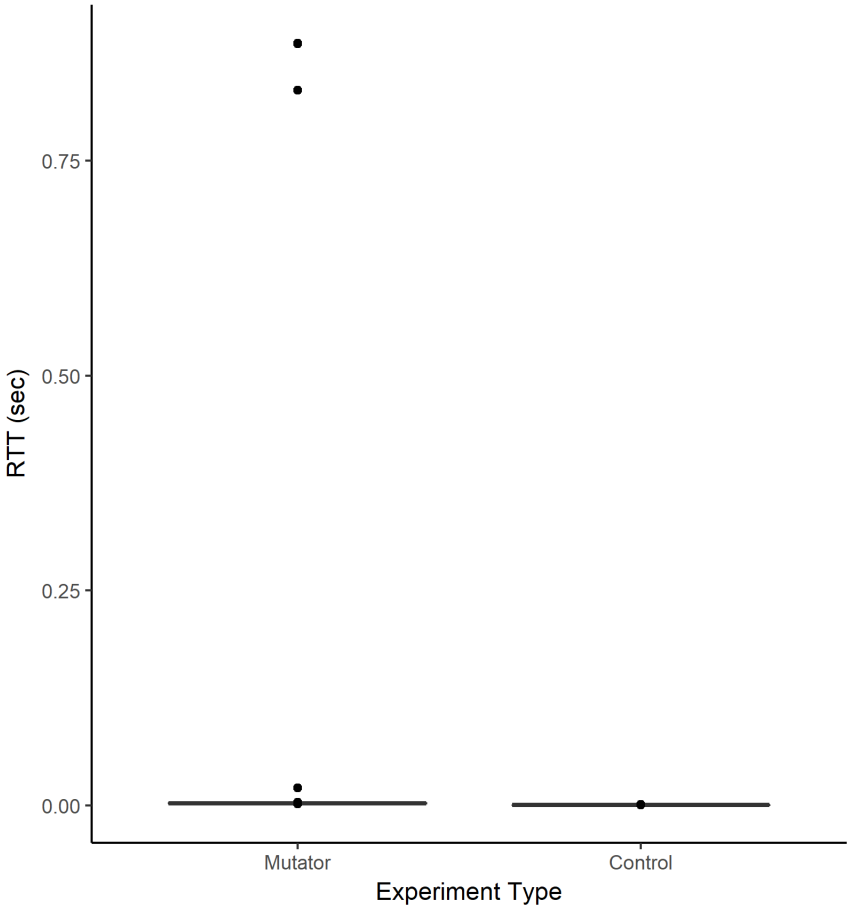



**Figure 30. Box & Whisker Plot of RTT Between Control and Mutator Trials for POP**

#### 5.3.4.3 Throughput.

Figure 31 details the throughput for data received by the server in terms of bits-per-second (bps). The y-axis represents bps. The x-axis separates trials with RHM or control trials without RHM. Average mutation throughput (318.667 bps) is greater than average throughput in control experiments (291.733 bps). The slight increase in average throughput can be explained by the additional packets required to make up for dropped packets recored in Section 5.3.4.4. There was a statistically-significant difference in the throughput for IMAP communications under a network using RHM; t(58)=10.189, p= 1.541e-14. With respect to throughput, the difference between control and mutator trials does not introduce enough extra traffic to the network to pose concerns about the impact on QoS.

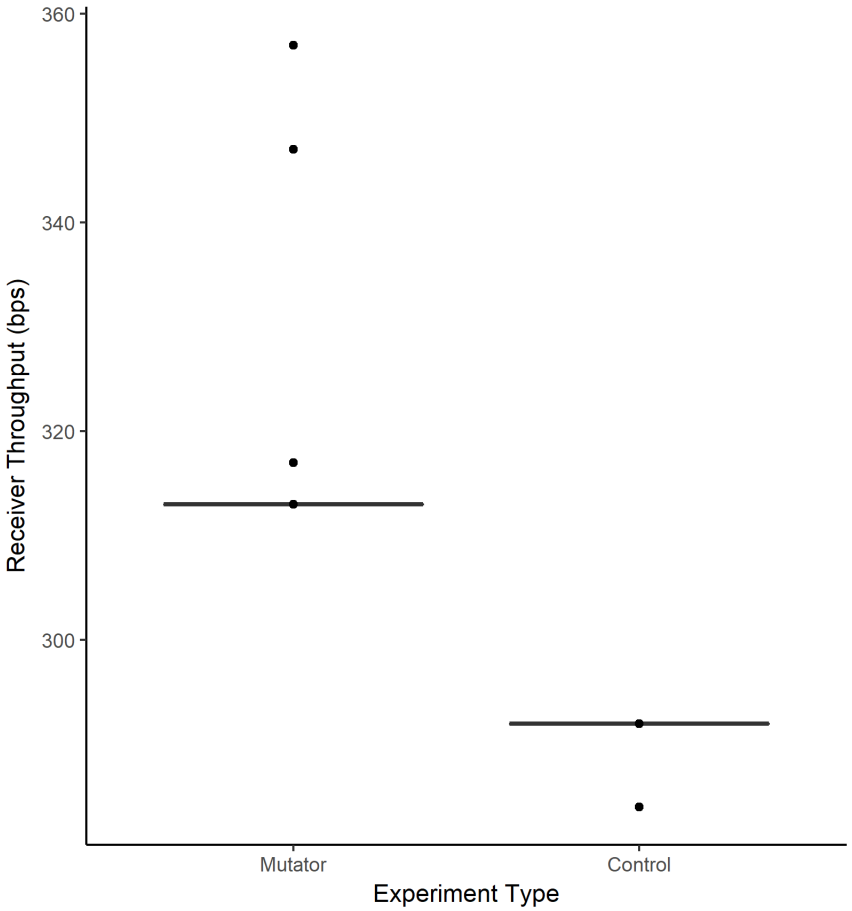



**Figure 31. Box & Whisker Plot of Throughput Between Control and Mutator Trials for POP**

#### 5.3.4.4 Dropped Packets.

Figure 32 illustrates the average amount of dropped packets for control and mutation trials. The y-axis represents the number of dropped packets, which is an integer value since packets cannot be partially dropped. The x-axis separates trials with RHM or control trials without RHM. The average mutation drop rate of 0.600 packets is the calculated average across all trials and is less than the average drop rate in control experiments (1.000 packets). Dropped packets are determined by the calculation of total packets perceived by the client minus total packets perceived by the server. In the case of POP, a reply to the `LIST` command from the client send messages that were longer than the standard Ethernet frame size of 1518 Bytes. Therefore, the server response was fragmented and perceived as two packets by the client. This means that no packets were dropped for the majority

of control experiments and the calculated average is misleading. The POP connection did not omit any data as TCP has built-in safeguards to handle dropped packets. There was not a statistically-significant difference in the amount of dropped packets for POP communications under a network using RHM; t(58)=1.4841, p= 0.1432.

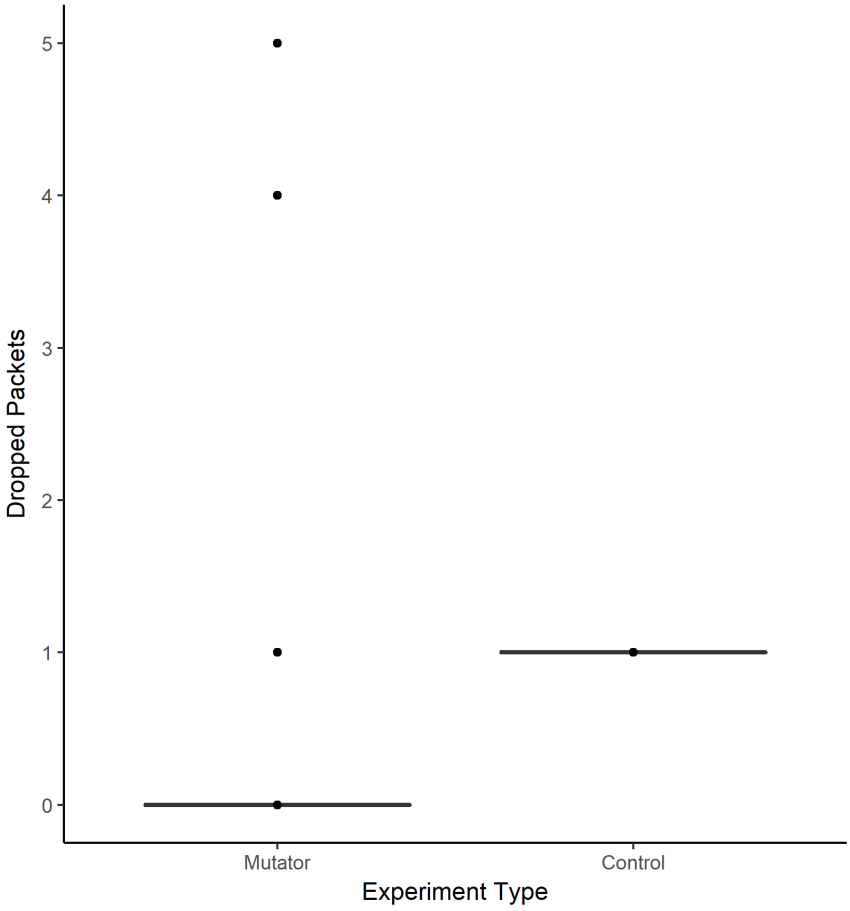



**Figure 32. Box & Whisker Plot of Dropped Packets Between Control and Mutator Trials for POP**

### 5.3.5 RTP.

Real-time Transport Protocol (RTP) delivers audio and video over IP networks. As such, it is prevalent in telephony, video teleconference, and television services. This protocol usually runs over User Datagram Protocol (UDP) to carry media streams. This protocol is one of the underpinnings of Voice over IP (VoIP). RFC 3550 provides the most current definition of the protocol [56]. Three metrics were assessed: jitter, throughput, and dropped

packets. Of these three metrics, only jitter indicated a difference between the control and mutator trials. The following sections provide box & whisker plots for tested metrics and an overview of t-test results. Unabridged output from t-tests and the R script used to generate them are available in Appendix B.5 and Appendix D, respectively.

#### 5.3.5.1 Jitter.

Figure 33 depicts the jitter perceived by the server that receives the RTP stream sent by the source. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation jitter (0.355 seconds) is greater than average jitter in control experiments (0.227 seconds). R was unable to conduct a t-test with the data supplied due to minimal variation in the data, which did not produce a meaningful t-statistic. Despite this, the difference in mean jitter is evidenced in Figure 33. Section 4.2.1 notes that maximum acceptable jitter is 50 milliseconds. However, part of the jitter data can be accounted for by the fact that packets were sent once per second. Examination of the increase in jitter from control to mutator, the average increase is 128 milliseconds. This is greater than the 50 millisecond maximum. This may present an adverse impact to QoS, but that determination should be made on a case by case basis dependent upon the network and application(s) in use.

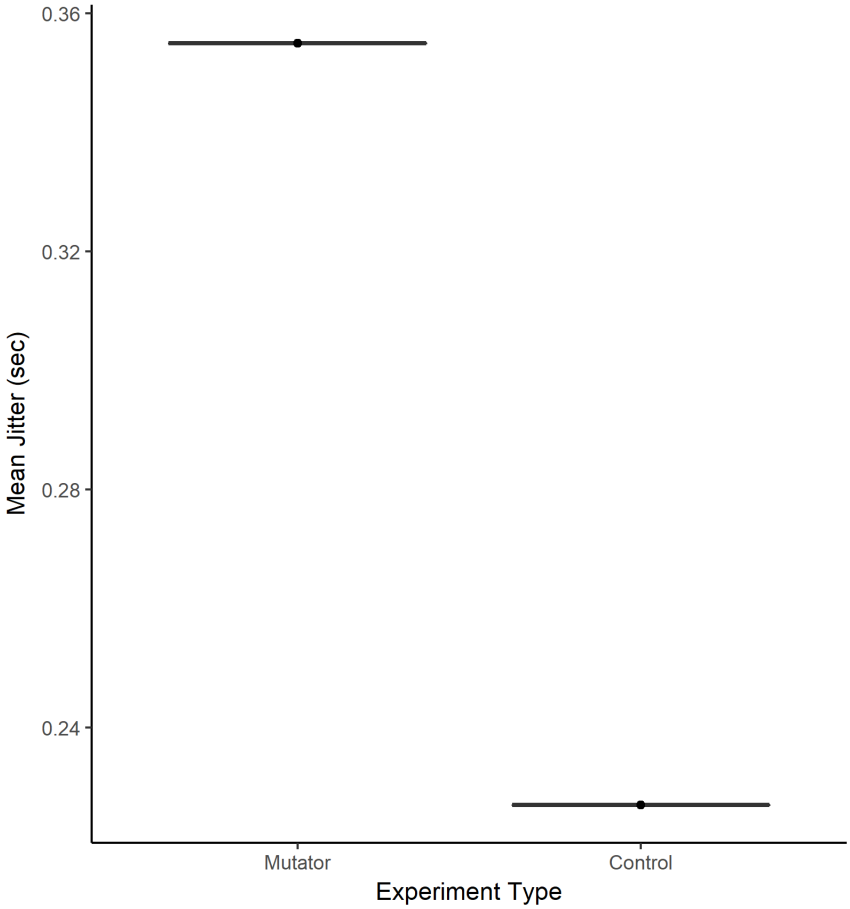



**Figure 33. Box & Whisker Plot of Jitter Experienced by Receiver Between Control and Mutator Trials for RTP**

#### 5.3.5.2 Throughput.

Figure 34 details the throughput for data received by the server in terms of bits-per-second (bps). The y-axis represents bps. The x-axis separates trials with RHM or control trials without RHM. Average mutation throughput (1551.800 bps) is greater than average throughput in control experiments (1427.733 bps). There was a statistically-significant difference in the throughput for RTP communications under a network using RHM; t(58)=6.1149, p= 8.804e-08. Much of the variety in throughput can be accounted for by the semi-automated testing procedure for RTP. Some of the variability is due to human imprecision since the packet capture and traffic generator were terminated by hand. The remaining outliers in Section 5.3.5.3 are primarily caused by dropped packets. A better

experimental design would have fully automated the capture and transmission of RTP data.

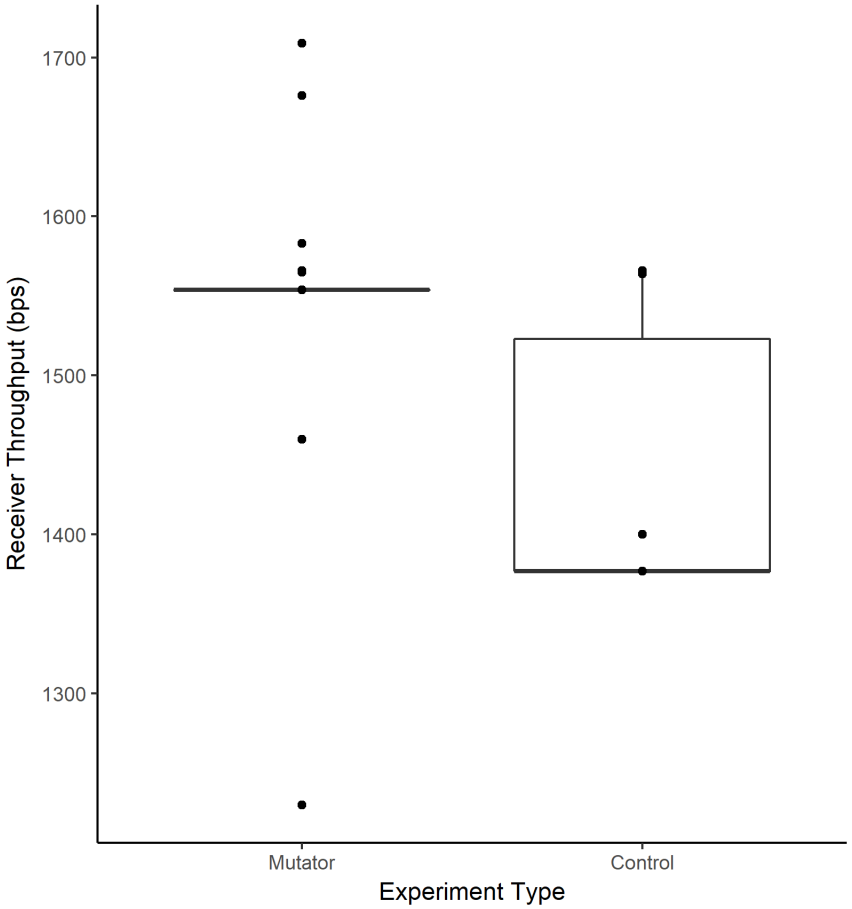



**Figure 34. Box & Whisker Plot of Throughput Between Control and Mutator Trials for RTP**

#### 5.3.5.3 Dropped Packets.

Figure 35 illustrates the average amount of dropped packets for control and mutation trials. The y-axis represents the number of dropped packets, which is an integer value since packets cannot be partially dropped. The x-axis separates trials with RHM or control trials without RHM. The mutation drop rate of 12.500 packets is the calculated average from all trials and is greater than the average drop rate in control experiments (-8.400 packets). Since RTP uses UDP at the transport layer, those dropped packets did not reach the server. There was a statistically-significant difference in the amount of dropped packets for RTP communications under a network using RHM; t(58)=9.1971, p= 6.28e-13. Negative results

in the dropped amount of packets are accounted for by the method in which dropped packets for RTP were calculated (i.e., $TotalSent - TotalReceived$). For RTP, the individual streams were isolated in Wireshark and the total amount of RTP packets sent by the client an server were computed. The client total subtracted from the server total indicated the amount of packets dropped. Timing issues in the semi-automated format of experiments likely caused negative results in situations where the packet capture began after transmission started. A better experimental design would have fully automated the capture and transmission of RTP data. Despite these issues, RTP packets still seem to successfully arrive in a RHM-enabled network.

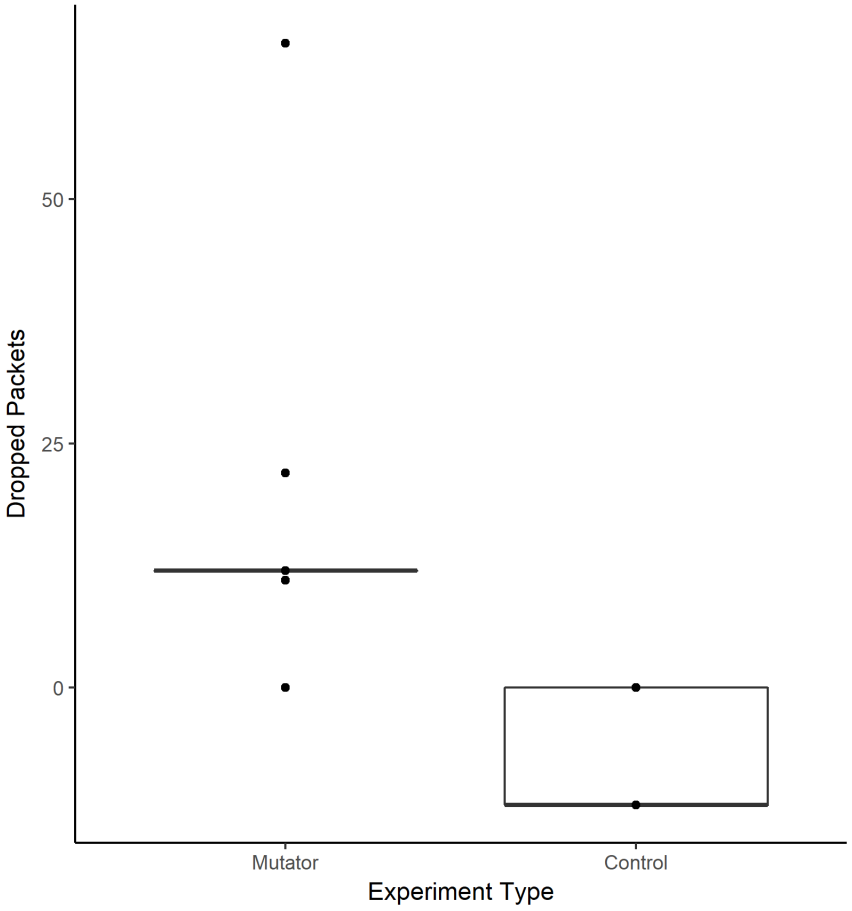



**Figure 35. Box & Whisker Plot of Dropped Packets Between Control and Mutator Trials for RTP**

### 5.3.6 SMTP.

SMTP is standardized form of email transmission defined in RFC 821 and updated in RFC 5321. Mail servers and other mail agents use SMTP to send or receive messages whereas clients may use IMAP, POP, or webmail. SMTP requires a reliable ordered data stream channel, so TCP is a common choice but other modes are possible [57]. Four metrics were assessed: latency, RTT, throughput, and dropped packets. Of these four metrics, only latency indicated a statistically-significant difference between the control and mutator trials. The following sections provide box & whisker plots for tested metrics and an overview of t-test results. Unabridged output from t-tests and the R script used to generate them are available in Appendix B.6 and Appendix D, respectively.

#### 5.3.6.1 Latency.

Figure 36 depicts the latency perceived by the server that hosts the SMTP service in use. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation latency (0.065 seconds) is greater than average latency in control experiments (0.061 seconds). There was a statistically-significant difference in the latency for SMTP communications under a network using RHM; t(58)=5.6527, p=5.056e-07. However, an increase in average latency of four milliseconds is unlikely to have a noticeable impact on applications that require SMTP.

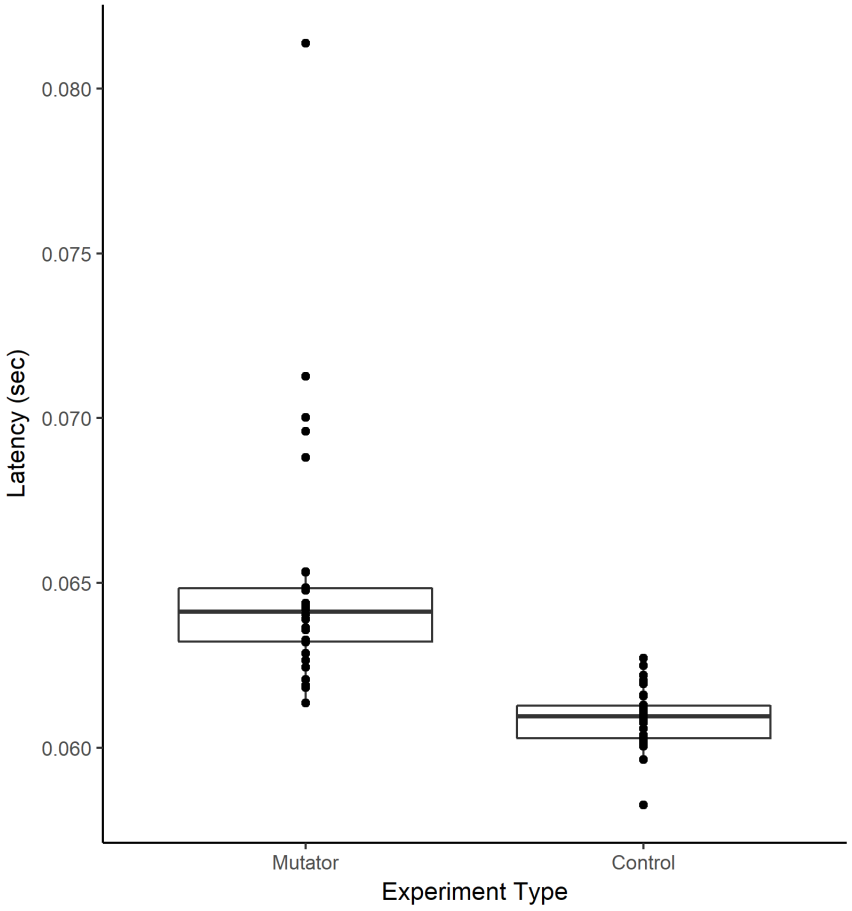



**Figure 36. Box & Whisker Plot of Latency Between Control and Mutator Trials for SMTP**

#### 5.3.6.2 RTT.

Figure 37 portrays the RTT for data sent by the client and subsequent server responses from the SMTP server. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation RTT (0.133 seconds) is greater than average RTT in control experiments (0.013 seconds). Four outliers skewed the mutator averages. Without the outliers, the mutation average is 0.014 seconds. Despite outliers, there was not a statistically-significant difference in the RTT for IMAP communications under a network using RHM; $t(58)=2.1415$, $p= 0.03645$. Without outliers, an average increase in RTT of 1 millisecond is unlikely to have an noticeable impact and rare cases where RTT is greater pose little threat to reliable QoS.

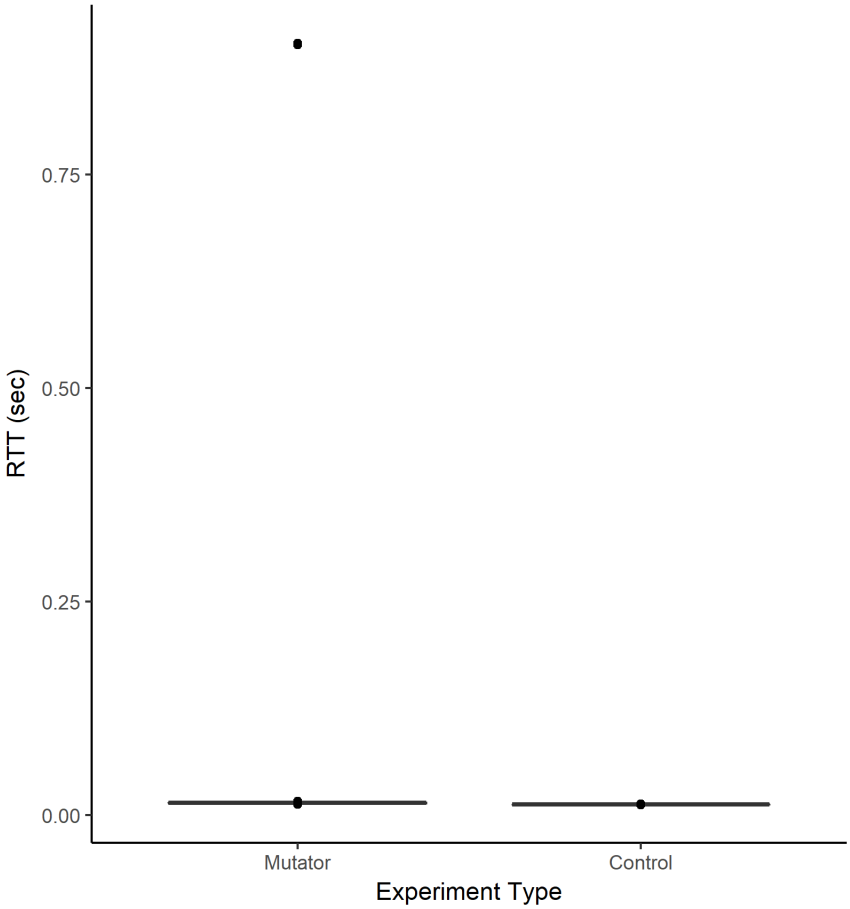



**Figure 37. Box & Whisker Plot of RTT Between Control and Mutator Trials for SMTP**

#### 5.3.6.3 Throughput.

Figure 38 details the throughput for data received by the server in terms of bps. The y-axis represents bps. The x-axis separates trials with RHM or control trials without RHM. Average mutation throughput (226.400 bps) is less than average throughput in control experiments (228.000 bps). There was not a statistically-significant difference in the throughput for IMAP communications under a network using RHM; t(58)=2.1122, p= 0.03898. Some of the outliers in throughput for mutator trials can be accounted for by trials that terminated packet captures before the SMTP connection was closed. Increased automation of the testing procedure would eliminate this discrepancy.

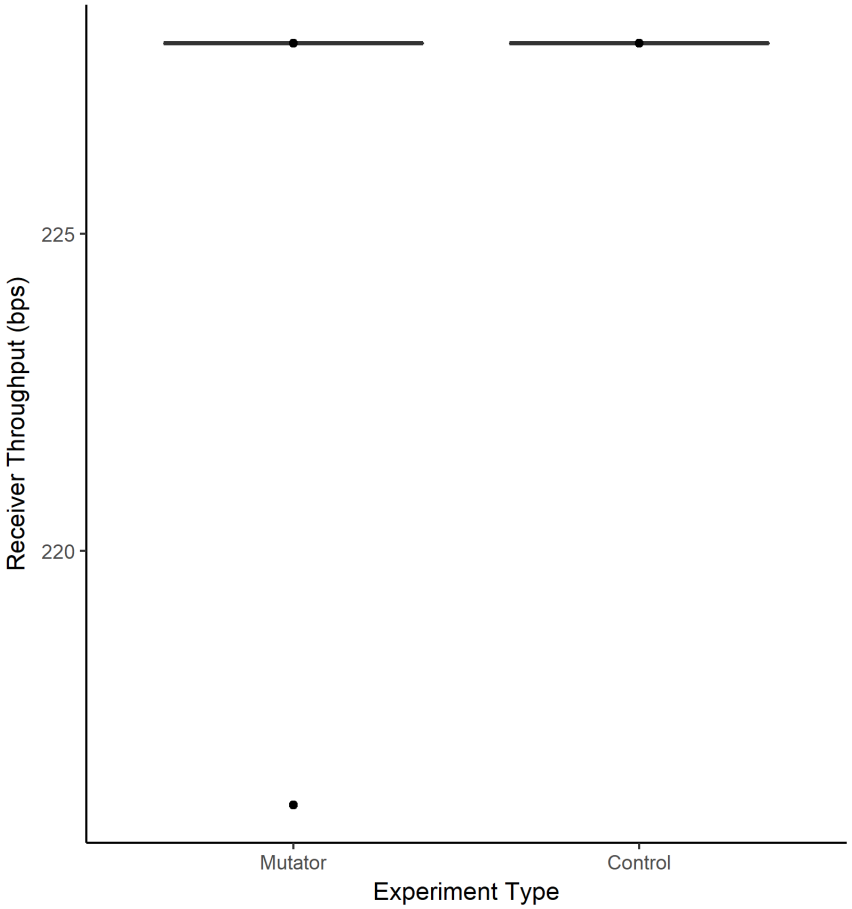



**Figure 38. Box & Whisker Plot of Throughput Between Control and Mutator Trials for SMTP**

#### 5.3.6.4 Dropped Packets.

Figure 39 illustrates the average amount of dropped packets for control and mutation trials. The y-axis represents the number of dropped packets, which is an integer value since packets cannot be partially dropped. The x-axis separates trials with RHM or control trials without RHM. The average mutation drop rate of 0.533 packets is the calculated average based upon all trials and is greater than the average drop rate in control experiments (0.000 packets). The SMTP connection did not omit any data as TCP has built-in safeguards to handle dropped packets. There was not a statistically-significant difference in the amount of dropped packets for SMTP communications under a network using RHM; $t(58)=2.1122$, $p= 0.03898$.

