A CASE FOR SOFTWARE-DEFINED NETWORKING IN THE UNITED STATES MARINE CORPS: AUTOMATING DISTRIBUTED FIREWALLS

Logan, Brent E.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A CASE FOR SOFTWARE-DEFINED NETWORKING IN THE UNITED STATES MARINE CORPS: AUTOMATING DISTRIBUTED FIREWALLS

by

Brent E. Logan

June 2019

Thesis Advisor:	Geoffrey G. Xie
Second Reader:	Justin P. Rohrer

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

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE A CASE FOR SOFTWARE-DEFINED NETWORKING IN THE UNITED STATES MARINE CORPS: AUTOMATING DISTRIBUTED FIREWALLS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Brent E. Logan	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

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

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

13. ABSTRACT (maximum 200 words)

Software Defined Networking (SDN) is a field in computer science that has seen rapid adoption in industry and academia. SDN reduces network administration and cost, empowers fine grain network control, and enables programmability and innovation in a relatively stagnant area of computer science. In this research, we make a case for more rapid adoption of software defined network (SDN) technology in the DoD by demonstrating that distributed firewall operation can be virtualized, automated, and assured of security properties with SDN. Specifically, we have developed and evaluated a distributed firewall application within the standard ONOS SDN control platform. The application enforces access control between arbitrary end points and intelligently distributes processing of filter rules across network devices, even after the network topology changes. The test bed evaluation results confirm the reachability control performance and show that the application and virtual switches built upon commodity computers are capable of handling more than 50,000 filter rules. The automated distributed firewall is a viable proof of concept that provides flexibility and improved security in a world where ubiquitous, ad hoc, and zero-trust networking are becoming the new normal. Lastly, we provide an acquisition heuristic for purchasing and fielding SDN solutions to the Marine Corps' operating forces.

14. SUBJECT TERMS software defined networking, firewall, distributed firewall, acquisition, United States Marine Corps, networking, information technology, programmable networks, hybrid networks, tactical networks, ubiquitous networking, reachability control, flow control, network segmentation, automation, ad hoc networking, Zero Trust, network segmentation	**15. NUMBER OF PAGES** 147
	16. PRICE CODE

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

A CASE FOR SOFTWARE-DEFINED NETWORKING IN THE UNITED STATES MARINE CORPS: AUTOMATING DISTRIBUTED FIREWALLS

Brent E. Logan
Major, United States Marine Corps
BS, U.S. Naval Academy, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2019

Approved by: Geoffrey G. Xie
Advisor

Justin P. Rohrer
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Software Defined Networking (SDN) is a field in computer science that has seen rapid adoption in industry and academia. SDN reduces network administration and cost, empowers fine grain network control, and enables programmability and innovation in a relatively stagnant area of computer science. In this research, we make a case for more rapid adoption of software defined network (SDN) technology in the DoD by demonstrating that distributed firewall operation can be virtualized, automated, and assured of security properties with SDN. Specifically, we have developed and evaluated a distributed firewall application within the standard ONOS SDN control platform. The application enforces access control between arbitrary end points and intelligently distributes processing of filter rules across network devices, even after the network topology changes. The test bed evaluation results confirm the reachability control performance and show that the application and virtual switches built upon commodity computers are capable of handling more than 50,000 filter rules. The automated distributed firewall is a viable proof of concept that provides flexibility and improved security in a world where ubiquitous, ad hoc, and zero-trust networking are becoming the new normal. Lastly, we provide an acquisition heuristic for purchasing and fielding SDN solutions to the Marine Corps' operating forces.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

List of Acronyms and Abbreviations

ACL	Access Control List
API	Application Planning Interface
ARP	Address Resolution Protocol
C2	Command and Control
CDMA	Code Division Multiple Access
COMMEX	Communications Exercise
COTS	Commercial Off the Shelf
CSN	Circuit Switched Networks
DHS	Department of Homeland Security
DHS	Department of Homeland Security
DoD	Department of Defense
DoN	Department of the Navy
EIGRP	Enhanced Interior Gateway Protocol
FDMA	Frequency Division Multiple Access
GUI	Graphical User Interface
HADR	Humanitarian Assistance Disaster Relief
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force

IOS	Inter-network Operating System
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IP	Internet Protocol
IRTF	Internet Research Task Force
ISR	Intelligence, Surveillance, and Reconnaissance
IT	Information Technology
IT	Information Technology
IoT	Internet of Things
MAC	Media Access Control
MAGTF	Marine Air Ground Task Force
MANET	Mobile Ad-Hoc Network
MASINT	measurement and signature intelligence
MCCES	Marine Corps Communication-Electronics School
MCDP	Marine Corps Doctrinal Publication
MCWP	Marine Corps Warfighting Publication
MEB	Marine Expeditionary Brigade
MEF	Marine Expeditionary Force
MEU	Marine Expeditionary Unit
MOC	Marine Operating Concept
MOOTW	Military Operations Other Than War
MOS	Military Occupational Specialty

MSC	Major Subordinate Command
NOS	Network Operating System
NOTM	Network on the Move
NPS	Naval Postgraduate School
OEF	Operation Enduring Freedom
OF	Open Flow
OIF	Operation Iraqi Freedom
OODA	Observe Orient Decide Act
OSI	Open System Interconnection
OS	Operating System
OVS	Open vSwitch
PIOM	plan, install, operate, and maintain
QoS	Quality of Service
REST	Representational State Transfer
REST	representational state transfer
RFC	Request for Comments
ROMO	Range of Military Operations
SDN	Software Defined Network
SRWBR	short range wide band radio
SSH	Secure Shell
STRAPEX	bootstrap exercise
SYSCON	Systems Control

TCP/IP	Transmission Control Protocol/Internet Protocol
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
Telnet	Terminal Network
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USAF	United States Air Force
USA	United States Army
USB	Universal Serial Bus
USCG	United States Coast Guard
USG	United States government
USMC	United States Marine Corps
USN	U.S. Navy
VM	Virtual Machine
VOIP	Voice over IP
VSAT	Very Small Aperture Terminal
WWW	World Wide Web
cURL	Client URL

Acknowledgments

I am deeply appreciative for the opportunity to attend the Naval Postgraduate School. This journey would not have been possible without the support of the leadership at the Marine Corps Communication-Electronics School, who encouraged me to pursue my graduate education. During my time at NPS, I would have struggled had it not been for the support of my cohort: Joe "Peepers" Plot, Lucas Burke, Teal "Koala" Peterson, Nate Richardson, Ryan "Dad" Keller, Britney Snelgrove, Michael Whitaker, Britt Campbell, and John Mutton. I was humbled every day by your work ethic, ingenuity, and creativity. Your friendship made attending NPS an experience I'll never forget. I would also like to thank Scott Tollefson and Terry Kvitchko, who helped me during my thesis with coding assistance, guidance, and pointers. I would not have been able to get as much done in such a short period without you. I am lucky to have had Dr. Geoffrey G. Xie as my advisor. Dr. Xie fostered my intellectual curiosity and developed my critical thinking, writing, and computer science skills. I am a better leader today because of Dr. Xie's guidance. Lastly, I'd like to thank my parents, Suzanne and Ed Logan, and my family for being there to lean on throughout my career. I love you guys very much.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: The Resurgence of Great Power Competition

For the first time since the fall of the Soviet Union, we are experiencing a return to great power competition. With a rising China and a resurgent Russia, the U.S. does not enjoy a monopoly on sea power or sea control. Rogue regimes like North Korea and Iran persist in taking actions that threaten regional and global stability.

—Admiral John M. Richardson, 31st Chief of Naval Operations [1]

A common definition of warfare is of two nation-state's meeting in combat (e.g., World War 2) [2]. The reality is modern warfare does not adhere to a strict definition. The global operating environment is becoming increasingly gray with the emergence of, "gray zones" defined as, "conceptual space[s] between peace and war ... threaten[ing] US and allied interests by challenging, undermining, or violating international customs, norms, or laws" [3], [4]. In particular, the global operating environment is becoming rapidly more dynamic and complex due to the proliferation of information technology, cyber operations, information warfare, and the pervasive use of disruptive technologies [1], [5]. Information technology proliferation has lowered the cost of entry into warfare [6]. Accordingly, information technology has created an explosion of war participants from those who are interested in viewing and reporting war to those who are actively engaged in it [7]. Rapid technology advancement has outpaced the Department of Defense (DoD)'s ability to purchase, field, and train to new equipment and concepts [8] forcing the U.S. armed services to reexamine their acquisition and war fighting paradigms.

As a result, Admiral Richardson, referencing Colonel John Boyd's Observe Orient Decide Act (OODA) loop (a decision making model) in an address at the 68th Current Strategy Forum at the U.S. Naval War College, stated that the United States' observation (raw information gathering) advantage is gone and that the winner of strategic competition will be decided by the team [or nation-state] who can "orient" (distill,filter and sort information) and "decide"(make an informed decision) faster than the other [9]. Admiral Richardson elaborated that the problem of “orientation” is a technical one, pointing to the need to rapidly sort through “avalanches” of information to make informed decisions [9]. Moreover, a “networked navy” is a pillar of Admiral Richardson’s plan to increase the lethality of the Naval service [1]. Admiral Richardson’s “A Design For Maintaining Maritime Superiority” states that the Navy and Marine Corps team *must* create the capability to act at any point on the Competition Conflict Spectrum emphasizing the need for capabilities that can be used in an increasing diverse and gray operating environment (Figure 1.1: a diagram that accounts for the shift from the old definition of war to the new by visualizing the shift from the older Joint Phasing Model to the non-linear sliding scale of the Competition Conflict Spectrum [10]).

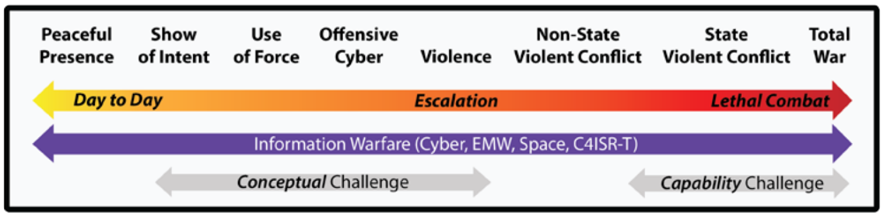


Figure 1.1. Competition-Conflict Spectrum for the Military Dimension of Power. Source: [10].

However, according to the Marine Operating Concept (MOC), the Marine Corps is not ready to meet the demands of the future and has fallen short of its congressional mandate to, "be ready when the nation is least ready" [11] citing technology proliferation and information warfare as drivers of change. The purpose of the MOC is to acknowledge the Marine Corps’ readiness deficit and provide a framework to begin building up the Marine Corps’ capabilities to address the future operating environment and the reemergence of the great power competition [11]. The Marine Corps is expected to undertake any mission within the Range of Military Operations (ROMO) on the continuum of operations outlined in the Joint Phasing Model referenced in [12]. ROMO includes everything from a peaceful presence mission, to humanitarian assistance, to total war. This research seeks to help close the orientation gap referenced by Admiral Richardson and realize the imperatives of the Marine Operating Concept by testing networking technology with the potential to improve orientation speed and multiply innovation potential.

1.1 Problem Statement

In the Marine Corps, the ability to begin the OODA loop decision cycle is entirely dependent on the availability of a Command and Control (C2) network consisting of telecommunications and information technology equipment enabling communication and data exchange between relevant parties. Marine Corps War fighting Publication 3-40.3, *MAGTF Communications*, states that, " the Marine Corps *requires* a robust C2 capability to execute actions across the range of joint military operations" [13]. Without an enabling C2 network, war fighting cannot occur.

Therefore, the time that it takes to plan, install, operate, and maintain (PIOM) a C2 network is a factor that must be considered in the OODA loop decision cycle speed. Computer network network engineering, installation, and management is part of C2 network installation time and is a difficult, manpower-intensive, and complex undertaking [14]. To exacerbate matters, the trend of networking nearly everything has manifested in the form of the Internet of Things (IoT) [15]. More devices, more users, and more sensors generally equates to more data making the task of administering and managing network traffic extremely challenging [16]. The state of networking appears to be approaching exponential growth in the amount of data and devices on the network [17], [18]. Networking in the United States Marine Corps (USMC) is no different and suffers from the same difficulties as the civilian sector, but is also compounded by time, resource, and policy constraints inherent in tactical environments. This is the challenge that Marine Corps and the armed services must overcome to win the "orientation" battles of the future. To compete and win in the next great power competition, the Marine Corps needs to rethink how it approaches networking and network operations.

To that end, this research studies Software Defined Networks because it is a key technology that could improve the USMC's OODA loop decision speed. SDN is a promising technology that is advancing rapidly outside of academic research and is taking strides in industry [15], [19]. Software Defined Network (SDN) promises to improve the limits of traditional networking through centralized control and network programmability giving operators efficient granular control of the network [14]. Moreover, SDN has applicability across a multitude of areas including, but not limited to, security [20], rapidly re-configurable networks [21], fifth generation (5G) cellular networks [22], and big data [23]. The promise and potential impact of SDN has motivated companies like Google, Microsoft, Facebook, Yahoo, Verizon, and Deutsche Telekom to support the Open Networking Foundation (ONF) with the hopes of creating SDN standards and promoting SDN on the internet [14]. SDN's potential make it a prime technology of interest to improve the Navy and Marine Corps ability to orient faster.

Parts of the United States Government have begun investing in SDN with benefits ranging from improved operating costs to improved network performance and efficiency [15]. The former USA Chief Information Officer (CIO), Lieutenant General Robert S. Ferrell, stated that SDN is part of the Army's strategic investments that will shape the battlefield of the Army in 2025—2040 [24]. Lieutenant General Ferrell also argued that the battlefield will be shaped by, "so-called leap-forward technologies" like SDN and warrant research and investment to maintain a leg up in combat [15]. SDN thus has the potential to make USMC networks agile, resilient in a contested environment, and rapidly configurable and programmable in response to requirements in "A Cooperative Strategy for 21st Century Seapower" [25], "The Marine Operating Concept" [11], "A Design for Maintaining Maritime Superiority" [10], and the Marine Corps' concept for "Expeditionary Advanced Base Operations" [26].

1.2 Research Questions

The following are the research questions that this research seeks to answer.

1. Using the greedy heuristic algorithm for Access Control List Access Control List (ACL) placement shown in [27], what are the effects of deploying an SDN distributed firewall application, in a traditional and hybrid USMC network? What are the effects in terms of security, throughput, and resilience in comparison to a traditional USMC network?
2. Given the work done in [28] how can the USMC begin incrementally purchasing and fielding SDN equipment to maximize immediate networking gains and minimize fiscal cost for the organization?

If SDN is a viable technology for improving the efficiency of network operations, it may serve as an enabler for solving the Navy and Marine Corps' "orientation" problem in support of the current great power competition.

1.3 Thesis Organization

This research is organized as follows:

Chapter II surveys the origins and technical underpinnings of Software Defined Networking to provide context for comparing SDN to existing networking technologies and for understanding the experimental design, set forth in Chapter III. Chapter II also details Firewall technologies and network access control as a primer for contextualizing the distributed firewall program. Lastly, it outlines the state of networking and information technology personnel in the United States Marine Corps USMC. Chapter III covers experimental design and is broken into two major parts. The first part articulates the algorithms that comprise the distributed firewall program as well as the heuristics used as decision making criteria for flow rule placement. The second part of the chapter outlines the methodology for testing the research questions and the motivations behind the various topological designs. Chapter III provides a detailed outline of how each variable is tested, measured, and evaluated. Chapter IV summarizes the results of the experiments. Chapter V examines the experimental results and evaluates their relevance to the research questions and the conclusions that are reached. Chapter V also discusses limitations of the experiments and future research areas and the impact the research has towards Marine Corps Networks.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2: Background

Fundamental to our character as a Marine Corps is our role as the Nation's force-in-readiness. We must continue to be ready for operations across the range of military operations (ROMO). At the same time, we recognize the current and future fight may not be what we experienced in the past. It will encompass not just the domains of land, air, and sea, but also space and the cyber domain. It will include information operations and operations across the electromagnetic spectrum. It will involve rapidly changing and evolving technologies and concepts, which will force us to be more agile, flexible, and adaptable. Most importantly, it will require Marines who are smart, fit, disciplined, resilient, and able to adapt to uncertainty and to the unknown.

—General Robert B. Neller, 37th Commandant of the Marine Corps [11]

This chapter will discuss the key capabilities and constraints of two foundational technologies: SDN and Distributed Firewalls. The purpose of describing these technologies is to lay a foundation for describing the current state of networking in the USMC and advocating for change and innovation. Lastly, related work will be addressed with respect to distributed firewall technologies, SDNs, and hybrid networks consisting of both SDN nodes and traditional switches and routers.

2.1 Software Defined Networks

In the information technology world, Software Defined Networking has gained traction as the successor to the traditional networking paradigm [14], [23]. The defining feature of an SDN is the separation of the data plane from the control plane and network programmability [23]. There are a handful of models that can be used to describe how networking works. The most common models are the Open System Interconnection (OSI) and Transmission Control Protocol/Internet Protocol (TCP/IP) models. Both frameworks are comprised of layers that provide a specific function that enable network communication to happen. The networking layer, also called layer 3, is the layer that deals with the Internet Protocol (IP) and the actual movement of packets through a network. Generally, networking can be divided into three fundamental "layers": the management, control, and data planes [14] which can be seen in Figure 2.1. The data plane's responsibility is to forward packets according to a set of rules while the control plane's role is to serve as the mechanism for communicating those rules to data plane switches. In an SDN the control plane and data plane are decoupled and the control plane functionality is removed from the network devices. The last plane is called the management plane which is the interface from which network operators implement policy and management at a high level.

According to the work done in [29], an SDN has two defining characteristics: In an SDN, the control plane is separated from the data plane. This means that SDN devices that route packets (data plane) are separate and distinct from devices that make control decisions (SDN controllers) for the network. Data plane devices are analogous to mail sorters. They simply store and forward mail to the appropriate destination. If they have an issue, they consult higher headquarters.

The control plane for an SDN, where network intelligence and logic are implemented, is amalgamated into a single device (the controller) that has centralized control over multiple data plane devices.

It is important to note that SDN enables innovation but does not offer capabilities “out of the box” beyond the aforementioned. SDN is simply a tool out of which innovation can grow bringing with it the limitless possibilities of programming [14]. Moreover, commercial networking devices today now typically include an OpenFlow Application Programming Interface (API) for SDN implementation [14]. The SDN wave is gaining inertia and pushing the networking industry forward.

2.1.1 A Brief History of SDN

SDNs improve the way networks are designed and managed [29]; however, the idea of software defined networking is not new and has roots in Circuit Switched Networks (CSN) and old computer networks present during the birth of the world wide web (at least publicly available internet) [29]. In CSN, also known as telephony networks, the control and data planes were separated enabling simplified management of network services [29] through a

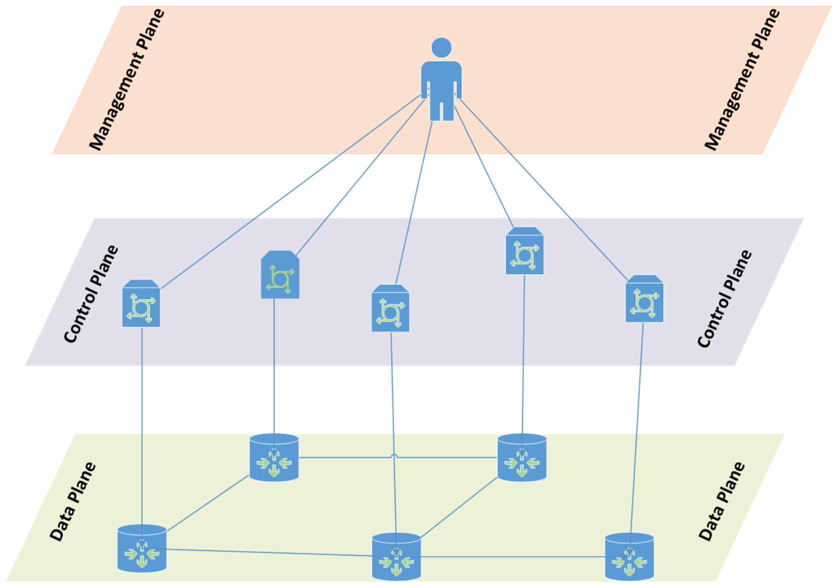


Figure 2.1. View of the Management, Control, and Data Planes. Source: [14].

combination of in band and out of band control channels. This allowed operators to make configuration changes without adversely affecting network traffic or its available bandwidth.

When the internet exploded into public view it became the victim of its own success by complicating networking management and infrastructure [29], [30] through rapid growth. The demand for services forced providers to rapidly expand to accommodate the burgeoning influx of traffic. The primary concerns of internet providers during this time period were a general frustration with the time it took to enable capabilities across the network, the deployment of middle-boxes, dynamic fine-grain control of network resources, and vendor lock in [29]. These concerns can be trivial on small networks but are complex, expensive, and onerous on large networks. Any large-scale change to the network architecture is viewed as an extremely difficult and task that borders on impossible in practice [14]. As an example, consider the task of upgrading the software on your network. The provider must install the software on every piece of equipment on the network. The equipment could be located in small geographic area but is likely to be spread across great distances in numerous communications closets and buildings. Installation of the software and subsequent reboot time are multiplied by the number of pieces of equipment on the network. The cumulative time, labor, and monetary cost of upgrading the network grows rapidly based on the number of devices within it. This process repeats itself anytime the provider needs to upgrade, replace, or remove network equipment. It is important to remember that this simple example does not examine how the provider plans to incrementally upgrade the equipment, failure plan, or testing of the upgrade in a sandbox environment. To say the least, managing and administering a network is a difficult and complex task [14], [29].

2.1.2 SDN Structure

SDN programmability translates into improved network management through computer science abstractions rather than complex scripts and instructions deployed throughout the network [20]. In layman's terms, an operator can change the whole network from a central point at the proverbial touch of button. According to Feamster et al., "Making computer networks more programmable enables innovation in network management and lowers the barrier to deploying new services" [29]. This is made possible from an SDN's structure. To understand SDN structure one needs to understand basic networking concepts and terminology in order to grasp the entire picture:

1. SDN: A new networking framework that enables network programmability, and consolidation of network control to a singular entity called the controller [14]. SDN accomplishes this by separating packet switching functionality (data plane) from routing decision intelligence (control plane) [29].
2. Control Plane: Traffic routing decisions and intelligence as well as the data structures that communicate these decisions form the data plane. Several studies describe the controller as the "brain" of the network [14], [23], [31]. Figure 2.2 depicts the Control Plane in with other SDN constructs.
3. Data Plane: Networking devices responsible for forwarding traffic throughout the network in accordance to the logic (forwarding tables) received from the control plan. [14]. Figure 2.2 depicts the Data Plane in with other SDN constructs.
4. Management Plane: Inclusive of the software and services, enabled by the northbound interface, that serve to configure network policy, monitor network health (status), and network functionality. Applications can be invoked from the interface to employ programs like firewalls, load balancers, and intrusion detection [14].
5. Network Operating System: A NOS is software that provides a user an interface into the management plane of a Software Defined Network. The NOS is the medium

through which holistic policy decisions are dictated to the network and subsequently translated into low level protocols to enable those decisions. Moreover, it is where operators can view SDN's global network view [29].

6. Open Flow: is a standard originally defined in [32], by researchers at Stanford, as a method of running and testing experimental networks over existing network infrastructure without adversely affecting production traffic. The OF standard took SDN from the theoretical to the practical by creating a standard for the data plane and enabling a control plane API from which real networks could be built [14].
7. Network Function Virtualization (NFV): is an abstraction of a part of the network from the physical infrastructure [29].
8. Forwarding Devices: Any network devices, hardware or software, whose role is sort and forward packets, based upon logic received from the control plane of the network (controller).
9. Middlebox: An intermediate network device, within the data plane, that serves in a capacity different from that of a traditional router or switch [33] (e.g., firewall, Network Address Translation [NAT] device, or Intrusion Detection System [IDS]).
10. Northbound Interface: The northbound interface is the API from which programmers can modify control plane logic via the NOS. This interface abstracts away the instructions used to program data plane devices [14]. Figure 2.2 depicts the Northbound Interface with other SDN abstractions.
11. Southbound Interface: The southbound interface is an API which defines the protocols which communicate between the controller and the devices within the data plane. Figure 2.2 depicts the Southbound Interface with other SDN abstractions.

2.1.3 Software Defined Networking Versus Traditional Networking

Traditional networks are comprised of devices whose control and forwarding logic are combined within the same machine. This means that a single device is in control of both forwarding and routing decisions. This makes the network highly decentralized as each device independently determines the best path to get from point a to point b. Using this method, it is extremely difficult for network administrators and engineers to design the network to behave in fine grain detail without meticulously configuring the network, device by device, to perform as intended [14]. Even after such a configuration is created, there is no absolute way to dictate behavior under varying network conditions without

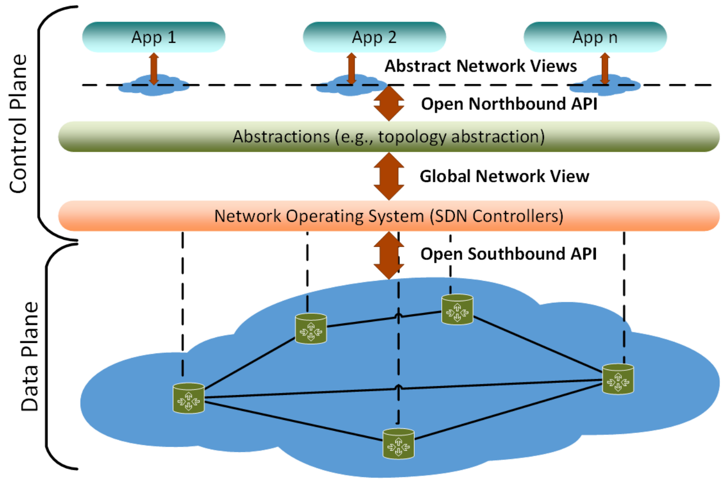


Figure 2.2. View of SDN Architecture and Abstractions. Source: [14].

complex configuration control using load balancing and quality of service techniques. The decentralized nature of traditional networking was considered a critical design feature of the internet in order to ensure network resilience [14]. Traditional networking is effective and has been successful in supporting the modern internet; however, the difficulty with the traditional networking is in configuration control, network management, and administration [14] caused by the aforementioned decentralization. Traditional networking has also long been beholden to vendor specific equipment, command line languages, and even protocols.

Vendor lock in occurs once a specific internet Service Provider (ISP), carrier, or network administrator purchases a particular piece of equipment and builds the network using that equipment set. For the administrator, it is more pragmatic to continue to purchase vendor specific equipment than another vendor's equipment for many reasons. Typically, compatibility between equipment from different vendors can be issue. Different vendors use proprietary protocols, command line languages, and closed operating systems [14]. It is very expensive to retrain personnel in different vendor specific languages and equipment [14]. Lastly, the closed operating systems and languages mean custom configurations, functionality, and protocols outside of included software are none. In order to obtain specialty functionality outside of vendor specific capabilities, specialized middleware is typically employed to fill in in-line gaps in a network [14] which simultaneously increases the logical and physical complexity of the network.

Traditional networks are difficult to configure and easily mis-configured [14], [29]. A network administrator could have thousands of network devices under their purview. Each device must be properly secured, configured, and operated to maintain network functionality. A single mis-configuration can devastate a network [14]. In the author's experience, a single command run on a distant node can reduce network usability drastically. For example, a local administrator advertised an entire class C vice a subnet within that network. The result was that every layer device in the class C network looked for addresses at the router advertising the class C. The problem was that node b was only reachable through high latency, low bandwidth links. The result was a classified network was unusable for a period of several hours while network admins troubleshot the issue.

SDNs are markedly different from traditional networks. As mentioned before, SDN have a control plane and a data plane that are decoupled [14] meaning data plane devices are dumb packet forwarders (switches). Secondly, all forwarding decisions are based on what is called flows rather than simple forwarding rules [14]. A flow is simply a set of fields that match information in the packet header that corresponds to an action taken by that forwarding device. All flows across the network are coordinated by the controller and all forwarding devices apply the same criteria to given flow creating a coordinated and holistic method of handling packets moving throughout a network [14]. Flows provide fine grain control over packet movements giving network administrators absolute control over traffic paths. Moreover, because the controller maintains a current view of the network state, any topology, load, or state changes within the network can be dynamically handled by the controller without operator intervention. This is in contrast to traditional networking where control is possible, with meticulous device configuration, under steady state conditions but difficult to control under dynamic ones.

2.1.4 The Advantages

Overall, SDN has many benefits that stem directly from the decoupling of the control plane from the data plane:

1. Applications benefit from a shared view of the network through the control plane (NOS and or controller) [14]. This means any application can appropriately apply policy, load balancing, and any programmable function even in dynamic networks.
2. Network policy modification is less error prone due to high level abstraction the control

layer provides [14]. Instead of device to device configuration, network operators can holistically define policy and exert control over the network via the control plane. This eliminates the potential for error and eases trouble shooting efforts.

3. The controller maintains a global view of the network and its state. This gives administrators a real-time view of the network without configuring a third party program to maintain this view [34].
4. The ability to program an SDN enables innovation [29] and gives network administrators the ability to modify and augment the network as they see fit.
5. SDN equipment and software are supported by most vendors and readily available [14], [29].
6. Explicit fine grain control over packet path through the flow abstraction [14].