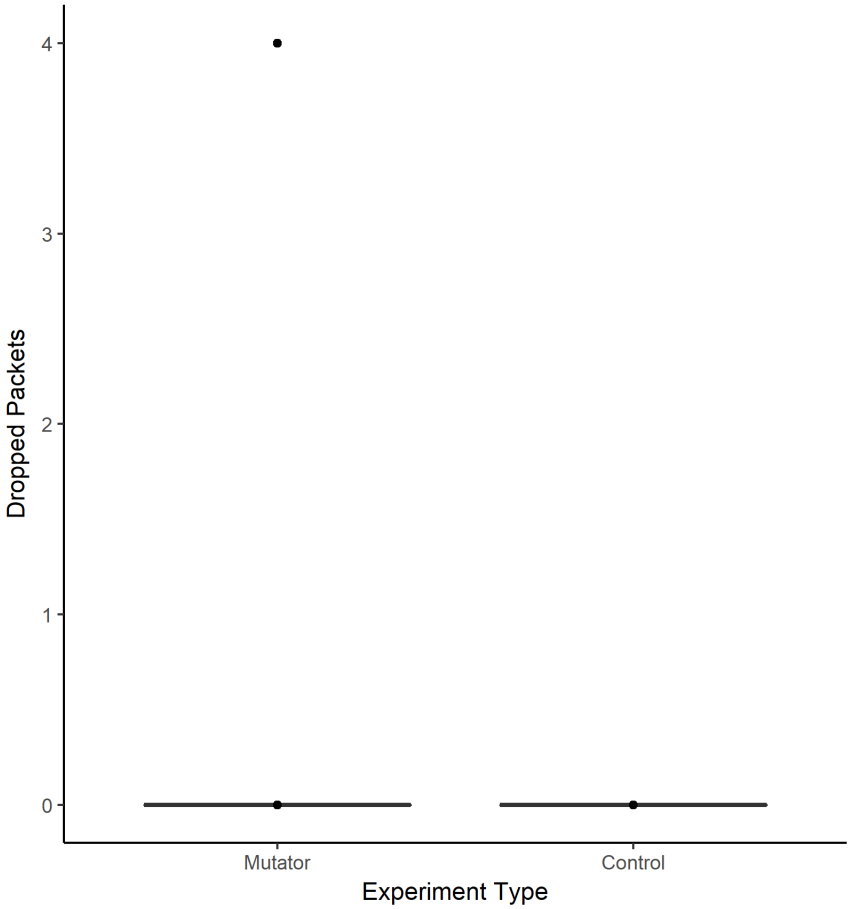



**Figure 39. Box & Whisker Plot of Dropped Packets Between Control and Mutator Trials for SMTP**

### 5.3.7 SSH.

SSH is a protocol for secure remote login over an insecure network. Designed as a replacement for telnet and other insecure remote shell protocols, SSH is used to provide confidentiality and integrity of data. Transport later aspects of SSH are defined in RFC 4253 [58]. Four metrics were assessed: latency, RTT, throughput, and dropped packets. Of these four metrics, two indicated a statistically-significant difference between the control and mutator trials. The following sections provide box & whisker plots for tested metrics and an overview of t-test results. Unabridged output from t-tests and the R script used to generate them are available in Appendix B.7 and Appendix D, respectively.

#### 5.3.7.1 Latency.

Figure 40 depicts the latency perceived by the server that hosts the SSH service used by a client. The y-axis represents time in seconds. The x-axis separates trials with RHM or control trials without RHM. Average mutation latency (0.003 seconds) is equal to the average latency in control experiments (0.003 seconds). There was not a statistically-significant difference in the latency for SSH communications under a network using RHM; t(58)=0.27478, p= 0.7845. The data does not suggest an adverse QoS impact to latency.

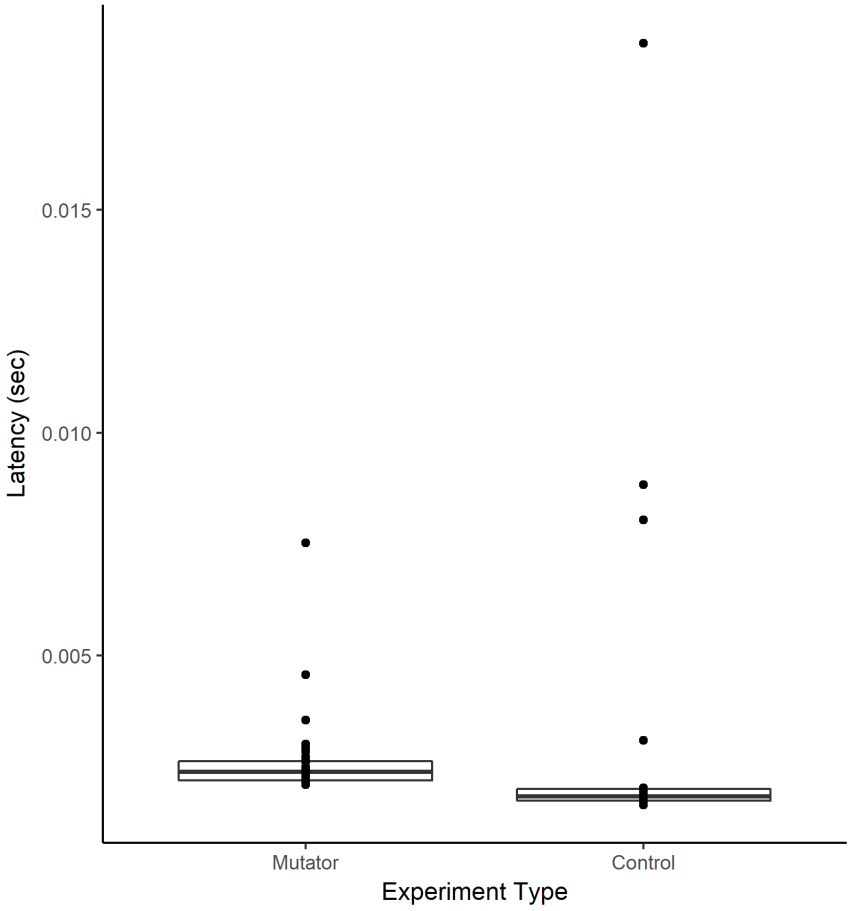



**Figure 40. Box & Whisker Plot of Latency Between Control and Mutator Trials for SSH**

#### 5.3.7.2 RTT.

Figure 41 portrays the RTT for data sent by the client and subsequent server responses from the SSH server. The y-axis represents time in seconds. The x-axis separates trials

with RHM or control trials without RHM. Average mutation RTT (0.0011 seconds) is greater than average RTT in control experiments (0.0008 seconds). There was a statistically-significant difference in the RTT for SSH communications under a network using RHM; t(58)=6.9831, p= 3.129e-09. Despite this difference, an increase in RTT of 0.3 milliseconds would not have a negative impact on QoS.

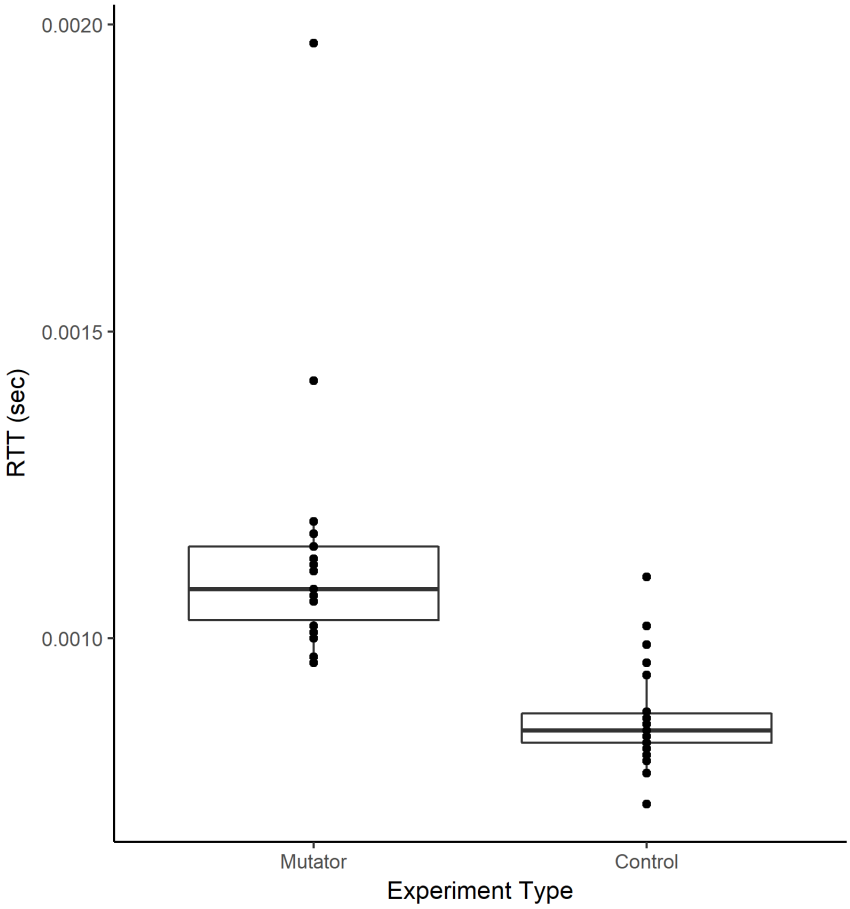



**Figure 41. Box & Whisker Plot of RTT Between Control and Mutator Trials for SSH**

#### 5.3.7.3 Throughput.

Figure 42 details the throughput for data received by the server in terms of bps. The y-axis represents bps. The x-axis separates trials with RHM or control trials without RHM. Average mutation throughput (3203.533 bps) is less than average throughput in control experiments (3339.267 bps). There was a statistically-significant difference in the through-

put for SSH communications under a network using RHM; t(58)=8.6838, p= 4.424e-12. However, a difference of 133.734 bps does not indicate a large enough change in network traffic to concern network infrastructure with a switch fabric capacity of 176 Gigabits per second (Gbps).

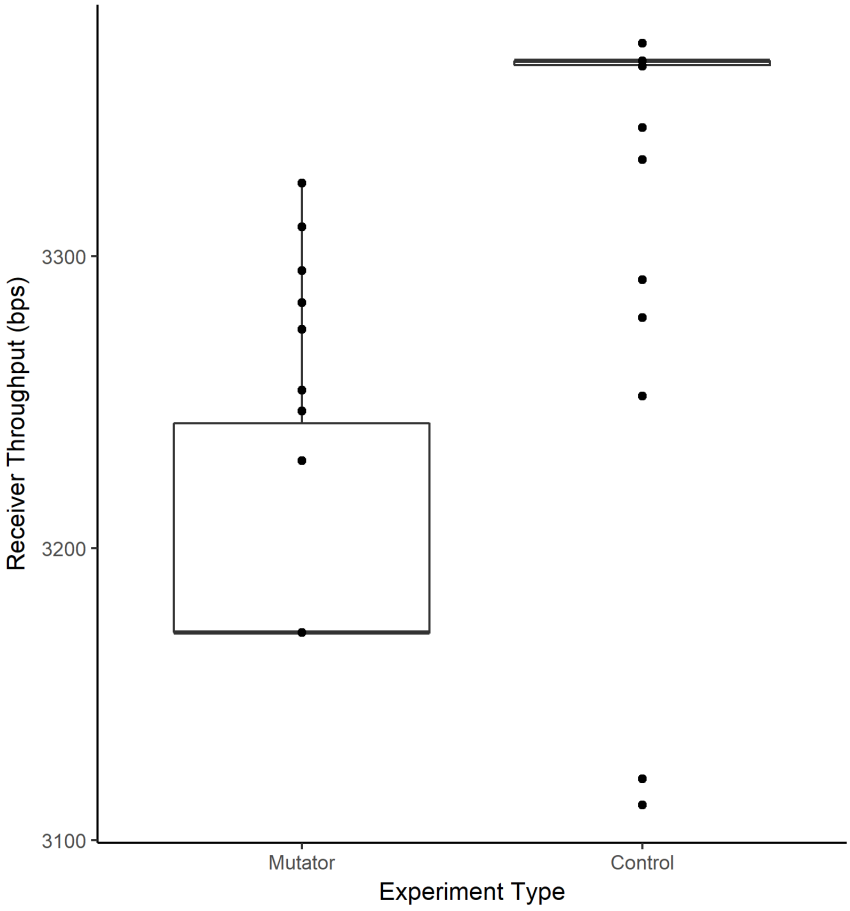



**Figure 42. Box & Whisker Plot of Throughput Between Control and Mutator Trials for SSH**

#### 5.3.7.4 Dropped Packets.

Figure 43 illustrates the average amount of dropped packets for control and mutation trials. The y-axis represents the number of dropped packets, which is an integer value since packets cannot be partially dropped. The x-axis separates trials with RHM or control trials without RHM. The average mutation drop rate of 0.033 packets is the calculated average from all trials and is less than the average drop rate in control experiments (1.000 packet).

As with POP, some of the messages sent were larger than the standard Ethernet frame size of 1518 Bytes. Specifically, the server key exchange initialization was 1714 Bytes. From the server perspective, this packet was not broken up into multiple packets but on the client packet captures, it appeared as two packets. This means that no packets were dropped by the control trials and the calculated average is misleading. The SSH connection did not omit any data as TCP has built-in safeguards to handle dropped packets. Therefore, despite a calculated statistically-significant difference in the amount of dropped packets of $t(58)=12.794$, $p < 2.2e\text{-}16$, the true difference in means is closer to 0.033 packets for mutator and 0 packets for the control. As a result, no adverse QoS impact should be expected when using an established SSH connection.

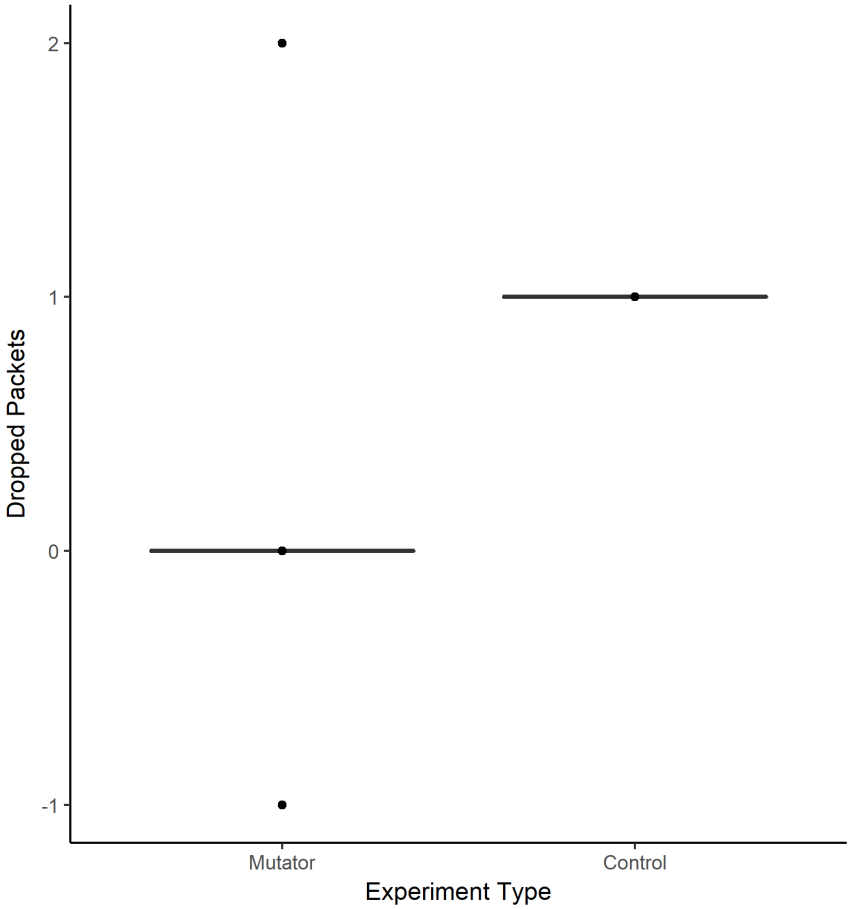



**Figure 43. Box & Whisker Plot of Dropped Packets Between Control and Mutator Trials for SSH**

# VI. Conclusions and Recommendations

## 6.1 Overview

This chapter provides a summary of the research conducted. Section 6.2 states the conclusions of the research. Section 6.3 explains how this research contributes to the field of study related to Software-Defined Networking (SDN). Section 6.4 presents new research paths for future exploration.

## 6.2 Research Conclusions

This research met the three goals of stability, efficacy, and Quality of Service (QoS) assessment identified in the design phase. Experimental data demonstrated the stability and efficacy of a Random Host Mutation (RHM)-enabled network and provided results on the QoS impact of this technique. The stable design of RHM ensured that all designated targets were reachable from inside their own subnets, could send successful Internet Control Message Protocol (ICMP) pings and Domain Name System (DNS) requests, and were exploitable by the adversary. RHM also reduced the ability of an adversary to discover hosts and ensured they were only accessible through their Virtual IP address (vIP). This line of research confirms the defensive efficacy of RHMs against scans as first determined by Aust.

QoS analysis reveals the impact that RHMs have on seven application layer protocols that use Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) at the transport layer. The experimental design successfully enabled the isolated assessment of various protocols without interference from other confounding factors. All network assets produced detailed logs of network traffic with Wireshark. This resulted in a controlled, static environment to produce verifiable conclusions about QoS. Three protocols exhibit a decrease in QoS that may not be acceptable, based upon network requirements. The other four do not show a difference in QoS large enough to create concern, if at all. Based upon the design goals of stability, efficacy, and QoS assessment, the experiments have met all three goals.

However, RHMs do not impede an adversary's ability to maintain persistence on compromised hosts. Aust's initial research into Proactive Host Mutation (PHM) produced code that does not maintain active connections after a mutation occurs. As a result, active connections were terminated after each mutation. This rendered the network unusable if a connection lasted longer than one mutation interval. This design flaw required a solution for SDN-enabled MTD to have a chance at use in real-world networks. The RHM technique described in this thesis created a version of Aust's work that overcame this critical issue and ensured greater usability. In combination with IDS-integration, which is presented as future work in Section 6.4, the disruption to adversary scanning attempts make the adversary less stealthy and make targets harder to reliably exploit. The compressed decision making cycle produced by frequent mutation intervals amplifies this effect, especially at shorter mutation intervals.

### 6.2.1 Stability.

Design of the testbed network showed all target hosts and network services remain accessible from inside the network and maintain connectivity as indicated by ICMP ping tests. DNS also resolved the Fully Qualified Domain Names (FQDNs) for network services. The adversary always exploited target machines given that it discovered the current vIP associated with the target and launched an attack before a mutation occurred. The occurrence of a mutation before completion of an attack accounts for the instances where the adversary failed to exploit a target. This is the expected behavior from a RHM-enabled network and not a stability issue.

### 6.2.2 Effectiveness.

A validation study of Aust's trials with 30 hosts confirms the efficacy of RHMs as a defensive technique against scans. While there is a discrepancy in the time required to conduct network scans, scans from validation trials report a smaller number of *perceived hosts* when compared to the number of *total hosts*. As with the PHM mutator created by Aust, the RHM mutator does not allow traffic to reach the Real IP address (rIP) of a target;

it only accepts traffic directed to the vIP associated with a host for the current mutation interval.

A key distinction between PHM implemented by Aust and RHM is the ability of RHM networks to allow connections that persist beyond one mutation. Initial experiments reveal that the original mutator code deletes all rIP:vIP mappings with each mutation and provides no means to maintain a connection that spanned multiple mutations. Without this ability, PHM does not have value as a MTD in realistic network scenarios. A key contribution of this thesis was the modification of mutator code to enable connections across multiple mutations. However, this ability does not distinguish between legitimate and malicious traffic which does not impede the adversary's ability to maintain persistence on a compromised host. Given the need to balance security and usability, this tradeoff is acceptable.

Based upon previous results and limitations in design of the RHM framework, one concludes that RHMs provide a means of defense against the reconnaissance and scanning phases of adversary action. RHMs do not reduce the ability of an adversary to maintain persistence through a host, as that requires the ability to reliably distinguish legitimate traffic from malicious traffic. This may be possible for well-known exploits, but becomes impractical given the wide array of malicious traffic which may be obfuscated by the adversary.

### 6.2.3 Quality of Service.

There is a statistically-significant difference in either latency or jitter for three of the seven protocols under test. After removal of outliers, no protocols indicate a difference in Round Trip Time (RTT). One protocol, Hypertext Transfer Protocol (HTTP), shows a difference in throughput and packet drop rate. The quantification of these metrics informs network engineers about the performance cost of RHMs. In some cases, the differences between RHM and control performance, while statistically significant, are minimal and may be an acceptable overhead cost for some applications. This information provides a foundation for further research into QoS with RHM and allows network engineers to make decisions based upon proven data to balance security and usability.

## 6.3 Significance of Research

### 6.3.1 Contributions.

This research focuses on validation of the studies conducted by Aust and the QoS implications of RHM on a network. The concept of RHM stems from researchers at University of North Carolina (UNC) where preliminary efforts were done in mininet, a network emulator [31]. Mininet serves as an excellent prototype platform, but does not indicate performance when applied to actual hardware and software. Based upon Aust's efforts, the research herein confirms the ability of RHMs to impede the scanning activity of an adversary. While Aust's research notes the effect that PHM has on attacks launched by adversaries, it does not describe how legitimate traffic may be affected. PHM also presents serious problems toward network usability by legitimate users. RHMs do not provide additional defense against exploits launched by an adversary, but they do create a limited window of opportunity for adversaries to launch attacks. Analysis of RHM also informs potential users how their networks may be affected by the examination of seven protocols in common use. File Transfer Protocol (FTP), HTTP, and Real-time Transport Protocol (RTP) show decreases in performance, whereas Internet Message Access Protocol (IMAP), Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), and Secure Shell (SSH) do not.

### 6.3.2 Applications.

The value of a MTD provides the primary impetus for RHMs as a defensive countermeasure. Frustration of an attacker and the creation of incomplete intelligence provides an edge skewed toward defensive efforts. Adversaries are left with two choices to launch successful attacks: increase the scan rate at the risk of detection by an Intrusion Detection System (IDS) and less precise intelligence, or stop scans entirely. The former presents a noisier adversary that an IDS could detect whereas the latter leaves an adversary with incomplete or incorrect data. The RHM framework also leverages the flexibility of SDN to enable modification for whatever specific implementation details may best suit a potential user (e.g., Supervisory Control and Data Acquisition (SCADA) applications, traditional

IT).

## 6.4 Future Work

The work by previous researchers and this thesis present several opportunities for future work in the MTD research area.

1. *Media Access Control (MAC) Address and Port mutation*: As noted in Section 1.5, a determined adversary could identify targets on the network via other defining characteristics. SDN allows modification of MAC address and port fields in packets. A similar approach as described in this thesis could add another layer of difficulty to adversary actions for a network that uses RHM.

2. *Multiple controller configurations*: The use of a single controller in the test network presents a centralized point of failure. A real-world implementation of SDN would benefit from multiple controller for load balancing and failure redundancy. In the case of time-sensitive mutations, synchronization of the mutation table must be implemented.

3. *Optimal mutation rate and address range determination*: As first indicated by Jafarian in 2012, allocation of virtual IP addresses is an instance of the knapsack problem [31]. While experiments in this thesis were on a small enough scale that naìve allocation techniques suffice, other RHM implementations would benefit from address allocation schemes custom-tailored to the specifics of the networks in which they are used. For example, a shifting mutation rate instead of the constant rates in experiments could further confound adversaries as they launch scans and attacks on the network. Address ranges also require one vIP for each rIP. In a standard /24 subnet, this means that up to half the address space is wasted on IP addresses that do not resolve to actual hosts. A comparative analysis of ways to determine the frequency at which mutations should occur, as well as ways to allocate vIPs , would prove useful to support real-world use of RHM as a MTD.

4. *IDS integration*: The ability of RHM to prompt noisier adversary actions on a network is mentioned several times. To capitalize on this change in adversary behavior, integration of statistics gathered by the SDN infrastructure with an IDS allows for better active defense. The specifics of how an SDN controller or switch could integrate with an IDS depends upon further research.

5. *Honeypot integration*: Mutations provide a layer of obfuscation to the true characteristics of a network. Honeypots create false targets to tempt exploitation. A combination of these two concepts may provide a way to further confuse and impede adversary actions. One potential application of honeypots with RHM could be a "follower" strategy. In this instance, honeypots would be assigned vIPs that belonged to legitimate hosts in the previous mutation. This can result in an adversary that launches an attack on a honeypot running the same set of services as legitimate users due to stale network intelligence.

6. *Graceful flow management*: Flows stored on the SDN switches are created by network traffic previously encountered by the controller. In this implementation of RHM, they are removed from the SDN devices after a timeout period that starts once no received traffic matches the flow. A more precise version of RHM examines characteristics such as header flags in the transport layer and then removes flows once a terminated connection is detected. Some SDN hardware may require specific versions of the Operating System (OS) in order to detect information with this degree of detail. This form of flow management can suffer from spoofed traffic, unless precautions are taken by the controller to be aware of such attempts.

7. *DNS and Address Resolution Protocol (ARP) updates*: As discussed in Section 5.3.1, some protocols (e.g., FTP) do not function when IP address resolution is in use. If the IP address exists in the payload of the packet, then RHM presents usability challenges. Use of FQDNs also presents an issue due to DNS caches on hosts in the network. One possible solution to this problem is the creation of DNS updates from the controller that update the mapping between FQDN and the IP address of

critical network services. As the controller calculates new rIP:vIP mappings, it would also update a separate list of IP addresses and DNS records. The mappings in this additional table then propagate to the hosts and keep their DNS cache in sync with the current network state. For applications reliant upon MAC addresses, a similar technique can update the ARP caches of hosts in the network. As with graceful flow management, this method of updating DNS and ARP caches may be vulnerable to spoofing efforts by an adversary.

## 6.5 Chapter Summary

RHM as a MTD is a cutting-edge technique that disrupts scanning activity of adversaries before they can launch attacks on network assets. If an adversary has a constrained window of opportunity to launch a successful attack, then there is a higher likelihood that the attacker commits an error in one of the steps of the cyberattack methodology (Figure 4). The conclusions made in this chapter stem from extensive tests conducted in a network that mimics what could be expected in a small enterprise network. This chapter also examines applications for RHMs and directions for future work.

# Appendix A. Validation Study Results

## A.1 Original Results

This section contains the data from all of Aust's experiments [24]. For the sake of brevity, all column headers in this section are described here. Trial indicates the mutation interval in use for the corresponding data in Appendix A.1.1 and the index of the associated data for all other tables. Mutator indicates if the data was gathered with PHMs active on the network. A 1 indicates that PHM was active, a 0 indicates that it was not. Scan - I refers to the number of seconds required to complete an intense scan. Scan - Q refers to the number of seconds required to complete a quick scan. Hosts - I refers to the number of *perceived hosts* based upon intense scan results. Hosts - Q refers to the number of *perceived hosts* based upon quick scan results. Pen Time - I displays the amount of time that an adversary maintained access on a target host following the results of an intense scan. Pen Time - Q indicates the amount of time that an adversary maintained access on a target host following the results of a quick scan. Cells that contain a "-" indicate that no data was collected for reasons explained in the section that contains the graph.

### A.1.1 Averages.

**Table 8. Averages from Aust's Experiments**

| Trial | Mutator | Scan - I | Scan - Q | Hosts - I | Hosts - Q | Pen Time - I | Pen Time - Q |
|---|---|---|---|---|---|---|---|
| 30S | 0 | 84.092 | 5.44 | 30 | 30 | 3600 | 3600 |
| 30S | 1 | 0 | 9.706 | 0 | 21 | 0 | 139.56 |
| 1M | 0 | 84.092 | 5.44 | 30 | 30 | 3600 | 3600 |
| 1M | 1 | 224.826 | 10.806 | 16.6 | 13.2 | 0 | 188.04 |
| 5M | 0 | 84.092 | 5.44 | 30 | 30 | 3600 | 3600 |
| 5M | 1 | 99.7 | 11.44 | 15.4 | 17.6 | 333 | 417.72 |
| 15M | 0 | 84.092 | 5.44 | 30 | 30 | 3600 | 3600 |
| 15M | 1 | 138.398 | 9.81 | 20.8 | 15.6 | 923.28 | 876 |

### A.1.2 Control.

**Table 9. Data from Aust's Control Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q | Pen Time - I | Pen Time - Q |
|---|---|---|---|---|---|---|
| 1 | 93.840 | 4.680 | 30 | 30 | 3600.000 | 3600.000 |
| 2 | 85.600 | 5.830 | 30 | 30 | 3600.000 | 3600.000 |
| 3 | 89.880 | 5.860 | 30 | 30 | 3600.000 | 3600.000 |
| 4 | 81.380 | 4.980 | 30 | 30 | 3600.000 | 3600.000 |
| 5 | 69.760 | 5.850 | 30 | 30 | 3600.000 | 3600.000 |

### A.1.3 30 Second.

Cells that contain a "-" indicate that no data was collected due to failure of either the scan or exploit.

**Table 10. Data from Aust's 30 Second Interval Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q | Pen Time - I | Pen Time - Q |
|---|---|---|---|---|---|---|
| 1 | - | 9.74 | - | 20 | - | 3.16 |
| 2 | - | 15.04 | - | 23 | - | - |
| 3 | - | 6.46 | - | 21 | - | 3.15 |
| 4 | - | 8.8 | - | 15 | - | 2.15 |
| 5 | - | 8.49 | - | 26 | - | 3.17 |

### A.1.4 1 Minute.

Cells that contain a "-" indicate that no data was collected due to failure of the exploit.

**Table 11. Data from Aust's 1 Minute Interval Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q | Pen Time - I | Pen Time - Q |
|---|---|---|---|---|---|---|
| 1 | 449.31 | 8.49 | 30 | 28 | - | 3.06 |
| 2 | 68.76 | 10.61 | 10 | 17 | - | 3.15 |
| 3 | 160.78 | 11.83 | 13 | 6 | - | 3.17 |
| 4 | 155.62 | 11.56 | 11 | 6 | - | 3.15 |
| 5 | 289.66 | 11.54 | 19 | 9 | - | 3.14 |

### A.1.5 5 Minutes.

**Table 12. Data from Aust's 5 Minute Interval Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q | Pen Time - I | Pen Time - Q |
|---|---|---|---|---|---|---|
| 1 | 154.050 | 8.670 | 23 | 27 | 5.150 | 6.160 |
| 2 | 89.600 | 11.250 | 15 | 13 | 6.160 | 7.170 |
| 3 | 89.150 | 8.190 | 15 | 28 | 6.140 | 7.160 |
| 4 | 74.900 | 19.110 | 10 | 11 | 5.160 | 7.150 |
| 5 | 90.800 | 9.980 | 14 | 9 | 5.140 | 7.170 |

### A.1.6 15 Minutes.

**Table 13. Data from Aust's 15 Minute Interval Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q | Pen Time - I | Pen Time - Q |
|---|---|---|---|---|---|---|
| 1 | 111.770 | 11.600 | 15 | 9 | 16.290 | 6.160 |
| 2 | 155.550 | 9.700 | 28 | 21 | 15.180 | 16.290 |
| 3 | 78.130 | 8.210 | 9 | 24 | 16.170 | 16.160 |
| 4 | 192.470 | 9.380 | 26 | 16 | 14.140 | 17.170 |
| 5 | 154.070 | 10.160 | 26 | 8 | 15.160 | 17.220 |

## A.2 Validation Results

This section contains the data from all of the validation experiments. For the sake of brevity, all column headers in this section are described here. Trial indicates the mutation interval in use for the corresponding data in Appendix A.2.1 and the index of the associated data for all other tables. Mutator indicates if the data was gathered with RHMs active on the network. A 1 indicates that RHM was active, a 0 indicates that it was not. Scan - I refers to the number of seconds required to complete an intense scan. Scan - Q refers to the number of seconds required to complete a quick scan. Hosts - I refers to the number of *perceived hosts* based upon intense scan results. Hosts - Q refers to the number of *perceived hosts* based upon quick scan results. As discussed in Section 2.6, a primary limitation of PHM was that it did not allow connections to persist beyond one mutation. RHM added this capability, which still provides a defensive benefit against scans but does not terminate connections after each mutation. This tradeoff means that the "Pen Time" data collected

by Aust has no counterpart in validation trials since adversary connections were never terminated.