2.2 Hybrid SDN

Migrating from an SDN from a traditional network can be fiscally prohibitive, complicated, and time consuming [28]. Research in Hybrid networks is promising [33], [28] and shows that hybrid networks work and SDN can be adopted incrementally creating immediate capability and cost savings. In this paper, when the term hybrid is used the author is referring to the physical makeup of the network. A hybrid SDN network is a network that uses SDN and non-SDN equipment together. Weitzel [33] showed that heterogeneous Marine Corps tactical networks are possible and functional. Moreover, Weitzel showed that SDN equipment is backward compatible and able to function as a traditional networking device in the absence of an SDN controller. The work done by Levin et al. in [28] suggests that SDN networks can be built using both SDN and traditional networking equipment. An example of hybrid network from is shown below:

Figure 2.3 is a hypothetical example of a network which includes traditional networking equipment and SDN equipment. This type of setup is within the scope of possibilities in today's environment because of the high cost of information technology equipment and likelihood that SDN adoption will be an incremental process [33]. Network Engineers may choose to adopt an incremental approach over time, due to monetary constraints, instead of a full overhaul in order to maintain service and cut down on costs [28]. The research outlined in this paper looks to expand upon the work shown in [33] and demonstrate SDN programmability at work SDN functionality in a hybrid network. Using the design

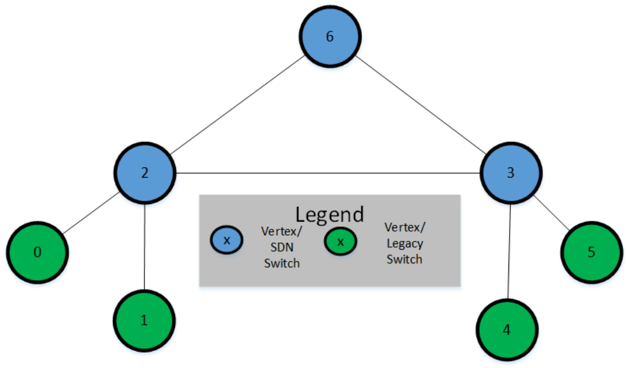


Figure 2.3. Example Hybrid Network SDN

principles outlined in [28] this research looks to outline a methodology for prescribing the minimum number of SDN devices in a hybrid network in order to realize a logical SDN using a heterogeneous equipment set. Secondly, the work will expand the hybrid research in [33] by evaluating an application running in a hybrid network.

2.3 Firewalls

Firewalls in computer networking are an homage to their less-complex cousins that operate in the physical world. Fire walls in building construction are walls meant to shield parts of a building from the more flammable sections such as a kitchen or a furnace [35]. In reality, firewalls slow down fires rather than completely stopping them in their tracks. Furthermore, walls in the real world served to segregate parts of a house into definable sections. Walls also kept bad entities out and provided a safe haven for those within it. Computer firewalls embody all of the characteristics of the aforementioned and serve as a barrier to keep malicious entities from entering your network. According to Kurose et al., Computer Networking: A Top Down Approach [36] explains firewalls typically have three primary goals:

1. Everything must pass through the firewall: Any traffic the goes between the internet and the intranet must pass through the firewall to enforce security policy [36].
2. Authorized traffic only: Only traffic authorized by the system administrator, through security policy, is allowed to pass through the firewall. All other traffic is blocked [36].
3. The firewall must be immune to penetration: if the firewall is not under your control then the security of the whole network is lost.

Firewalls today, can filter traffic at multiple layers within the ISO network model with a predominant focus on data-link, network, transport, and application layers [35]. Firewalls are typically classified in three categories: traditional packet filters (stateless), Stateful filters, and application gateways [36]. This paper will address four types of firewalls. The simplest firewall is a traditional packet filter, also known as a stateless firewall, which simply filters traffic based on header information contained within the IP packet. The traditional packet filter will look at the header, and go down a sequential list of rules until a match is obtained. If match happens, the traditional packet filter will allow the packet access to the network. If a match does not happen, the firewall will typically drop (discard) the packet. The aforementioned matching occurs using a construct called an ACL. The ACL is the filter through which each packet is inspected and compared to. If a packet's criteria matches the criteria listed within the ACL then the Firewall will allow or drop the packet. Each interface on a router can have an ACL applied to it to properly filter traffic on a network. ACLs typically start from the least restrictive rules to the most restrictive and look visually akin to a funnel. A generic ACL is shown in Table 2.1 illustrating rules that might be applied in a stateless firewall from [36].

Action	**Source Address**	**Dest Address**	**Protocol**	**Source Port**	**Dest Port**	**Flag Bit**
Allow	222.22/16	Outside of 222.22/16	TCP	>1023	80	Any
Allow	Outside of 222.22/16	222.22/16	TCP	80	>1023	ACK
Allow	222.22/16	Outside of 222.22/16	UDP	>1023	52	-
Allow	Outside of 222.22/16	222.22/16	UDP	53	>1023	-
Deny	All	All	All	All	All	All

Table 2.1. Example Access Control List Table. Source: [36].

The next type of firewall is a stateful filter. Stateful filters have the same functionality as a stateless firewall plus some more advanced features. Stateful filters can keep track of information exchanges between entities outside and inside the network. This provides some intelligence to the firewall helping it sort between legitimately initiated "conversations" and social engineering like attacks initiated by malicious entities outside of the network. Application gateways go beyond packet filtering. Application gateways are servers that provide application specific security to the network [36]. An example use case would require a host to authenticate themselves to the application gateway which would subsequently allow connections to the outside world under the gateway's supervision.

2.4 Distributed Firewalls

The final type of firewall is a distributed firewall. A distributed firewall is based upon the assumption that those within your network are potentially untrustworthy [35]. Therefore, it is pragmatic to take internal precautions to block illegitimate behavior by posting rules within your network to stop such activity.

Firewalls are typically placed at the edge of the network between the outside world, the internet, and the inside world, the intranet. In the first distributed firewall proposed in [37] described a basic framework where a policy file was distributed to each host within a network. The policy file outlined what was allowed on a particular host and each host used a cryptographic authentication scheme to validate sender communication. This form of firewall reduced the bottleneck encountered by traditional firewalls and introduced important communication between an intranet host and its distant end receiver (e.g., a host knows if it did or did not start a conversation between itself and another entity) [37]. In later implementations, such as in Sung et al. Towards Systematic Design of Enterprise Networks 2011 [27], access control (the core of firewall functionality) is specified by distributing ACLs to an edge cut set of network devices.

2.5 Mininet Emulation Tool

Network simulation and emulation are effective tools for testing and evaluating networks because they abstract away details and provide greater flexibility and control over a test network [38]. The Mininet emulation tool is a program written in Python (with a little C) to support research, design, testing and evaluation of networks [39]. Mininet is useful for modeling Software Defined Networks because it provides a platform to cheaply test SDN networks at scale without requiring SDN enabled equipment [40]. Each virtual network switch in Mininet is emulated using Open vSwitch [an SDN enabled virtual switch] [41]. Mininet creates a network utilizing virtual hosts, devices, links, and controllers running in a Linux environment [39]. Mininet is useful because it is faster than equivalent full virtualized networks because its scalability, speed, installation speed, and bandwidth [39].

2.6 USMC Networking Technology and Policy

The USMC is a unique organization within the U.S. DoD. They are the smallest of the Armed Services (with the exception of the United States Coast Guard who normally falls underneath the Department of Homeland Security (Department of Homeland Security (DHS)) and can fall under the Department of the Navy Department of the Navy (DoN) during times of war) [42]. The legal roles and responsibilities of the United States Marine Corps are outlined by the United States Code Title 10 section 5063 [43]. However, these responsibilities are best articulated by the USMC's 35th Commandant James F. Amos:

> The Marine Corps is America's Expeditionary Force in Readiness - a balanced air-ground-logistics team. We are forward deployed and forward engaged, shaping, training, deterring, and responding to all manner of crises and contingencies. We create options and decision space for our Nation's leaders. Alert and ready, we respond to today's crisis with forces available today. Responsive and scalable, we team with other Services, inter-agency partners, and allies. We enable and participate in joint and combined operations of any magnitude. A "middleweight force", we are light enough to carry the day upon arrival, and capable of operating independent of local infrastructure. We operate throughout the spectrum of threats - irregular, hybrid, conventional - particularly in the gray areas where they overlap. Marines are always ready to respond whenever the nation call...wherever the President may direct. Source: [44].

USMC units operate in groups called a Marine Air Ground Task Force (MAGTF). The MAGTF is a symbiotic grouping of different Marines with complementary capabilities. Grouping these Marines into MAGTFs creates self sustaining and supporting task organized units who are capable of executing a wide variety of missions from Humanitarian Assistance / Disaster Relief Humanitarian Assistance Disaster Relief (HADR) to Counter Insurgency Operations to full scale war. The word MAGTF "Marine Air Ground Task Force" is self describing and illustrates the basic capabilities that come organically to each MAGTF. Each MAGTF has four elements, a Ground Combat Element (GCE), a Air Combat Element (ACE), a Logistics Combat Element (LCE), and a Command Element (CE). These elements each bring skills and capabilities that create the, "balanced, air-ground, combined-arms formations under a single commander" [45]. Not explicitly stated, but critical to the functionality of the MAGTF, are the communications Marines whose responsibilities lie in the, “planning, installation, operation, displacement and maintenance” [46] of C2 networks.

2.6.1 The Communications Occupational Field

The communications field is large and multifaceted and encompasses every telecommunications task from operating hand-held radios, managing enterprise information technology systems, and maintaining equipment to enable command and control for the MAGTF commander [46]. Each occupational field within the Marine Corps is given a four digit code that represents the breadth of job specialties that have similar functions and capabilities [46]. The communications OccFld has the code 06xx and accounts for the more than 18 job specialties within the community. The Marines who enable C2 for the Marine Corps are known as Communicators.

The communications field enables C2 for every unit within the Marine Corps from a fire team of four Marines to the Marine Expeditionary ForceMarine Expeditionary Force (MEF) of over forty-five thousand Marines. This includes everything from tactical hand-held radios to large scale enterprise information technology networks. Communications and communicators tie the MAGTF together. Moreover,“MAGTF C2 enhances lethality and effectiveness across the ROMO through better decision making and shared understanding” [13]. Artillery and aircraft cannot drop precision guided munitions without positive communications between the requester and the pilot. Logisticians cannot resupply Marines in need nor can intelligence analysts disseminate much needed information without reliable communications. Communications and communicators are the proverbial nervous system to the body of the Marine Corps and the MAGTF

All Marine Communicators are trained at the Marine Corps Communication-Electronics School Marine Corps Communication-Electronics School (MCCES) in 29 Palms, California. Each prospective communicator learns universal 06xx skills at the Basic Communication Course (BCC) [46], [47]. A Marine’s performance in BCC dictates their follow on training and, ultimately, their career path. Most communicators have three career paths: transmissions (radio frequency), networking, or data systems. Other career paths include spectrum management and information security. The primary supervisors of communications Marines are Communications Officers (0602) and Chiefs (0699) who are responsible for the Planning, Installation, Operations, and maintenance PIOM of C2 architectures in the Marine Corps [13], [46].

A large part of the 06xx skill set involves planning, engineering, configuring and maintaining IP-based networks and technologies. In fact, approximately one half of the communications officer training curriculum is dedicated towards IP-based technologies. In the experience of the author, the Marine Corps has heavy reliance on networked systems to enhance decision making, enable collaboration, and speed up war fighting processes. The Marine Corps C2 strategy states that, "Marines around the globe, both in garrison and forward deployed, require a networked C2 environment that is ready, responsive, and resilient" [48].

2.6.2 Marine Corps Networks

Networking is a critical part of USMC C2 architectures and is arguably the center of gravity of current Marine Corps C2 plans. The Marine war fighting functions of fires, intelligence, maneuver, force protection, and logistics all utilize USMC IP networks to quickly and accurately coordinate their functions, in real time, while providing cognizance of their actions to a higher headquarters. Each one of these functions can be run without IP-based networks; however, the preferred method of operation within the Marine Corps is with an IP-based solutions.

Marine Corps operations are heavily invested in and reliant upon a redundant and effective C2 infrastructure [48]. Due to the complex nature of telecommunications and information technology enterprise networks, communications units are either in an exercise or preparing for one. The complexity and importance of Marine Corps communications networks do not allow for procrastination or ill-conceived plans. To successfully provide services on "game day" communications Marines must constantly prepare and hone their skills sets to successfully execute their mission. This means a series of preparatory evolutions occur prior to every exercise and operation. The first preparatory evolution is called a bootstrap exercise or bootstrap exercise (STRAPEX).

The STRAPEX in the simplest sense is a formal validation of equipment settings. A STRAPEX involves systematically connecting every piece of equipment that is going to be used and validating its functionality while in a garrison environment (office). STRAPEXs involves setting up radio networks, internet services, and transmission systems together to form a coherent C2 architecture. The STRAPEX is essential to ensuring that all of the right equipment is available, functional, and configured properly. The STRAPEX serves as an opportunity to shake out planning, equipment, and configuration shortfalls before a unit uses them operationally. If the system works in garrison, it serves as a confidence builder and tangible measure of performance that the system will work in the field.

In a STRAPEX every issue, equipment/personnel/ or process-based, is troubleshot with the full complement of Marine communicators in a unit. The STRAPEX itself is conducted indoors, where possible, and all the equipment is setup in a localized space to assist in equipment setup, configuration documentation, troubleshooting, tear-down, and packing. A STRAPEX can be more difficult and tedious than the operation itself. It is the point where communicators finalize and iron out any and all configuration settings for an exercise. This can involve deploying hundreds of computers, radios, and phones on top of dozens of transmissions pieces, server racks, networking devices, and encryption devices. A STRAPEX can take as long as a month for a large exercise. The largest amount of time is typically spent setting up the network and aligning the myriad of equipment pieces and suites to the overall Quality of Service (QoS) and security strategy. Once the architecture is functional, every setting for every piece of equipment is meticulously documented, shut down, and packed away for immediate use in a COMMEX, exercise, or operation.

A communications operation to prepare to support Marine Corps Operations is called a communications exercise or Communications Exercise (COMMEX). The purpose of the COMMEX is to ensure digital services are operational down to the user level and serves to sharpen the skills of the communicators and prepare the unit for an upcoming operation. The primary difference between the COMMEX and the STRAPEX is the environment [33]. A COMMEX is executed in the field to replicate or simulate the actual environment that the operations will be conducted in. Because each site is separated by large distances, the whole C2 system must be configured and troubleshot over the radio. This method of installation is how the communications architectures are built in reality and test each Marine's knowledge, skill, and patience beyond what can be done in a STRAPEX. Any shortfall in equipment, personnel, or knowledge must be worked around until services are validated at each site.

The USMC uses traditional networking and employs complicated plans to deploy network wide capabilities. Thousands of man hours are spent planning, installing, operating, displacing, and maintaining these networked systems. The amount of time troubleshooting and administering these systems is only going to grow worse as the number of technologies and services increases. The Marine Corps needs to either reduce its information technology footprint and or streamline the manner in which deploys and manages information technology systems. This is whereSDN could benefit the USMC. SDN could be the means to alleviate the cumbersome administrative and operational load that traditional networking presents and creates an opportunity to leverage the flexibility of programmatic networking. Any policy, application, tool, or middleware box could be distilled into an application. Moreover, conducting network operations and maintaining cognizance over network health could easily be maintained through the control plane.