### A.2.1 Averages.

**Table 14. Averages from Validation Experiments**

| Trial | Mutator | Scan - I | Scan - Q | Hosts - I | Hosts - Q |
|---|---|---|---|---|---|
| 30S | 0 | 901.2 | 71.1 | 30 | 30 |
| 30S | 1 | 0 | 39.5 | 0 | 18.5 |
| 1M | 0 | 901.2 | 71.1 | 30 | 30 |
| 1M | 1 | 0 | 44 | 0 | 20.4 |
| 5M | 0 | 901.2 | 71.1 | 30 | 30 |
| 5M | 1 | 0 | 50.9 | 0 | 22 |
| 15M | 0 | 901.2 | 71.1 | 30 | 30 |
| 15M | 1 | 724.3 | 56.6 | 22 | 22 |

### A.2.2 Control.

**Table 15. Data from Control Validation Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q |
|---|---|---|---|---|
| 1 | 868 | 71 | 30 | 30 |
| 2 | 861 | 73 | 30 | 30 |
| 3 | 933 | 72 | 30 | 30 |
| 4 | 858 | 69 | 30 | 30 |
| 5 | 876 | 70 | 30 | 30 |
| 6 | 954 | 73 | 30 | 30 |
| 7 | 954 | 73 | 30 | 30 |
| 8 | 933 | 72 | 30 | 30 |
| 9 | 902 | 70 | 30 | 30 |
| 10 | 873 | 68 | 30 | 30 |

### A.2.3 30 Second.

**Table 16. Data from 30 Second Interval Validation Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q |
|---|---|---|---|---|
| 1 | 120 | 120 | 0 | 0 |
| 2 | 120 | 120 | 0 | 0 |
| 3 | 120 | 31 | 0 | 19 |
| 4 | 120 | 120 | 0 | 0 |
| 5 | 120 | 120 | 0 | 0 |
| 6 | 120 | 120 | 0 | 0 |
| 7 | 120 | 120 | 0 | 0 |
| 8 | 120 | 120 | 0 | 0 |
| 9 | 120 | 120 | 0 | 0 |
| 10 | 120 | 48 | 0 | 18 |

### A.2.4 1 Minute.

**Table 17. Data from 1 Minute Interval Validation Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q |
|---|---|---|---|---|
| 1 | 240 | 55 | 0 | 21 |
| 2 | 240 | 34 | 0 | 17 |
| 3 | 240 | 64 | 0 | 27 |
| 4 | 240 | 56 | 0 | 24 |
| 5 | 240 | 41 | 0 | 18 |
| 6 | 240 | 33 | 0 | 22 |
| 7 | 240 | 28 | 0 | 16 |
| 8 | 240 | 46 | 0 | 21 |
| 9 | 240 | 43 | 0 | 21 |
| 10 | 240 | 40 | 0 | 17 |

### A.2.5 5 Minutes.

**Table 18. Data from 5 Minute Interval Validation Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q |
|---|---|---|---|---|
| 1 | 1200 | 51 | 0 | 21 |
| 2 | 1200 | 50 | 0 | 23 |
| 3 | 1200 | 50 | 0 | 23 |
| 4 | 1200 | 47 | 0 | 23 |
| 5 | 1200 | 35 | 0 | 19 |
| 6 | 1200 | 37 | 0 | 17 |
| 7 | 1200 | 75 | 0 | 27 |
| 8 | 1200 | 53 | 0 | 23 |
| 9 | 1200 | 42 | 0 | 18 |
| 10 | 1200 | 69 | 0 | 26 |

### A.2.6 15 Minutes.

**Table 19. Data from 15 Minute Interval Validation Experiments**

| Trial | Scan - I | Scan - Q | Hosts - I | Hosts - Q |
|---|---|---|---|---|
| 1 | 781 | 45 | 21 | 19 |
| 2 | 771 | 51 | 24 | 21 |
| 3 | 751 | 49 | 21 | 20 |
| 4 | 915 | 53 | 22 | 22 |
| 5 | 796 | 73 | 23 | 27 |
| 6 | 892 | 95 | 25 | 23 |
| 7 | 662 | 44 | 23 | 25 |
| 8 | 308 | 53 | 17 | 23 |
| 9 | 883 | 63 | 26 | 23 |
| 10 | 484 | 40 | 18 | 17 |

# Appendix B. QoS Study Results

## B.1 FTP

Due to the inability of FTP to establish a connection based upon IP address, no meaningful test data was gathered. The entire FTP exchange conducted in mutator trials is included in this section. The "No." column indicates the frame number. "Source" and "Destination" show the source IP address of the client and server. "Protocol" indicates the protocol associated with a given frame. "Length" states the length of a frame in bytes. "Info" provides an overview of the data transmitted in a frame.

**Figure 44. Full FTP Stream**

| No. | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 9 | 10.13.1.8 | 10.13.1.44 | TCP | 74 | 41248 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=373596937 TSecr=0 WS=128 |
| 10 | 10.13.1.44 | 10.13.1.8 | TCP | 74 | 21 → 41248 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=614783254 TSecr=373596937 |
| 11 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=373596942 TSecr=614783254 |
| 12 | 10.13.1.44 | 10.13.1.8 | FTP | 93 | Response: 220 Microsoft FTP Service |
| 13 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=1 Ack=28 Win=29312 Len=0 TSval=373596943 TSecr=614783256 |
| 14 | 10.13.1.8 | 10.13.1.44 | FTP | 72 | Request: FEAT |
| 15 | 10.13.1.44 | 10.13.1.8 | FTP | 100 | Response: 211-Extended features supported: |
| 16 | 10.13.1.44 | 10.13.1.8 | FTP | 84 | Response:  LANG EN* |
| 17 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=7 Ack=80 Win=29312 Len=0 TSval=373596943 TSecr=614783256 |
| 18 | 10.13.1.44 | 10.13.1.8 | FTP | 119 | Response:  AUTH TLS;TLS-C;SSL;TLS-P; |
| 19 | 10.13.1.44 | 10.13.1.8 | FTP | 73 | Response:  HOST |
| 20 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=7 Ack=140 Win=29312 Len=0 TSval=373596943 TSecr=614783256 |
| 21 | 10.13.1.44 | 10.13.1.8 | FTP | 103 | Response:  SIZE |
| 22 | 10.13.1.8 | 10.13.1.44 | FTP | 72 | Request: LANG |
| 23 | 10.13.1.44 | 10.13.1.8 | FTP | 112 | Response: 200 Language is now English, UTF-8 encoding. |
| 24 | 10.13.1.8 | 10.13.1.44 | FTP | 80 | Request: OPTS UTF8 ON |
| 25 | 10.13.1.44 | 10.13.1.8 | FTP | 124 | Response: 200 OPTS UTF8 command successful - UTF8 encoding now ON. |
| 26 | 10.13.1.8 | 10.13.1.44 | FTP | 83 | Request: HOST 10.13.1.44 |
| 27 | 10.13.1.44 | 10.13.1.8 | FTP | 102 | Response: 504 Server cannot accept argument. |
| 28 | 10.13.1.8 | 10.13.1.44 | FTP | 82 | Request: USER anonymous |
| 29 | 10.13.1.44 | 10.13.1.8 | FTP | 138 | Response: 331 Anonymous access allowed, send identity (e-mail name) as password. |
| 30 | 10.13.1.8 | 10.13.1.44 | FTP | 78 | Request: PASS lftp@ |
| 31 | 10.13.1.44 | 10.13.1.8 | FTP | 87 | Response: 230 User logged in. |
| 32 | 10.13.1.8 | 10.13.1.44 | FTP | 71 | Request: PWD |
| 33 | 10.13.1.44 | 10.13.1.8 | FTP | 97 | Response: 257 "/" is current directory. |
| 34 | 10.13.1.8 | 10.13.1.44 | FTP | 74 | Request: TYPE I |
| 35 | 10.13.1.44 | 10.13.1.8 | FTP | 86 | Response: 200 Type set to I. |
| 36 | 10.13.1.8 | 10.13.1.44 | FTP | 87 | Request: SIZE Data50_FTP.txt |
| 37 | 10.13.1.44 | 10.13.1.8 | FTP | 80 | Response: 213 50000000 |
| 38 | 10.13.1.8 | 10.13.1.44 | FTP | 87 | Request: MDTM Data50_FTP.txt |
| 39 | 10.13.1.44 | 10.13.1.8 | FTP | 86 | Response: 213 20171020133747 |
| 40 | 10.13.1.8 | 10.13.1.44 | FTP | 72 | Request: PASV |
| 41 | 10.13.1.44 | 10.13.1.8 | FTP | 113 | Response: 227 Entering Passive Mode (10,13,1,4,19,253). |
| 42 | Vmware_9d:… | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 43 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=133 Ack=542 Win=29312 Len=0 TSval=373596958 TSecr=614783258 |
| 44 | Vmware_9d:… | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 45 | Vmware_9d:… | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 46 | Vmware_9d:… | Broadcast | ARP | 42 | Who has 10.13.1.4? Tell 10.13.1.8 |
| 47 | 10.13.1.8 | 10.13.1.44 | FTP | 69 | Request: \377\364\377 |
| 48 | 10.13.1.8 | 10.13.1.44 | FTP | 67 | Request: \362 |
| 49 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [FIN, ACK] Seq=137 Ack=542 Win=29312 Len=0 TSval=373597698 TSecr=614783258 |
| 50 | 10.13.1.44 | 10.13.1.8 | TCP | 66 | 21 → 41248 [ACK] Seq=542 Ack=137 Win=66304 Len=0 TSval=614783558 TSecr=373597698 |
| 51 | 10.13.1.44 | 10.13.1.8 | TCP | 66 | 21 → 41248 [ACK] Seq=542 Ack=138 Win=66304 Len=0 TSval=614783558 TSecr=373597698 |
| 52 | 10.13.1.44 | 10.13.1.8 | TCP | 66 | 21 → 41248 [FIN, ACK] Seq=542 Ack=138 Win=66304 Len=0 TSval=614783558 TSecr=373597698 |
| 53 | 10.13.1.8 | 10.13.1.44 | TCP | 66 | 41248 → 21 [ACK] Seq=138 Ack=543 Win=29312 Len=0 TSval=373597699 TSecr=614783558 |
| 54 | 10.13.1.44 | 10.13.1.8 | TCP | 60 | 21 → 41248 [RST, ACK] Seq=543 Ack=138 Win=0 Len=0 |

## B.2 HTTP

This section contains the data from all QoS experiments for HTTP. For the sake of brevity, all column headers in this section are described here. Trial indicates the index associated with the data in a row. Latency shows the average time in milliseconds that it took information from the client to reach the server for the trial. RTT indicates the amount of time in seconds from when a packet was sent by the server to the client and the receipt of the ACK from the client. Duration shows the length of the connection in seconds. BPSS represents the throughput of the client in Bits per second (bps). BPSR indicates the throughput of the server in bps. PacketsS displays the number of packets sent by the client during the transmission. PacketsR reports the number of packets sent by the server during the transmission.

ACKlostR displays the number of packets lost by the server during a trial. DupACKR shows the count of duplicate ACKS received during a trial. RetransR, FRetransR, and SRetransR indicate the amount of retransmissions, fast retransmissions, and spurious retransmissions by the server in a trial, respectively. OutOfOrderR indicates the number of packets received out of order in a single trial. WinUpdateR reports the number of window updates sent in a transmission. WinFullR shows how many times a notification that the TCP receive window was at capacity were send in a single trial.

### B.2.1 Control.

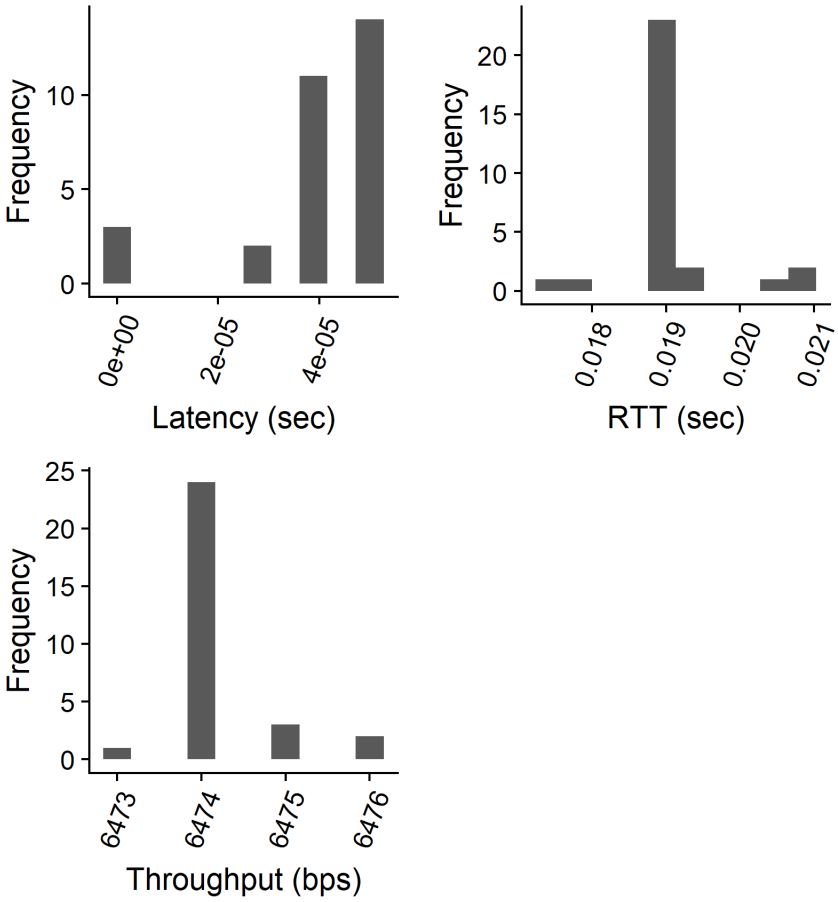



**Figure 45. Histograms of HTTP QoS Control Data**

**Table 20. HTTP Control QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR |
|---|---|---|---|---|---|---|---|
| 1 | 5.00E-05 | 0.0189 | 62.198 | 6750 | 6474 | 37 | 5054 |
| 2 | 4.00E-05 | 0.01902 | 62.188 | 6751 | 6474 | 37 | 5028 |
| 3 | 5.00E-05 | 0.02069 | 62.187 | 6751 | 6475 | 37 | 5052 |
| 4 | 4.00E-05 | 0.01912 | 62.177 | 6752 | 6475 | 37 | 5012 |
| 5 | 4.00E-05 | 0.01907 | 62.175 | 6753 | 6476 | 37 | 5042 |
| 6 | 4.00E-05 | 0.01765 | 62.178 | 6753 | 6476 | 37 | 5156 |
| 7 | 4.00E-05 | 0.01908 | 62.201 | 6750 | 6473 | 37 | 5025 |
| 8 | 5.00E-05 | 0.01909 | 62.194 | 6750 | 6474 | 37 | 5027 |
| 9 | 5.00E-05 | 0.01895 | 62.178 | 6752 | 6475 | 37 | 5038 |
| 10 | 4.00E-05 | 0.01903 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 11 | 5.00E-05 | 0.01891 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 12 | 4.00E-05 | 0.02046 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 13 | 3.00E-05 | 0.0193 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 14 | 5.00E-05 | 0.01892 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 15 | 5.00E-05 | 0.01889 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 16 | 5.00E-05 | 0.0191 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 17 | 5.00E-05 | 0.0189 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 18 | 5.00E-05 | 0.01878 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 19 | 5.00E-05 | 0.01891 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 20 | 5.00E-05 | 0.01896 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 21 | 4.00E-05 | 0.01907 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 22 | 4.00E-05 | 0.02093 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 23 | 4.00E-05 | 0.01894 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 24 | 3.00E-05 | 0.01752 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 25 | 4.00E-05 | 0.01919 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 26 | 5.00E-05 | 0.01895 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 27 | 4.00E-05 | 0.01892 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 28 | 4.00E-05 | 0.01907 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 29 | 5.00E-05 | 0.01891 | 62.194 | 6751 | 6474 | 37 | 5042 |
| 30 | 5.00E-05 | 0.01897 | 62.194 | 6751 | 6474 | 37 | 5042 |

**Table 21. HTTP Control Lost, Duplicate, Retransmitted, and Out of Order Packets**

| Trial | ACKlostR | DupACKR | RetransR | FRetransR | SRetransR | OutOfOrderR |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 | 0 | 0 | 0 |
| 23 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 22. HTTP Control Window Update Data**

| Trial | WinUpdateR | WinFullR |
|---|---|---|
| 1 | 1108 | 15 |
| 2 | 1126 | 0 |
| 3 | 1108 | 15 |
| 4 | 1119 | 0 |
| 5 | 1111 | 15 |
| 6 | 1118 | 0 |
| 7 | 1128 | 0 |
| 8 | 1103 | 15 |
| 9 | 1101 | 20 |
| 10 | 1109 | 14 |
| 11 | 1116 | 3 |
| 12 | 1086 | 1 |
| 13 | 1102 | 17 |
| 14 | 1112 | 7 |
| 15 | 1109 | 14 |
| 16 | 1109 | 12 |
| 17 | 1111 | 4 |
| 18 | 1085 | 106 |
| 19 | 1116 | 3 |
| 20 | 1108 | 13 |
| 21 | 1124 | 0 |
| 22 | 1112 | 4 |
| 23 | 1109 | 15 |
| 24 | 985 | 4 |
| 25 | 1107 | 15 |
| 26 | 1119 | 2 |
| 27 | 1114 | 4 |
| 28 | 1105 | 14 |
| 29 | 1119 | 0 |
| 30 | 1110 | 15 |

### B.2.2 Mutator.

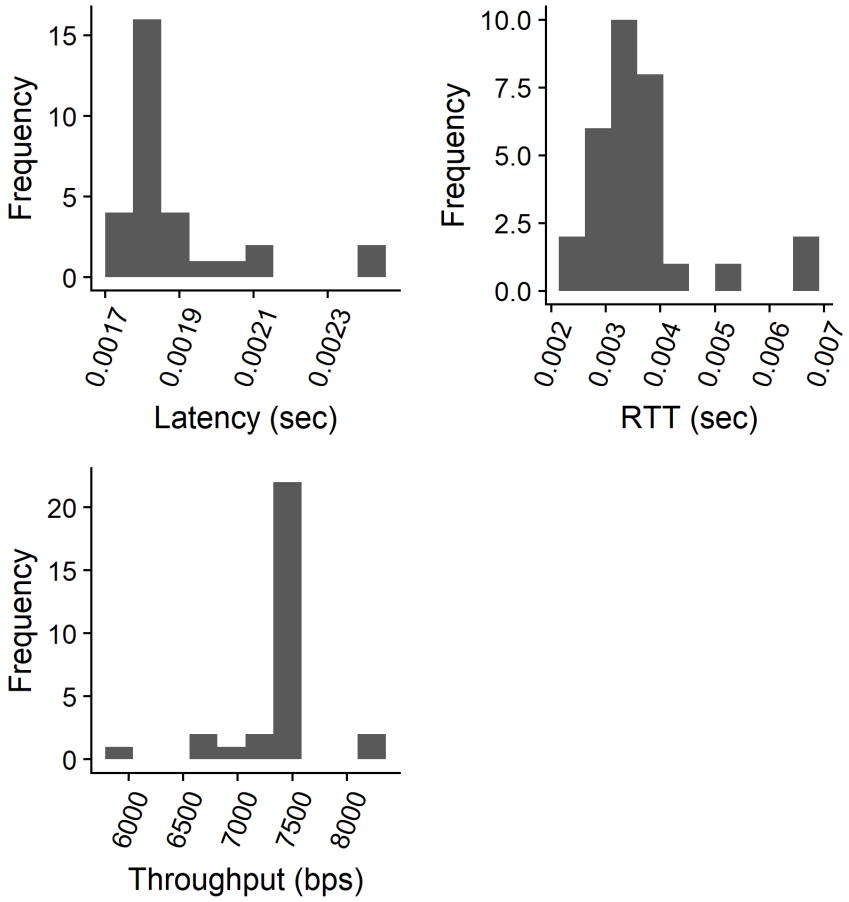



**Figure 46. Histograms of HTTP QoS Mutator Data**

**Table 23. HTTP Mutator QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR |
|---|---|---|---|---|---|---|---|
| 1 | 0.00187 | 0.00315 | 66.409 | 6606 | 7304 | 69 | 71 |
| 2 | 0.00183 | 0.00335 | 64.666 | 6786 | 7515 | 69 | 71 |
| 3 | 0.00239 | 0.00686 | 83.13 | 5282 | 5848 | 69 | 71 |
| 4 | 0.00239 | 0.00686 | 67.23 | 5282 | 7254 | 69 | 72 |
| 5 | 0.00171 | 0.00259 | 59.452 | 7378 | 8162 | 69 | 71 |
| 6 | 0.00198 | 0.00397 | 69.08 | 6353 | 7043 | 69 | 71 |
| 7 | 0.00208 | 0.00511 | 72.571 | 6050 | 6687 | 69 | 71 |
| 8 | 0.00171 | 0.00257 | 59.647 | 7354 | 8133 | 69 | 71 |
| 9 | 0.00206 | 0.00413 | 71.713 | 6121 | 6774 | 69 | 71 |
| 10 | 0.00186 | 0.00353 | 65.034 | 6747 | 7484 | 69 | 71 |
| 11 | 0.00178 | 0.00265 | 65.034 | 6747 | 7484 | 69 | 71 |
| 12 | 0.00178 | 0.00306 | 65.034 | 6747 | 7484 | 69 | 71 |
| 13 | 0.00187 | 0.00373 | 65.034 | 6747 | 7484 | 69 | 71 |
| 14 | 0.0018 | 0.00349 | 65.034 | 6747 | 7484 | 69 | 71 |
| 15 | 0.0018 | 0.00304 | 65.034 | 6747 | 7484 | 69 | 71 |
| 16 | 0.00183 | 0.0037 | 65.034 | 6747 | 7484 | 69 | 71 |
| 17 | 0.0018 | 0.00327 | 65.034 | 6747 | 7484 | 69 | 71 |
| 18 | 0.00179 | 0.00353 | 65.034 | 6747 | 7484 | 69 | 71 |
| 19 | 0.00179 | 0.003 | 65.034 | 6747 | 7484 | 69 | 71 |
| 20 | 0.00179 | 0.00301 | 65.034 | 6747 | 7484 | 69 | 71 |
| 21 | 0.00176 | 0.00291 | 65.034 | 6747 | 7484 | 69 | 71 |
| 22 | 0.00179 | 0.00311 | 65.034 | 6747 | 7484 | 69 | 71 |
| 23 | 0.00213 | 0.00405 | 65.034 | 6747 | 7484 | 69 | 71 |
| 24 | 0.00177 | 0.00401 | 65.034 | 6747 | 7484 | 69 | 71 |
| 25 | 0.00181 | 0.00338 | 65.034 | 6747 | 7484 | 69 | 71 |
| 26 | 0.00178 | 0.00316 | 65.034 | 6747 | 7484 | 69 | 71 |
| 27 | 0.00189 | 0.00385 | 65.034 | 6747 | 7484 | 69 | 71 |
| 28 | 0.0018 | 0.00315 | 65.034 | 6747 | 7484 | 69 | 71 |
| 29 | 0.00181 | 0.00366 | 65.034 | 6747 | 7484 | 69 | 71 |
| 30 | 0.00178 | 0.00382 | 65.034 | 6747 | 7484 | 69 | 71 |

**Table 24. HTTP Mutator Lost, Duplicate, Retransmitted, and Out of Order Packets**

| Trial | ACKlostR | DupACKR | RetransR | FRetransR | SRetransR | OutOfOrderR |
|---|---|---|---|---|---|---|
| 1 | 0 | 22073 | 3568 | 3435 | 0 | 324 |
| 2 | 1 | 22987 | 3613 | 3485 | 0 | 343 |
| 3 | 0 | 26067 | 3491 | 3288 | 0 | 441 |
| 4 | 0 | 25003 | 3734 | 3592 | 0 | 328 |
| 5 | 1 | 21671 | 3619 | 3510 | 0 | 293 |
| 6 | 0 | 23740 | 3624 | 3474 | 0 | 369 |
| 7 | 1 | 25110 | 3446 | 3273 | 0 | 425 |
| 8 | 0 | 21778 | 3590 | 3481 | 0 | 306 |
| 9 | 0 | 23824 | 3539 | 3379 | 0 | 391 |
| 10 | 0 | 23108 | 3678 | 3543 | 0 | 333 |
| 11 | 0 | 20797 | 3597 | 3483 | 0 | 295 |
| 12 | 1 | 22130 | 3766 | 3641 | 0 | 315 |
| 13 | 0 | 23525 | 3599 | 3463 | 0 | 350 |
| 14 | 0 | 23620 | 3669 | 3537 | 0 | 303 |
| 15 | 0 | 22259 | 3663 | 3538 | 0 | 325 |
| 16 | 0 | 23856 | 3984 | 3852 | 0 | 296 |
| 17 | 0 | 23062 | 3691 | 3565 | 0 | 300 |
| 18 | 1 | 23712 | 3820 | 3695 | 0 | 324 |
| 19 | 0 | 22146 | 3723 | 3601 | 0 | 315 |
| 20 | 0 | 22130 | 3739 | 3617 | 0 | 328 |
| 21 | 1 | 21322 | 3731 | 3611 | 0 | 329 |
| 22 | 0 | 21993 | 3931 | 3805 | 0 | 333 |
| 23 | 0 | 22225 | 3848 | 3659 | 0 | 406 |
| 24 | 2 | 24871 | 3912 | 3780 | 0 | 309 |
| 25 | 1 | 23098 | 3694 | 3565 | 0 | 326 |
| 26 | 0 | 22215 | 3775 | 3656 | 0 | 294 |
| 27 | 0 | 23866 | 3689 | 3546 | 0 | 366 |
| 28 | 0 | 22297 | 4023 | 3884 | 0 | 292 |
| 29 | 0 | 24268 | 3798 | 3672 | 0 | 309 |
| 30 | 1 | 24349 | 3829 | 3699 | 0 | 306 |

**Table 25. HTTP Mutator Window Update Data**

| Trial | WinUpdateR | WinFullR |
|---|---|---|
| 1 | 160 | 0 |
| 2 | 115 | 0 |
| 3 | 196 | 0 |
| 4 | 103 | 0 |
| 5 | 119 | 0 |
| 6 | 106 | 0 |
| 7 | 97 | 0 |
| 8 | 119 | 0 |
| 9 | 89 | 0 |
| 10 | 105 | 0 |
| 11 | 202 | 0 |
| 12 | 196 | 0 |
| 13 | 130 | 0 |
| 14 | 125 | 0 |
| 15 | 151 | 0 |
| 16 | 103 | 0 |
| 17 | 123 | 0 |
| 18 | 148 | 0 |
| 19 | 144 | 0 |
| 20 | 165 | 0 |
| 21 | 196 | 0 |
| 22 | 193 | 0 |
| 23 | 234 | 0 |
| 24 | 107 | 0 |
| 25 | 136 | 0 |
| 26 | 170 | 0 |
| 27 | 119 | 0 |
| 28 | 177 | 0 |
| 29 | 100 | 0 |
| 30 | 106 | 0 |

### B.2.3 T-test results.

```
[1] "Latency"

 Two Sample t-test

data:  Control and Mutation
t = -57.229, df = 58, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.001928181 -0.001756697
sample estimates:
   mean of x    mean of y
3.248836e-05 1.874927e-03

[1] "RTT"

 Two Sample t-test

data:  Control and Mutation
t = 69.219, df = 58, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 0.01482287 0.01600918
sample estimates:
  mean of x   mean of y
0.019073117 0.003657091

[1] "Duration"

 Two Sample t-test

data:  Control and Mutation
t = -5.0827, df = 58, p-value = 4.161e-06
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
```

```
 -5.783837 -1.806545
sample estimates:
mean of x mean of y
 62.19161  65.98680

[1] "Throughput"

 Two Sample t-test

data:  Control and Mutation
t = -12.483, df = 58, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -1118.6378  -725.2289
sample estimates:
mean of x mean of y
 6474.200  7396.133

[1] "Dropped Packets"

 Two Sample t-test

data:  Control and Mutation
t = -1215, df = 58, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -5015.804 -4993.863
sample estimates:
   mean of x    mean of y
-5006.866667    -2.033333
```

### B.3 IMAP

This section contains the data from all QoS experiments for IMAP. For the sake of brevity, all column headers in this section are described here. Trial indicates the index associated with the data in a row. Latency shows the average time in milliseconds that

it took information from the client to reach the server for the trial. RTT indicates the amount of time in seconds from when a packet was sent by the server to the client and the receipt of the ACK from the client. Duration shows the length of the connection in seconds. BPSS represents the throughput of the client in bps. BPSR indicates the throughput of the server in bps. PacketsS displays the number of packets sent by the client during the transmission. PacketsR reports the number of packets sent by the server during the transmission. PktsDrop indicates the sum of all dropped packets during one trial.