2.6.3 DoD Acquisitions

The military industrial complex is a large bureaucratic process known for its inefficiency [49], [50]. It can take years, even decades, from inception to conception of a program of record [50]. This means that personnel working within the acquisition branches of the armed services must be able to anticipate needs of their organizations years and even decades into the future. Moreover, the whole process is driven by stringent fiscal constraints that do not account for change. The budget of the DoD is the responsibility of the legislative branch. This creates two issues: first congress must approve the budget. Secondly, the budget has to accurately predict development, production, and maintenance costs for a future technology. To put this in perspective, congress has only passed a budget four times in the last 42 years (Fiscal years 1977, 1989, 1995, and 1997) necessitating continuing resolutions every time the budget wasn't passed [51]. The point here is that the process, while both legal and well meaning at its inception, hinders the purchase of timely, useful, and fiscally responsible equipment for the armed forces. Congress has corroborated the fact that the acquisition process is broken and needs fixing [49], [50].

Generally, the larger and more expensive the need, the more time and oversight it will take to field that need. Moreover, as stewards of the American public, fiscal responsibility is an inherent guideline when procuring new equipment. Military viable systems that fit within budgetary constraints are an ideal and fall directly in line with the fiscal values of the USMC. The USMC has a historical precedence for frugality canonized in General Victor H. Krulak's "First to Fight" [52]. Given this precedence, and the difficulties of the DoD acquisitions process, the logical method of improving USMC networks is a frugal approach that can be aligned with the current acquisitions construct and a slow fielding process. This research seeks to show that SDN networks are a fiscally responsible choice because they can be fielded incrementally and implemented in the form of hybrid networks consisting of traditional networking equipment and SDN equipment. Moreover, this research is looking to show that SDN procurement can save the Marine Corps money while improving operating cost, efficiency, and capability.

2.7 Related Work

Similar SDN research has been conducted by Weitzel in [33] and Levin et al. in [28]. Weitzel focused on primarily on the feasibility of integrating SDN technology into Marine Corps tactical networks as a means to alleviate the administrative load of administering a large tactical network. Weitzel demonstrated that an SDN can help to automate many low level tasks that would otherwise take up operators time and effort [33]. More importantly, Weitzel showed that hybrid SDN networks work [33]. Levin et al. focused on innovative ways to incorporate SDN technology into traditional networking architectures. Levin et al. stated that deploying an SDN is difficult and a complete replacement of traditional equipment is impractical if not impossible [28]. Therefore, Levin et al. proposed a hybrid SDN architecture called Panopticon to realize the benefits of SDN without necessitating the replacement of the whole architecture [28]. Lastly, Sung et al. in [27] proposed a systematic design approach for VLANs and reachability control [27]. Sung et al.'s algorithm for reachability control and VLANs demonstrated that systematic approach is necessary and feasible for large enterprise networks to improve performance, reduce configuration error and the resulting vulnerabilities that come from them [27].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3: Experimentation

We need to streamline our ability to evaluate and acquire advanced technologies to ensure we gain advantages from innovations faster than our competitors and adversaries.

—The Marine Operating Concept [11]

The following chapter outlines the research's experimental design to provide sufficient technical depth and adequate coverage of the design space to allow for follow on research. The experiments sought to simulate Marine Corps networks, examine the strengths and weaknesses of the distributed firewall program and Open vSwitch, and offer a software defined solution utilizing both a hybrid and homogeneous equipment set. The homogeneous network design reflects an idealized equipment set comprised solely of SDN equipment while the heterogeneous architecture reflects the fiscal and physical realities of incremental network upgrade and improvements as stated in [28].

3.1 Design of Experiments

The experiments were designed to provide empirical evidence to answer the research questions. Does the distributed firewall algorithm, adapted from [27], work in an SDN environment and how well? How viable is Open vSwitch as a virtual switching platform for the United States Marine Corps? Are Software Defined Networks a viable network model for the United States Marine Corps and how should they be implemented? How well does a hybrid SDN network operate utilizing the distributed firewall program?

3.1.1 The Hardware and Software

The experiments were run on one physical system using a virtual machine to emulate the computer networking environment. The experiment is considered an emulation because it replicates the networking environment and behavior in software using the Mininet emulation tool. The Mininet emulation tool has been shown to be a reliable, scalable, and accurate tool for SDN research [40], [53], [54], [55]. This made Mininet an ideal choice for the SDN research. The following is the specification of the physical system running the experiments:

Computer Model ASUS Strix Republic of Gamers (ROG) GL502VMK Laptop
Processor Intel Core i7 7700HQ @ 2.80GHz
System Type 64-bit OS, x64 based processor
Memory 24 GB RAM: 8 GB DDR4 2400MHz SDRAM Onboard Memory, 16GB DDR4 2400MHz SDRAM SO-DIMM socket
Storage Hard Drive: 1 TB 7200RPM SATA HDD, Solid State: 256GB PCIE Gen3x4 SSD
Operating System Windows 10 Home

The following is a list of the software used to build and create. Detailed instructions on how to build and learn the software suites are detailed in the appendices. All of the software is open source and freely available online.

Hypervisor Oracle VM VirtualBox Version 5.2.26 r128414 (Qt5.6.2)
SDN Network Operating System Open Networking Operating System (ONOS version 1.15.0d7b6c33) is a "carrier-grade" SDN Network Operating System (NOS) designed with transitioning from traditional to SDN networking environments. [56]
Network Emulation Software Mini-net Emulation Software (version 2.2.1) is used to create virtual networks running actual code (such as Open VSwitch) on a VM. [39]
Virtual SDN Switch Open vSwitch (version 2.5.5) [41]
Network Protocol Analyzer Wireshark (version 2.6.6)

3.1.2 Demonstrating SDN's Programmability: Distributed Access Control

One of the benefits of SDNs is its programmability [15]. The bar for entry is a basic understanding of programming. Using these skills, a network administrator or another interested party is capable of creating a program and incorporating it onto an SDN. Traditional networks do not have the flexibility of SDNs and are constrained to vendor specifications. Programs cannot be written in high level programming languages and subsequently loaded onto network devices much less installed onto a single machine and distributed network wide. As an example, a distributed firewall is not a feature that is available in traditional networking equipment. Distributed firewalls are a service that you can purchase or a capability that you can create using a combination of software, hardware, and or careful configuration. To enable this capability in a traditional network, additional equipment and or software would need to be incorporated into the existing architecture. This is not the case in an SDN environment. To demonstrate the programmable nature of SDN and experiment with automated distributed firewall operations, a program was written to create a distributed firewall in accordance with the reachability control specifications described in [27].

The program is installed on the controller (ONOS controller) which populates selected network devices with operator defined flow rules (modeling access control lists) using the systematic reachability framework described in [27]. Furthermore, the program uses an algorithm to determine the minimum edge cut set of devices to place flow rules on based upon a source ***s*** and destination node ***t***.

1. Let $R(s,t)$ be defined as the set of flow rules that controls traffic from source ***s*** to sink ***t***.
2. Let $E\{R\}$ be defined as the set of routers with flow rules to control traffic from source ***s*** to sink ***t***.

The effect of the program is that any IP-based traffic originating at the source *s*, destined for the sink *t* will be filtered by a network device in the set $E\{R\}$ containing $R(s,t)$ (flow rules) within its flow tables. In layman's terms, the program ensures all traffic from point a to point b is filtered. The difference between the "firewall" in this research and a traditional firewall is that ***any*** SDN enabled switch can act as a firewall through flow rules. In a traditional setup, middle-boxes, host-based software, and network monitoring tools would be required to enable a similar capability.

3.1.3 Program Components

Cormen et al. in [57] define a minimum cut of a network as, "is a cut whose capacity is minimum over all cuts of the network." The minimum edge cut in the case of the aforementioned distributed firewall, is similar to a border checkpoint that all cars must pass through or a passport check at an international airport arrivals section. In a network, a flow is defined as the rate at which some unit moves (in this case packets) [57]. In a computer network graph, edge weights can be defined as any metric other than distance such as cost or time [57]. In this case, the summation of edge weights along a specific path equate to a flow and can be used to ascertain useful information about a network. For example, the Ford-Fulkerson method was developed in the to determine the minimum number of locations to bomb (minimum edge cut set) on the Soviet rail system to completely and accurately cut off Eastern Europe from the Soviet Union [58].

The maximum flow is the process of finding a possible flow from source ***s*** to sink ***t*** that is the maximum [57]. In the case of maximum flow, the term maximum is defined as the capacity of the minimum cut (the summation of the weights of all the edges in the minimum cut) [57]. The algorithm that was used to find the Maximum Flow Minimum Cut was a variation of the Ford-Fulkerson algorithm called the Edmonds-Karp Algorithm (Figure 3.1 adapted from [59]). The basic premise of Edmonds-Karp is that it uses a Breadth First Search (BFS) in the Ford-Fulkerson implementation to pick a minimum edge path [59], [57]. The overall time complexity of Edmonds-Karp algorithm, in the worst case is $O(VE^2)$ [57]. The specific implementation adapted from [59] in the code has a time complexity of $O(EV^3)$ because it uses an adjacency matrix for the BFS.

```
initialize flow f to 0;
while there exists an augmenting path p : using breadth first search do
    augment flow f along p ;
    return f;
```

Figure 3.1. Ford Fulkerson: Edmonds-Karp variation (G, s, t). Source: [57].

As an example, examine Figure 3.2 from [59]. The source ***s*** is node 0 while the sink is ***t*** node 5. The min-edge cut set in this case is the set of edges: 1 -> 3, 4 -> 3, and 4 -> 5. If this graph was a computer network, then devices 1,3,4, and 5 would be the vertices of the edges in the minimum-edge cut set. In the context of a distributed firewall, devices 1,3,4, and 5 would seem to require flow rules. The set of vertices in the minimum edge cut set (1,3,4,5) works; however, it is not optimized in the sense that it needlessly installs rules on devices that do not need it.

Visually you can see in Figure 3.2 that vertexes 3 and 5 ***do not*** need flow rules as vertices 1 and 4 will filter incoming traffic before it reaches either vertex 3 and 5; therefore, placing rules

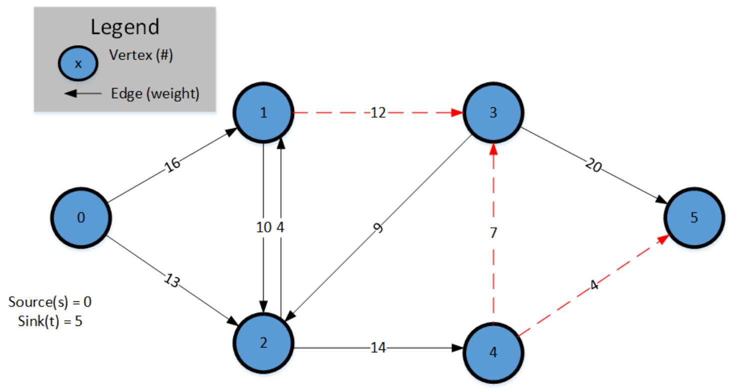


Figure 3.2. Sample Flow Network (Directed Graph) with Max Flow Min Cut Set Calculated. Source: [59].

on vertices 3 and 5 is a waste of system resources. In the case of a small enterprise, vertices like 3 and 5 will likely be handling large amounts of IP traffic meaning high throughput is a desirable performance specification. Each packet that comes into an interface is inspected and run through the long list of flow rules that is present on each device. The longer the list the longer the inspection time. If the traffic is high, a network will experience an increase a significant growth in delay as the number of rules grows [60]. Given the aforementioned problem, optimization was required to prevent overburdening the network.

As mentioned before, sometimes only one vertex of an edge requires flow rule installation to successfully filter all traffic. Moreover, a single vertex can satisfy multiple edges in a minimum edge cut set. Algorithms do exist to determine what is the called the minimum node/vertex cut set such as algorithm 11 in [61], which is used in the Python Library NetworkX [62]. These algorithms were not used in this research. To minimize the number of nodes that required flow rules, Dijkstra's shortest path algorithm (See Figures 3.3 & 3.4 for algorithms — See [63] for source code) was used to determine the shortest path from the source ***s*** to the set of vertices in the minimum edge cut set. The running time Dijkstra's algorithm is OV^2 [57]. The premise is to find the first vertices in the edge cut set that would be encountered along all known paths from ***s*** to ***t***. Vertex 1 has a cumulative weight of 16, vertex 3: 28, vertex 4: 27, and vertex 5: 31. For edge 1->3 vertex 1 has the lowest weight. For edge 4->3 vertex 4 has the lowest weight. For edge 4->5 vertex 4 has the lowest weight. This implies that vertexes 1 and 4 are the best candidates for flow placement.

```
Data: $u, v, w$
/* $nodeu, nodev, weightw$ */
Result: $void$
/* $d[v]$ is a shortest path estimate */
/* $w(u, v)$ is the weight of the edge between $u$ and $v$ */
/* $\pi isthenewnodevaluedistance$ */
1 if $d[v] > d[u] + w(u, v)$ then
2     $d[v] \longleftarrow d[u] + w(u, v)$;
3     $\pi[v] \longleftarrow u$;
```

Figure 3.3. $Relax(u, v, w)$ Pseudocode. Source: [57]. The process of relaxing an edge consists of systematically testing whether we can improve the shortest path to v by going though u and updating $d[v]$ and $\pi[v]$ accordingly.

```
Data: $G, s$
/* Graph representing a network and source node s */
Result: $D$
/* vector representing distance of shortest path from $s$ to every
   other node in $G$ */
1 Initialize: Single Source Shortest path $(G, s)$;
2 $S \longleftarrow \emptyset$ ;
3 $Q \longleftarrow V[G]$;
  /* Min-Priority Queue of vertices in $G$ */
4 while $Q \neq \emptyset$ do
5     $u \longleftarrow EXTRACT - MIN(Q)$;
      /* extract minimum value */
6     $S \longleftarrow S \cup \{u\}$;
7     for each vertex $v \in Adj[u]$ do
8         Relax $(u, v, w)$;
```

Figure 3.4. $Dijkstra(G, w, s)$ Algorithm Pseudocode. Source: [57]. Dijkstra's algorithm solves single source shortest paths problems on weighted directed graph $G = (V, E)$ for non-negative weight edges *w*.

Another way to calculate the optimal flow placement is to use hop count from the source s.

Vertex 1 has a count of 1, vertex 3: 2, vertex 4: 2, and vertex 4: 2, and vertex 5: 3. For edge 1->3 vertex 1 is the lowest hop count. For edge 4->3 both vertexes are equal; however, vertex 1 already accounts for edge 1->3 therefore, vertex 4 is optimal. Lastly, for edge 4->5 edge 4 has the lowest hop count and account for both edges 4->3 and 4->5. Figure 3.2 illustrates the flow rule placement optimization. Flow rule optimization, in this case, results in a 50% cost savings in terms of reachability control flow rule placement.

To make the maximum flow minimum cut theorem applicable to computer networks we made the assumption that a computer network could be modeled as an undirected graph versus a directed graph in 3.2. This was implemented by making reverse edges of equal weight (see Figure 3.5).