### B.3.1 Control.

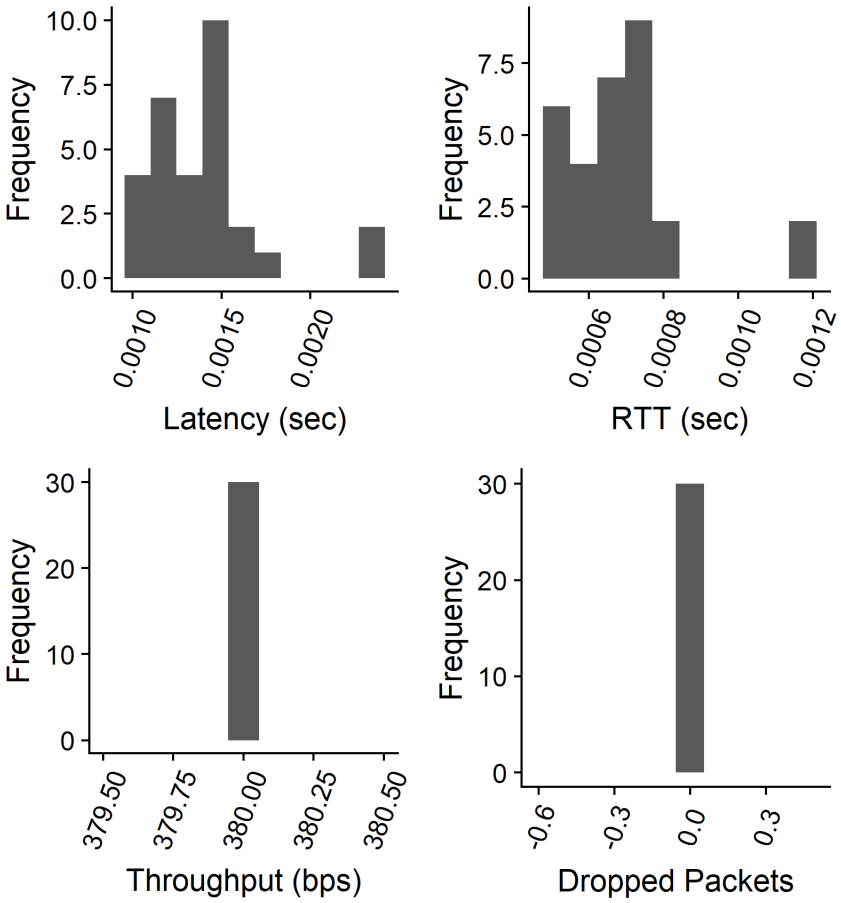



**Figure 47. Histograms of IMAP QoS Control Data**

**Table 26. IMAP Control QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.00168 | 0.00083 | 122.027 | 380 | 380 | 50 | 50 | 0 |
| 2 | 0.00142 | 7.00E-04 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 3 | 0.00141 | 0.00068 | 122.023 | 380 | 380 | 50 | 50 | 0 |
| 4 | 0.00145 | 0.00074 | 122.026 | 380 | 380 | 50 | 50 | 0 |
| 5 | 0.00169 | 8.00E-04 | 122.025 | 380 | 380 | 50 | 50 | 0 |
| 6 | 0.00134 | 0.00064 | 122.025 | 380 | 380 | 50 | 50 | 0 |
| 7 | 0.00114 | 0.00053 | 122.025 | 380 | 380 | 50 | 50 | 0 |
| 8 | 0.0011 | 0.00056 | 122.025 | 380 | 380 | 50 | 50 | 0 |
| 9 | 0.0011 | 0.00055 | 122.026 | 380 | 380 | 50 | 50 | 0 |
| 10 | 0.00115 | 0.00053 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 11 | 0.00148 | 0.00077 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 12 | 0.0013 | 0.00065 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 13 | 0.00238 | 0.00119 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 14 | 0.00149 | 0.00071 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 15 | 0.00154 | 0.00075 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 16 | 0.00118 | 0.00058 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 17 | 0.00154 | 0.00075 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 18 | 0.00141 | 0.00066 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 19 | 0.00144 | 0.00071 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 20 | 0.00118 | 0.00054 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 21 | 0.00157 | 0.00075 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 22 | 0.00119 | 0.00059 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 23 | 0.00132 | 0.00068 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 24 | 0.00238 | 0.00119 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 25 | 0.0014 | 0.00071 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 26 | 0.00106 | 0.00053 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 27 | 0.00108 | 0.00055 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 28 | 0.0012 | 0.00056 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 29 | 0.00124 | 0.00063 | 122.024 | 380 | 380 | 50 | 50 | 0 |
| 30 | 0.00135 | 0.00064 | 122.024 | 380 | 380 | 50 | 50 | 0 |

### B.3.2 Mutator.

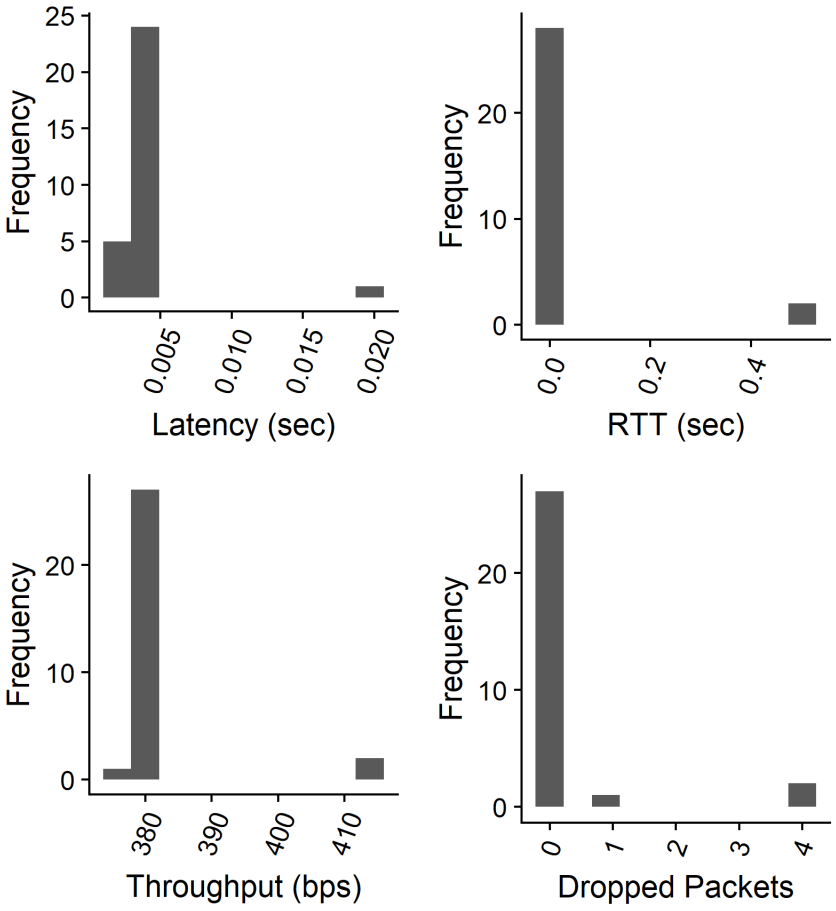



**Figure 48. Histograms of IMAP QoS Mutator Data**

**Table 27. IMAP Mutator QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.00425 | 0.00203 | 122.025 | 380 | 380 | 50 | 50 | 0 |
| 2 | 0.00414 | 0.00213 | 121.99 | 380 | 380 | 50 | 50 | 0 |
| 3 | 0.00308 | 0.00153 | 122.009 | 380 | 380 | 50 | 50 | 0 |
| 4 | 0.00281 | 0.50282 | 106.986 | 382 | 414 | 50 | 46 | 4 |
| 5 | 0.00344 | 0.0017 | 122.003 | 380 | 380 | 50 | 50 | 0 |
| 6 | 0.00484 | 0.00234 | 122.019 | 380 | 380 | 50 | 50 | 0 |
| 7 | 0.00389 | 0.00195 | 122.003 | 380 | 376 | 50 | 49 | 1 |
| 8 | 0.00432 | 0.00207 | 122.02 | 380 | 380 | 50 | 50 | 0 |
| 9 | 0.00281 | 0.50282 | 106.986 | 382 | 414 | 50 | 46 | 4 |
| 10 | 0.0042 | 0.00218 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 11 | 0.00304 | 0.00156 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 12 | 0.00326 | 0.00164 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 13 | 0.00308 | 0.00154 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 14 | 0.00303 | 0.00152 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 15 | 0.00293 | 0.0015 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 16 | 0.00343 | 0.00172 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 17 | 0.00289 | 0.00148 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 18 | 0.00374 | 0.00193 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 19 | 0.0033 | 0.00166 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 20 | 0.00322 | 0.00166 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 21 | 0.00356 | 0.00182 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 22 | 0.00314 | 0.0016 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 23 | 0.00289 | 0.00148 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 24 | 0.00313 | 0.00161 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 25 | 0.02052 | 0.00175 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 26 | 0.00336 | 0.0017 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 27 | 0.00306 | 0.00156 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 28 | 0.00312 | 0.00161 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 29 | 0.0034 | 0.00172 | 122.022 | 380 | 380 | 50 | 50 | 0 |
| 30 | 0.00317 | 0.00162 | 122.022 | 380 | 380 | 50 | 50 | 0 |

### B.3.3 T-test results.

```
[1] "Latency"

 Two Sample t-test

data:  Control and Mutation
t = -4.4067, df = 58, p-value = 4.593e-05
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.004108429 -0.001013097
sample estimates:
  mean of x   mean of y
0.001407335 0.003968098

[1] "RTT"

 Two Sample t-test

data:  Control and Mutation
t = -1.4843, df = 58, p-value = 0.1431
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.09626904  0.02736483
sample estimates:
   mean of x    mean of y
0.0006895218 0.0351416282

[1] "Duration"

 Two Sample t-test

data:  Control and Mutation
t = 1.4468, df = 58, p-value = 0.1533
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
```

```
 -0.8470467  2.8620049
sample estimates:
mean of x mean of y
 122.0244  121.0169

[1] "Throughput"

 Two Sample t-test

data:  Control and Mutation
t = -1.3442, df = 58, p-value = 0.1841
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -6.360270  2.093604
sample estimates:
mean of x mean of y
 380.0000  382.1333

[1] "Dropped Packets"

 Two Sample t-test

data:  Control and Mutation
t = -1.6075, df = 58, p-value = 0.1134
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.7970266  0.1970266
sample estimates:
mean of x mean of y
      0.0       0.3
```

### B.4 POP

This section contains the data from all QoS experiments for POP. For the sake of brevity, all column headers in this section are described here. Trial indicates the index associated with the data in a row. Latency shows the average time in milliseconds that it

took information from the client to reach the server for the trial. RTT indicates the amount of time in seconds from when a packet was sent by the server to the client and the receipt of the ACK from the client. Duration shows the length of the connection in seconds. BPSS represents the throughput of the client in bps. BPSR indicates the throughput of the server in bps. PacketsS displays the number of packets sent by the client during the transmission. PacketsR reports the number of packets sent by the server during the transmission.

### B.4.1 Control.

The result of one dropped packet in each trial for POP control experiments occurred because some of the messages sent were larger than the standard Ethernet frame size of 1518 Bytes. Specifically, the server key exchange initialization was 1714 Bytes. From the server perspective, this packet was not broken up into multiple packets but on the client packet captures, it appeared as two packets. This means that no packets were dropped by the control trials and the calculated average is misleading. The POP connection did not omit any data as TCP has built-in safeguards to handle dropped packets.

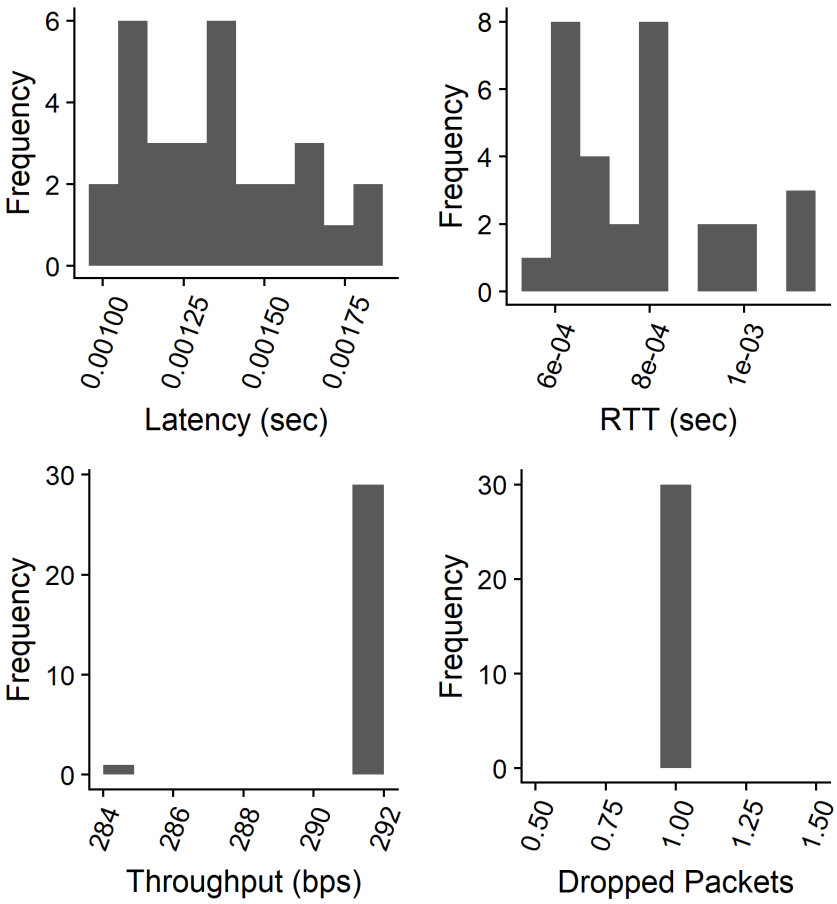



**Figure 49. Histograms of POP QoS Control Data**

**Table 28. POP Control QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.0016 | 0.00092 | 120.011 | 296 | 292 | 33 | 32 | 1 |
| 2 | 0.00113 | 0.00062 | 120.012 | 296 | 292 | 33 | 32 | 1 |
| 3 | 0.00137 | 0.00077 | 120.011 | 296 | 292 | 33 | 32 | 1 |
| 4 | 0.0014 | 0.00082 | 120.011 | 296 | 292 | 33 | 32 | 1 |
| 5 | 0.00113 | 0.00064 | 120.01 | 296 | 292 | 33 | 32 | 1 |
| 6 | 0.00154 | 0.00094 | 120.009 | 288 | 284 | 31 | 30 | 1 |
| 7 | 0.00133 | 8.00E-04 | 120.011 | 296 | 292 | 33 | 32 | 1 |
| 8 | 0.0011 | 0.00065 | 120.011 | 296 | 292 | 33 | 32 | 1 |
| 9 | 0.00119 | 0.00076 | 120.01 | 296 | 292 | 33 | 32 | 1 |
| 10 | 0.00181 | 0.00115 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 11 | 0.0017 | 0.00112 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 12 | 0.00105 | 0.00066 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 13 | 0.00105 | 0.00062 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 14 | 0.00133 | 8.00E-04 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 15 | 0.00181 | 0.00112 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 16 | 0.00102 | 0.00061 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 17 | 0.00144 | 0.00081 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 18 | 0.00144 | 0.00084 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 19 | 0.00156 | 0.00102 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 20 | 0.00122 | 0.00063 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 21 | 0.00107 | 0.00063 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 22 | 0.00137 | 0.00078 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 23 | 0.00099 | 6.00E-04 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 24 | 0.00136 | 0.00071 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 25 | 0.00126 | 0.00059 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 26 | 0.0013 | 0.00069 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 27 | 0.0013 | 8.00E-04 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 28 | 0.00166 | 0.00081 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 29 | 0.00164 | 0.001 | 120.009 | 296 | 292 | 33 | 32 | 1 |
| 30 | 0.00118 | 7.00E-04 | 120.009 | 296 | 292 | 33 | 32 | 1 |

### B.4.2 Mutator.

The result of one dropped packet in each trial for POP mutator experiments occurred because some of the messages sent were larger than the standard Ethernet frame size of 1518 Bytes. Specifically, the server key exchange initialization was 1714 Bytes. From the server perspective, this packet was not broken up into multiple packets but on the client packet captures, it appeared as two packets. This means that no packets were dropped by the control trials and the calculated average is misleading. The POP connection did not omit any data as TCP has built-in safeguards to handle dropped packets.

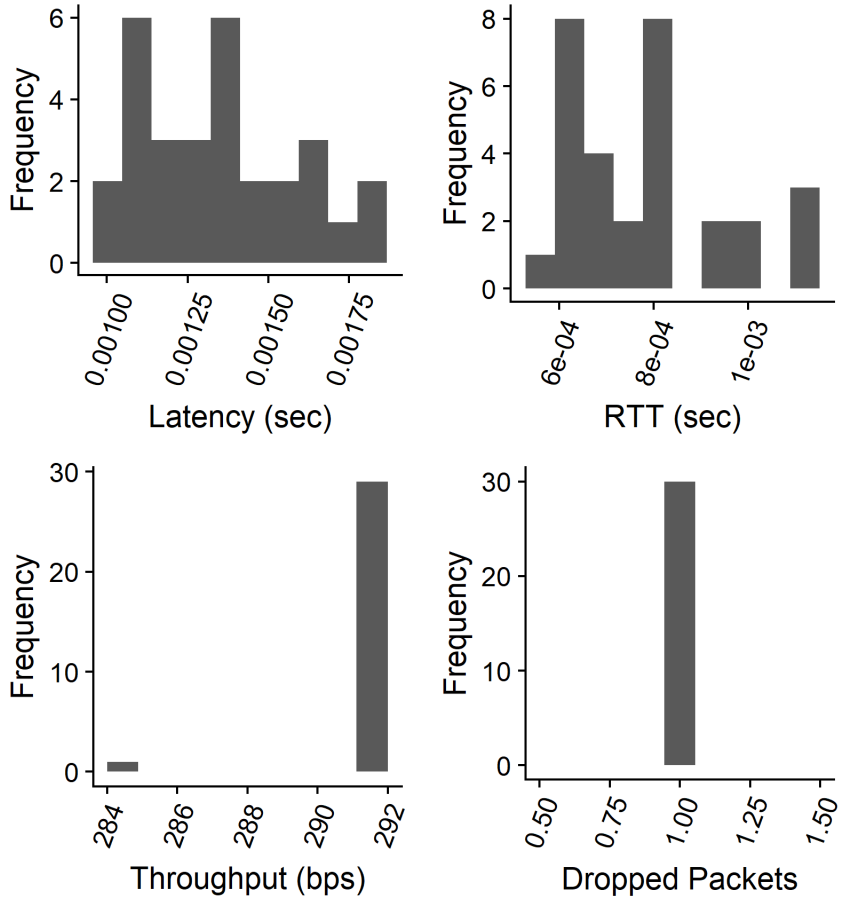



**Figure 50. Histograms of POP QoS Mutator Data**

**Table 29. POP Mutator QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.02657 | 0.00339 | 120.003 | 300 | 313 | 34 | 33 | 1 |
| 2 | 0.02732 | 0.88682 | 104.976 | 316 | 357 | 37 | 33 | 4 |
| 3 | 0.02832 | 0.00246 | 119.99 | 296 | 313 | 33 | 33 | 0 |
| 4 | 0.02836 | 0.00267 | 120.007 | 296 | 313 | 33 | 33 | 0 |
| 5 | 0.02889 | 0.00308 | 119.988 | 296 | 313 | 33 | 33 | 0 |
| 6 | 0.02979 | 0.00289 | 100.011 | 296 | 347 | 33 | 28 | 5 |
| 7 | 0.02751 | 0.88645 | 104.972 | 316 | 357 | 37 | 33 | 4 |
| 8 | 0.02733 | 0.88628 | 104.976 | 316 | 357 | 37 | 33 | 4 |
| 9 | 0.02828 | 0.00304 | 120.002 | 300 | 317 | 34 | 34 | 0 |
| 10 | 0.02776 | 0.00254 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 11 | 0.0276 | 0.00226 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 12 | 0.02678 | 0.00195 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 13 | 0.02811 | 0.00238 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 14 | 0.02746 | 0.00214 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 15 | 0.02652 | 0.00187 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 16 | 0.01242 | 0.02072 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 17 | 0.02756 | 0.00228 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 18 | 0.02713 | 0.00181 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 19 | 0.02757 | 0.00196 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 20 | 0.02735 | 0.00212 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 21 | 0.02646 | 0.00187 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 22 | 0.02725 | 0.83208 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 23 | 0.02707 | 0.00193 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 24 | 0.02767 | 0.00227 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 25 | 0.02739 | 0.00241 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 26 | 0.02651 | 0.00213 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 27 | 0.02746 | 0.00217 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 28 | 0.02701 | 0.00175 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 29 | 0.02776 | 0.00239 | 119.991 | 296 | 313 | 33 | 33 | 0 |
| 30 | 0.02696 | 0.00231 | 119.991 | 296 | 313 | 33 | 33 | 0 |

### B.4.3 T-test results.

```
[1] "Latency"

 Two Sample t-test

data:  Control and Mutation
t = -184.96, df = 57, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.02653919 -0.02578531
sample estimates:
  mean of x   mean of y
0.001345462 0.027507710

[1] "RTT"

 Two Sample t-test

data:  Control and Mutation
t = -2.18, df = 57, p-value = 0.0334
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.27027763  0.02704714
sample estimates:
  mean of x   mean of y
0.000787852 0.122403094

[1] "Duration"

 Two Sample t-test

data:  Control and Mutation
t = 2.1077, df = 58, p-value = 0.03939
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
```

```
 -0.5759626  4.9456326
sample estimates:
mean of x mean of y
 120.0094  117.8246

[1] "Throughput"

 Two Sample t-test

data:  Control and Mutation
t = -10.189, df = 58, p-value = 1.541e-14
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -33.97309 -19.89358
sample estimates:
mean of x mean of y
 291.7333  318.6667

[1] "Dropped Packets"

 Two Sample t-test

data:  Control and Mutation
t = 1.4841, df = 58, p-value = 0.1432
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.3178222  1.1178222
sample estimates:
mean of x mean of y
      1.0       0.6
```

### B.5 RTP

This section contains the data from all QoS experiments for RTP. For the sake of brevity, all column headers in this section are described here. Trial indicates the index associated with the data in a row. Jitter represents the average of the difference between

the forwarding delay of two consecutive packets in the same stream in seconds for each trial. Duration shows the length of the connection in seconds. BPS represents the throughput of the client in bps. PacketsS displays the number of packets sent by the client during the transmission. PacketsR reports the number of packets sent by the server during the transmission. Figure 51 indicates that a negative number of packets were dropped for some trials. This is a result of a limitation in the semi-automated method used to gather RTP data as the packet captures did not always begin with sufficient lead time before data was sent.

### B.5.1 Control.

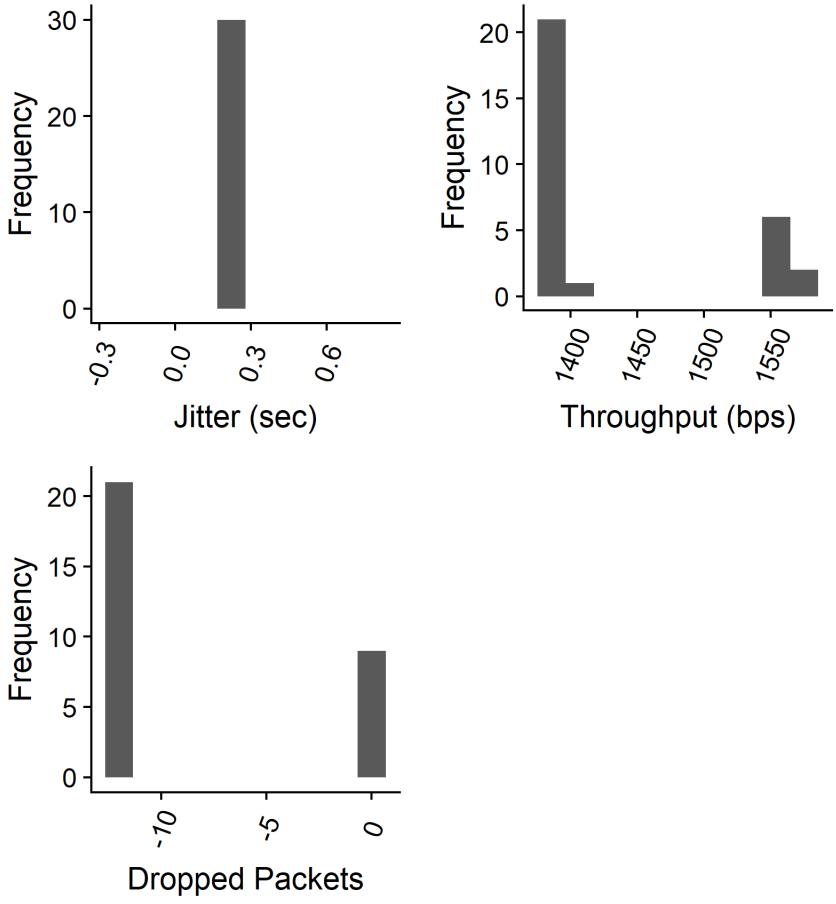



**Figure 51. Histograms of RTP QoS Control Data**

**Table 30. RTP Control QoS Data**

| Trial | MaxJitterR | jitterR | Duration | BPS | PacketsS | PacketsR | Dropped |
|---|---|---|---|---|---|---|---|
| 1 | 0.264 | 0.227 | 107.014703 | 1566 | 108 | 108 | 0 |
| 2 | 0.264 | 0.227 | 119.025531 | 1564 | 120 | 120 | 0 |
| 3 | 0.264 | 0.227 | 120.026317 | 1564 | 121 | 121 | 0 |
| 4 | 0.264 | 0.227 | 120.026437 | 1564 | 121 | 121 | 0 |
| 5 | 0.264 | 0.227 | 110.024517 | 1565 | 111 | 111 | 0 |
| 6 | 0.264 | 0.227 | 118.025383 | 1564 | 119 | 119 | 0 |
| 7 | 0.264 | 0.227 | 120.02717 | 1564 | 121 | 121 | 0 |
| 8 | 0.264 | 0.227 | 119.027055 | 1564 | 120 | 120 | 0 |
| 9 | 0.264 | 0.227 | 137.246447 | 1400 | 128 | 128 | 0 |
| 10 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 11 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 12 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 13 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 14 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 15 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 16 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 17 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 18 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 19 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 20 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 21 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 22 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 23 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 24 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 25 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 26 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 27 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 28 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 29 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |
| 30 | 0.264 | 0.227 | 140.682634 | 1377 | 120 | 132 | -12 |

### B.5.2 Mutator.

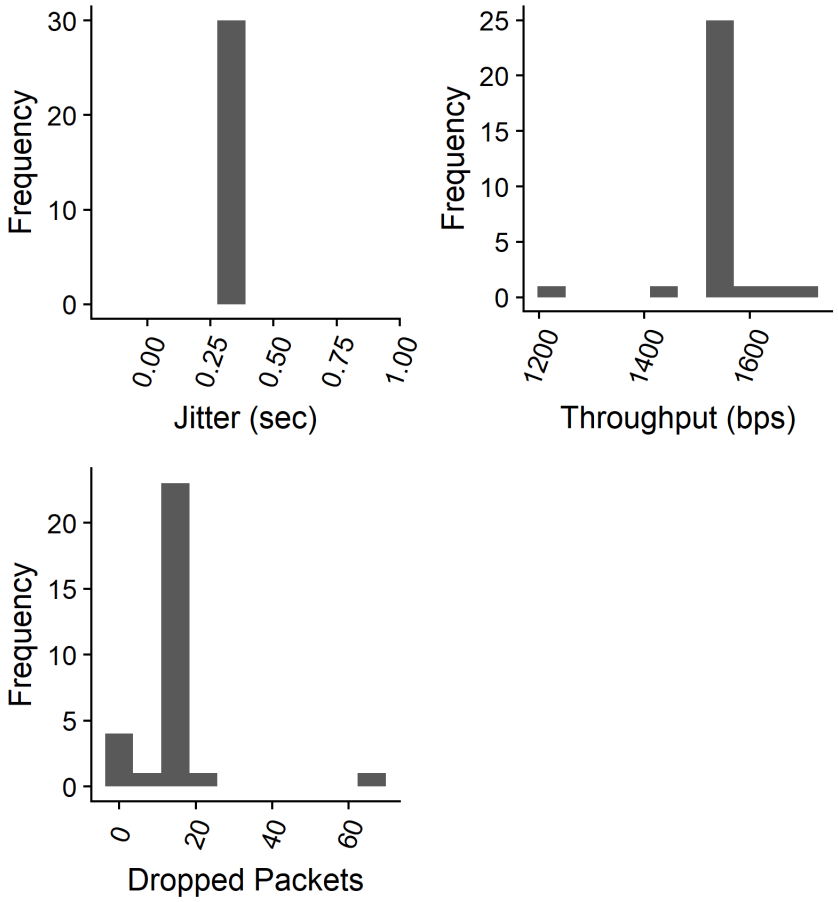



**Figure 52. Histograms of RTP QoS Mutator Data**

**Table 31. RTP Mutator QoS Data**

| Trial | MaxJitterR | jitterR | Duration | BPS | PacketsS | PacketsR | Dropped |
|---|---|---|---|---|---|---|---|
| 1 | 1.053 | 0.355 | 109.012512 | 1566 | 110 | 110 | 0 |
| 2 | 1.053 | 0.355 | 118.02527 | 1583 | 188 | 122 | 66 |
| 3 | 1.053 | 0.355 | 104.019535 | 1566 | 127 | 105 | 22 |
| 4 | 1.053 | 0.355 | 99.998782 | 1709 | 121 | 109 | 12 |
| 5 | 1.053 | 0.355 | 113.016961 | 1565 | 114 | 114 | 0 |
| 6 | 1.053 | 0.355 | 111.997591 | 1565 | 125 | 113 | 12 |
| 7 | 1.053 | 0.355 | 111.997824 | 1676 | 120 | 120 | 0 |
| 8 | 1.053 | 0.355 | 141.674235 | 1230 | 123 | 112 | 11 |
| 9 | 1.053 | 0.355 | 112.018064 | 1460 | 107 | 107 | 0 |
| 10 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 11 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 12 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 13 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 14 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 15 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 16 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 17 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 18 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 19 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 20 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 21 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 22 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 23 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 24 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 25 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 26 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 27 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 28 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 29 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |
| 30 | 1.053 | 0.355 | 109.111131 | 1554 | 121 | 109 | 12 |

### B.5.3 T-test results.

```
1  [1] "Max Jitter"
2  [1] "T test error. Is your data essentially constant?"
3  [1] "Mean Jitter"
4  [1] "T test error. Is your data essentially constant?"
5  [1] "Duration"
6
7   Two Sample t-test
8
9  data:  Control and Mutation
10 t = 10.124, df = 58, p-value = 1.959e-14
11 alternative hypothesis: true difference in means is not equal to 0
12 99 percent confidence interval:
13  17.48243 29.96319
14 sample estimates:
15 mean of x mean of y
16  134.1593  110.4365
17
18 [1] "Throughput"
19
20  Two Sample t-test
21
22 data:  Control and Mutation
23 t = -6.1149, df = 58, p-value = 8.804e-08
24 alternative hypothesis: true difference in means is not equal to 0
25 99 percent confidence interval:
26  -178.10276  -70.03057
27 sample estimates:
28 mean of x mean of y
29  1427.733  1551.800
30
31 [1] "Dropped Packets"
32
33  Two Sample t-test
34
```

```
35 data:  Control and Mutation
36 t = -9.1971, df = 58, p-value = 6.28e-13
37 alternative hypothesis: true difference in means is not equal to 0
38 99 percent confidence interval:
39  -26.95218 -14.84782
40 sample estimates:
41 mean of x mean of y
42       -8.4       12.5
```

### B.6 SMTP

This section contains the data from all QoS experiments for SMTP. For the sake of brevity, all column headers in this section are described here. Trial indicates the index associated with the data in a row. Latency shows the average time in milliseconds that it took information from the client to reach the server for the trial. RTT indicates the amount of time in seconds from when a packet was sent by the server to the client and the receipt of the ACK from the client. Duration shows the length of the connection in seconds. BPSS represents the throughput of the client in bps. BPSR indicates the throughput of the server in bps. PacketsS displays the number of packets sent by the client during the transmission. PacketsR reports the number of packets sent by the server during the transmission.

### B.6.1 Control.

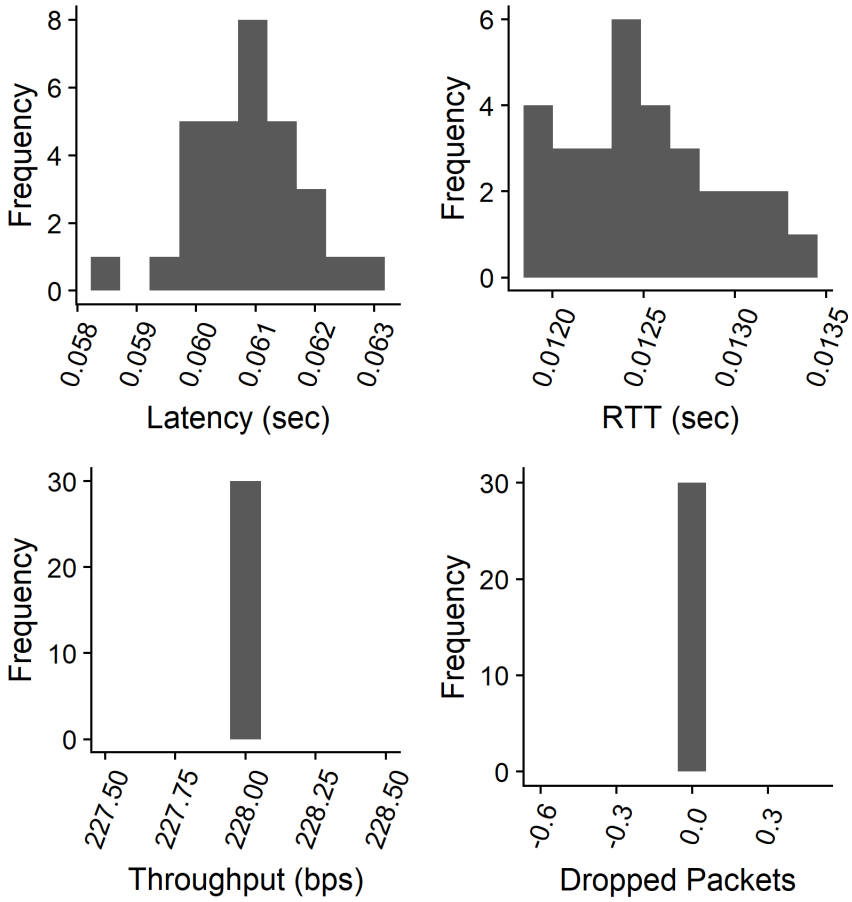



**Figure 53. Histograms of SMTP QoS Control Data**

**Table 32. SMTP Control QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.06203 | 0.01271 | 120.023 | 227 | 228 | 45 | 45 | 0 |
| 2 | 0.06121 | 0.01324 | 120.024 | 227 | 228 | 45 | 45 | 0 |
| 3 | 0.06058 | 0.01266 | 120.024 | 227 | 228 | 45 | 45 | 0 |
| 4 | 0.0612 | 0.01254 | 120.023 | 227 | 228 | 45 | 45 | 0 |
| 5 | 0.06037 | 0.0125 | 120.024 | 227 | 228 | 45 | 45 | 0 |
| 6 | 0.06091 | 0.01234 | 120.025 | 227 | 228 | 45 | 45 | 0 |
| 7 | 0.06272 | 0.01318 | 120.023 | 227 | 228 | 45 | 45 | 0 |
| 8 | 0.06004 | 0.01185 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 9 | 0.06157 | 0.01288 | 120.023 | 227 | 228 | 45 | 45 | 0 |
| 10 | 0.06219 | 0.01229 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 11 | 0.06098 | 0.0124 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 12 | 0.06075 | 0.01309 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 13 | 0.0602 | 0.01211 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 14 | 0.06027 | 0.01216 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 15 | 0.06111 | 0.01186 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 16 | 0.06034 | 0.01192 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 17 | 0.06078 | 0.01294 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 18 | 0.05964 | 0.01233 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 19 | 0.06248 | 0.01306 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 20 | 0.05826 | 0.01206 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 21 | 0.06038 | 0.01253 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 22 | 0.06016 | 0.01244 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 23 | 0.06098 | 0.01232 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 24 | 0.06021 | 0.01249 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 25 | 0.06018 | 0.0123 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 26 | 0.06193 | 0.01243 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 27 | 0.06119 | 0.01269 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 28 | 0.06159 | 0.0133 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 29 | 0.06121 | 0.01233 | 120.022 | 227 | 228 | 45 | 45 | 0 |
| 30 | 0.06129 | 0.01198 | 120.022 | 227 | 228 | 45 | 45 | 0 |

### B.6.2 Mutator.

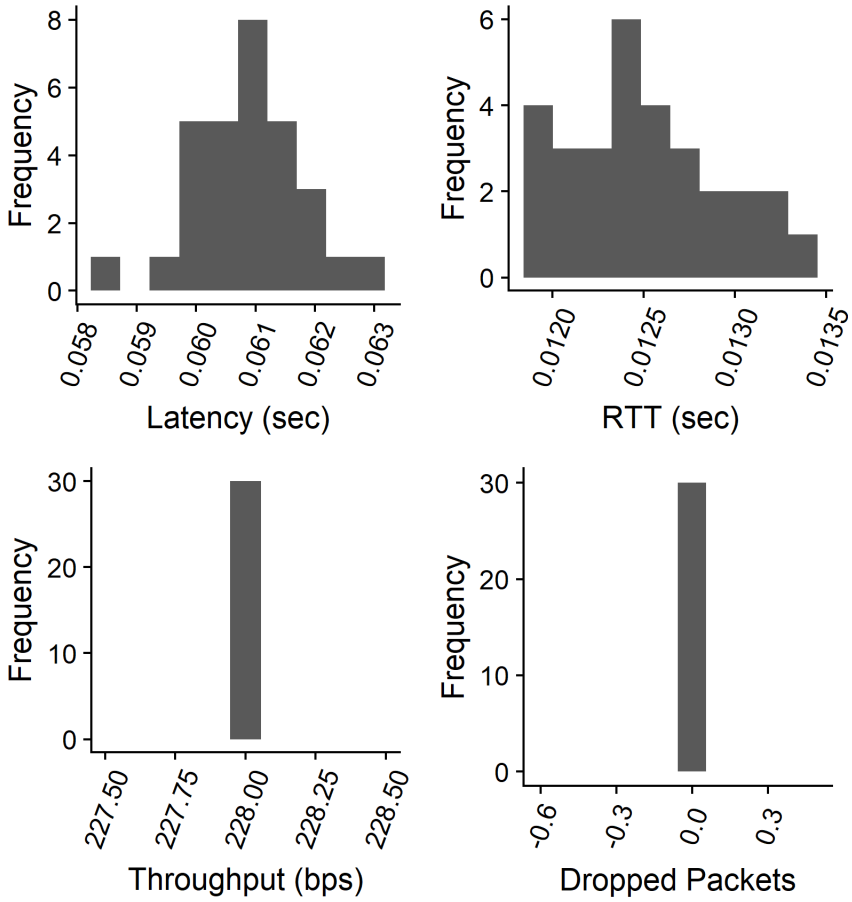



**Figure 54. Histograms of SMTP QoS Mutator Data**

**Table 33. SMTP Mutator QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.06534 | 0.01469 | 120.018 | 227 | 228 | 45 | 45 | 0 |
| 2 | 0.0632 | 0.01439 | 120.004 | 227 | 228 | 45 | 45 | 0 |
| 3 | 0.07002 | 0.90205 | 104.995 | 207 | 216 | 40 | 36 | 4 |
| 4 | 0.0642 | 0.0143 | 120.015 | 227 | 228 | 45 | 45 | 0 |
| 5 | 0.06485 | 0.01479 | 120.018 | 227 | 228 | 45 | 45 | 0 |
| 6 | 0.0688 | 0.90224 | 104.978 | 207 | 216 | 40 | 36 | 4 |
| 7 | 0.06439 | 0.01451 | 120.019 | 227 | 228 | 45 | 45 | 0 |
| 8 | 0.07127 | 0.90421 | 104.966 | 207 | 216 | 40 | 36 | 4 |
| 9 | 0.0696 | 0.90325 | 104.988 | 207 | 216 | 40 | 36 | 4 |
| 10 | 0.06532 | 0.01497 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 11 | 0.06182 | 0.01401 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 12 | 0.06362 | 0.01435 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 13 | 0.06363 | 0.01379 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 14 | 0.06477 | 0.01384 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 15 | 0.06357 | 0.01533 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 16 | 0.06265 | 0.01359 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 17 | 0.06482 | 0.01594 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 18 | 0.06405 | 0.01324 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 19 | 0.06327 | 0.01497 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 20 | 0.06421 | 0.01399 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 21 | 0.06188 | 0.01315 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 22 | 0.0643 | 0.01504 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 23 | 0.06286 | 0.01325 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 24 | 0.06244 | 0.0129 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 25 | 0.06389 | 0.01325 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 26 | 0.08138 | 0.01406 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 27 | 0.06136 | 0.01244 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 28 | 0.0636 | 0.01436 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 29 | 0.06423 | 0.01472 | 120.02 | 227 | 228 | 45 | 45 | 0 |
| 30 | 0.06207 | 0.01374 | 120.02 | 227 | 228 | 45 | 45 | 0 |

### B.6.3 T-test results.

```
1  [1] "Latency"
2
3   Two Sample t-test
4
5  data:  Control and Mutation
6  t = -5.6527, df = 58, p-value = 5.056e-07
7  alternative hypothesis: true difference in means is not equal to 0
8  99 percent confidence interval:
9   -0.006111195 -0.002196831
10 sample estimates:
11  mean of x  mean of y
12 0.06089210 0.06504611
13
14 [1] "RTT"
15
16  Two Sample t-test
17
18 data:  Control and Mutation
19 t = -2.1415, df = 58, p-value = 0.03645
20 alternative hypothesis: true difference in means is not equal to 0
21 99 percent confidence interval:
22  -0.26957080  0.02927669
23 sample estimates:
24  mean of x  mean of y
25 0.01249851 0.13264557
26
27 [1] "Duration"
28
29  Two Sample t-test
30
31 data:  Control and Mutation
32 t = 2.116, df = 58, p-value = 0.03865
33 alternative hypothesis: true difference in means is not equal to 0
34 99 percent confidence interval:
```

```
 -0.5194891  4.5364688
sample estimates:
mean of x mean of y
 120.0223  118.0138

[1] "Throughput"

 Two Sample t-test

data:  Control and Mutation
t = 2.1122, df = 58, p-value = 0.03898
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.4174168  3.6174168
sample estimates:
mean of x mean of y
    228.0     226.4

[1] "Dropped Packets"

 Two Sample t-test

data:  Control and Mutation
t = -2.1122, df = 58, p-value = 0.03898
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -1.2058056  0.1391389
sample estimates:
mean of x mean of y
0.0000000 0.5333333
```

### B.7 SSH

This section contains the data from all QoS experiments for SSH. For the sake of brevity, all column headers in this section are described here. Trial indicates the index associated with the data in a row. Latency shows the average time in milliseconds that it

took information from the client to reach the server for the trial. RTT indicates the amount of time in seconds from when a packet was sent by the server to the client and the receipt of the ACK from the client. Duration shows the length of the connection in seconds. BPSS represents the throughput of the client in bps. BPSR indicates the throughput of the server in bps. PacketsS displays the number of packets sent by the client during the transmission. PacketsR reports the number of packets sent by the server during the transmission.

### B.7.1 Control.

The result of one dropped packet in each trial for SSH control experiments occurred because some of the messages sent were larger than the standard ethernet frame size of 1518 Bytes. Specifically, the server key exchange initialization was 1714 Bytes. From the server perspective, this packet was not broken up into multiple packets but on the client packet captures, it appeared as two packets. This means that no packets were dropped by the control trials and the calculated average is misleading. The SSH connection did not omit any data as TCP has built-in safeguards to handle dropped packets.

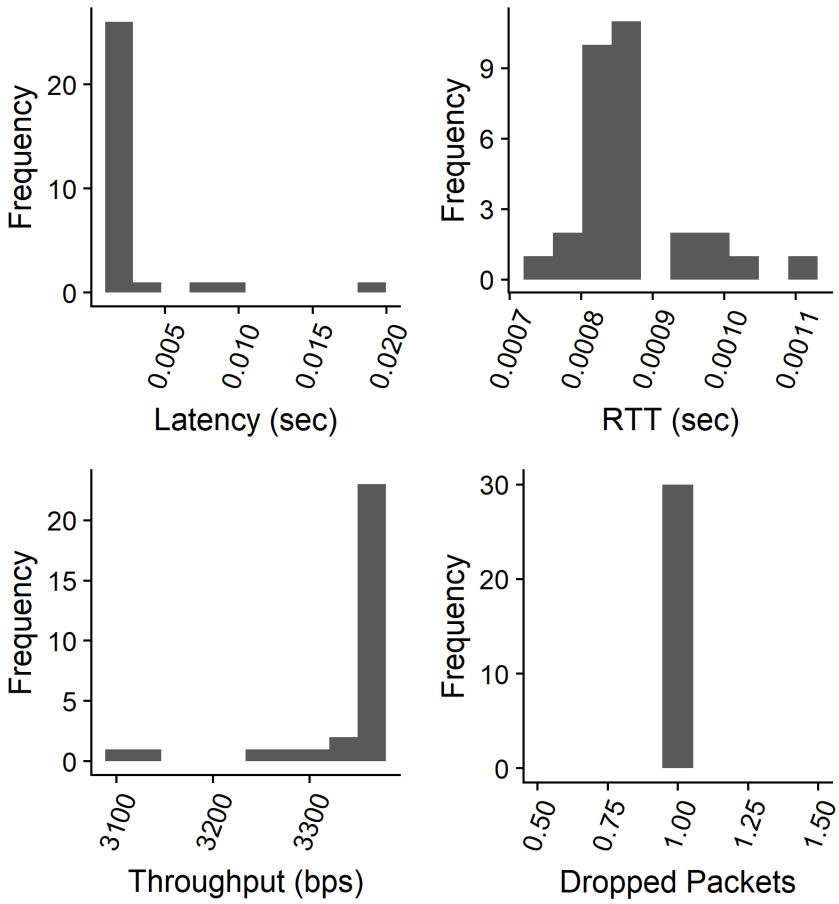



**Figure 55. Histograms of SSH QoS Control Data**

**Table 34. SSH Control QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.01874 | 0.00073 | 148.945 | 3125 | 3121 | 582 | 581 | 1 |
| 2 | 0.00186 | 0.00082 | 132.385 | 3377 | 3373 | 562 | 561 | 1 |
| 3 | 0.00184 | 0.00099 | 140.238 | 3116 | 3112 | 546 | 545 | 1 |
| 4 | 0.00192 | 0.00084 | 134.196 | 3296 | 3292 | 555 | 554 | 1 |
| 5 | 0.0019 | 0.00096 | 131.572 | 3369 | 3365 | 558 | 557 | 1 |
| 6 | 0.00194 | 0.00085 | 132.547 | 3337 | 3333 | 556 | 555 | 1 |
| 7 | 0.0017 | 0.00081 | 136.171 | 3256 | 3252 | 558 | 557 | 1 |
| 8 | 0.00174 | 0.00082 | 134.715 | 3283 | 3279 | 556 | 555 | 1 |
| 9 | 0.00188 | 0.00085 | 132.117 | 3348 | 3344 | 556 | 555 | 1 |
| 10 | 0.0017 | 0.00087 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 11 | 0.0031 | 0.00088 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 12 | 0.00883 | 0.00102 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 13 | 0.00203 | 0.00078 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 14 | 0.00203 | 0.00082 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 15 | 0.00171 | 0.00085 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 16 | 0.00165 | 0.0011 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 17 | 0.00175 | 0.00086 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 18 | 0.00174 | 8.00E-04 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 19 | 0.00198 | 0.00083 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 20 | 0.00183 | 0.00085 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 21 | 0.00804 | 0.00094 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 22 | 0.00202 | 0.00083 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 23 | 0.00175 | 0.00084 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 24 | 0.00181 | 0.00083 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 25 | 0.00177 | 0.00083 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 26 | 0.00202 | 0.00086 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 27 | 0.00169 | 0.00085 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 28 | 0.00185 | 0.00099 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 29 | 0.00182 | 0.00088 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |
| 30 | 0.00172 | 0.00086 | 129.615 | 3371 | 3367 | 546 | 545 | 1 |

### B.7.2 Mutator.

The single negative result for dropped packets as shown in Figure 56 occurred during trial 5. In this trial, the packet capture of the client terminated out-of-sync with the server. This experimental error accounts for the negative result. In reality, zero packets were dropped.

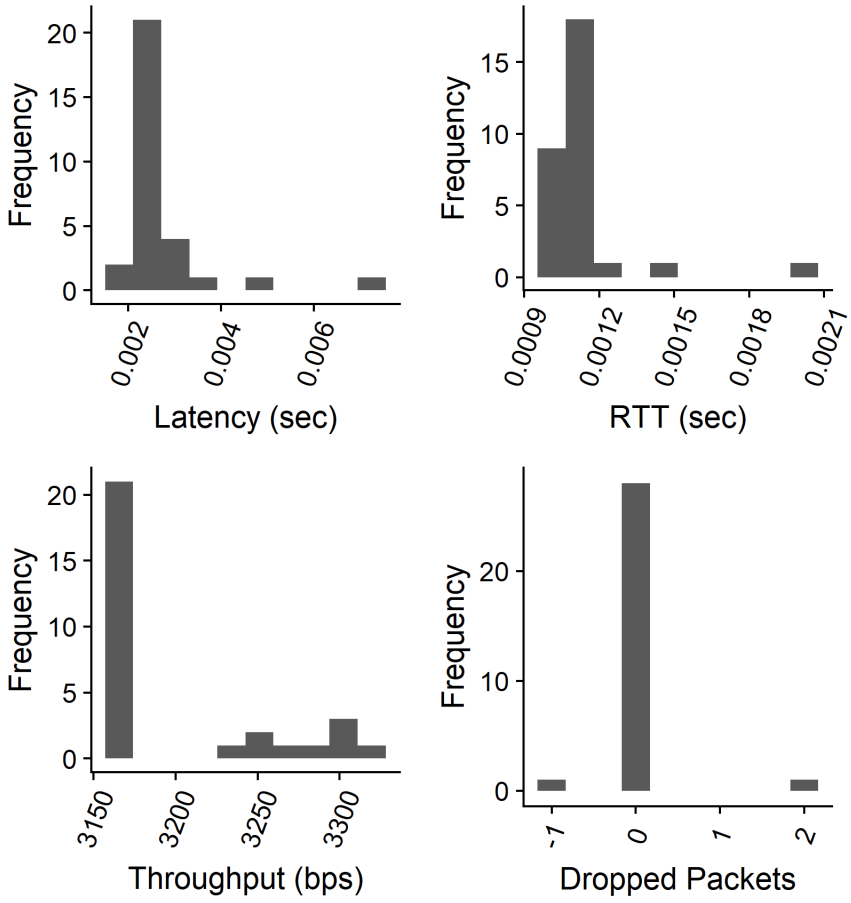



**Figure 56. Histograms of SSH QoS Mutator Data**

**Table 35. SSH Mutator QoS Data**

| Trial | Latency | RTT | Duration | BPSS | BPSR | PacketsS | PacketsR | PktsDrop |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.00233 | 0.00112 | 128.592 | 3310 | 3310 | 532 | 532 | 0 |
| 2 | 0.00249 | 0.00117 | 129.839 | 3295 | 3295 | 534 | 534 | 0 |
| 3 | 0.00302 | 0.00115 | 135.189 | 3254 | 3254 | 546 | 546 | 0 |
| 4 | 0.00242 | 0.00197 | 133.387 | 3335 | 3325 | 556 | 554 | 2 |
| 5 | 0.00218 | 0.00107 | 128.011 | 3287 | 3295 | 520 | 521 | -1 |
| 6 | 0.00273 | 0.00115 | 127.689 | 3284 | 3284 | 519 | 519 | 0 |
| 7 | 0.00248 | 0.00108 | 126.591 | 3229 | 3230 | 498 | 498 | 0 |
| 8 | 0.00262 | 0.00117 | 137.37 | 3313 | 3247 | 560 | 560 | 0 |
| 9 | 0.00217 | 0.00106 | 137.371 | 3274 | 3275 | 566 | 566 | 0 |
| 10 | 0.00753 | 0.00097 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 11 | 0.00217 | 0.00107 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 12 | 0.0025 | 0.00119 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 13 | 0.00236 | 0.00108 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 14 | 0.00233 | 0.00107 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 15 | 0.00285 | 0.00101 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 16 | 0.00264 | 0.00096 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 17 | 0.00211 | 0.00102 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 18 | 0.00355 | 0.00142 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 19 | 0.0021 | 0.00108 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 20 | 0.00218 | 0.00102 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 21 | 0.00214 | 0.00111 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 22 | 0.00226 | 0.00115 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 23 | 0.00262 | 0.00113 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 24 | 0.00457 | 0.00101 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 25 | 0.00249 | 0.00096 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 26 | 0.00294 | 0.001 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 27 | 0.00234 | 0.00108 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 28 | 0.00218 | 0.00115 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 29 | 0.0023 | 0.00115 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |
| 30 | 0.00234 | 0.00111 | 138.291 | 3170 | 3171 | 545 | 545 | 0 |

### B.7.3 T-test results.

```
1  [1] "Latency"
2
3   Two Sample t-test
4
5  data:  Control and Mutation
6  t = -5.6527, df = 58, p-value = 5.056e-07
7  alternative hypothesis: true difference in means is not equal to 0
8  99 percent confidence interval:
9   -0.006111195 -0.002196831
10 sample estimates:
11  mean of x  mean of y
12 0.06089210 0.06504611
13
14 [1] "RTT"
15
16  Two Sample t-test
17
18 data:  Control and Mutation
19 t = -2.1415, df = 58, p-value = 0.03645
20 alternative hypothesis: true difference in means is not equal to 0
21 99 percent confidence interval:
22  -0.26957080  0.02927669
23 sample estimates:
24  mean of x  mean of y
25 0.01249851 0.13264557
26
27 [1] "Duration"
28
29  Two Sample t-test
30
31 data:  Control and Mutation
32 t = 2.116, df = 58, p-value = 0.03865
33 alternative hypothesis: true difference in means is not equal to 0
34 99 percent confidence interval:
```

```
 -0.5194891  4.5364688
sample estimates:
mean of x mean of y
 120.0223  118.0138

[1] "Throughput"

 Two Sample t-test

data:  Control and Mutation
t = 2.1122, df = 58, p-value = 0.03898
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -0.4174168  3.6174168
sample estimates:
mean of x mean of y
    228.0     226.4

[1] "Dropped Packets"

 Two Sample t-test

data:  Control and Mutation
t = -2.1122, df = 58, p-value = 0.03898
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -1.2058056  0.1391389
sample estimates:
mean of x mean of y
0.0000000 0.5333333
```

# Appendix C. Experiment Scripts

## C.1 Mutator Code

### C.1.1 Mutator_PlebeNet.py.

```
1  # Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #     http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12 # implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.
15
16 from ryu.base import app_manager
17 from ryu.controller import ofp_event
18 from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
19 from ryu.controller.handler import set_ev_cls
20 from ryu.ofproto import ofproto_v1_3
21 from ryu.lib.packet import packet
22 from ryu.lib.packet import ethernet
23 from ryu.lib.packet import tcp
24 from ryu.lib.packet import udp
25 from ryu.lib.packet import ipv4
26 from ryu.lib.packet import arp
27 from ryu.lib.packet import icmp
28 from ryu.lib.packet import ether_types
29 from ryu import cfg
30 from Mutator import Mutator  #Import Mutator class
```

```
31 from ActiveConnection import ActiveConnection
32 import logging
33 import schedule
34
35 #Set up Logging
36 logger = logging.getLogger('SDNMutator')
37 hdlr = logging.FileHandler('SDNMutator.log')
38 formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s')
39 hdlr.setFormatter(formatter)
40 logger.addHandler(hdlr)
41 logger.setLevel(logging.DEBUG)
42
43 class MutationController(app_manager.RyuApp):
44     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
45
46     def __init__(self, *args, **kwargs):
47         super(MutationController, self).__init__(*args, **kwargs)
48
49         self.mac_to_port = {}
50         self.RIP_VIP = {}
51         self.VIP_RIP = {}
52         self.actives = []
53         self.m = 0  # Track mutation number
54
55         self.logger = logging.getLogger('SDNMutator.Logger')
56         self.logger.info('Creating instance of Logger')
57
58         # Set up argument parsing
59         CONF = cfg.CONF
60         CONF.register_opts([
61             cfg.IntOpt('frequency', default=60, help=('Mutation rate, in
                seconds')),
62             cfg.ListOpt('networks', default=None, help=('First three octets of
                 IPv4 address ranges to mutate. Comma separated.')),
63             cfg.IntOpt('timeout', default=240, help=('Seconds until flow
```

```
                timeout'))])
64          self.logger.info('Arguments received: %s', CONF.list_all_sections())
65          networks = CONF.networks
66          frequency = CONF.frequency
67          self.timeout = CONF.timeout
68          print(frequency)
69          print(networks)
70          print("Adding mutations to scheduler.")
71          self.mutator = Mutator(networks)
72
73          schedule.every(frequency).seconds.do(self.triggerMutation, self.
                mutator, networks, False, self.m)
74          schedule.run_continuously()
75          print("Scheduler active.")
76
77      def triggerMutation(self, mutator, networks, first, m_in):
78          m_in = m_in + 1
79          mutations = mutator.mutate(networks, first, m_in)
80          self.RIP_VIP = mutations[0]
81          self.VIP_RIP = mutations[1]
82          self.actives = mutations[2]
83
84      # <editor-fold desc="Mutation processing">
85      def add_flow(self, datapath, priority, match, actions, buffer_id=None):
86          ofproto = datapath.ofproto
87          parser = datapath.ofproto_parser
88          timeout = self.timeout
89
90          inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                actions)]
91          if buffer_id:
92              mod = parser.OFPFlowMod(datapath=datapath, idle_timeout=timeout,
                    hard_timeout=timeout, buffer_id=buffer_id,
93                                      priority=priority, match=match,
94                                      instructions=inst)
```

```
95          else:
96              mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                    idle_timeout=timeout, hard_timeout=timeout,
97                                          match=match, instructions=inst)
98          datapath.send_msg(mod)
99
100     def addActive(self, src_rip, dst_rip, src_vip, dst_vip, sPort, dPort,
            protocol):
101         conn = ActiveConnection(src_rip, src_vip, dst_rip, dst_vip, dst_vip,
                sPort, dPort, protocol)
102
103         print "Adding new entry to Actives"
104         if not any(str(x) == str(conn) for x in self.actives):
105             self.actives.append(ActiveConnection(src_rip, src_vip, dst_rip,
                    dst_vip, dst_vip, sPort, dPort, protocol))
106             if protocol == "TCP":
107                 self.actives.append(ActiveConnection(dst_rip, dst_vip, src_rip
                        , src_vip, src_vip, dPort, sPort, protocol))
108         else:
109             self.logger.info("Duplicate entry in self.actives avoided")
110
111     def address_translation(self, RIP, VIP):
112         if RIP not in self.RIP_VIP:
113             return False
114         elif VIP not in self.VIP_RIP:
115             return False
116         else:
117             return True
118
119     #translate RIPs & VIPs
120     def lookupAddresses(self, src_rip, dst_vip):
121         translation = {}
122
123         # Check actives table before searching mappings
124         conn = ActiveConnection.find_by_rSrc_pDst(src_rip, dst_vip)
```

```
125
126         if conn is None:
127             self.mutator.printActives()
128             print str(conn)
129             print "No entry found. Search RIP:VIP table."
130
131             # Catch if there doesn't exist a translation in RIP:VIP
132             if not self.address_translation(src_rip, dst_vip):
133                 print "Translation does not exist."
134                 return
135             else:
136                 src_vip = self.RIP_VIP[src_rip]
137                 dst_rip = self.VIP_RIP[dst_vip]
138         else:
139             print "Entry found in actives."
140             src_rip = conn.rSrc
141             src_vip = conn.vSrc
142             dst_rip = conn.rDst
143             dst_vip = conn.pDst
144
145         translation.update({'src_rip': src_rip})
146         translation.update({'src_vip': src_vip})
147         translation.update({'dst_rip': dst_rip})
148         translation.update({'dst_vip': dst_vip})
149         print translation
150         return translation
151     # </editor-fold>
152
153     # <editor-fold desc="Packet translation methods">
154     def arpTranslation(self, arpPkt, dpid, parser, out_port, ofproto, msg,
        datapath, in_port):
155         arpPkt = arpPkt[0]
156
157         map = self.lookupAddresses(arpPkt.src_ip, arpPkt.dst_ip)
158
```

```
159         # Catch if there doesn't exist a translation in RIP:VIP
160         # if not self.address_translation(map["src_rip"], map["dst_vip"]):
161         #     return
162
163         self.logPacket("ARP", map["src_rip"], map["dst_rip"], map["src_vip"],
                map["dst_vip"])
164
165         actions = [parser.OFPActionSetField(arp_tpa=map["dst_rip"]), parser.
                OFPActionSetField(arp_spa=map["src_vip"]),
166                    parser.OFPActionOutput(out_port)]
167
168         # install a flow to avoid the controller having to decide
169         if out_port != ofproto.OFPP_FLOOD:
170             match = parser.OFPMatch(in_port=in_port, eth_type=0x806, arp_tpa=
                    map["dst_vip"], arp_spa=map["src_rip"])
171             # verify if we have a valid buffer_id, if yes avoid to send both
                    flow_mod & packet_out
172             if msg.buffer_id != ofproto.OFP_NO_BUFFER:
173                 self.add_flow(datapath, 1, match, actions, msg.buffer_id)
174                 return
175             else:
176                 self.add_flow(datapath, 1, match, actions)
177             self.addActive(map["src_rip"], map["dst_rip"], map["src_vip"], map
                    ["dst_vip"], "ARP")
178         self.packet_out(msg, ofproto, parser, datapath, in_port, actions)
179
180     def icmpTranslation(self, ipv4Pkt, icmpPkt, dpid, parser, out_port,
            ofproto, msg, datapath, in_port):
181         ipv4Pkt = ipv4Pkt[0]
182         icmpPkt = icmpPkt[0]
183
184         map = self.lookupAddresses(ipv4Pkt.src, ipv4Pkt.dst)
185
186         # Catch if there is no translation
187         # if not self.address_translation(map["src_rip"], map["dst_vip"]):
```

```
188             #       return
189
190             self.logPacket("ICMP", map["src_rip"], map["dst_rip"], map["src_vip"],
                    map["dst_vip"])
191
192             actions = [parser.OFPActionSetField(ipv4_dst=map["dst_rip"]), parser.
                    OFPActionSetField(ipv4_src=map["src_vip"]),
193                             parser.OFPActionOutput(out_port)]
194
195             # install a flow to avoid the controller having to decide
196             if out_port != ofproto.OFPP_FLOOD:
197                 match = parser.OFPMatch(in_port=in_port, eth_type=0x800, ipv4_dst=
                        map["dst_vip"], ipv4_src=map["src_rip"],
198                                                 ip_proto=1, icmpv4_code=icmpPkt.code,
                                                        icmpv4_type=icmpPkt.type)
199                 # verify a valid buffer_id, if yes avoid to send both flow_mod &
                        packet_out
200                 if msg.buffer_id != ofproto.OFP_NO_BUFFER:
201                     self.add_flow(datapath, 1, match, actions, msg.buffer_id)
202
203                     return
204                 else:
205                     self.add_flow(datapath, 1, match, actions)
206                 print "hit"
207                 #self.addActive(map["src_rip"], map["dst_rip"], map["src_vip"],
                        map["dst_vip"], "ICMP")
208             self.packet_out(msg, ofproto, parser, datapath, in_port, actions)
209
210         def ipv4Translation(self, ipv4Pkt_in, tcpPkt_in, udpPkt_in, dpid, parser,
                out_port, ofproto, msg, datapath, in_port):
211             ipv4Pkt = ipv4Pkt_in[0]
212             map = self.lookupAddresses(ipv4Pkt.src, ipv4Pkt.dst)
213             match = parser.OFPMatch(in_port=in_port, eth_type=0x800, ipv4_dst=map
                    ["dst_vip"], ipv4_src=map["src_rip"])  # Default match rule
214
```

```
215         # Catch if there exists a translation
216         # if not self.address_translation(map["src_rip"], map["dst_vip"]):
217         #     return
218
219         #TCP Packet Processing
220         try:
221             tcpPkt = tcpPkt_in[0]
222             print "TCP: " + str(ipv4Pkt.src) + ":" + str(tcpPkt.src_port) + ",
                    " + str(ipv4Pkt.dst) + ":" + str(tcpPkt.dst_port)
223             if tcpPkt.has_flags(tcp.TCP_SYN, tcp.TCP_ACK):
224                 self.addActive(map["src_rip"], map["dst_rip"], map["src_vip"],
                        map["dst_vip"],
225                                tcpPkt.src_port, tcpPkt.dst_port, "TCP")
226                 match = parser.OFPMatch(in_port=in_port, eth_type=0x800,
                        ipv4_dst=map["dst_vip"], ipv4_src=map["src_rip"],
227                                         ip_proto=6, tcp_src=tcpPkt.src_port,
                                            tcp_dst=tcpPkt.dst_port)
228             elif tcpPkt.has_flags(tcp.TCP_FIN, tcp.TCP_ACK):
229                 sender = ActiveConnection.find_by_rSrc_port(map["src_rip"],
                        tcpPkt.src_port, tcp.dst_port)
230                 receiver = ActiveConnection.find_by_rSrc_port(map["dst_rip"],
                        tcpPkt.dst_port, tcpPkt.src_port)
231                 print "\n\n\nHIT\n\n\n"
232                 self.logger.info("Received FINACK on:\n" + sender + "\n" +
                        receiver)
233                 match = parser.OFPMatch(in_port=in_port, eth_type=0x800,
                        ipv4_dst=map["dst_vip"], ipv4_src=map["src_rip"],
234                                         ip_proto=6, tcp_src=tcpPkt.src_port,
                                            tcp_dst=tcpPkt.dst_port)
235             # elif tcpPkt.has_flags(tcp.TCP_RST):
236             #     Do Something
237         except IndexError:
238             print "No TCP info detected"
239         except:
240             raise
```

```
241
242         #UDP Packet Processing
243         try:
244             udpPkt = udpPkt_in[0]
245             print "UDP: " + str(ipv4Pkt.src) + ":" + str(udpPkt.src_port) + "
                    -> " + str(ipv4Pkt.dst) + ":" + str(udpPkt.dst_port)
246             self.addActive(map["src_rip"], map["dst_rip"], map["src_vip"], map
                    ["dst_vip"], udpPkt.src_port, udpPkt.dst_port, "UDP")
247             match = parser.OFPMatch(in_port=in_port, eth_type=0x800, ipv4_dst=
                    map["dst_vip"], ipv4_src=map["src_rip"],
248                                     ip_proto=17, udp_src=udpPkt.src_port,
                                            udp_dst=udpPkt.dst_port)
249         except IndexError:
250             print "No UDP info detected"
251         except:
252             raise
253
254         self.logPacket("IPv4", map["src_rip"], map["dst_rip"], map["src_vip"],
                map["dst_vip"])
255
256         # Modify IP address fields of packet
257         actions = [parser.OFPActionSetField(ipv4_dst=map["dst_rip"]), parser.
                OFPActionSetField(ipv4_src=map["src_vip"]),
258                    parser.OFPActionOutput(out_port)]
259
260         # install a flow to avoid the controller having to decide
261         if out_port != ofproto.OFPP_FLOOD:
262             # verify a valid buffer_id, if yes avoid to send both flow_mod &
                    packet_out
263             if msg.buffer_id != ofproto.OFP_NO_BUFFER:
264                 self.add_flow(datapath, 1, match, actions, msg.buffer_id)
265                 return
266             else:
267                 self.add_flow(datapath, 1, match, actions)
268         self.packet_out(msg, ofproto, parser, datapath, in_port, actions)
```

```
269     # </editor-fold>
270
271     def packet_out(self, msg, ofproto, parser, datapath, in_port, actions):
272         data = None
273
274         if msg.buffer_id == ofproto.OFP_NO_BUFFER:
275             data = msg.data
276
277         out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
278                                   in_port=in_port, actions=actions, data=data)
279         datapath.send_msg(out)
280
281     # <editor-fold desc="Event-driven methods">
282     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
283     def _packet_in_handler(self, ev):
284         # If you hit this you might want to increase the "miss_send_length" of
                your switch
285         if ev.msg.msg_len < ev.msg.total_len:
286             self.logger.debug("packet truncated: only %s of %s bytes", ev.msg.
                    msg_len, ev.msg.total_len)
287
288         ''' General Setup '''
289         msg = ev.msg
290         datapath = msg.datapath
291         ofproto = datapath.ofproto
292         parser = datapath.ofproto_parser
293         in_port = msg.match['in_port']
294         dpid = datapath.id
295
296         # Setup the packet
297         pkt = packet.Packet(msg.data)
298
299         # Get the different protocols, if the exist
300         eth = pkt.get_protocols(ethernet.ethernet)[
301             0]  # We know that there will be an ethernet component, so we can
```

```
                get the first element
302             icmpPkt = pkt.get_protocols(icmp.icmp)
303             ipv4Pkt = pkt.get_protocols(ipv4.ipv4)
304             arpPkt = pkt.get_protocols(arp.arp)
305
306             '''Basic Switch Functionality'''
307             if eth.ethertype == ether_types.ETH_TYPE_LLDP:
308                 # ignore lldp packet
309                 return
310             dst = eth.dst
311             src = eth.src
312
313             '''Functionality for mac to port mappings'''
314             # Default behavior for mac to port mappings
315             self.mac_to_port.setdefault(dpid, {})
316
317             # learn a mac address to avoid FLOOD next time.
318             self.mac_to_port[dpid][src] = in_port
319
320             if dst in self.mac_to_port[dpid]:
321                 out_port = self.mac_to_port[dpid][dst]
322             else:
323                 out_port = ofproto.OFPP_FLOOD
324
325             '''Translation Rules'''
326             if arpPkt:
327                 self.arpTranslation(arpPkt, dpid, parser, out_port, ofproto, msg,
                        datapath, in_port)
328             elif icmpPkt:
329                 self.icmpTranslation(ipv4Pkt, icmpPkt, dpid, parser, out_port,
                        ofproto, msg, datapath, in_port)
330             elif ipv4Pkt:
331                 tcpPkt = pkt.get_protocols(tcp.tcp)
332                 udpPkt = pkt.get_protocols(udp.udp)
333                 self.ipv4Translation(ipv4Pkt, tcpPkt, udpPkt, dpid, parser,
```

```
                out_port, ofproto, msg, datapath, in_port)
334         else:
335             actions = [parser.OFPActionOutput(out_port)]
336 
337             # install a flow to avoid packet_in next time
338             if out_port != ofproto.OFPP_FLOOD:
339                 match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
340                 # verify if we have a valid buffer_id, if yes avoid to send
                        both flow_mod & packet_out
341                 if msg.buffer_id != ofproto.OFP_NO_BUFFER:
342                     self.add_flow(datapath, 1, match, actions, msg.buffer_id)
343                     return
344                 else:
345                     self.add_flow(datapath, 1, match, actions)
346             self.packet_out(msg, ofproto, parser, datapath, in_port, actions)
347 
348     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
349     def switch_features_handler(self, ev):
350         datapath = ev.msg.datapath
351         ofproto = datapath.ofproto
352         parser = datapath.ofproto_parser
353 
354         # install table-miss flow entry
355         #
356         # We specify NO BUFFER to max_len of the output action due to
357         # OVS bug. At this moment, if we specify a lesser number, e.g.,
358         # 128, OVS will send Packet-In with invalid buffer_id and
359         # truncated packet data. In that case, we cannot output packets
360         # correctly.  The bug has been fixed in OVS v2.1.0.
361         match = parser.OFPMatch()
362         actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
363                                           ofproto.OFPCML_NO_BUFFER)]
364         self.add_flow(datapath, 0, match, actions)
365 
366     @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
```

```
367     def flow_stats_reply_handler(self, ev):
368         flows = []
369         for stat in ev.msg.body:
370             flows.append('table_id=%s '
371                          'duration_sec=%d duration_nsec=%d '
372                          'priority=%d '
373                          'idle_timeout=%d hard_timeout=%d flags=0x%04x '
374                          'cookie=%d packet_count=%d byte_count=%d '
375                          'match=%s instructions=%s' %
376                          (stat.table_id,
377                           stat.duration_sec, stat.duration_nsec,
378                           stat.priority,
379                           stat.idle_timeout, stat.hard_timeout, stat.flags,
380                           stat.cookie, stat.packet_count, stat.byte_count,
381                           stat.match, stat.instructions))
382         print('FlowStats: %s', flows)
383     # </editor-fold>
384
385     # <editor-fold desc="Logging">
386     def logPacket(self, protocol, src_rip, src_vip, dst_rip, dst_vip):
387         self.logger.info(protocol)
388         self.logger.info('src_RIP: %s, src_VIP: %s', src_rip, src_vip)
389         self.logger.info('dst_RIP: %s, dst_VIP: %s', dst_rip, dst_vip)
390     # </editor-fold>
391
392     def send_flow_stats_request(self, datapath, match):
393         ofp = datapath.ofproto
394         ofp_parser = datapath.ofproto_parser
395
396         cookie = cookie_mask = 0
397         req = ofp_parser.OFPFlowStatsRequest(datapath, 0,
398                                              ofp.OFPTT_ALL,
399                                              ofp.OFPP_ANY, ofp.OFPG_ANY,
400                                              cookie, cookie_mask,
401                                              match)
```

```
402             datapath.send_msg(req)
```