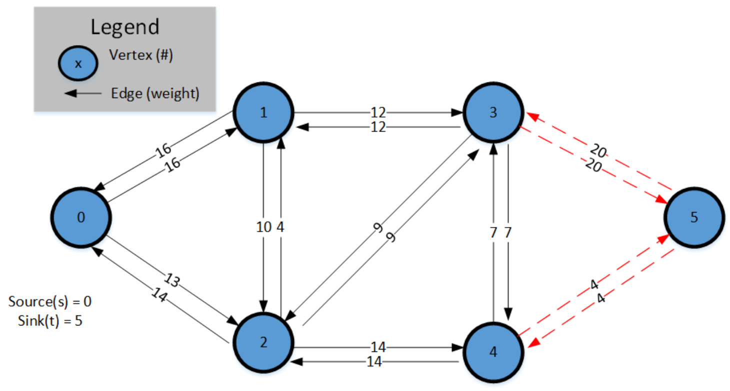


Figure 3.5. Undirected Graph Converted from a Directed Graph, from [59], with Max Flow Min Cut Set Calculated.

In the case of the undirected graph depicted in 3.5 the ***s-t*** cut with ***s=0*** and ***t=5*** results in the edges between vertices 3, 4, and 5. The optimal choice of flow placement in this example would be nodes 3 and 4.

The aforementioned flow rule placement criteria are a hybrid of several placement strategies proposed in [27] for reachability control. [27] proposed four different reachability control strategies for ACL placement. The variable $b(r)$ represents the number of ACL/Flow rules placed on a switch and $c(r)$ represents the ACL/Flow Rule limit that can be configured on switch ***r***. The fifth strategy is the strategy for the distributed firewall program.

1. **Minimum rules (MIN) strategy**: Minimize the total number of rules installed

network wide [27].

$$Minimize \sum_{r} b(r) \tag{3.1}$$

2. **Load Balancing (LB) strategy**: Spread load the processing overhead across the network to avoid over taxing any singular network device [27].

$$Minimize \max_{r} \{b(r)\} \tag{3.2}$$

3. **Capability-based (CB) strategy**: Provision reachability controls in accordance with network device capability (filtering capacity) [27].

$$Maximize \min_{r} \{c(r) - b(r)\} \tag{3.3}$$

4. **Security centric (SEC) strategy**: Place reachability controls (ACL/Flow Rules) as close as possible to the source to reduce security risk. Let ***h(f)*** represent hop count from the router on which ***flow rule*** is installed to the gateway router of the traffic sources targeted by $\boldsymbol{f}$. $\boldsymbol{H}$ represents the average h values averaged over all rules placed on to the network [27].

$$Minimize\ H \tag{3.4}$$

5. **Per-Rule SEC strategy (Min Min Cut)**: The previous four strategies are called offline solutions since they assume the entire set of flow rules required for the network to be known *a priori* and they aim to find optimal solutions for placing the entire set in one iteration. For this thesis, we focus on an online placement strategy where firewall flow rules are placed one rule at a time when and in the order that they become necessary. This is because we consider placing firewall rules at the internal switches of the network, not simply at its ingress point(s). The internal firewall rules are heavily driven by internal traffic, which are sometimes difficult to anticipate, particularly upon topology changes. More specifically, we enforce the SEC strategy per rule, by placing the rule at the switch closer to source **s** for each edge in the given min edge-cut set. Consider the more general case where multiple min edge-cut sets exist for a source destination pair. Let **EC(s, t)** denote the set of min edge-cut sets. Let **h(u)** denote the minimum hop count from **s** to a switch **u**. The following defines

a collective minimum distance to **s** for all edges in an edge-cut set **c**:

$$H(c) = \sum_{\text{edge } <a,b> \in c} \min(h(a), h(b)) \tag{3.5}$$

Now the Min Min Cut strategy can be formulated as a two-step process as follows. Step 1 finds the min edge-cut set **EC*** whose edges are collectively closest to **s**:

$$EC^* = \arg\min_{c \in EC(s,t)}(H(c)) \tag{3.6}$$

Step 2 then chooses the closer switch to **s** for each edge in **EC***.

The distributed firewall program combines several aspects of the aforementioned ACL/Flow Rule placement strategies. First, the program places flow rules as close as possible to the source ***s*** which is a characteristic of the SEC strategy. Secondly, the program minimizes the amount of the vertices (SDN switches) that have reachability control mechanisms (flow rules) thereby minimizing the total number of rules installed on the network. Minimizing rules on the network is line with the MIN strategy. Lastly, the program pulls each edges weight from an edge weigher method within the ONOS Java Application Planning Interface (API) and assigns that weight as an integer representing reachability and weight in the adjacency matrix data structure. The weight of each edge can be used to devise a capability-based strategy that combines the weight of the link (cost) and the $c(r)$ of the SDN device as a metric to place flow rules on. Overall, the distributed firewall program is a combination of the MIN and SEC methods from [27] with the ability to incorporate the CB strategy if necessary. The correctness and feasibility criterion for reachability control from [27] state:

- ***Reachability matrix:*** Consider a network with ***N*** VLANS. The network's reachability policy can be completely described by an N by N reachability matrix, denoted by M_R, where element M_R***(i,j)*** denotes the maximum RS that will always reach an intended destination host in VLAN ***j*** if originated by a host of VLAN ***i***.
- ***Managed event set:*** The resilience requirement of a networks reachability control policy can be completely described by a managed set, denoted by E_m, with each element in the set specifying a topology-changing event to which the network must respond without causing the reachability matrix to change.
- ***Correctness criterion:*** The network's reachability matrix is invariant and as specified

in M_r under all events E_m.

- ***Feasibility criterion:*** Let ***c(r)*** represent the limit on the total number of ACL rules that can be configured on a router ***r***, including all of its interfaces and in both traffic directions, without overloading ***r***. Let ***b(r)*** be the number of ACL rules that has been configured on router ***r***. Then $\forall r, b(r) \le c(r)$. [27]

The same feasibility and correctness criteria apply to the distributed firewall program and will be measures of effectiveness for program evaluation. The, the Min Min Cut strategy runs in polynomial time as each of its sub-components (Dijkstra, Edmonds-Karp, etc.) run in polynomial time. The pseudo code for the distributed firewall program is shown in Figure 3.6 and serves a polynomial time heuristic for node selection.

```
Data: $s, t, G$
/* input source, sink, and adjacency matrix */
Result: flow rule placement on min min cut
$s, t \longleftarrow$input source and sink;
$G \longleftarrow adjacencymatrix$ with link weights from ONOS;
Array result $\longleftarrow \emptyset$;
Hashmap nodedistance;
$shortestpath \longleftarrow Dijkstra(G, w, s)$;
$result \longleftarrow EdmondsKarp(G, s, t)$;
for edge: result do
    for vertex: edge do
        $distance1 \longleftarrow shortestpath[vertex1]$;
        $distance2 \longleftarrow shortestpath[vertex2]$;
        if $distance1 < distance2$ then
            $nodeddistance.add(vertex1, distance1)$;
        if $distance1 > distance2$ then
            $nodeddistance.add(vertex2, distance2)$;
        if $distance1 == distance2$ then
            $nodeddistance.add(vertex2, distance2)$;
            $nodeddistance.add(vertex1, distance1)$;
for object : nodeddistance.keyset() do
    add flow rule(s);
```

Figure 3.6. Distributed Firewall Program Code

3.2 Marine Corps Infantry Task Organization

The networks in these experiments are modeled after Marine Corps Infantry units. The base unit of Marine Infantry is the Marine fire team. The fire team has four members: one fire team leader, one automatic rifleman, one assistant automatic rifleman, and one rifleman. The next unit is the squad which consists of three fire teams and one squad leader. The next unit up is the rifle platoon, led by a Second Lieutenant which consists of three rifle squads. Three rifle platoons and a weapons platoon makes a company and three companies and a headquarters company makes a battalion. The pattern of three subordinate units and one headquarters unit repeats itself throughout the Marine Corps up to the MEF level (keep in mind there are exceptions to this rule). Figure 3.7 depicts the organization of the Infantry Battalion and Regiment to include the number of enlisted and officer personnel for both the U.S. Navy (USN) and the USMC. These numbers indicate a high upper limit of potential users on the tactical network; however, take these numbers with a grain of salt as the majority of users are going to be leaders which is going to be approximately 1/8 of the total population.

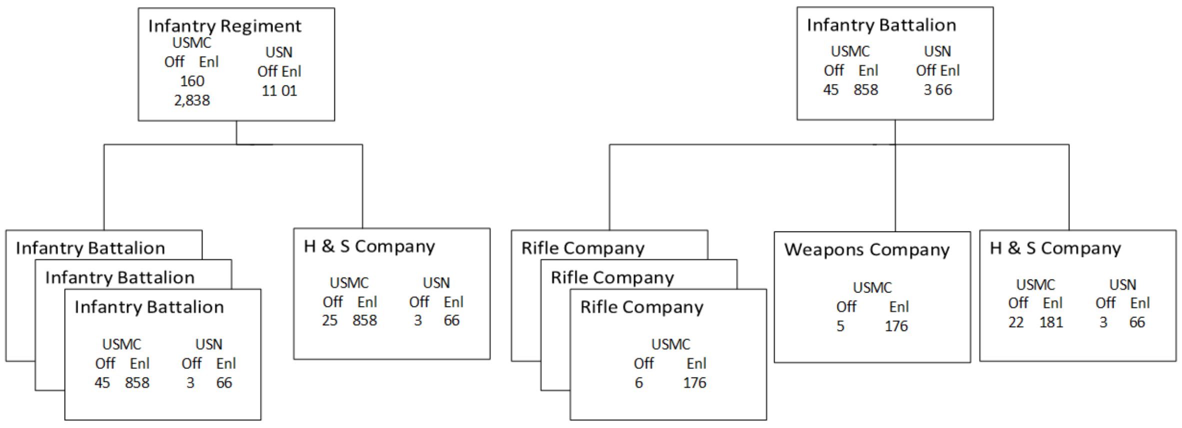


Figure 3.7. Task Organization of Marine Infantry Battalions and Regiments. Source: [64].

A Marine infantry battalion network topology will vary greatly from unit to unit and will depend on the mission. High mobility operations will depend almost entirely on tactical radios making very little use of enterprise Information Technology (IT) equipment organic to the unit. High mobility operations are more likely to utilize convenient short range and “long” haul communications such as satellite, High Frequency (HF), and Very High Frequency (VHF) communications over tactical radio. It is likely the only networking equipment that a Battalion will use will be Mobile Ad-Hoc Networking (MANET) radios, a Very Small Aperture Terminal Very Small Aperture Terminal (VSAT) (Small Satellite Dish) and or a Network on the Move (NOTM). The aforementioned equipment enables a battalion to communicate quickly and conveniently while on the move. Military Operations Other Than War (MOOTW), such as most of Operation Iraqi Freedom Operation Iraqi Freedom (OIF) and Operation Enduring Freedom Operation Enduring Freedom (OEF) tended to depend heavily on IP based technologies requiring long installation, operation, and maintenance hours. Network topologies in MOOTW environments are larger and more robust due to the permissiveness of the environment and high demand for services. Both types of operations require a network; however, for the purposes of testing a larger and more robust network, these experiments will assume that the infantry battalion is operating in a permissive MOOTW and the network architectures reflect the experiences of the author.

3.3 Experiment 1: Validation and Stress Test

The purpose of Experiment 1 is to validate the distributed firewall program's functionality in a controlled setting. The topology is a tree, as shown in Figure 3.8, and there is only one data path from end to end. The maximum flow minimum cut set in this case will naturally be an edge between two devices closest to the source s and therefore simple to validate flow rule addition and enforcement. Lastly, network performance parameters can easily be measured in this environment and used as a baseline for the rest of the experiments to follow. Remember that SDNs only allow traffic that is explicitly defined within an SDN switch's flow table. Any traffic that a switch receives for which there is no match within its flow table is either dropped or sent to the controller for inspection. By default in ONOS, all flows are denied unless explicitly allowed. For the purposes of testing, a reactive forwarding program is enabled by default. The reactive forwarding program allows any traffic on the network by installing flow rules for any traffic that is encountered on the data plane. This program allows us to use commonly used networking utilities such as iperf, ping, and trace route without ONOS denying the traffic.

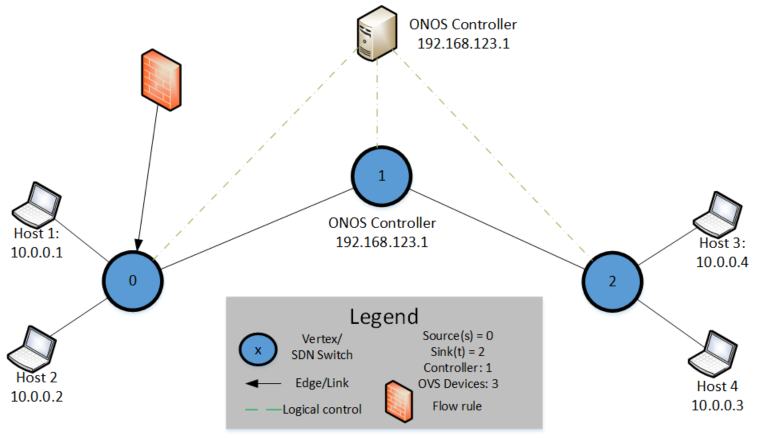


Figure 3.8. Experiment 1: 2 x 2 Tree Topology with Three Network Devices

3.3.1 Experiment 1 — Test 1: Program Functionality

Hypothesis: The distributed firewall program will determine the Min Min Cut set, pick the closest Open vSwitch (OVS) (Open vSwitch) and apply flow rules on it. The effect on throughput should be negligible.

Procedure: The Distributed Firewall program is a command line program that takes three arguments. The first two arguments are ***s*** and ***t*** respectively with the third argument taking the number of flow rules that you want to install. The flow rules used in this experiment simply deny traffic. Denying traffic from a specific IP space is easy to verify and configure. Figure 3.10 depicts a sample Client URL (cURL) request containing a JSON file to add a flow rule to a specific device on the network.

The cURL request takes advantage of ONOS's Representational State Transfer representational state transfer (REST) API. The REST API is a method of incorporating outside programs to interact with ONOS. ONOS's flow rules can also be created within the ONOS Java API. Both methods are valid for deleting, modifying, and or adding flow rules to ONOS. For the purposes of experimentation, the ONOS Java API will be used to create rules for speed, repeatability, and automation through shell scripting. Once the flow rule is added to a specified device, its functionality will be validated. The speed of the flow rule deployment will also be measured using metrics that will be gathered during the experiment using Wireshark.