### C.1.2 Mutator.py.

```
1  from random import randint
2  import logging
3  import logging.config
4  from ActiveConnection import ActiveConnection
5  import pprint
6
7  #Set up Logging
8  logger = logging.getLogger('Interval')
9  hdlr = logging.FileHandler('Interval.log')
10 formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s')
11 hdlr.setFormatter(formatter)
12 logger.addHandler(hdlr)
13 logger.setLevel(logging.DEBUG)
14
15 class Mutator:
16
17     def __init__(self, *args):
18         self.logger = logging.getLogger('Interval.Logger')
19         self.logger.info('Creating instance of Logger')
20
21         ## Set up argument parsing
22         # parser = argparse.ArgumentParser()
23         # parser.add_argument('--networks', nargs='*')
24         # args = parser.parse_args()
25         # self.logger.info('Arguments received: %s', args)
26         self.mac_to_port = {}
27         self.RIP_VIP = {}
28         self.VIP_RIP = {}
29         self.actives = []
30
31         # schedule.every(10).seconds.do(self.mutate, networks, False, -1)
32         # schedule.run_continuously()
```

```
33
34      # TODO: Parameterize hardcoded ranges.
35      def mutate(self, networks, first, mutation):
36
37          #Clear translation tables for next mutation
38          self.RIP_VIP = {}
39          self.VIP_RIP = {}
40
41          self.logger.info("Calculating mutation " + str(mutation))
42
43          print("Calculating mutation " + str(mutation))
44          for net in networks:
45
46              exclude = set()  # Exclude members of set from inactive mutations
47              self.RIP_VIP["10.13.1.1"] = "10.13.1.1"
48              self.VIP_RIP["10.13.1.1"] = "10.13.1.1"
49              self.RIP_VIP["10.13.1.2"] = "10.13.1.2"
50              self.VIP_RIP["10.13.1.2"] = "10.13.1.2"
51              self.RIP_VIP["10.13.1.255"] = "10.13.1.255"
52              self.VIP_RIP["10.13.1.255"] = "10.13.1.255"
53
54              # Mutate Active connections
55              for conn in self.actives:
56                  VIP = self.generateVIP(net, 11, 30)
57                  results = ActiveConnection.find_by_rDst(conn.rDst)
58                  for entry in results:
59                      # May repeatedly assign RIP:VIP mapping if multiple active
                            conns.
60                      entry.vDst = VIP
61                      self.updateRIPVIP(entry.rDst, VIP)
62
63                  exclude.add(conn.rSrc)  # Add to ensure active addresses aren'
                        t mutated 2x.
64
65              # Mutate inactive connections
```

```
66              for address in range(3, 10):
67                  # VIP=RIP for first run to make testing easier.
68                  # TODO: Remove from final implementation.
69                  RIP = net + str(address)
70                  if(first):
71                      if (net + str(address) in exclude):
72                          continue
73                      else:
74                          VIP = (net + str(address))
75                          self.RIP_VIP[net + str(address)] = VIP
76                          self.VIP_RIP[VIP] = net + str(address)
77                  else:
78                      if (net + str(address) in exclude):
79                          # Ignore active conns; they're calculated above.
80                          continue
81                      else:
82                          VIP = self.generateVIP(net, 31, 50)
83                          self.updateRIPVIP(RIP, VIP)
84
85          self.logger.info("RIP:VIP mappings:\n\n\tRIP\t\t:\t\tVIP\n"+pprint.
                pformat(self.RIP_VIP, indent=1, width=100)+"\n")
86          self.logger.info(self.printActives())
87          self.logger.info("Mutations calculated for networks")
88          mutationResults = [self.RIP_VIP, self.VIP_RIP, self.actives, self.
                mac_to_port]
89          print("Done")
90          return mutationResults
91
92      # Check if candidate VIP is in use
93      def VIP_used(self, VIP):
94          if VIP in self.RIP_VIP or VIP in self.VIP_RIP:
95              return True
96          else:
97              return False
98
```

```
99      # Generate VIP from address pool of range poolStart-poolEnd
100     def generateVIP(self, net, poolStart, poolEnd):
101         VIP = net + str(randint(poolStart, poolEnd))
102         while (self.VIP_used(VIP)):
103             VIP = net + str(randint(poolStart, poolEnd))
104         return VIP
105
106     # Update mutation mappings
107     def updateRIPVIP(self,RIP,VIP):
108         try:
109             self.RIP_VIP[RIP] = VIP
110             self.VIP_RIP[VIP] = RIP
111         except:
112             self.logger.fatal("Unable to update RIP:VIP mapping")
113             raise
114
115     # Get actives table as string.
116     def printActives(self):
117         activesString = "\n\nActives Table after mutation:\nIndex \tReal Src \
                tVirtual Src \tReal Dst \tPerceived Dst \tVirtual Dst \tSrc Port \
                tDst Port \tProtocol\n"
118         i = 1
119         for conn in self.actives:
120             activesString += (str(i) + "\t" + str(conn) + "\n")
121             i = i + 1
122         return activesString
```

### C.1.3 ActiveConnection.py.

```
1  from collections import defaultdict
2
3
4  class ActiveConnection:
5      # Pre-index attributes for later lookup
6      rSrc_index = defaultdict(list)
7      vSrc_index = defaultdict(list)
```

```
8       rDst_index = defaultdict(list)
9       pDst_index = defaultdict(list)
10      vDst_index = defaultdict(list)
11      protocol_index = defaultdict(list)
12
13      def __init__(self, rSrc, vSrc, rDst, pDst, vDst, sPort, dPort, protocol):
14          self.rSrc = rSrc  # Real source IP address
15          self.vSrc = vSrc  # Virtual source IP address
16          self.rDst = rDst  # Real destination IP address
17          self.pDst = pDst  # Perceived destination IP address of the connection
                b/t rSrc & rDst
18          self.vDst = vDst  # Current virtual IP address for new connections
19          self.sPort = sPort  # Source port
20          self.dPort = dPort  # Destination port
21          self.protocol = protocol  # Protocol in use (i.e. TCP or UDP)
22          self.fin = 0  # Number of FINs received in connection's lifetime (TCP
               only)
23
24          # Update indices
25          ActiveConnection.rSrc_index[rSrc].append(self)
26          ActiveConnection.vSrc_index[vSrc].append(self)
27          ActiveConnection.rDst_index[rDst].append(self)
28          ActiveConnection.pDst_index[pDst].append(self)
29          ActiveConnection.vDst_index[vDst].append(self)
30          ActiveConnection.protocol_index[protocol].append(self)
31
32      def __str__(self):
33          return (self.rSrc + "\t" + self.vSrc + "\t" + self.rDst + "\t " + self
                .pDst + "\t" + self.vDst
34                  + "\t" + str(self.sPort) + "\t" + str(self.dPort) + "\t" +
                        self.protocol + "\t" + str(self.fin))
35
36      # Search functions for attributes
37      @classmethod
38      def find_by_rSrc(cls, rSrc):
```

```
39          return ActiveConnection.rSrc_index[rSrc]
40
41      @classmethod
42      def find_by_rSrc_pDst(cls, rSrc, pDst):
43          connections = ActiveConnection.rSrc_index[rSrc]
44          # This loop MUST never return more than one item
45          for conn in connections:
46              print "Looking for"
47              print str(conn)
48              if conn.pDst == pDst:
49                  print "Found it"
50                  return conn
51              else:
52                  print ("Connection rSrc %s with pDst %s not found", rSrc, pDst
                        )
53                  return
54      @classmethod
55      def find_by_rSrc_port(cls, rSrc, sPort, dPort):
56          connections = ActiveConnection.rSrc_index[rSrc]
57          # This loop MUST never return more than one item
58          for conn in connections:
59              if conn.sPort == sPort and conn.dPort == dPort:
60                  return conn
61              else:
62                  print ("Connection rSrc %s with sPort %d and dPort %d not
                        found", rSrc, sPort, dPort)
63                  return
64
65      @classmethod
66      def find_by_vSrc(cls, vSrc):
67          return ActiveConnection.vSrc_index[vSrc]
68
69      @classmethod
70      def find_by_rDst(cls, rDst):
71          return ActiveConnection.rDst_index[rDst]
```

```
72
73      @classmethod
74      def find_by_pDst(cls, pDst):
75          return ActiveConnection.pDst_index[pDst]
76
77      @classmethod
78      def find_by_vDst(cls, vDst):
79          return ActiveConnection.vDst_index[vDst]
80
81      @classmethod
82      def find_by_protocol(cls, protocol):
83          return ActiveConnection.protocol_index[protocol]
```

### C.1.4 params.conf.

```
1  #Parameters for PlebeNet testbed
2
3  [DEFAULT]
4
5  frequency = 30
6  networks = 10.13.1.
7  timeout = 240
8  #networks = 10.13.1.,10.13.2.,10.13.37.,10.13.3.
9  #networks = 10.13.1.,10.13.2.,10.13.37.
10 #networks = 10.13.37.
```

## C.2 Adversary Scripts

This section contains the scripts used by the Kali adversary to scan and exploit machines on the network.

```
1  <ruby>
2
3  def time_diff(start_time, end_time)
4   seconds_diff = (start_time - end_time).to_i.abs
5   hours = seconds_diff/3600
6   seconds_diff -= hours * 3600
```

```
 7   minutes = seconds_diff / 60
 8   seconds_diff -= minutes * 60
 9   seconds = seconds_diff
10
11   puts "#{hours.to_s.rjust(2, '0')}:#{minutes.to_s.rjust(2, '0')}:#{seconds.
         to_s.rjust(2, '0')}"
12 end
13
14 #Start Time
15 start_time = Time.now
16
17 #Nmap Scan
18 run_single("db_nmap --min-hostgroup 96 -T4 -A -v -n 10.13.1.0/24")
19
20 endTime = Time.now
21
22 puts "Start Time: " + startTime.inspect
23 puts "End Time: " + endTime.inspect
24 puts "Total Time: "
25 time_diff(startTime, endTime)
```

```
 1 <ruby>
 2
 3 def time_diff(start_time, end_time)
 4   seconds_diff = (start_time - end_time).to_i.abs
 5   hours = seconds_diff/3600
 6   seconds_diff -= hours * 3600
 7   minutes = seconds_diff / 60
 8   seconds_diff -= minutes * 60
 9   seconds = seconds_diff
10
11   puts "#{hours.to_s.rjust(2, '0')}:#{minutes.to_s.rjust(2, '0')}:#{seconds.
         to_s.rjust(2, '0')}"
12 end
13
14 #Start Time
```

```
15 start_time = Time.now
16
17 #Nmap Scan
18 run_single("db_nmap --min-hostgroup 96 -T4 -n -F 10.13.1.0/24")
19
20 endTime = Time.now
21
22 puts "Start Time: " + startTime.inspect
23 puts "End Time: " + endTime.inspect
24 puts "Total Time: "
25 time_diff(startTime, endTime)
```

```
1 <ruby>
2
3 def time_diff(start_time, end_time)
4  seconds_diff = (start_time - end_time).to_i.abs
5  hours = seconds_diff/3600
6  seconds_diff -= hours * 3600
7  minutes = seconds_diff / 60
8  seconds_diff -= minutes * 60
9  seconds = seconds_diff
10
11  puts "#{hours.to_s.rjust(2, '0')}:#{minutes.to_s.rjust(2, '0')}:#{seconds.
       to_s.rjust(2, '0')}"
12 end
13
14 #Start Timer
15
16 run_single("use exploit/windows/smb/ms08_067_netapi")
17 run_single("set PAYLOAD windows/meterpreter/bind_tcp")
18 run_single("set LHOST 10.13.2.5")
19 run_single("set RPORT 445")
20
21 puts "Enter RHOST:"
22 target = gets
23 attack = "set RHOST " +target
```

```
24 puts attack
25 run_single(attack)
26
27 run_single("set SMBPIPE BROWSER")
28 startTime = Time.now
29 run_single("exploit")
30 run_single("exit")
31 endTime = Time.now
32
33 puts "Start Time: " + startTime.inspect
34 puts "End Time: " + endTime.inspect
35 puts "Total Time: "
36 time_diff(startTime, endTime)
```

## C.3 Legitimate User Scripts

### C.3.1 SSH.

Send_ssh.sh contains the script open, maintain, and close a SSH connection to a specified target.

```
1 #!/bin/bash
2 for i in `seq 1 20`;
3 do
4  date
5  ls
6  sleep 6
7 done
```

### C.3.2 IMAP.

imap_script.sh contains the script to retrieve content from the IMAP server.

```
1 #!/bin/sh
2
3 HOST=$1
4 (
```

```
5  echo open "$HOST 143"
6  sleep 1
7  echo "? LOGIN starbuck@sdn.local Password!123"
8  sleep 1
9  echo "? LIST INBOX *"
10 sleep 10
11 echo "? SELECT INBOX"
12 sleep 10
13 echo "? LIST INBOX *"
14 sleep 10
15 echo "? SELECT INBOX"
16 sleep 10
17 echo "? LIST INBOX *"
18 sleep 10
19 echo "? SELECT INBOX"
20 sleep 10
21 echo "? LIST INBOX *"
22 sleep 10
23 echo "? SELECT INBOX"
24 sleep 10
25 echo "? LIST INBOX *"
26 sleep 10
27 echo "? SELECT INBOX"
28 sleep 10
29 echo "? LIST INBOX *"
30 sleep 10
31 echo "? SELECT INBOX"
32 sleep 10
33 echo "? LOGOUT"
34 sleep 1
35 echo "exit"
36 ) | telnet
```

### C.3.3 POP.

pop_script.sh contains the script to retrieve content from the POP server.

```
1  #!/bin/sh
2
3  HOST=$1
4  (
5  echo open "$HOST 110"
6  sleep 20
7  echo "USER starbuck@sdn.local"
8  sleep 20
9  echo "PASS Password!123"
10 sleep 20
11 echo "STAT"
12 sleep 20
13 echo "LIST"
14 sleep 20
15 echo "RETR 1"
16 sleep 20
17 echo "QUIT"
18 ) | telnet
```

### C.3.4 SMTP.

smtp_script.sh contains the script to retrieve content from the SMTP server.

```
1  #!/bin/sh
2
3  HOST=$1
4  (
5  echo open "$HOST 25"
6  sleep 1
7  echo "EHLO $HOST"
8  sleep 1
9  echo "AUTH LOGIN"
10 sleep 1
```

```
11 echo "YXBvbGxvQHNkbi5sb2NhbA=="
12 sleep 1
13 echo "UGFzc3dvcmQhMTIz"
14 sleep 1
15 echo "MAIL FROM: <apollo@sdn.local>"
16 sleep 5
17 echo "RCPT TO: <starbuck@sdn.local>"
18 sleep 20
19 echo "DATA"
20 sleep 10
21 echo "FROM: apollo@sdn.local"
22 sleep 20
23 echo "TO: starbuck@sdn.local"
24 sleep 10
25 echo "SUBJECT: Message title"
26 sleep 20
27 echo "This is the message.\r\n"
28 sleep 10
29 echo "."
30 sleep 20
31 echo "QUIT"
32 ) | telnet
```

### C.3.5 HTTP.

http_script.sh contains the script to retrieve content from the HTTP server.

```
1 #!/bin/bash
2
3
4 curl -O $2/Data50.txt --limit-rate 785k
5 sha1sum Data50.txt > Hash_$1.txt
6 rm Data50.txt
```

### C.3.6 RTP.

message.txt contains the contents of a single packet that was continuously transmitted during RTP trials.

```
1 This is the captain. We have a little problem with our entry sequence, so we
      may experience some slight turbulence and then - explode.
```

### C.3.7 Data Collection Scripts.

Capture_Svcs.ps1 allows for semi-automated collection of network performance data from the servers on the network.

```
1 #Generates .pcapng files on a loop with a pause for user input to begin each
      loop.
2 #PARAMS
3 #Duration - length of packet capture in seconds
4 #Trials - how many files to generate
5 #Interface - Interface to listen on
6 #SVC_Name - name of VM to run packet captures on
7 #Protocol - What protocol is being assessed
8 #Control - Is this a control or not?
9 #Must be logged in to vCenter Server. Executed from PowerCLI.
10 param(
11   [parameter(Mandatory=$false)][int]$Duration = 60,
12   [parameter(Mandatory=$true)][int]$Trials = 1,
13   [parameter(Mandatory=$true)][string]$Interface,
14   [parameter(Mandatory=$true)][string]$SVC_Name = "",
15   [parameter(Mandatory=$true)][string]$Protocol = "",
16   [parameter(Mandatory=$true)][switch]$Control
17   )
18
19 #Password!122 for Overlord
20 #Password!123 for Others
21 $LocalUser = "administrator"
22 $LocalPWord = ConvertTo-SecureString -String "Password!123" -AsPlainText -
      Force
```

```
23 $LocalCredential = New-Object -TypeName System.Management.Automation.
       PSCredential -ArgumentList $LocalUser, $LocalPWord
24
25 For($i = 1; $i -le $Trials; $i++)
26 {
27  Write-Verbose -Message "Getting ready to start capture on $SVC_Name" -Verbose
28
29  if($Control)
30  {
31   $Script = "tshark.exe -i $Interface -a duration:$Duration -w C:\Users\
        Administrator\Captures\$Protocol-C-$i.pcapng"
32  }
33  else
34  {
35   $Script = "tshark.exe -i $Interface -a duration:$Duration -w C:\Users\
        Administrator\Captures\$Protocol-M-$i.pcapng"
36  }
37
38  Write-Host "Starting script for $SVC_Name..."
39  Invoke-VMScript -VM $SVC_Name -GuestCredential $LocalCredential -ScriptType
       bat -ScriptText $Script
40  Write-Host "Script completed."
41
42  $remaining = $Trials - $i
43
44  Read-Host -Prompt "$remaining Trials remain. Press <enter> to continue."
45 }
```

Capture_Kali.ps1 allows for semi-automated collection of network performance data from a simulated legitimate user on the network.

```
1 #Generates .pcapng files on a loop with a pause for user input to begin each
      loop.
2 #PARAMS
3 #Duration - length of packet capture in seconds
4 #Trials - how many files to generate
```

```
5  #Interface - Interface to listen on
6  #SVC_Name - name of VM to run packet captures on
7  #Protocol - What protocol is being assessed
8  #Control - Is this a control or not?
9  #Must be logged in to vCenter Server. Executed from PowerCLI.
10 param(
11  [parameter(Mandatory=$false)][int]$Duration = 150,
12  [parameter(Mandatory=$true)][int]$Trials = 1,
13  [parameter(Mandatory=$true)][string]$Interface,
14  [parameter(Mandatory=$true)][string]$SVC_Name = "",
15  [parameter(Mandatory=$true)][string]$Protocol = "",
16  [parameter(Mandatory=$true)][switch]$Control
17  )
18
19 $LocalUser = "root"
20 $LocalPWord = ConvertTo-SecureString -String "toor" -AsPlainText -Force
21 $LocalCredential = New-Object -TypeName System.Management.Automation.
       PSCredential -ArgumentList $LocalUser, $LocalPWord
22
23 For($i = 1; $i -le $Trials; $i++)
24 {
25  $target = Read-Host -Prompt "Enter IP of target machine..."
26  Write-Verbose -Message "Getting ready to start capture on $SVC_Name" -Verbose
27
28  if($Control)
29  {
30   switch($Protocol)
31   {
32    "IMAP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
          Captures/imap_captures/IMAP_C/$Protocol-C-$i-Kali.pcapng & sleep 5; /
          root/Captures/imap_captures/imap_script.sh $target"}
33    "POP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
          Captures/pop_captures/POP_C/$Protocol-C-$i-Kali.pcapng & sleep 5; /root
          /Captures/pop_captures/pop_script.sh $target"}
34    "SMTP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
```

```
        Captures/smtp_captures/SMTP_C/$Protocol-C-$i-Kali.pcapng & sleep 5; /
        root/Captures/smtp_captures/smtp_script.sh $target"}
35      "HTTP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
        Captures/http_captures/HTTP_C/$Protocol-C-$i-Kali.pcapng & sleep 5; /
        root/Captures/http_captures/http_script.sh $i"}
36
37     }
38    }
39    else
40    {
41     switch($Protocol)
42     {
43      "IMAP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
        Captures/imap_captures/IMAP_M/$Protocol-M-$i-Kali.pcapng & sleep 5; /
        root/Captures/imap_captures/imap_script.sh $target"}
44      "POP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
        Captures/pop_captures/POP_M/$Protocol-M-$i-Kali.pcapng & sleep 5; /root
        /Captures/pop_captures/pop_script.sh $target"}
45      "SMTP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
        Captures/smtp_captures/SMTP_M/$Protocol-M-$i-Kali.pcapng & sleep 5; /
        root/Captures/smtp_captures/smtp_script.sh $target"}
46      "HTTP"{$Script = "sudo tshark -i $Interface -a duration:$Duration -w /root/
        Captures/http_captures/HTTP_M/$Protocol-M-$i-Kali.pcapng & sleep 5; /
        root/Captures/http_captures/http_script.sh $i"}
47
48     }
49    }
50
51    Write-Host "Starting script for $SVC_Name trial $i"
52    Invoke-VMScript -VM $SVC_Name -GuestCredential $LocalCredential -ScriptType
        bash -ScriptText $Script
53    Write-Host "Script completed."
54
55    $remaining = $Trials - $i
56
```

```
57    Read-Host -Prompt "$remaining Trials remain. Press <enter> to continue."
58  }
```

# Appendix D. Data Processing Scripts

## D.1 Stream Isolation

Filter_Packet_Stream.ps1 isolates the TCP or UDP stream relevant to the protocol under test.

```
1  #Will create a new file following a supplied naming scheme (with hardcoded
       name manipulation)
2  #tshark filter isolated the first tcp stream in the .pcapng file.
3  #Currently only filters into Control!
4
5  # Character offsets for use in Remove() and Insert()
6  # HTTP_C_Receiver_#.pcapng 16,17
7  # IMAP_C_Receiver_#.pcapng 16,17
8  # SMTP_C_Receiver_#.pcapng 16,17
9  # RTP_C_Receiver_#.pcapng 15,16
10 # POP_C_Receiver_#.pcapng 15,16
11 # SSH_C_Receiver_#.pcapng 15,16
12 # HTTP_C_Sender_#.pcapng 14,15
13 # IMAP_C_Sender_#.pcapng 14,15
14 # SMTP_C_Sender_#.pcapng 14,15
15 # RTP_C_Sender_#.pcapng 13,14
16 # POP_C_Sender_#.pcapng 13,14
17 # SSH_C_Sender_#.pcapng 13,14
18
19 param(
20  [parameter(Mandatory=$true)][switch]$Control
21 )
22
23 if($Control) {
24  $path = "Control"
25  $Type = "C"
26 }
27 else {
28  $Path = "Mutator"
29  $Type = "M"
```

```
30  }
31
32  $Protocol = "HTTP", "IMAP", "SMTP", "RTP", "POP", "SSH"
33  $FileTypes = "HTTP_@_Receiver_#OLD.pcapng","IMAP_@_Receiver_#OLD.pcapng","
        SMTP_@_Receiver_#OLD.pcapng","RTP_@_Receiver_#OLD.pcapng","POP_@_Receiver_
        #OLD.pcapng","SSH_@_Receiver_#OLD.pcapng","HTTP_@_Sender_#OLD.pcapng","
        IMAP_@_Sender_#OLD.pcapng","SMTP_@_Sender_#OLD.pcapng","RTP_@_Sender_#OLD.
        pcapng","POP_@_Sender_#OLD.pcapng","SSH_@_Sender_#OLD.pcapng"
34
35  $offset1 = 17
36  $offset2 = 18
37
38  For($j = 0; $j -le 11; $j++)
39  {
40   $p = $j % 6
41   if($j % 3 -eq 0)
42   {
43    $offset1--
44    $offset2--
45   }
46
47   For($i = 1; $i -le 10; $i++)
48   {
49    $FileTypes[$j] = $FileTypes[$j].Remove($offset1).Insert($offset1,"$i")+"OLD.
          pcapng"
50
51    $sb = new-object system.text.stringbuilder
52    $sb.append($FileTypes[$j].split('_')[0])
53    # Since we use _@_ to cut on, we put it back with the proper type (C or M)
54    $sb.append("_$Type") | out-null
55
56    # Grab the second half of the original text [index 1]
57    $s2 = $FileTypes[$j].split('@')[1]
58    # the .ToCharArray() method of a string breaks the string into individual
          characters
```

```
59    # there's a bit more to char vs string; but, that's unnecessary information.
60    foreach($c in $s2.ToCharArray()) {
61
62      # Add each character to the string builder 1 at a time, followed by a
          period
63      $sb.append($c.ToString()) | out-null
64    }
65
66    # finally, spit the whole thing back out as a string object (think in
        objects)
67    $NewFile = $sb.ToString()
68
69    if($i -ge 10){$offset1++}
70    $NewFile = $NewFile.Remove($offset1 + 1)+".pcapng"
71    Write-Host $FileTypes[$j]
72    Write-Host $NewFile
73    if($Protocol[$p] -eq "RTP")
74    {
75     tshark -r "C:\Users\smayer.CDN\Documents\PCAPS\Control\$($Protocol[$p])\$(
          $FileTypes[$j])" -2 -R "udp.stream eq 0" -w "C:\Users\smayer.CDN\
          Documents\PCAPS\Control\$($Protocol[$p])\$NewFile"
76    }
77    else
78    {
79     tshark -r "C:\Users\smayer.CDN\Documents\PCAPS\Control\$($Protocol[$p])\$(
          $FileTypes[$j])" -2 -R "tcp.stream eq 0" -w "C:\Users\smayer.CDN\
          Documents\PCAPS\Control\$($Protocol[$p])\$NewFile"
80    }
81    if($i -ge 10){$offset1--}
82   }
83  }
```

### D.2 Data Extraction

GatherQoSData.ps1 takes a filtered TCP or UDP stream as input and produces a .csv with information about the transmission for later analysis.

```
1  #1. Apply filter to get latency and RTT
2  #tshark -r .\[file].pcapng -T fields -e frame.number -e ip.src -e ip.dst -e
       tcp.time_delta -e tcp.analysis.ack_rtt -E header=y > [file]_results.csv
3  #2. capinfos on sender and receiver to get byte rates and total packets
4  #capinfos [file].pcapng > [file]_info.txt
5  #OUTPUT: QoS Data in a tab-separated csv
6  #HTTP and RTP Commented for analysis later.
7
8  param(
9   [parameter(Mandatory=$true)][switch]$Control,
10  [parameter(Mandatory=$true)][string]$Protocol
11 )
12
13 if($Control) {
14  $path = "Control"
15  $Type = "C"
16 }
17 else {
18  $Path = "Mutator"
19  $Type = "M"
20 }
21
22 $TCP = $True
23
24 switch($Protocol)
25 {
26  "FTP"{$TCP = $True}
27  "HTTP"{$TCP = $True}
28  "IMAP"{$TCP = $True}
29  "POP"{$TCP = $True}
30  "RTP"{$TCP = $False}
```

```
31   "SMTP"{$TCP = $True}
32   "SSH"{$TCP = $True}
33  }
34
35  function gatherTCPData ($path, $protocol, $type, $i)
36  {
37   tshark -r C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Receiver\
         Filtered\$protocol`_$type`_Receiver_$i`.pcapng -T fields -e frame.number
         -e ip.src -e ip.dst -e tcp.time_delta -e tcp.analysis.ack_rtt -e tcp.
         analysis.retransmission -e tcp.analysis.fast_retransmission -e tcp.
         analysis.ack_lost_segment -e tcp.analysis.out_of_order -e tcp.analysis.
         spurious_retransmission -e tcp.analysis.duplicate_ack -e tcp.analysis.
         window_update -e tcp.analysis.window_full -E header=y > C:\Users\smayer.
         CDN\Documents\GitHub\Mayer_Thesis\Experiments\Results\$protocol`\$path`\
         $protocol`_$type`_Receiver_$i`_results.csv
38   tshark -r C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Sender\
         Filtered\$protocol`_$type`_Sender_$i`.pcapng -T fields -e frame.number -e
          ip.src -e ip.dst -e tcp.time_delta -e tcp.analysis.ack_rtt -e tcp.
         analysis.retransmission -e tcp.analysis.fast_retransmission -e tcp.
         analysis.ack_lost_segment -e tcp.analysis.out_of_order -e tcp.analysis.
         spurious_retransmission -e tcp.analysis.duplicate_ack -e tcp.analysis.
         window_update -e tcp.analysis.window_full -E header=y > C:\Users\smayer.
         CDN\Documents\GitHub\Mayer_Thesis\Experiments\Results\$protocol`\$path`\
         $protocol`_$type`_Sender_$i`_results.csv
39   capinfos C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Receiver\
         Filtered\$protocol`_$type`_Receiver_$i`.pcapng | Select-String "File name
         :","Number of packets:","Capture duration:","Data bit rate:" | Add-
         Content C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Experiments\
         Results\$protocol`\$path`\$protocol`_$type`_$i`_Info.txt
40   capinfos C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Sender\
         Filtered\$protocol`_$type`_Sender_$i`.pcapng | Select-String "File name
         :","Number of packets:","Capture duration:","Data bit rate:" | Add-
         Content C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Experiments\
         Results\$protocol`\$path`\$protocol`_$type`_$i`_Info.txt
41
```

```
42   Write-Host $protocol $type $i complete.
43  }
44
45  function gatherUDPData ($path, $protocol, $type, $i)
46  {
47   tshark -r C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Receiver\
         Filtered\$protocol`_$type`_Receiver_$i`.pcapng -T fields -e frame.number
         -e ip.src -e ip.dst -e tcp.time_delta -e tcp.analysis.ack_rtt -e tcp.
         analysis.retransmission -e tcp.analysis.fast_retransmission -e tcp.
         analysis.ack_lost_segment -e tcp.analysis.out_of_order -e tcp.analysis.
         spurious_retransmission -e tcp.analysis.duplicate_ack -e tcp.analysis.
         window_update -e tcp.analysis.window_full -E header=y > C:\Users\smayer.
        CDN\Documents\GitHub\Mayer_Thesis\Experiments\Results\$protocol`\$path`\
         $protocol`_$type`_Receiver_$i`_results.csv
48   tshark -r C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Sender\
         Filtered\$protocol`_$type`_Sender_$i`.pcapng -T fields -e frame.number -e
          ip.src -e ip.dst -e tcp.time_delta -e tcp.analysis.ack_rtt -e tcp.
         analysis.retransmission -e tcp.analysis.fast_retransmission -e tcp.
         analysis.ack_lost_segment -e tcp.analysis.out_of_order -e tcp.analysis.
         spurious_retransmission -e tcp.analysis.duplicate_ack -e tcp.analysis.
         window_update -e tcp.analysis.window_full -E header=y > C:\Users\smayer.
        CDN\Documents\GitHub\Mayer_Thesis\Experiments\Results\$protocol`\$path`\
         $protocol`_$type`_Sender_$i`_results.csv
49   capinfos C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Receiver\
         Filtered\$protocol`_$type`_Receiver_$i`.pcapng | Select-String "File name
         :","Number of packets:","Capture duration:","Data bit rate:" | Add-
         Content C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Experiments\
         Results\$protocol`\$path`\$protocol`_$type`_$i`_Info.txt
50   capinfos C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Sender\
         Filtered\$protocol`_$type`_Sender_$i`.pcapng | Select-String "File name
         :","Number of packets:","Capture duration:","Data bit rate:" | Add-
         Content C:\Users\smayer.CDN\Documents\GitHub\Mayer_Thesis\Experiments\
         Results\$protocol`\$path`\$protocol`_$type`_$i`_Info.txt
51
52   Write-Host $protocol $type $i complete.
```

```
53   Write-Host C:\Users\smayer.CDN\Documents\PCAPS\$path`\$protocol`\Receiver\
         Filtered\$protocol`_$type`_Receiver_$i`.pcapng
54 }
55
56 For($iter = 1; $iter -le 10; $iter++)
57 {
58  if($TCP){gatherTCPData $Path $Protocol $Type $iter}
59  else{gatherUDPData $Path $Protocol $Type $iter}
60 }
```

### D.3 Data Aggregation

ReadResults.R takes the output of GatherQoSData.ps1 and calculates averages for each measured piece of information. Results are then stored in another .csv for statistical analysis.

```
1 require(tidyverse)
2 require(readxl)
3 require(stringr)
4
5 ctrldirs <- c("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/
      Results/FTP/Control","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
      Experiments/Results/HTTP/Control","C:/Users/smayer.CDN/Documents/GitHub/
      Mayer_Thesis/Experiments/Results/IMAP/Control","C:/Users/smayer.CDN/
      Documents/GitHub/Mayer_Thesis/Experiments/Results/POP/Control","C:/Users/
      smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/RTP/Control
      ","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/
      SMTP/Control","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
      Experiments/Results/SSH/Control")
6 mutatedirs <- c("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments
      /Results/FTP/Mutator","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
      Experiments/Results/HTTP/Mutator","C:/Users/smayer.CDN/Documents/GitHub/
      Mayer_Thesis/Experiments/Results/IMAP/Mutator","C:/Users/smayer.CDN/
      Documents/GitHub/Mayer_Thesis/Experiments/Results/POP/Mutator","C:/Users/
      smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/RTP/Mutator
```

```
    ","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/
    SMTP/Mutator","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
    Experiments/Results/SSH/Mutator")
protocol <- c("FTP","HTTP","IMAP","POP","RTP","SMTP","SSH")
#RIPs of sender used for each trial. 1:1 mapping with protocol vector
sender_rip <- c
    ("10.13.1.8","10.13.1.8","10.13.1.8","10.13.1.8","10.13.1.8","10.13.1.8","10.13.1.8")


#Generic filename format
udp_str <- "AAA_X_Jitter.csv"
sender_str <- "AAA_X_Sender_#_results.csv"
receiver_str <- "AAA_X_Receiver_#_results.csv"
info <- "AAA_X#_Info.txt"

#Tibble for intermediate results _S = Sender, _R = Receiver
TCP_Avgs <- tibble(latency=0, RTT=0, duration=0, BPSSender=0, BPSReceiver=0,
    PktsSender=0, PktsReceiver=0, PktsDrop = (PktsSender-PktsReceiver),
    RetransR = 0, RetransS = 0, FastRetransR = 0, FastRetransS = 0, ACKlostR =
     0, ACKlostS = 0, OutOfOrderR = 0, OutOfOrderS = 0, SRetransR = 0,
    SRetransS = 0, DupACKR = 0, DupACKS = 0, WinUpdateR = 0, WinUpdateS = 0,
    WinFullR = 0, WinFullS = 0)

UDP_Avgs <- tibble(MaxjitterS=0, MaxjitterR=0, jitterS=0, jitterR=0, duration
    =0, BPSSender=0, BPSReceiver=0, PktsSender=0, PktsReceiver=0, PktsDrop = (
    PktsSender-PktsReceiver))

#Pull data from files to create tibbles for TCP data
parseTCP <- function(TCP_Avgs, sender_str, receiver_str, info, sender_rip)
{
  #Import files
   Sender <- as_tibble(read.csv(file=sender_str, header=TRUE, na.strings = (c
       ("", "NA")), sep='\t', fileEncoding = "UTF-16LE"))
   Receiver <- as_tibble(read.csv(file=receiver_str, header=TRUE, na.strings =
       (c("", "NA")), sep='\t', fileEncoding = "UTF-16LE"))
```

```
28  print(nrow(Receiver))
29  #Isolate latency data
30  Sender_Latency <- tibble (ip.dst = Sender$ip.dst, delta = Sender$tcp.
        time_delta) %>% filter(ip.dst == sender_rip)
31  Sender_Latency$delta[Sender_Latency$delta >3.5] = NA #Ignore pauses due to
        test script
32  Latency_mean <- round(mean(Sender_Latency$delta, na.rm = TRUE), 5)
33
34  #Isolate RTT data
35  Sender_RTT <- tibble (frame = Sender$frame.number, RTT = Sender$tcp.analysis
        .ack_rtt) %>% drop_na(RTT)
36  RTT_mean <- round(mean(Sender_RTT$RTT), 5)
37
38  #Isolate pkts, duration, and bps from text files
39  info <- readLines(info)
40  pkt_rec <- as.numeric(word(info, start = 6)[2])
41  pkt_send <- as.numeric(word(info, start = 6)[6])
42  dur <- round(as.numeric(word(info, start = 6)[3]), 3)
43  bps_r <- as.numeric(word(info, start = 10)[4])
44  bps_s <- as.numeric(word(info, start = 10)[8])
45
46  print(sender_str)
47  #Gather TCP Issue Data (Sender)
48  RT_S = sum(Sender$tcp.analysis.retransmission, na.rm = TRUE)
49  FR_S = sum(Sender$tcp.analysis.fast_retransmission, na.rm = TRUE)
50  ACKl_S = sum(Sender$tcp.analysis.ack_lost_segment, na.rm = TRUE)
51  OO_S = sum(Sender$tcp.analysis.out_of_order, na.rm = TRUE)
52  SR_S = sum(Sender$tcp.analysis.spurious_retransmission, na.rm = TRUE)
53  DA_S = sum(Sender$tcp.analysis.duplicate_ack, na.rm = TRUE)
54  WU_S = sum(Sender$tcp.analysis.window_update, na.rm = TRUE)
55  WF_S = sum(Sender$tcp.analysis.window_full, na.rm = TRUE)
56
57  print(receiver_str)
58  #Gather TCP Issue Data (Receiver)
59  RT_R = sum(Receiver$tcp.analysis.retransmission, na.rm = TRUE)
```

```
60    FR_R = sum(Receiver$tcp.analysis.fast_retransmission, na.rm = TRUE)
61    ACKl_R = sum(Receiver$tcp.analysis.ack_lost_segment, na.rm = TRUE)
62    OO_R = sum(Receiver$tcp.analysis.out_of_order, na.rm = TRUE)
63    SR_R = sum(Receiver$tcp.analysis.spurious_retransmission, na.rm = TRUE)
64    DA_R = sum(Receiver$tcp.analysis.duplicate_ack, na.rm = TRUE)
65    WU_R = sum(Receiver$tcp.analysis.window_update, na.rm = TRUE)
66    WF_R = sum(Receiver$tcp.analysis.window_full, na.rm = TRUE)
67
68    add_row(TCP_Avgs, latency = Latency_mean, RTT = RTT_mean, duration = dur,
          BPSSender = bps_s, BPSReceiver = bps_r, PktsSender = pkt_send,
          PktsReceiver = pkt_rec, PktsDrop=(pkt_send-pkt_rec), RetransR = RT_R,
          RetransS = RT_S, FastRetransR = FR_R, FastRetransS = FR_S, ACKlostR =
          ACKl_R, ACKlostS = ACKl_S, OutOfOrderR = OO_R, OutOfOrderS = OO_S,
          SRetransR = SR_R, SRetransS = SR_S, DupACKR = DA_R, DupACKS = DA_S,
          WinUpdateR = WU_R, WinUpdateS = WU_S, WinFullR = WF_R, WinFullS = WF_S)
69  }
70
71  #Pull data from files to create tibbles for TCP data
72  parseUDP <- function(UDP_Avgs, udp_str, info, sender_rip)
73  {
74    #Import files (Expects UTF-8)
75    Jitter_Data <- as_tibble(read.csv(file=udp_str, header=TRUE, na.strings = (c
          ("", "NA")), sep=',', fileEncoding = "UTF-8"))
76    #Isolate Jitter data
77    mjs <- round(mean(Jitter_Data$Send.Max.Jitter..ms.), 5)
78    mjr <- round(mean(Jitter_Data$Rec.Max.Jitter..ms.), 5)
79    js <- round(mean(Jitter_Data$Send.Mean.Jitter..ms.), 5)
80    jr <- round(mean(Jitter_Data$Rec.Mean.Jitter..ms.), 5)
81
82    #Isolate pkts, duration, and bps from text files
83    info <- readLines(info)
84    pkt_rec <- as.numeric(word(info, start = 6)[2])
85    pkt_send <- as.numeric(word(info, start = 6)[6])
86    dur <- as.numeric(word(info, start = 6)[3])
87    bps_r <- as.numeric(word(info, start = 10)[4])
```

```
88    bps_s <- as.numeric(word(info, start = 10)[8])
89    print(pkt_rec)
90
91    add_row(UDP_Avgs, MaxjitterS=mjs, MaxjitterR=mjr, jitterS=js, jitterR=jr,
          duration = dur, BPSSender = bps_s, BPSReceiver = bps_r, PktsSender =
          pkt_send, PktsReceiver = pkt_rec, PktsDrop=(pkt_send-pkt_rec))
92  }
93
94  #Creates the appropriate filenames & destinations for output
95  parseProtocol <- function(directory, protocol, sender_rip)
96  {
97    print(protocol)
98    print(directory)
99    setwd(directory)
100
101   for(i in c(1:30))
102   {
103
104     #Update filenames for the current trial
105     udp_str <- sub("^[[:upper:]]{3,}", protocol, udp_str)
106     sender_str <- sub("^[[:upper:]]{3,}", protocol, sender_str)
107     receiver_str <- sub("^[[:upper:]]{3,}", protocol, receiver_str)
108     info <- sub("^[[:upper:]]{3,}", protocol, info)
109
110     if(grepl("Control",directory))
111     {
112       udp_str <- sub("_X_", "_C_", udp_str)
113       sender_str <- sub("_X_", "_C_", sender_str)
114       receiver_str <- sub("_X_", "_C_", receiver_str)
115       info <- sub("_X_", "_C_", info)
116     }
117     else if(grepl("Mutator",directory))
118     {
119       udp_str <- sub("_X_", "_M_", udp_str)
120       sender_str <- sub("_X_", "_M_", sender_str)
```

```
121         receiver_str <- sub("_X_", "_M_", receiver_str)
122         info <- sub("_X_", "_M_", info)     }
123       else
124       {
125         print("Error assigning output file name.")
126       }
127
128       print(i)
129       #Update for each trial
130       sender_str <- sub("_(\\d|#|\\d\\d)_", capture.output(cat("_",i,"_",sep="")
              ), sender_str)
131       receiver_str <- sub("_(\\d|#|\\d\\d)_", capture.output(cat("_",i,"_",sep
              =="")), receiver_str)
132       info <- sub("_(\\d|#|dd)_", capture.output(cat("_",i,"_",sep="")), info)
133
134       #Get actual data
135       if(protocol != "RTP")
136       {
137         TCP_Avgs <- parseTCP(TCP_Avgs, sender_str, receiver_str, info,
                sender_rip)
138       }
139       else
140       {
141         UDP_Avgs <- parseUDP(UDP_Avgs, udp_str, info, sender_rip)
142       }
143     }
144
145     if(protocol != "RTP")
146     {
147       TCP_Avgs <- TCP_Avgs[-1,]
148       print(TCP_Avgs)
149       #Output results to proper directory
150       if(grepl("Control",directory))
151       {
```

```
152         write.csv(TCP_Avgs, file = capture.output(cat(protocol,"_C_Avgs.csv",
                sep="")))
153       }
154       else if(grepl("Mutator",directory))
155       {
156         write.csv(TCP_Avgs, file = capture.output(cat(protocol,"_M_Avgs.csv",
                sep="")))
157       }
158       else
159       {
160         print("Error parsing directory name.")
161       }
162     }
163     else
164     {
165       UDP_Avgs <- UDP_Avgs[-1,]
166       print(UDP_Avgs)
167       #Output results to proper directory
168       if(grepl("Control",directory))
169       {
170         write.csv(UDP_Avgs, file = capture.output(cat(protocol,"_C_Avgs.csv",
                sep="")))
171       }
172       else if(grepl("Mutator",directory))
173       {
174         write.csv(UDP_Avgs, file = capture.output(cat(protocol,"_M_Avgs.csv",
                sep="")))
175       }
176       else
177       {
178         print("Error parsing directory name.")
179       }
180     }
181   }
182
```

```
183 for(j in c(2:2))
184 {
185   parseProtocol(directory = ctrldirs[j],protocol = protocol[j], sender_rip =
        sender_rip[j])
186   parseProtocol(directory = mutatedirs[j],protocol = protocol[j], sender_rip =
         sender_rip[j])
187 }
```

### D.4 Validation Analysis

MakeGraphs.R takes Aust's original data and the validation data from experiments and produces graphs of the data for comparison.

```
1  require(tidyverse)
2  require(readxl)
3  #require(gridExtra)
4  require(ggplot2)
5  require(cowplot)
6  require(extrafont)
7
8  setwd("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments")
9
10 Avgs <- as_tibble(read_excel("Results.xlsx", sheet = "AustAvgs"))
11
12 MutationTime = c("30S","1M","5M","15M")
13
14 #outputdir <- "C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Latex/Figures
       "
15
16 #Slides directory
17 outputdir <- "C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Slides/"
18
19 IS <- ggplot(data=Avgs, aes(x=Time, y=IntenseScan, group=Mutate)) +
20   theme_classic() +
21   labs(y="Seconds", col="Treatment\nCondition") +
22   geom_line(aes(color=factor(Mutate)), size = 2) +
```

```
23    geom_point(aes(color=factor(Mutate)), size = 3) +
24    scale_x_discrete(limits=MutationTime) +
25    theme(text=element_text(family = "Century Gothic"), axis.text.x  =
          element_text(size=16), axis.text.y  = element_text(size=16),
26          axis.title.x = element_text(face="bold", size=18), axis.title.y =
                element_text(face="bold", size=18))
27
28  ggsave("IScan.png", IS, path = outputdir, height = 5.75, width = 5.4, units =
        "in")
29
30  IH <- ggplot(data=Avgs, aes(x=Time, y=IntenseHosts, group=Mutate)) +
31    theme_classic() +
32    labs(y="Hosts", col="Treatment\nCondition") +
33    geom_line(aes(color=factor(Mutate)), size = 2) +
34    geom_point(aes(color=factor(Mutate)), size = 3) +
35    scale_x_discrete(limits=MutationTime) +
36    theme(text=element_text(family = "Century Gothic"), axis.text.x  =
          element_text(size=16), axis.text.y  = element_text(size=16),
37          axis.title.x = element_text(face="bold", size=18), axis.title.y =
                element_text(face="bold", size=18))
38
39  ggsave("IHosts.png", IH, path = outputdir, height = 5.75, width = 5.4, units =
        "in")
40
41  IPT <- ggplot(data=Avgs, aes(x=Time, y=`PenTime-I`, group=Mutate)) +
42    theme_classic() +
43    labs(y="Seconds", col="Treatment\nCondition") +
44    geom_line(aes(color=factor(Mutate)), size = 2) +
45    geom_point(aes(color=factor(Mutate)), size = 3) +
46    scale_x_discrete(limits=MutationTime) +
47    theme(text = element_text(family = "Century Gothic"), axis.text.x  =
          element_text(size=16), axis.text.y  = element_text(size=16),
48          axis.title.x = element_text(face="bold", size=18), axis.title.y =
                element_text(face="bold", size=18))
49
```

```
50 ggsave("IPT.png", IPT, path = outputdir, height = 5.75, width = 5.4, units = "
       in")
51
52 QS <- ggplot(data=Avgs, aes(x=Time, y=QuickScan, group=Mutate)) +
53   theme_classic() +
54   labs(y="Seconds", col="Treatment\nCondition") +
55   geom_line(aes(color=factor(Mutate)), size = 2) +
56   geom_point(aes(color=factor(Mutate)), size = 3) +
57   scale_x_discrete(limits=MutationTime) +
58   theme(text=element_text(family = "Century Gothic"), axis.text.x =
         element_text(size=16), axis.text.y = element_text(size=16),
59         axis.title.x = element_text(face="bold", size=18), axis.title.y =
             element_text(face="bold", size=18))
60
61 ggsave("QScan.png", QS, path = outputdir, height = 5.75, width = 5.4, units =
       "in")
62
63
64 QH <- ggplot(data=Avgs, aes(x=Time, y=QuickHosts, group=Mutate)) +
65   theme_classic() +
66   labs(y="Hosts", col="Treatment\nCondition") +
67   geom_line(aes(color=factor(Mutate)), size = 2) +
68   geom_point(aes(color=factor(Mutate)), size = 3) +
69   scale_x_discrete(limits=MutationTime) +
70   theme(text=element_text(family = "Century Gothic"), axis.text.x =
         element_text(size=16), axis.text.y = element_text(size=16),
71         axis.title.x = element_text(face="bold", size=18), axis.title.y =
             element_text(face="bold", size=18))
72
73 ggsave("QHosts.png", QH, path = outputdir, height = 5.75, width = 5.4, units =
       "in")
74
75
76 QPT <- ggplot(data=Avgs, aes(x=Time, y=`PenTime-Q`, group=Mutate)) +
77   theme_classic() +
```

```
78    labs(y="Seconds", col="Treatment\nCondition") +
79    geom_line(aes(color=factor(Mutate)), size = 2) +
80    geom_point(aes(color=factor(Mutate)), size = 3) +
81    scale_x_discrete(limits=MutationTime) +
82    theme(text=element_text(family = "Century Gothic"), axis.text.x =
          element_text(size=16), axis.text.y = element_text(size=16),
83          axis.title.x = element_text(face="bold", size=18), axis.title.y =
              element_text(face="bold", size=18))
84
85  ggsave("QPT.png", QPT, path = outputdir, height = 5.75, width = 5.4, units = "
        in")
```

### D.5 Validation T-tests

ValidationTTests.R takes control and mutation data and conducts t-tests at the 99% confidence level to look for a difference in the two reported means.

```
1   require(tidyverse)
2   require(dplyr)
3   require(readxl)
4   require(stringr)
5
6   #Read Replication Data
7   setwd("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/")
8
9   MayerControl <- as_tibble(read_excel("Results.xlsx", sheet = "Control - 30
        Hosts"))
10  Trial30S <- as_tibble(read_excel("Results.xlsx", sheet = "30 sec - 30 Hosts"))
11  Trial1M <- as_tibble(read_excel("Results.xlsx", sheet = "60 sec - 30 Hosts"))
12  Trial5M <- as_tibble(read_excel("Results.xlsx", sheet = "5 min - 30 Hosts"))
13  Trial15M <- as_tibble(read_excel("Results.xlsx", sheet = "15 min - 30 Hosts"))
14
15  #setwd("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/
        Validation")
16
```

```
17 #MayerControl <- as_tibble(read.csv(file="MayerControl.csv", header=TRUE, sep
       =',', na.strings = "-", fileEncoding = "UTF-8"))
18 #Trial30S <- as_tibble(read.csv(file="Mayer30S.csv", header=TRUE, sep=',', na.
       strings = "-", fileEncoding = "UTF-8"))
19 #Trial1M <- as_tibble(read.csv(file="Mayer30S.csv", header=TRUE, sep=',', na.
       strings = "-", fileEncoding = "UTF-8"))
20 #Trial5M <- as_tibble(read.csv(file="Mayer30S.csv", header=TRUE, sep=',', na.
       strings = "-", fileEncoding = "UTF-8"))
21 #Trial15M <- as_tibble(read.csv(file="Mayer30S.csv", header=TRUE, sep=',', na.
       strings = "-", fileEncoding = "UTF-8"))
22
23 treatments <- c("Control","30S", "1M", "5M", "15M")
24
25 #Change to output directory
26 setwd("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/
       Validation")
27
28 t.test.robust <- function(Control, Mutation, hyp, confidence) {
29   obj<-try(t.test(Control, Mutation, var.equal = hyp, conf.level = confidence)
         , silent=TRUE)
30   if (is(obj, "try-error")) return(warnings()) else return(obj)
31 }
32
33 tTests <- function(ControlData, MutationData, Trial)
34 {
35   print(paste("Conducting t-tests for", Trial, sep = " "))
36   #Do t-tests 99% confidence levels for all metrics
37   sink(paste("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/
         Results/Validation/", Trial, "_99_T-Tests.txt", sep = ""), append =
         FALSE)
38
39   print("Scan - I")
40   print(t.test.robust(ControlData$`Intense Scan`, MutationData$`Intense Scan`,
         hyp = TRUE, confidence = 0.99))
41   print("Scan - Q")
```

```
42    print(t.test.robust(ControlData$`Quick Scan`, MutationData$`Quick Scan`, hyp
          = TRUE, confidence = 0.99))
43    print("Hosts - I")
44    print(t.test.robust(ControlData$`Intense Hosts`, MutationData$`Intense Hosts
          `, hyp = TRUE, confidence = 0.99))
45    print("Hosts - Q")
46    print(t.test.robust(ControlData$`Quick Hosts`, MutationData$`Quick Hosts`,
          hyp = TRUE, confidence = 0.99))
47    print("Pen Time - I")
48    print(t.test.robust(ControlData$`Pen Time - I`, MutationData$`Pen Time - I`,
          hyp = TRUE, confidence = 0.99))
49    print("Pen Time - Q")
50    print(t.test.robust(ControlData$`Pen Time - Q`, MutationData$`Pen Time - Q`,
          hyp = TRUE, confidence = 0.99))
51    sink()
52  }
53
54  tTests(MayerControl, Trial30S, "30S")
55  tTests(MayerControl, Trial1M, "1M")
56  tTests(MayerControl, Trial5M, "5M")
57  tTests(MayerControl, Trial15M, "15M")
```