Flow rule deployment and installation speed will be measured through detailed examination; the pcap files produced through shell scripting. Pcap files are capture files produced by the Wireshark packet analyzation program that contain information about every packet that passes a specific interface(s) on a network. This includes the header and payload information of every packet as well as timing information in reference to the first packet that passes the specified interface. Every rule *r*, or set of rules *r1, r2, ... , rn*, that is installed on a switch from a controller takes time *t* for installation to complete. The installation time is a product of the handshake that takes place between controller and switch. The handshake consists of the following (see Figure: 3.9): first the controller sends a `OFPT_FLOW_MOD`, at time *i*, message to the switch dictating flow rule installation. Every `OFPT_FLOW_MOD` message has a transaction ID associated with it. Transaction IDs can be unique to each flow rule or can be used across a set of flow rules. Following the `OFPT_FLOW_MOD` message, an `OFPT_BARRIER_REQUEST`, with the same transaction ID as the `OFPT_FLOW_MOD` message, is issued to the switch forcing the switch to process all previously received messages prior to the `OFPT_BARRIER_REQUEST` [65]. The `OFPT_BARRIER_REQUEST` ensures the `OFPT_FLOW_MOD` is processed. Lastly, once the switch has complied with the `OFPT_BARRIER_REQUEST` it sends an `OFPT_BARRIER_REPLY`, at time *j*, (with the requisite transaction ID) to the controller indicating all messages have been processed up to the `OFPT_BARRIER_REQUEST`. The difference in time between sending the `OFPT_FLOW_MOD` and receiving the `OFPT_BARRIER_REPLY` is the installation time. Therefore, flow rule installation time t for rule *r* is $t = j - i$. For multiple rules, the total installation time is the difference between the first `OFPT_FLOW_MOD` request time and the last `OFPT_BARRIER_REPLY` (keeping in mind all rules or set of rules must be replied to).

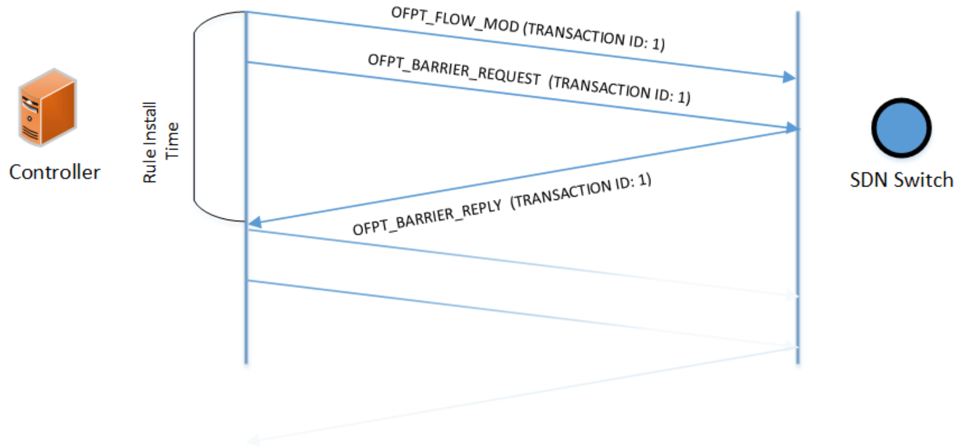

Controller
Rule Install Time
OFPT_FLOW_MOD (TRANSACTION ID: 1)
OFPT_BARRIER_REQUEST (TRANSACTION ID: 1)
OFPT_BARRIER_REPLY (TRANSACTION ID: 1)
SDN Switch

Figure 3.9. Experiment 1: Rule Installation Process and Timing

```

curl -X POST --user onos:rocks --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
   "priority": 40000,
imeout": 0,
   "isPermanent": true,
   "deviceId": "of:0000000000000001",
   "treatment": {
    "action": "DENY"
   },
   "selector" :{
     "criteria": [
        {
        "type": "ETH_TYPE",
        "ethType": "0x0800"
      },
        {
         "type": "IPV4_SRC",
         "ip": "10.0.0.4/32"
        }
     ]
   }
 }' 'http://192.168.123.1:8181/onos/v1/flows/of%3A0000000000000001?appId=org.onosproject.flows'
```

Figure 3.10. ONOS cURL Request With Deny Rule.

Data Collection: End to end throughput and latency will be captured by analyzing iperf captures. Validating that the program works will be done through several mechanisms. The number, type, and specification of flow rules installed on a device can be validated through the same methods that flow rules can be created/modified/ or deleted: through the REST, Graphical User Interface (GUI), and ONOS Java API.

3.3.2 Experiment 1 — Test 2: Stress Testing

The purpose of test 2 will be to stress test the distributed firewall program, Mininet, and Open vSwitch to evaluate their limits and functionality under realistic network flow rules. **Hypothesis Test 2:** The same topology will be utilized in the second test; however, the number of rules installed on the Min Min Cut will be varied on a logarithmic scale up to 100,000 rules to determine the limits of single OVS switch, deployment speed and end to end latency. **Procedure Test 2:** The number of rules will be increased until the network experiences notable increases in latency and deployment speed. The type of rules will also be varied from the strict deny rules used in Figure 3.10 to rules changing traffic from one VLAN ID to another.

In order to stress test Open vSwitch, we plan to install flow rules following a logarithmic scale starting at one rule and ending at one hundred thousand rules for a total of 6 planned tests. Each test was repeated twenty-five times for statistical significance. After rule installation, network performance data was gathered by measuring bandwidth and CPU utilization performance.

Data Collection Test 2: End to end latency will be measured using Wireshark packet traces. Deployment speed of flow rules will be gathered using the ONOS Java API to record rule creation and implementation times. Wireshark captures can also be used to validate flow rule updates by validating receipt of Open Flow Modification *OPEN_FLOW_MOD* messages. Lastly, the iperf and iostat utilities will be used to measure throughput and cpu usage.

3.3.3 Experiment 1 — Test 3:

The purpose of test 3 is to evaluate the distributed firewall program's ability to react to network topology changes that change the Maximum Flow Minimum Cut Set.

Hypothesis Test 3: The distributed firewall program is programmed to react dynamically to topology changes. Once the topology changes in such a manner to change the Maximum Flow Minimum Cut set, the distribute firewall deploys flow rules appropriately. The speed of the deployment will depend on the size of the cut and the number of rules being deployed.

Procedure Test 3: The procedure will remain the same as test 2 and the only new variable will be topology changes that change the Max Flow Min Cut simulating additions/losses in connectivity. The topology will be changed such that an edge is created between the first and last node. The topology changes will simulate natural changes in network connectivity due to environmental effects, traffic, malicious entities, and insider threats. **Data Collection Test 3:** The data collection method for test 3 will be the same as the previous experiments.

3.4 Experiment 2: Infantry Regiment Testing

The Infantry Regiment reflects the current size of the infantry component of SPMAGTF deployments in support of Operation Secure Resolve. The purpose of this experiment is to test the limits of the distributed firewall program when managing larger networks of twenty-five or more SDN switches. In this experiment one regimental topology will be evaluated: Figure 3.11. Links will be created and torn down in order to simulate link losses and additions outside and inside of network operations control. Because new links cannot be added to a Mininet topology post startup we chose to create links ahead of time and tear them down prior to testing our baseline topology. Core links are depicted in green and reflect approved links between units. Adjacent links are depicted in red and reflect the links that will be used to circumvent normal traffic flow. Test 2 evaluates the distributed firewall program under realistic network conditions using conservative bandwidths and latency. LAN links use a 100 Mbits/s bandwidth and 1 millisecond delay while core links have 10 milliseconds in delay and bandwidths varying between 4 and 45 Mbits/s to simulate microwave line of sight links.

3.4.1 Experiment 2: Baseline Test

Hypothesis: The purpose of test 1 is to validate the functionality of the distributed firewall program under realistic conditions by deploying the $MinMinCut$ after traffic has begun to flow utilizing valid flow rules (ONOS Intents) and the iperf utility. $MinMinCut$ will block traffic in accordance with the deployment speeds discovered in experiment 1.

Procedure: We will utilize a shell script to automate the testing. We will start up the network and establish intents (flows) between two sets of hosts. One point to point intent will be for legitimate traffic while the other intent be illegitimate (once the distributed firewall program is deployed). The number of flow rules in this experiment will be low (100 rules) and the test will be repeated twenty-five times for statistical significance.

Data Collection: In this test we are primarily concerned with blocking illegitimate flows and allowing legitimate ones. Iperf output between two sets of hosts will be used to evaluate throughput and effectiveness of the distributed firewall program. Iperf output will give coarse grain indication of flow stop speed and impact on throughput.

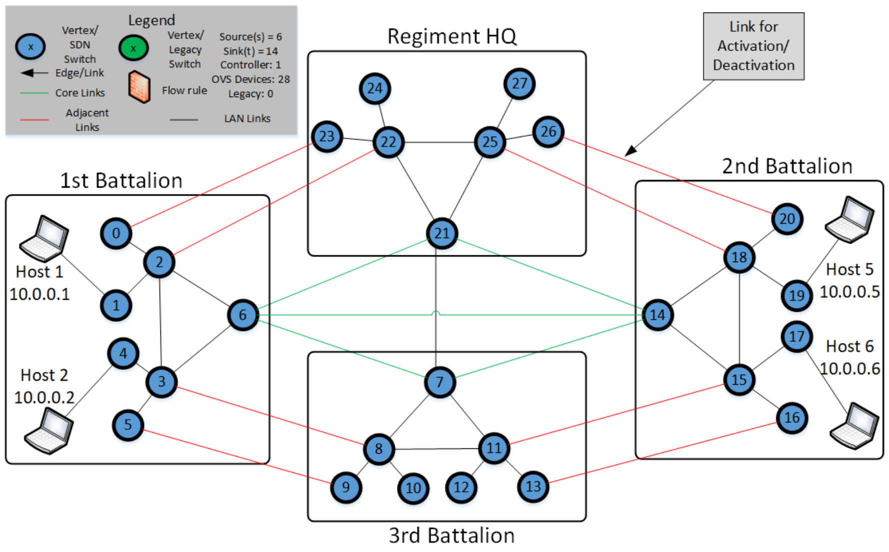


Figure 3.11. Experiment 2: Homogeneous SDN Infantry Regiment Topology

3.4.2 Experiment 2: Random Adjacent Link Establishment Test

Hypothesis: The purpose of the adjacent link establishment test is to validate that the distributed firewall program can quickly block traffic when the network topology changes in such a manner as to purposely or inadvertently create flows around the $MinMinCut$. We hypothesize that the distributed firewall program will block traffic and stop any flows within the speeds discovered in experiment 1 taking into account the latency of the network.

Procedure: We will utilize a shell script to automate the testing. We will start with connectivity using only core links. The distributed firewall program will be run and then point to point intents will be established between two sets of hosts. One intent will be legitimate traffic while the other intent be blocked by flow rules (once the distributed firewall program is deployed). The number of flow rules in this experiment will be 1000 rules. Once intents and flow rules are set, an adjacent link will be established that changes the $MinMinCut$.

Data Collection: In this test we are primarily concerned with blocking illegitimate flows and allowing legitimate ones. Iperf output between two sets of hosts will be used to evaluate throughput and effectiveness of the distributed firewall program. Iperf output will give coarse grain indication of flow stop speed and impact on throughput.

3.4.3 Experiment 2: Random Core Link Loss Test

Hypothesis : The purpose of this test is to see whether or not core link loss has a significant impact on throughput. I hypothesize that the impact will be in accordance to the rule set size as discovered in experiment 1.

Procedure: The procedure for test 1 is the same as the previous with the exception that instead of adding an adjacent link, we will tear down a core link and evaluate the effects.

Data Collection: In this test we are evaluating the throughput impact on a core link loss. Iperf output between two sets of hosts will be used to evaluate throughput. Iperf output will give insight into core link loss statistics.

3.4.4 Experiment 2: Malicious Adjacent Link Establishment

Hypothesis: Malicious Adjacent Link Establishment is to validate that the distributed firewall program can quickly block traffic when the network topology and intents are aligned such that the intent shortest path is only path on the network. This simulates using an intent that specifically defines paths along specific interfaces to circumvent flow rules. This test pits ONOS's intent framework speed against the distributed firewall program speed. We hypothesize that the distributed firewall program will block traffic and stop any flows within the speeds discovered in experiment 1 taking into account the latency of the network.

Procedure: We will utilize a shell script to automate the testing. We will start with connectivity using only core links. The distributed firewall program will be run and then intents will be established between two sets of hosts. One intent will be legitimate traffic while the other intent be blocked by flow rules (once the distributed firewall program is deployed). The number of flow rules in this experiment will be 1000 rules. Once intents and flow rules are set, all core links will be cut off segregating the origin network from the destination network. Iperf will be started as in previous experiments. Lastly, an adjacent link will be created forcing the intent framework to recalculate the shortest path and allow for iperf traffic to traverse that path. Simultaneously the distributed firewall program will calculate the *MinMinCut* and install flow rules. Traffic will be appropriately blocked at the speeds discovered in experiment 1. Illegitimate traffic may reach the destination for small periods of time.

Data Collection: In this test we are primarily concerned with blocking illegitimate flows and allowing legitimate ones. Iperf output between two sets of hosts will be used to evaluate throughput and effectiveness of the distributed firewall program. Iperf output will give coarse grain indication of flow stop speed and impact on throughput.

3.5 Experiment 3: Hybrid Network Evaluation

As mentioned by previous research [33], SDN is still an emergent technology and as such the specification is still under review by the Internet Research Task Force (IRTF) [66] and the Internet Engineering Task Force (IETF) [67]. The purpose of experiment 3 is to evaluate hybrid topologies to gain insight to provide a procurement recommendation to the Marine Corps. Infantry battalions also continuously deploy on the Marine Expeditionary Unit (MEU), the smallest MAGTF, and reflect the current standard of force for Marine quick reactionary forces. Figures 3.12 and 3.13 illustrate the structure of the tested networks. These networks were designed keeping in mind the principles from [28]. Some of the primary points of [28] are stated below:

1) SDN's benefits can be seen in hybrid networks consisting of SDN and traditional networking equipment (reflecting incremental network upgrades).
2) Upgrading the network can occur under a constrained budget and impact the network minimally.
3) Reduce network disruption while building self-assurance in SDN capabilities during incremental network upgrades.
4) "***The benefits of SDN to enterprise networks can be realized for every source-destination path that includes at least one SDN switch*** " [28] (bold emphasis added).

The topology depicted in Figure 3.12 reflects an idealized homogeneous SDN architecture

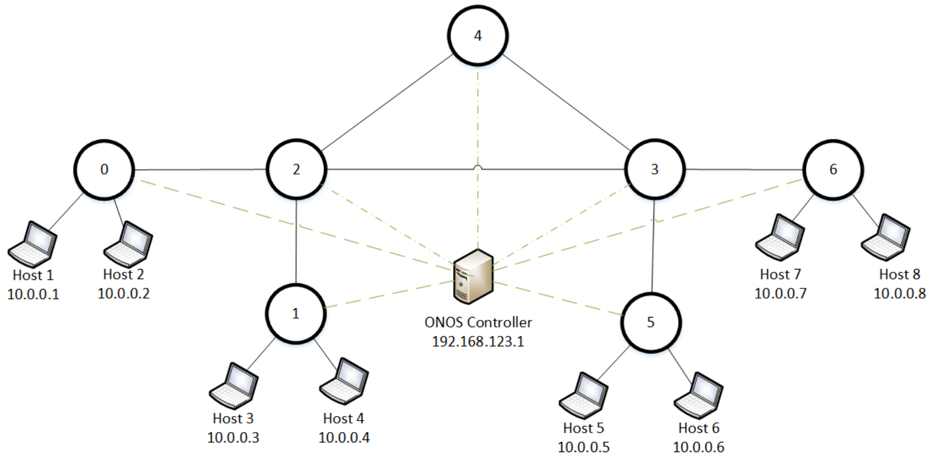


Figure 3.12. Experiment 3: Homogeneous SDN Infantry Battalion Topology with Seven Network Devices

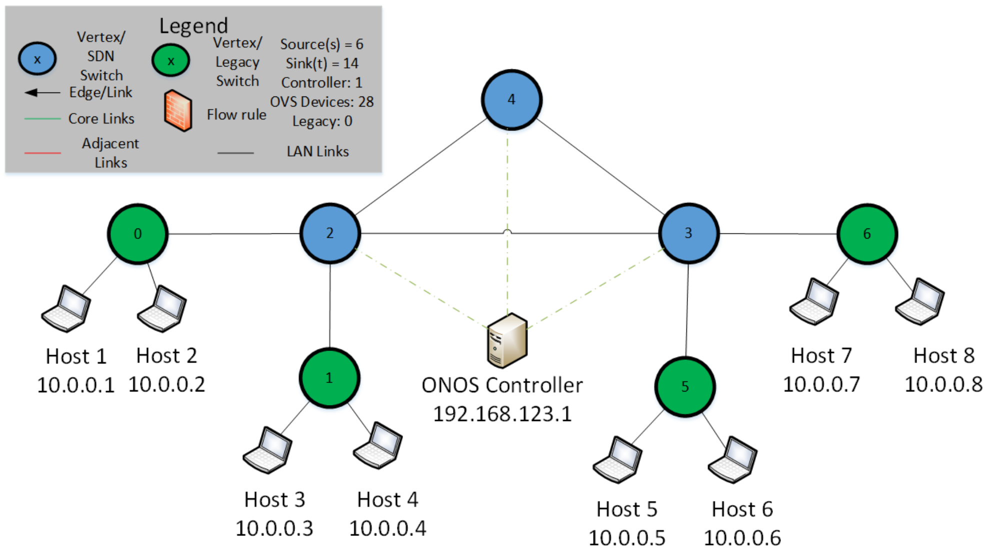


Figure 3.13. Experiment 3: Heterogeneous SDN Infantry Battalion Topology with Seven Network Devices

while Figure 3.13 reflects the reality of an incremental upgrade to SDN.

Hypothesis: The hybrid network, see Figure 3.13, will perform in exactly the same manner as the homogeneous network except for the flexibility and granularity that comes with having more SDN capable devices. The logical SDN will “appear” like an SDN of smaller size. The topology shown in Figure 3.13 reflects the principles from [28] dictating that the benefits of SDN can only be realized as long as the path between two devices contains at least one SDN switch. Therefore, the outer nodes or access layer switches will generally be the best candidates for legacy devices. This is why the access layer switches of the topology are legacy and the core and distribution layer switches are SDN.

Procedure: We will demonstrate the functionality and flexibility of a hybrid network and discuss the benefits of a hybrid network and demonstrate the negatives aspects.

3.6 Recap

The purpose of this research is to demonstrate the programmability of SDNs and their effectiveness in homogeneous and hybrid networks. The experimental design addresses topologies of varying sizes, SDN and hybrid equipment build outs, and realistic Marine Infantry networks. The design varies experimental variables to provide meaningful metrics of latency, deployment speed, and resilience to topology changes. The program will be measured against the same correctness and feasibility criterion listed in [27] and basic principles stated in [28]. The testing methodology will also provide evidence in support of Open vSwitch as a virtual testing platform and a platform for use in the Marine Corps.

CHAPTER 4: Findings

We live in an age that is driven by information. Technological breakthroughs...are changing the face of war and how we prepare for war.

—William Perry, 19th United States Secretary of Defense [6]

The following are experimental results. As a recap, the purpose of these experiments is to validate the functionality of the distributed firewall program, stress test Open vSwitch (Mininet's native virtual switch platform), and evaluate both the firewall and Open vSwitch on simulated homogeneous and heterogeneous Marine Corps topologies. The driving purpose behind the research is to evaluate software defined networking, reachability control frameworks from [27], and hybrid topologies for use and acquisition within the United States Marine Corps. Remember, that ONOS by default denies all traffic on the network unless explicitly allowed through flow rules. To enable testing, we enabled a reactive forwarding program which installs flows for all traffic encountered on the network (white listing everything).

4.1 Experiment 1: Validation and Stress Test

The purpose of experiment 1 is threefold. First, validate that the distributed firewall program appropriately filters traffic and responds to topology changes when invoked. Second, find Open vSwitch's experimentally constrained upper limit for flow rules to gather insight for DoD utilization. And finally, measure any effects on throughput and CPU utilization.

4.1.1 Experiment 1 — Test 1

The purpose of Experiment 1 is to validate the distributed firewall program's functionality. To answer this question, the program was installed, activated, and invoked on ONOS using a basic tree topology (see Figure 3.8) with four hosts and three SDN switches (Open vSwitch). Each host had an IP address ranging from 10.0.0.1 (host1) to 10.0.0.4 (host4).

Once ONOS and Mininet were activated, the ONOS GUI was visible at 192.168.123.1: 8181/onos/ui/index.html#/topo (see Figure 4.1). The ONOS-Mininet command line during activation and invocation where are also visible on the right side of Figure 4.1. The output of the program's invocation revealed a correct adjacency matrix, $MinMinCut$ (edge 0 - 2 : node 3), and deployed rule (deny IP traffic with destination 10.0.0.4/32 - read in at run time from a file).

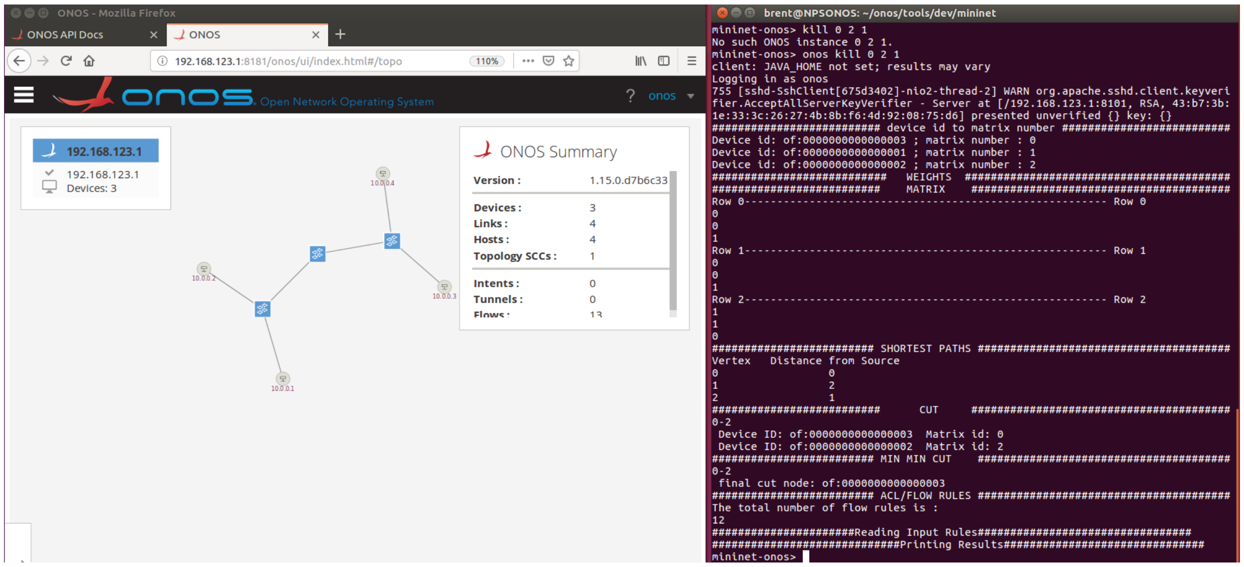
A shell script automated the testing process by iterating the following sequence twenty- five times for a specified number of rules :

1. Launch ONOS
2. Activate tshark (CLI Wireshark) and collect Openflow traffic on port 6653 or 6633 for a specified amount of time
3. Activate the distributed firewall program to begin ***Min Min Cut*** and rule installation
4. tshark ends and outputs a pcap file
5. Activate TCP iperf bandwidth test for 15 seconds between hosts on opposite sides of the ***Min Min Cut*** and output the result in a text file
6. Activate iostat utility for CPU utilization data during the TCP iperf bandwidth test and output the result to a text file
7. Close ONOS and Mininet

The results were then gathered using a Python script to parse through the data in the following sequence:

1. read iostat output files and identify CPU utilization for a specified number of rules and output the result to a CSV
2. read TCP iperf output files and identify overall throughput for a specified number of rules and output the result to CSV
3. for each pcap file, for a specified number of rules, convert the file to xml and calculate individual rule installation time and overall total rule installation time (see Figure 3.9. Average rule installation time and overall rule installation time were output to a CSV file.

The validity of the deployed flow rule was tested using ping and iperf tools. Any ping from 10.0.0.1 to 10.0.0.4 and from 10.0.0.4 to 10.0.0.1 failed due to the functionality of Internet


ONOS - Mozilla Firefox
ONOS API Docs
ONOS
192.168.123.1:8181/onos/ui/index.html#/topo
110%
onos. Open Network Operating System
onos
192.168.123.1
192.168.123.1
Devices: 3
10.0.0.4
10.0.0.2
10.0.0.3
10.0.0.1
brent@NPSONOS: ~/onos/tools/dev/mininet

ONOS Summary

Version : 1.15.0.d7b6c33

Devices : 3
Links : 4
Hosts : 4
Topology SCCs : 1

Intents : 0
Tunnels : 0
Flows : 13

```
mininet-onos> kill 0 2 1
No such ONOS instance 0 2 1.
mininet-onos> onos kill 0 2 1
client: JAVA_HOME not set; results may vary
Logging in as onos
755 [sshd-SshClient[675d3402]-nio2-thread-2] WARN org.apache.sshd.client.keyveri
fier.AcceptAllServerKeyVerifier - Server at [/192.168.123.1:8101, RSA, 43:b7:3b:
1e:33:3c:26:27:4b:8b:f6:4d:92:08:75:d6] presented unverified {} key: {}
########################## device id to matrix number ##########################
Device id: of:0000000000000003 ; matrix number : 0
Device id: of:0000000000000001 ; matrix number : 1
Device id: of:0000000000000002 ; matrix number : 2
###########################   WEIGHTS  #########################################
##########################    MATRIX    ########################################
Row 0------------------------------------------------------- Row 0
0
0
1
Row 1------------------------------------------------------- Row 1
0
0
1
Row 2------------------------------------------------------- Row 2
1
1
0
######################### SHORTEST PATHS #######################################
Vertex   Distance from Source
0                  0
1                  2
2                  1
##########################      CUT     ########################################
0-2
 Device ID: of:0000000000000003  Matrix id: 0
 Device ID: of:0000000000000002  Matrix id: 2
######################### MIN MIN CUT    #######################################
0-2
 final cut node: of:0000000000000003
######################### ACL/FLOW RULES #######################################
The total number of flow rules is :
12
######################Reading Input Rules##################################
#############################Printing Results################################
mininet-onos> 
```

Figure 4.1. Experiment 1: Firewall Invocation. ONOS GUI (left) and CLI (right) when the Distributed Firewall is Invoked on the Tree Topology (Tree Topology — Figure: 3.8).

Control Message Protocol. No ICMP echo-replies are given for either set of pings resulting in blank console messages. To further validate functionality, iperf UDP tests where ran. Any traffic destined for 10.0.0.4 was blocked while any traffic from 10.0.0.4 for 10.0.0.1 made it to its destination.

4.1.2 Experiment 1 — Test 2

The purpose of test 2 is to stress test the distributed firewall program and Open vSwitch to evaluate their limits and functionality under realistic network flow rules.

The maximum number of rules that were installed on Open vSwitch was 75,000 flow rules. The original procedure called for logarithmic rule growth; however, 100,000 rules exceeded the Open vSwitch's capacity under experimental constraints (hardware and software specifications). While the method to install 100,000 rules worked, it caused the network to crash. The first indications of this failure was a lack of TCP iperf data for every test using 100,000 rules. Upon closer inspection, the failure occurred because 100,000 rules overloaded the network and resulted in a dump of all rules (GUI showed 0 rules). Using the Linux utility top, we discovered that the DFA Java process monopolized CPU time, spending 90% of its time in user space. Keep in mind, the experimental constraints are fairly modest. As the number of rules approached 100,000, the links from the ***Min Min Cut*** node turned red indicating link failure. After the links failed, total flow rules across the network dropped to zero. Once the limit of 100,000 rule was discovered, we backed off by 50,000 rules and worked back towards the limit. Ultimately, 50,000, 64,000, and 75,000 rules were tested. 64,000 rules was chosen because it is representative of the number of null routes at an enterprise level firewall in the Marine Corps.

Flow rule installation time was measured from an individual rule and a total rule set installation time perspective. The statistics for individual rule installation time across the set of tests is illustrated in Figures 4.2 and 4.3. Starting at 10,000 rules there is an increase in per rule and total installation time (Figures 4.2 and 4.4), due to a $75ms$ pause for every 100 rules beyond 1000. The pause was instituted during testing as a congestion control mechanism due to the low latency (bus latency of the test laptop) and high bandwidth (correlated to computer clock speed [68]) of the test environment. Figure 4.3 is a regression of all average per rule installation times across the range of tests. Theoretically, the per rule installation time should be relatively flat across all tests but we saw in increase in time from 1 to 2 ms for 1 - 10,000 rules to 6 - 10 ms for 50,000 - 75,000 rules. Average per rule installation time, regardless of total rules, was $3.3394 \pm 3.471ms$. The total rule installation time increased linearly (Figure 4.5) despite the almost logarithmic growth of rules indicating a better than theoretical installation time despite congestion control mechanisms.

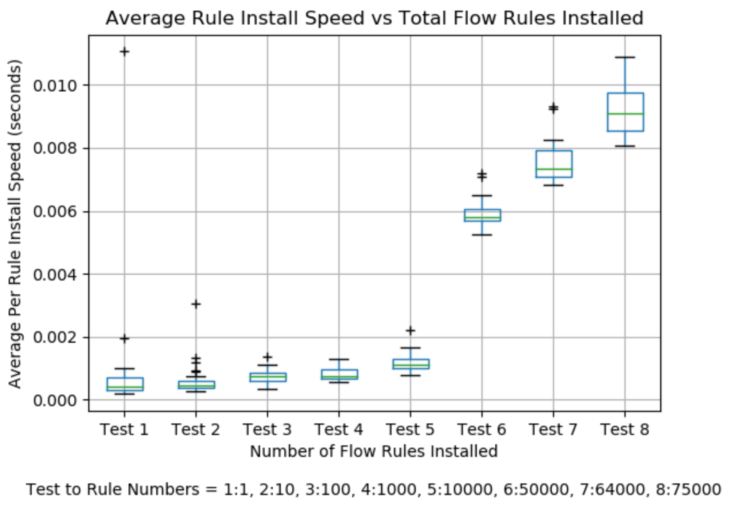


Figure 4.2. Average Per Rule Installation Time Comparisons. The depicted bars are standard box plots showing the inter-quartile range, min, max, median and outliers of each test.