### D.6 QoS Analysis

StatsTests.R takes the input from ReadResults.R and then charts the differences between control and mutation data. It also conducts t-tests and is capable of producing histograms.

```
1  require(tidyverse)
2  require(readxl)
3  require(stringr)
4  require(ggplot2)
5  require(psych)
6  require(reshape2)
7  require(cowplot)
8  require(matrixStats)
9  require(extrafont)
```

```
10
11 #Process avgs .csv files for all the protocols under test and conduct t-tests
       as well as make box & whisker plots.
12 #Has capability to make histograms (currently commented out call to function
       histograms())
13
14 ctrldirs <- c("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/
       Results/FTP/Control","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
       Experiments/Results/HTTP/Control","C:/Users/smayer.CDN/Documents/GitHub/
       Mayer_Thesis/Experiments/Results/IMAP/Control","C:/Users/smayer.CDN/
       Documents/GitHub/Mayer_Thesis/Experiments/Results/POP/Control","C:/Users/
       smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/RTP/Control
       ","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/
       SMTP/Control","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
       Experiments/Results/SSH/Control")
15 mutatedirs <- c("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments
       /Results/FTP/Mutator","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
       Experiments/Results/HTTP/Mutator","C:/Users/smayer.CDN/Documents/GitHub/
       Mayer_Thesis/Experiments/Results/IMAP/Mutator","C:/Users/smayer.CDN/
       Documents/GitHub/Mayer_Thesis/Experiments/Results/POP/Mutator","C:/Users/
       smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/RTP/Mutator
       ","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/Results/
       SMTP/Mutator","C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/
       Experiments/Results/SSH/Mutator")
16 protocol <- c("FTP","HTTP","IMAP","POP","RTP","SMTP","SSH")
17 #RIPs of sender used for each trial. 1:1 mapping with protocol vector
18
19 #Generic filename format
20 avgs_str <- "AAA_X_Avgs.csv"
21
22 updateFilename <- function(avgs_str, directory, protocol)
23 {
24   #Update filenames for the current protocol
25    avgs_str <- sub("^[[:upper:]]{3,}", protocol, avgs_str)
26
```

```
27    if(grepl("Control",directory))
28    {
29      return (avgs_str <- sub("_X_", "_C_", avgs_str))
30    }
31    else if(grepl("Mutator",directory))
32    {
33      return (avgs_str <- sub("_X_", "_M_", avgs_str))
34    }
35    else
36    {
37      print("Error assigning output file name.")
38      return ("Error in updateFilename")
39    }
40  }
41
42
43  t.test.robust <- function(Control, Mutation, hyp, confidence) {
44    obj<-try(t.test(Control, Mutation, var.equal = hyp, conf.level = confidence)
          , silent=TRUE)
45    if (is(obj, "try-error")) return("T test error. Is your data essentially
          constant?") else return(obj)
46  }
47
48
49  boxplots <- function(ExptData, protocol)
50  {
51    print(paste("Creating boxplots for", protocol, sep = " "))
52    #Set output directory
53    outputdir <- "C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Latex/
          Figures"
54    #Slides Dir
55    outputdir <- "C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Slides/"
56
57    #Generate box & whisker plots for data
58    if(protocol != "RTP")
```

```
59  {
60    boxylatency <- ggplot(ExptData, aes(ExptData$control, ExptData$latency)) +
          geom_boxplot() + geom_point() +labs(x = "Experiment Type", y = "
          Latency (sec)") + theme_classic() + theme(legend.position = "none") +
          theme(text=element_text(family = "Century Gothic"), axis.text.x =
          element_text(size=18), axis.text.y = element_text(size=18), axis.
          title.x = element_text(face="bold", size=20), axis.title.y =
          element_text(face="bold", size=20))
61    ggsave(paste(protocol, "_latency.png", sep = ""), boxylatency, path =
          outputdir, height = 5.5, width = 4.7, units = "in")
62
63    boxyRTT <- ggplot(ExptData, aes(ExptData$control, ExptData$RTT)) +
          geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "RTT (
          sec)") + theme_classic() + theme(legend.position = "none") + theme(
          text=element_text(family = "Century Gothic"), axis.text.x =
          element_text(size=18), axis.text.y = element_text(size=18), axis.
          title.x = element_text(face="bold", size=20), axis.title.y =
          element_text(face="bold", size=20))
64    ggsave(paste(protocol, "_RTT.png", sep = ""), boxyRTT, path = outputdir,
          height = 5.5, width = 4.7, units = "in")
65  }
66  else
67  {
68    boxyMJ_S <- ggplot(ExptData, aes(ExptData$control, ExptData$MaxjitterS)) +
          geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "
          Maximum Jitter (sec)") + theme_classic() + theme(legend.position = "
          none") + theme(text=element_text(family = "Century Gothic"), axis.text
          .x = element_text(size=18), axis.text.y = element_text(size=18),
          axis.title.x = element_text(face="bold", size=20), axis.title.y =
          element_text(face="bold", size=20))
69    ggsave(paste(protocol, "_Max_Jitter_S.png", sep = ""), boxyMJ_S, path =
          outputdir, height = 5.5, width = 4.7, units = "in")
70
71    boxyJ_S <- ggplot(ExptData, aes(ExptData$control, ExptData$jitterS)) +
          geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "Mean
```

```
        Jitter (sec)") + theme_classic() + theme(legend.position = "none") +
        theme(text=element_text(family = "Century Gothic"), axis.text.x =
        element_text(size=18), axis.text.y = element_text(size=18), axis.
        title.x = element_text(face="bold", size=20), axis.title.y =
        element_text(face="bold", size=20))
72      ggsave(paste(protocol, "_Jitter_S.png", sep = ""), boxyJ_S, path =
        outputdir, height = 5.5, width = 4.7, units = "in")
73
74      boxyMJ_R <- ggplot(ExptData, aes(ExptData$control, ExptData$MaxjitterR)) +
         geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "
        Maximum Jitter (sec)") + theme_classic() + theme(legend.position = "
        none") + theme(text=element_text(family = "Century Gothic"), axis.text
        .x = element_text(size=18), axis.text.y = element_text(size=18),
        axis.title.x = element_text(face="bold", size=20), axis.title.y =
        element_text(face="bold", size=20))
75      ggsave(paste(protocol, "_Max_Jitter_R.png", sep = ""), boxyMJ_R, path =
        outputdir, height = 5.5, width = 4.7, units = "in")
76
77      boxyJ_R <- ggplot(ExptData, aes(ExptData$control, ExptData$jitterR)) +
        geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "Mean
        Jitter (sec)") + theme_classic() + theme(legend.position = "none") +
        theme(text=element_text(family = "Century Gothic"), axis.text.x =
        element_text(size=18), axis.text.y = element_text(size=18), axis.
        title.x = element_text(face="bold", size=20), axis.title.y =
        element_text(face="bold", size=20))
78      ggsave(paste(protocol, "_Jitter_R.png", sep = ""), boxyJ_R, path =
        outputdir, height = 5.5, width = 4.7, units = "in")
79    }
80
81    boxyduration <- ggplot(ExptData, aes(ExptData$control, ExptData$duration)) +
         geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "
       Duration (sec)") + theme_classic() + theme(legend.position = "none") +
       theme(text=element_text(family = "Century Gothic"), axis.text.x =
       element_text(size=18), axis.text.y = element_text(size=18), axis.title.
       x = element_text(face="bold", size=20), axis.title.y = element_text(face
```

```
    ="bold", size=20))
82  ggsave(paste(protocol, "_duration.png", sep = ""), boxyduration, path =
    outputdir, height = 5.5, width = 4.7, units = "in")
83
84  boxyBPS_S <- ggplot(ExptData, aes(ExptData$control, ExptData$BPSSender)) +
    geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "Sender
    Throughput (bps)") + theme_classic() + theme(legend.position = "none") +
     theme(text=element_text(family = "Century Gothic"), axis.text.x =
    element_text(size=18), axis.text.y = element_text(size=18), axis.title.
    x = element_text(face="bold", size=20), axis.title.y = element_text(face
    ="bold", size=20))
85  ggsave(paste(protocol, "_BPS_S.png", sep = ""), boxyBPS_S, path = outputdir,
     height = 5.5, width = 4.7, units = "in")
86
87  boxyBPS_R <- ggplot(ExptData, aes(ExptData$control, ExptData$BPSReceiver)) +
     geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "
    Receiver Throughput (bps)") + theme_classic() + theme(legend.position =
    "none") + theme(text=element_text(family = "Century Gothic"), axis.text.
    x = element_text(size=18), axis.text.y = element_text(size=18), axis.
    title.x = element_text(face="bold", size=20), axis.title.y =
    element_text(face="bold", size=20))
88  ggsave(paste(protocol, "_BPS_R.png", sep = ""), boxyBPS_R, path = outputdir,
     height = 5.5, width = 4.7, units = "in")
89
90  boxyPktS <- ggplot(ExptData, aes(ExptData$control, ExptData$PktsSender)) +
    geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "Packets
     Sent") + theme_classic() + theme(legend.position = "none") + theme(text
    =element_text(family = "Century Gothic"), axis.text.x = element_text(
    size=18), axis.text.y = element_text(size=18), axis.title.x =
    element_text(face="bold", size=20), axis.title.y = element_text(face="
    bold", size=20))
91  ggsave(paste(protocol, "_PktsS.png", sep = ""), boxyPktS, path = outputdir,
    height = 5.5, width = 4.7, units = "in")
92
```

```
93    boxyPktR <- ggplot(ExptData, aes(ExptData$control, ExptData$PktsReceiver)) +
          geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "
          Packets Received")+ theme_classic() + theme(legend.position = "none") +
          theme(text=element_text(family = "Century Gothic"), axis.text.x =
          element_text(size=18), axis.text.y = element_text(size=18), axis.title.
          x = element_text(face="bold", size=20), axis.title.y = element_text(face
          ="bold", size=20))
94    ggsave(paste(protocol, "_PktsR.png", sep = ""), boxyPktR, path = outputdir,
          height = 5.5, width = 4.7, units = "in")
95
96    boxyDrop <- ggplot(ExptData, aes(ExptData$control, ExptData$PktsDrop)) +
          geom_boxplot() + geom_point() + labs(x = "Experiment Type", y = "Dropped
          Packets") + theme_classic() + theme(legend.position = "none") + theme(
          text=element_text(family = "Century Gothic"), axis.text.x =
          element_text(size=18), axis.text.y = element_text(size=18), axis.title.
          x = element_text(face="bold", size=20), axis.title.y = element_text(face
          ="bold", size=20))
97    ggsave(paste(protocol, "_Drop.png", sep = ""), boxyDrop, path = outputdir,
          height = 5.5, width = 4.7, units = "in")
98
99    print(paste("Graphs created in: ", outputdir, sep = ""))
100 }
101
102 tTests <- function(ControlData, MutationData, protocol)
103 {
104
105
106   print(paste("Conducting t-tests for", protocol, sep = " "))
107   #Do t-tests 99% confidence levels for all metrics
108   #sink(paste("C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Experiments/
          Results/",protocol, "/", protocol, "_99_T-Tests.txt", sep = ""), append
          = FALSE)
109
110   if(protocol != "RTP")
111   {
```

```
112       print("Latency")
113       print(t.test.robust(ControlData$latency, MutationData$latency, hyp = TRUE,
              confidence = 0.99))
114       print("RTT")
115       print(t.test.robust(ControlData$RTT, MutationData$RTT, hyp = TRUE,
              confidence = 0.99))
116     }
117     else
118     {
119       #print(t.test.robust(ControlData$MaxjitterS, MutationData$MaxjitterS, hyp
              = TRUE, confidence = 0.99))
120       #print(t.test.robust(ControlData$jitterS, MutationData$jitterS, hyp = TRUE
              , confidence = 0.99))
121       print("Max Jitter")
122       print(t.test.robust(ControlData$MaxjitterR, MutationData$MaxjitterR, hyp =
              TRUE, confidence = 0.99))
123       print("Mean Jitter")
124       print(t.test.robust(ControlData$jitterR, MutationData$jitterR, hyp = TRUE,
              confidence = 0.99))
125     }
126     print("Duration")
127     print(t.test.robust(ControlData$duration, MutationData$duration, hyp = TRUE,
            confidence = 0.99))
128     print(t.test.robust(ControlData$BPSSender, MutationData$BPSSender, hyp =
          TRUE, confidence = 0.99))
129     print("Throughput")
130     print(t.test.robust(ControlData$BPSReceiver, MutationData$BPSReceiver, hyp =
            TRUE, confidence = 0.99))
131     print("Dropped Packets")
132     print(t.test.robust(ControlData$PktsDrop, MutationData$PktsDrop, hyp = TRUE,
            confidence = 0.99))
133     sink()
134   }
135
136   histograms <- function(ControlData, MutationData, protocol)
```

```
137 {
138   print(paste("Creating histograms for", protocol, sep = " "))
139   #Set output directory
140   outputdir <- "C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Latex/
          Figures"
141
142   if(protocol != "RTP")
143   {
144     cLatency <- ggplot(ControlData, aes(latency)) + geom_histogram(bins = 10)
          + theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("
          Latency (sec)") + ylab("Frequency")
145     cRTT <- ggplot(ControlData, aes(RTT)) + geom_histogram(bins = 10) + theme(
          axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("RTT (sec)") +
           ylab("Frequency")
146
147     mLatency <- ggplot(MutationData, aes(latency)) + geom_histogram(bins = 10)
           + theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("
          Latency (sec)") + ylab("Frequency")
148     mRTT <- ggplot(MutationData, aes(RTT)) + geom_histogram(bins = 10) + theme
          (axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("RTT (sec)")
          + ylab("Frequency")
149   }
150   else
151   {
152     cJitter <- ggplot(ControlData, aes(jitterR)) + geom_histogram(bins = 10) +
           theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("Jitter
           (sec)") + ylab("Frequency")
153     mJitter <- ggplot(MutationData, aes(jitterR)) + geom_histogram(bins = 10)
          + theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("
          Jitter (sec)") + ylab("Frequency")
154   }
155
156   cBPSR <- ggplot(ControlData, aes(BPSReceiver)) + geom_histogram(bins = 10) +
         theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("
        Throughput (bps)") + ylab("Frequency")
```

```
157     cDrop <- ggplot(ControlData, aes(PktsDrop)) + geom_histogram(bins = 10) +
            theme(axis.text.x = element_text(angle=70, vjust=0.5))  + xlab("Dropped
            Packets") + ylab("Frequency")
158
159     mBPSR <- ggplot(MutationData, aes(BPSReceiver)) + geom_histogram(bins = 10)
            + theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("
            Throughput (bps)") + ylab("Frequency")
160     mDrop <- ggplot(MutationData, aes(PktsDrop)) + geom_histogram(bins = 10) +
            theme(axis.text.x = element_text(angle=70, vjust=0.5)) + xlab("Dropped
            Packets") + ylab("Frequency")
161
162     if(protocol != ("RTP"))
163     {
164       if(protocol == "HTTP")
165       {
166         #Special case since HTTP drop graph isn't descriptive
167         cGrid <- plot_grid(cLatency, cRTT, cBPSR, labels = c("", "", ""))
168         mGrid <- plot_grid(mLatency, mRTT, mBPSR, labels = c("", "", ""))
169       }
170       else
171       {
172         cGrid <- plot_grid(cLatency, cRTT, cBPSR, cDrop, labels = c("", "", "",
                ""))
173         mGrid <- plot_grid(mLatency, mRTT, mBPSR, mDrop, labels = c("", "", "",
                ""))
174       }
175     }
176     else
177     {
178       cGrid <- plot_grid(cJitter, cBPSR, cDrop, labels = c("", "", "", ""))
179       mGrid <- plot_grid(mJitter, mBPSR, mDrop, labels = c("", "", "", ""))
180     }
181     ggsave(paste(protocol, "_C_Hist.png", sep = ""), cGrid, path = outputdir)
182     ggsave(paste(protocol, "_M_Hist.png", sep = ""), mGrid, path = outputdir)
183 }
```

```
184
185 pieCharts <- function(ControlData, MutationData, protocol)
186 {
187   print(paste("Creating pie charts for", protocol, sep = " "))
188   #Set output directory
189   outputdir <- "C:/Users/smayer.CDN/Documents/GitHub/Mayer_Thesis/Latex/
          Figures"
190
191   QoSNames = c("Retransmission", "Fast Retransmission", "Lost ACK", "Out of
          Order", "Spurious Retransmission", "Duplicate ACK", "Window Update", "
          Window Full", "Data")
192
193   #Control
194
195   QoSDataC = c(
196     mean(ControlData$RetransS),
197     mean(ControlData$FastRetransR),
198     mean(ControlData$ACKlostR),
199     mean(ControlData$OutOfOrderR),
200     mean(ControlData$SRetransR),
201     mean(ControlData$DupACKR),
202     mean(ControlData$WinUpdateR),
203     mean(ControlData$WinFullR)
204   )
205
206   datapkts <- mean(ControlData$PktsReceiver)-sum(QoSDataC)
207
208   QoSDataC = c(QoSDataC, datapkts)
209
210   QoSdfC <- data.frame(QoS_Metric = QoSNames, control = round(QoSDataC, 0))
211
212   print(QoSdfC)
213   print(sum(QoSdfC$control))
214
```

```
215 QoSbp <- ggplot(QoSdfC, aes(x="Average Packet Distribution", y=control, fill
        =QoS_Metric)) +
216  geom_bar(width = 1, stat = "identity") +
217  theme_classic()
218 ggsave(paste(protocol, "_C_Pie.png", sep = ""), QoSbp, path = outputdir)
219
220
221 #Mutate
222
223 QoSDataM = c(
224   mean(MutationData$RetransS),
225   mean(MutationData$FastRetransR),
226   mean(MutationData$ACKlostR),
227   mean(MutationData$OutOfOrderR),
228   mean(MutationData$SRetransR),
229   mean(MutationData$DupACKR),
230   mean(MutationData$WinUpdateR),
231   mean(MutationData$WinFullR)
232 )
233
234 datapkts <- mean(MutationData$PktsReceiver)-sum(QoSDataM)
235
236 QoSDataM = c(QoSDataM, datapkts)
237
238 QoSdfM <- data.frame(QoS_Metric = QoSNames, mutator = QoSDataM)
239
240 print(QoSdfM)
241 print(sum(QoSdfM$mutator))
242
243 QoSbp <- ggplot(QoSdfM, aes(x="Average Packet Distribution", y=control, fill
        =QoS_Metric)) +
244   geom_bar(width = 1, stat = "identity") +
245   theme_classic()
246 ggsave(paste(protocol, "_M_Pie.png", sep = ""), QoSbp, path = outputdir)
247
```

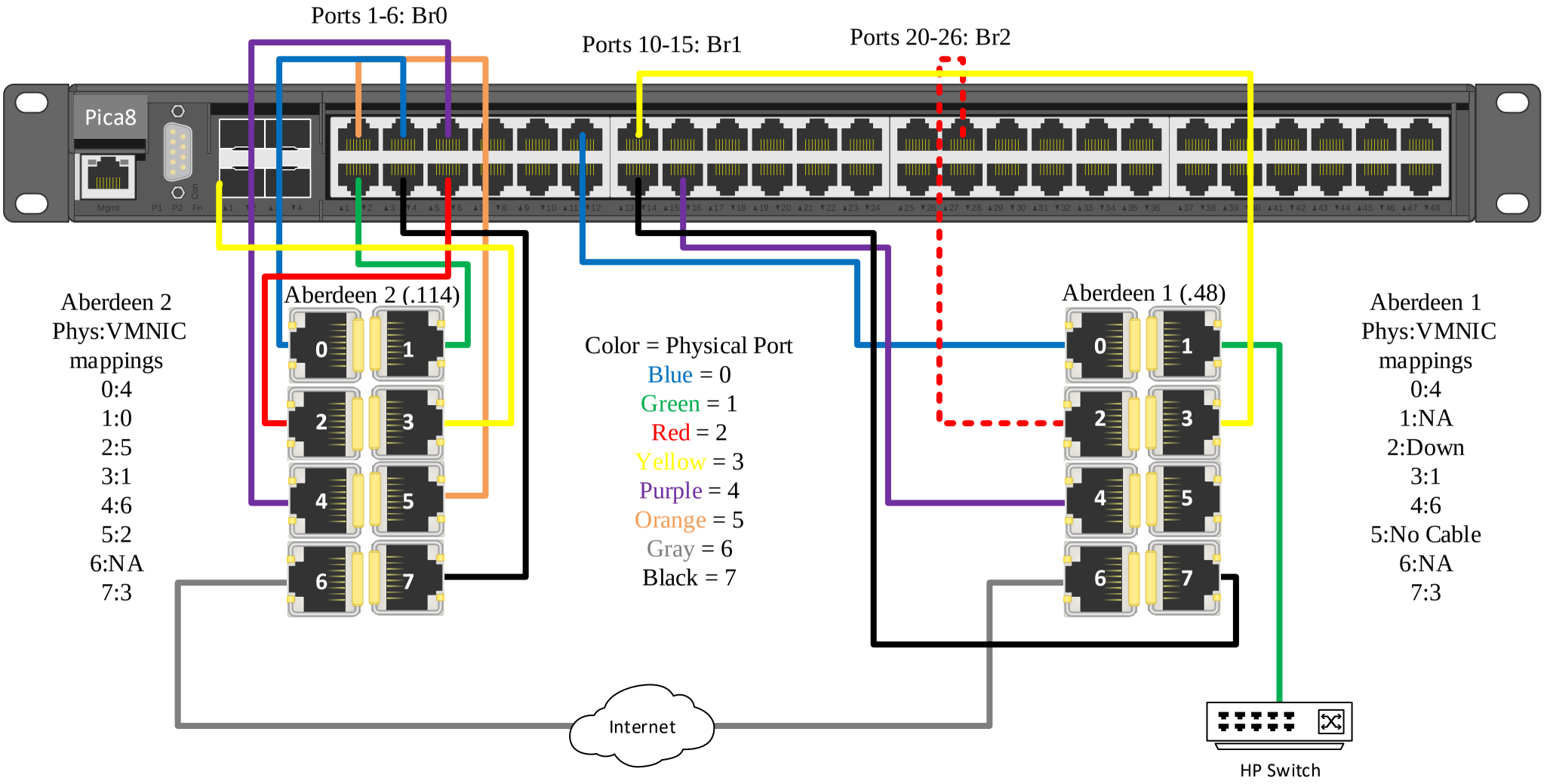
```
248     QoSData <- QoSdfC
249     QoSData$mutator <- round(QoSdfM$mutator, 0)
250     print(QoSData)
251
252     QoSData.m <- melt(QoSData, id.vars = "QoS_Metric")
253     print(QoSData.m)
254
255     QoSChart <- ggplot(data=QoSData.m, aes(x=QoS_Metric, y=value))+
256       geom_bar(aes(fill=variable), position = position_dodge(), stat = "identity
              ") +
257       geom_text(aes(label = value, group = variable),
258                 size = 3, angle = 0, position = position_dodge(width=0.9)) +
259       theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
260       coord_flip() +
261       theme(legend.position="bottom")
262
263     ggsave(paste(protocol, "_Drop.png", sep = ""), QoSChart, path = outputdir)
264   }
265
266
267   processProtocol <- function(ctl_dir, mutate_dir, filename, protocol)
268   {
269     print(protocol)
270
271     #Import & prepare Data (expects .csv stored in UTF-8)
272     #File is different from filename so that it can survive being regexe'd
273     print(ctl_dir)
274     file <- updateFilename(filename, ctl_dir, protocol)
275     ControlData <- as_tibble(read.csv(file=paste(ctl_dir, "/", file, sep = ""),
            header=TRUE, sep=',', fileEncoding = "UTF-8"))
276     print(mutate_dir)
277     file <- updateFilename(filename, mutate_dir, protocol)
278     MutationData <- as_tibble(read.csv(file=paste(mutate_dir, "/", file, sep =
            ""), header=TRUE, sep=',', fileEncoding = "UTF-8"))
279     print("Data read from files.")
```

```
280
281     #Print & describe data
282     print(ControlData)
283     print(MutationData)
284
285     ControlStats <- describe(ControlData)
286     MutationStats <- describe(MutationData)
287
288     ControlData <- add_column(ControlData, control = TRUE)
289     MutationData <- add_column(MutationData, control = FALSE)
290     ExptData <- rbind(ControlData, MutationData)
291     bool <- factor(ExptData$control==1, labels = c("Mutator","Control"))
292     print(bool)
293     ExptData$control <- bool
294     print(ExptData)
295
296     myPlots <- boxplots(ExptData, protocol)
297     #tTests(ControlData, MutationData, protocol)
298     #histograms(ControlData, MutationData, protocol)
299     #pieCharts(ControlData, MutationData, protocol)
300 }
301
302 for(j in c(2:7))
303 {
304     processProtocol(ctl_dir = ctrldirs[j], mutate_dir = mutatedirs[j], filename
          = avgs_str, protocol = protocol[j])
305 }
```

# Appendix E. Network Wiring Diagram

This appendix includes the physical connections between the SDN switch and two servers used to conduct experiments. The rack diagram used an IBM switch due to a lack of available Pica rack diagrams. On the real Pica switch, the numbering for ports is flipped (i.e., top row ports are even and bottom row ports are odd). For this thesis, the dotted red line on Port 2 of Aberdeen 1 indicates an inactive interface. The HP switch on port 1 of Aberdeen 1 did not play a role in this thesis.

Bottom row of RJ45 ports are Odd, Top row are Even.



# Bibliography


[1] *Metasploit*. [Online]. Available: https://www.metasploit.com/ (visited on 01/18/2018).

[2] P. Göransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*, 2nd ed. Cambridge, MA: Morgan Kaufman, 2017, ISBN: 9780128045558.

[3] T. Nadeau and K. Gray, *SDN: Software Defined Networks*. O'Reilly Media, 2013, ISBN: 9781449342302.

[4] K. Kirkpatrick, *Software-defined networking*, 2013. DOI: 10.1145/2500468.2500473. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2500468.2500473 (visited on 01/18/2018).

[5] N. Feamster, J. Rexford, and E. Zegura, *The Road to SDN: An Intellectual History of Programmable Networks*, New York, NY, USA, 2014. DOI: 10.1145/2602204.2602219. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602219 (visited on 01/18/2018).

[6] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014, ISSN: 1553877X. DOI: 10.1109/SURV.2014.012214.00180. arXiv: 1406.0440.

[7] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, *Advanced study of SDN/OpenFlow controllers*, 2013. DOI: 10.1145/2556610.2556621. [Online]. Available: http://dl.acm.org/citation.cfm?id=2556610.2556621 (visited on 01/18/2018).

[8] *Boost C++ Libraries*, 2017. [Online]. Available: http://www.boost.org/ (visited on 01/18/2018).

[9] *OSGi: The Dynamic Module System for Java*, 2018. [Online]. Available: https://www.osgi.org/developer/architecture/ (visited on 01/18/2018).

[10] *Spring*, 2018. [Online]. Available: https://spring.io/ (visited on 01/18/2018).

[11] *Netty*, 2018. [Online]. Available: https://netty.io/ (visited on 01/18/2018).

[12] N. Mathewson, A. Khuzhin, and N. Provos, *Libevent*, 2017. [Online]. Available: http://libevent.org/ (visited on 01/18/2018).

[13] *GLib Reference Manual*, 2014. [Online]. Available: https://developer.gnome.org/glib/ (visited on 01/18/2018).

[14] D. Bilenko, *Gevent*, 2015. [Online]. Available: http://www.gevent.org/ (visited on 01/18/2018).

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, *OpenFlow: Enabling Innovation in Campus Networks*, 2008. DOI: 10.1145/1355734.1355746. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1355734.1355746 (visited on 01/18/2018).

[16] C. Berndtson, *25 Software-Defined Networking Players to Know*, 2012. [Online]. Available: http://www.crn.com/slide-shows/networking/232900998/25-software-defined-networking-players-to-know.htm (visited on 04/26/2017).

[17] *Software-Defined Networking: Why We Like It and How We Are Building On It*, 2013. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/cis13090_sdn_sled_white_paper.pdf (visited on 01/18/2018).

[18] L. Plant, *Network and Cloud: SDN and NFV 101*, 2015. [Online]. Available: https://www.ibm.com/blogs/insights-on-business/telecom-media-entertainment/network-and-cloud-sdn-and-nfv-101/ (visited on 01/18/2018).

[19] E. Skoudis, *Counter Hack Reloaded, Second Edition: A Step-by-step Guide to Computer Attacks and Effective Defenses*, Second. Upper Saddle River, NJ, USA: Prentice Hall Press, 2005, ISBN: 9780131481046.

[20] *Security by Design Principles*, 2017. [Online]. Available: https://www.owasp.org/index.php/Security_by_Design_Principles#Principle_of_Defense_in_depth (visited on 01/01/2017).

[21] V Nagaraju, L Fiondella, and T Wandji, "A survey of fault and attack tree modeling and analysis for cyber risk management," *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pp. 1–6, 2017. DOI: 10.1109/THS.2017.7943455.

[22] NSTC, *Trustworthy Cyberspace: Strategic Plan for the Federal Cybersecurity Research and Development Program*, 2011. [Online]. Available: https://www.nitrd.gov/SUBCOMMITTEE/csia/Fed_Cybersecurity_RD_Strategic_Plan_2011.pdf (visited on 01/18/2018).

[23] A. Duby, *Moving Target Defense: Evasive Maneuvers in Cyberspace*, Augusta, GA, 2016. [Online]. Available: https://www.youtube.com/watch?v=HZWOyZEScys (visited on 01/18/2018).

[24] M. Aust, "Proactive Host Mutation in Software-Defined Networking," Master's thesis, Air Force Institute of Technology, Dayton, OH, 2017.

[25] J. A. Jerkins, "Motivating a market or regulatory solution to IoT insecurity with the Mirai botnet code," *2017 IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC 2017*, 2017. DOI: 10.1109/CCWC.2017.7868464.

[26] A. Stavrou, J. Voas, and I. Fellow, *DDoS in the IoT*, 2017. [Online]. Available: https://www.computer.org/csdl/mags/co/2017/07/mco2017070080.html (visited on 01/18/2018).

[27] M. Reith, S. Penecost, D. Celebucki, and R. Kaufman, *Operationalizing Cyber: Recommendations for Future Research*, 2017. [Online]. Available: https://search.proquest.com/openview/0c3e05994e4a362d80ad6374fb1b10e9/1?pq-origsite=gscholar&cbl=396500 (visited on 01/18/2018).

[28] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan, *Spatio-temporal Address Mutation for Proactive Cyber Agility against Sophisticated Attackers*, 2014. DOI: 10.1145/2663474.2663483. [Online]. Available: http://dl.acm.org/citation.cfm?id=2663474.2663483 (visited on 01/18/2018).

[29] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware IP address randomization for proactive agility against sophisticated attackers," *Proceedings - IEEE INFOCOM*, vol. 26, pp. 738–746, 2015, ISSN: 0743166X. DOI: 10.1109/INFOCOM.2015.7218443.

[30] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, vol. 106 LNICS, 2013, pp. 310–327, ISBN: 9783642368820. DOI: 10.1007/978-3-642-36883-7_19.

[31] J. H. Jafarian, E. Al-Shaer, and Q. Duan, *Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking*, 2012. DOI: 10.1145/2342441.2342467. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342467 (visited on 01/18/2018).

[32] E. Cole and S. Northcutt, *Honeypots: A Security Manager's Guide to Honeypots*, 2017. [Online]. Available: https://www.sans.edu/cyber-research/security-laboratory/article/honeypots-guide (visited on 08/31/2017).

[33] J. Dearien, *OTSDN: What is it and why do you need it?* Minneapolis, MN, 2017. [Online]. Available: https://github.com/samayer12/OTSDNSlides (visited on 01/18/2018).

[34] M. Aharoni, D. Kearns, and R. Hertzog, *Kali*, 2018. [Online]. Available: https://www.kali.org/ (visited on 01/23/2018).

[35] *Nmap*. [Online]. Available: https://nmap.org/ (visited on 01/18/2018).

[36] S. Poretsky, J. Perser, S. Erramilli, and S. Khurana, *Terminology for Benchmarking Network-layer Traffic Control Mechanisms*, 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4689.txt (visited on 01/18/2018).

[37] J. Anuskiewicz, *Measuring Jitter Accurately*, 2008. [Online]. Available: http://www.lightwaveonline.com/articles/2008/04/measuring-jitter-accurately-54886317.html (visited on 09/01/2017).

[38] R. Chen, *Computing over a high-latency network means you have to bulk up*, 2006. [Online]. Available: https://blogs.msdn.microsoft.com/oldnewthing/20060407-25/?p=31613 (visited on 09/01/2017).

[39] J. E. Burge, J. M. Carroll, and R. McCall, *What is Latency and Why Does It Matter?* 2008. DOI: 10.1007/978-3-540-77583-6_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-77583-6_1 (visited on 01/18/2018).

[40] *Round-Trip Delay Time*. [Online]. Available: https://www.its.bldrdoc.gov/fs-1037/dir-031/_4641.htm (visited on 01/18/2018).

[41] SilverPeak, *How to Properly Measure and Correct Packet Loss*, 2017. [Online]. Available: https://www.silver-peak.com/sites/default/files/infoctr/silver-peak_wp_measuringloss.pdf (visited on 08/17/2017).

[42] K. Mansfield and J. Antonakos, *Computer Networking from LANs to WANs: Hardware, Software, and Security*. Boston: Cengae Learning, 2010, p. 501, ISBN: 978-1423903161.

[43] E. Weisstein, *Central Limit Theorem*, 2018. [Online]. Available: http://mathworld.wolfram.com/CentralLimitTheorem.html (visited on 01/18/2018).

[44] C. Lung, *Getting started with OpenStack Oslo Config (oslo.config)*, 2014. [Online]. Available: http://www.giantflyingsaucer.com/blog/?p=4822 (visited on 01/18/2018).

[45] *Curl*. [Online]. Available: https://curl.haxx.se/ (visited on 01/18/2018).

[46] *Rtpgen*. [Online]. Available: https://github.com/kevana/rtpgen (visited on 01/18/2018).

[47] *Rtpdump*. [Online]. Available: http://www.cs.columbia.edu/irt/software/rtptools/ (visited on 01/18/2018).

[48] J Postel and J Reynolds, *RFC 959: File Transfer Protocol*, 1985. [Online]. Available: https://tools.ietf.org/html/rfc959 (visited on 01/18/2018).

[49] *Active FTP vs. Passive FTP, a Definitive Explanation*, 2018. [Online]. Available: http://slacksite.com/other/ftp.html (visited on 01/31/2018).

[50] *Lftp*, 2017. [Online]. Available: https://lftp.yar.ru/ (visited on 01/18/2018).

[51] R. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7230.txt (visited on 01/18/2018).

[52] *TCP Analyze Sequence Numbers*. [Online]. Available: https://wiki.wireshark.org/TCP_Analyze_Sequence_Numbers (visited on 01/18/2018).

[53] P. Verma, *What is the meaning of "TCP Spurious Retransmission" label in Wireshark TCP packet?* 2016. [Online]. Available: https://www.quora.com/What-is-the-meaning-of-TCP-Spurious-Retransmission-label-in-Wireshark-TCP-packet.

[54] M. Crispin, *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*, 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3501.txt (visited on 01/18/2018).

[55] J. Myers and M. Rose, *Post Office Protocol - Version 3*, 1996. [Online]. Available: https://tools.ietf.org/html/rfc1939 (visited on 01/18/2018).

[56] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, 2003. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3550.txt (visited on 01/18/2018).

[57] J. Klensin, *Simple Mail Transfer Protocol*, 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5321.txt (visited on 01/18/2018).

[58] T. Ylonen and C. Lonvick, *The Secure Shell (SSH) Transport Layer Protocol*, 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4253.txt (visited on 01/18/2018).

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704–0188*



| 1. REPORT DATE *(DD–MM–YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From — To)* |
|---|---|---|
| 22–03–2018 | Master's Thesis | Oct 2017 — Mar 2018 |

**4. TITLE AND SUBTITLE**

Quality of Service Impacts of a Moving Target Defense with Software-Defined Networking

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

Mayer, Samuel, A, 2d Lt, USAF

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

**8. PERFORMING ORGANIZATION REPORT NUMBER**

AFIT-ENG-MS-18-M-045

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

This field intentionally left blank.

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION / AVAILABILITY STATEMENT**

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**13. SUPPLEMENTARY NOTES**



**14. ABSTRACT**

An analysis of the impact a defensive network technique implemented with software-defined networking has upon quality of service experienced by legitimate users. The research validates previous work conducted at AFIT to verify claims of defensive efficacy and then tests network protocols in common use (FTP, HTTP, IMAP, POP, RTP, SMTP, and SSH) on a network that uses this technique. Metrics that indicate the performance of the protocols under test are reported with respect to data gathered in a control network. The conclusions of these experiments enable network engineers to determine if this defensive technique is appropriate for the quality of service requirements on their network.

**15. SUBJECT TERMS**

Software-Defined Networking, Moving Target Defense, Random Host Mutation, Quality of Service

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Dr. Barry E. Mullins, AFIT/ENG |
| U | U | U | U | 242 | 19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x7979; barry.mullins@afit.edu |