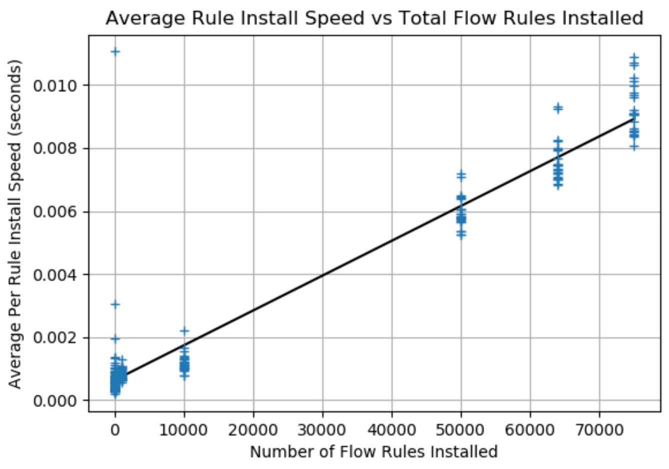
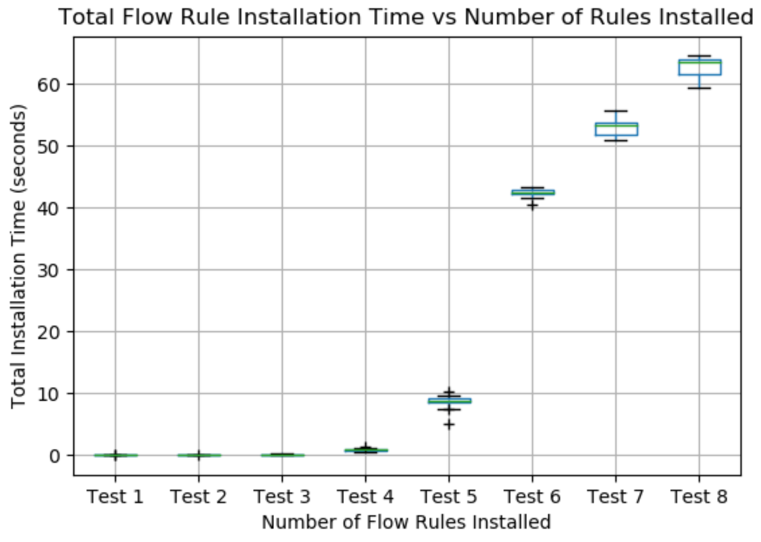
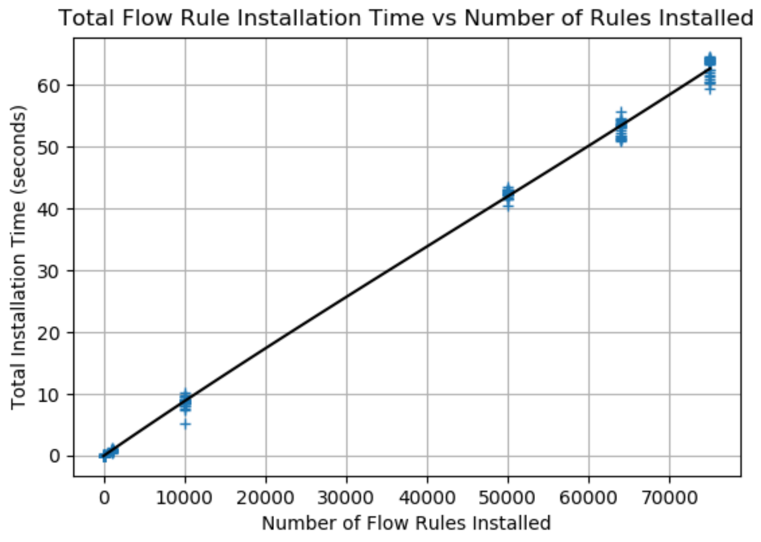


Figure 4.3. Linear Regression of Average Rule Installation Time Over All Tests. The r^2 Value for this regression was 0.930135.



Test to Rule Numbers = 1:1, 2:10, 3:100, 4:1000, 5:10000, 6:50000, 7:64000, 8: 75000

Figure 4.4. Overall Delay Test Comparison. The depicted bars are standard box plots showing the inter-quartile range, min, max, median and outliers of each test.



Test to Rule Numbers = 1:1, 2:10, 3:100, 4:1000, 5:10000, 6:50000, 7:64000, 8:75000

Figure 4.5. Overall Delay with Linear Regression. The r^2 value for this regression was 0.99885

The measured overall times are accurate; however, there is error present in the per rule installation time. The parsing program to calculate per rule installation time calculated the time based off having the complete set of packets including `OFPT_FLOW_MOD`, `OFPT_BARRIER_REQUEST`, and `OFPT_BARRIER_REPLY` and the time difference between the `OFPT_FLOW_MOD` and `OFPT_BARRIER_REPLY`. If any of the packets from the set listed above were not present, the program ignored the set in the total calculation of average per rule time and output an error message stating the transaction ID of the set. Due to the low latency and high bandwidth of the environment, TCP would bundle multiple OpenFlow Packets into single TCP segments consisting of multiple transaction IDs. This bundling effectively hid multiple transaction IDs from the parsing program. Additionally, it was discovered that there were multiple transaction IDs with the same ID of zero. Transaction ID's with same number also threw off the parser as these IDs had legitimate payloads despite the seemingly erroneous ID. The total number of ignored sets was low; however, the average per rule installation time was based off of what was found. Future work needs to correct this error and further investigate the phenomenon. The total rule installation time is based off of the first `OFPT_FLOW_MOD` and the last `OFPT_BARRIER_REPLY` (assuming a full correct set for both). We know the overall rule set installation times to be accurate because the method call to return the number of installed rules always returned the requested number of rules.

We performed bandwidth tests using iperf after each rule set installation. Increasing the number of flow rules forces the affected Open vSwitch to inspect each incoming packet until a match occurs. Iperf tests were performed between hosts whose flows were not filtered on the rule list forcing each packet to be compared to each rule on the flow rule list theoretically resulting in increased queuing delay (and latency) and a decrease in throughput. Figures 4.6 and 4.7 depict throughput versus total number of flow rules installed. There is a decrease in throughput as rules are increased.

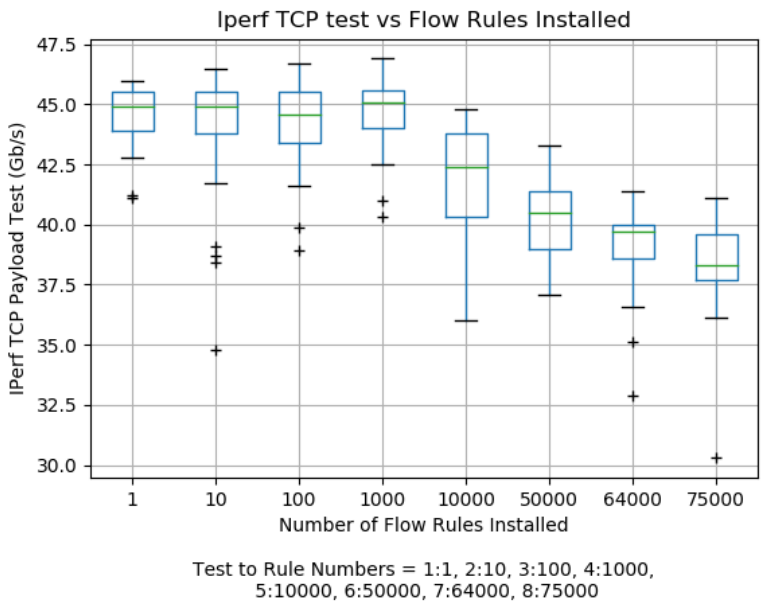


Figure 4.6. Iperf TCP Throughput Test Comparison. The depicted bars are standard box plots showing the inter-quartile range, min, max, median and outliers of each test. Notice that for 10,000 rules the range of values is much higher than for other rules.

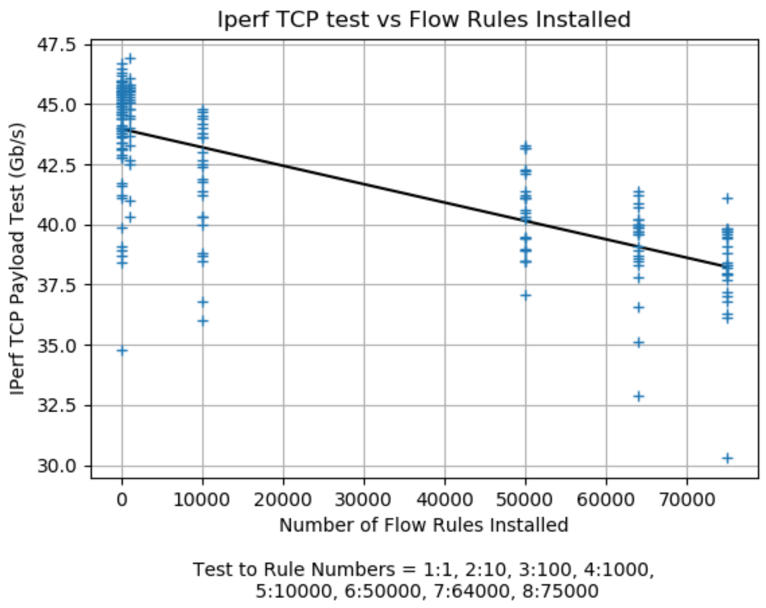


Figure 4.7. Iperf TCP Throughput with Linear Fit. The r^2 value for this regression was 0.21137 indicating the line is not useful for prediction.

CPU utilization versus total rules installed can be seen in Figure 4.8. There is an increase in CPU utilization from 1 to 1000 rules and then a sharp reduction from 1000 to 10000 to the baseline utilization after that (due to the 75*ms* pause). Given a linear regression (see Figure 4.9, there is a decrease of approximately five percent across the 8 tests; however, the curved regression doesn't appear to model the data accurately given the known 75*ms* pause. Lastly, IOStat data was collected at large and future work should isolate the individual process ID of the switch in question to more precisely capture per process CPU usage.

To analyze the data further, the IOstat and throughput data were averaged, across each test, then compared as arrays using the Pearson correlation coefficient. The correlation coefficient between the two sets of data is 0.4015 meaning there is a low to moderate correlation between the data. Using a 4th degree polynomial in a curved regression reveals an inverse relationship between CPU utilization and throughput. More data points are needed to verify this relationship. The causality seems low given the fact the relationship is likely linear.

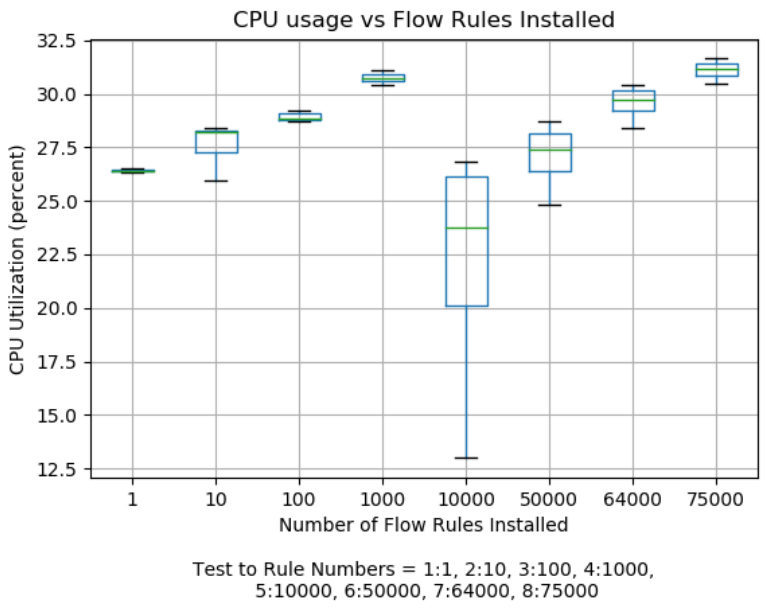


Figure 4.8. CPU Utilization per Test Comparison. Notice that for 10,000 rules the range of values is large.

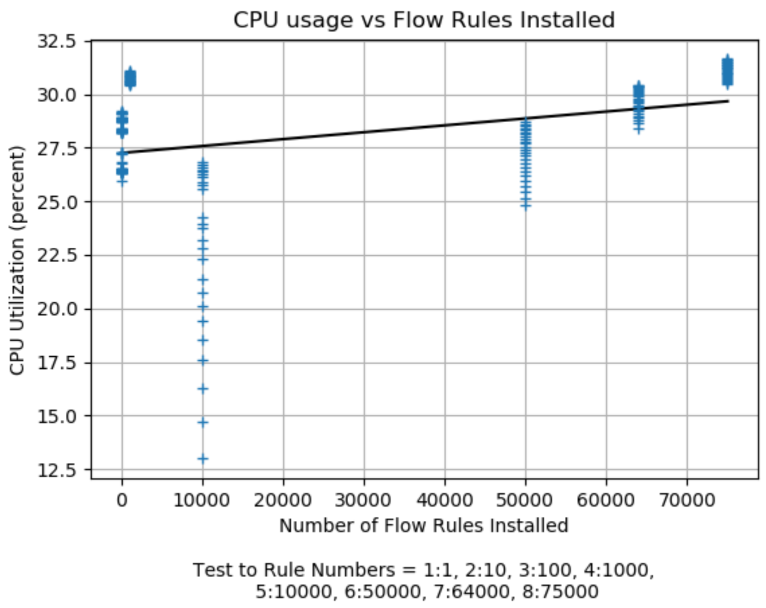


Figure 4.9. CPU Utilization with Linear Regression. r^2 for this fit was low at 0.061892

Examining Figures 4.9, 4.8, and 4.6, it is clear that the installation of 10,000 rules had a much larger range of values than any other test and was an outlier. The individual box plots in the figures above do not indicate any outliers because the values are so spread out and no individual point within the tests lie more than 1.5 times away from the inter-quartile range. This spreading effect is most pronounced in Figure 4.9. The contributor to CPU utilization drop at 10,000 rules is likely our congestion control algorithm which instituted a $75ms$ pause per 100 rules after 1000; however, this high range of values disappears again as the rules increase. We believe this to be an effect of the law of large numbers (numbers being rules in this case) post congestion control institution. Moreover, it may be experimental error. Overall, the data shows that installation time was a useful metric based off of the r^2 values of 4.5 and 4.3 but that CPU utilization and throughput were not.

4.1.3 Experiment 1 — Test 3

The purpose of test 3 is to evaluate the distributed firewall program's ability to react to network topology changes that change the Maximum Flow Minimum Cut Set. The program is configured to react to any link change (Link update/add/drop) using the ONOS Java API. Anytime a link change occurs, the program logs the link change and recalculates the $MinMinCut$ and reapplies flow rules. The results of the link change and $MinMinCut$ are logged and saved to file. To test link changes, a triangular topology was created (see Figure 4.10) to simulate the addition or deletion of a third link to the tree topology (Tree Topology: see Figure 3.8). Anytime the link was brought down or brought up (or any link in the topology) the change was detected and $MinMinCut$ was reran an rules deployed as necessary. The output file contains the link change, the new adjacency matrix and $MinMinCut$. One drawback with the reactive topology method is that the current iteration of the code does not maintain state and completely re-installs the $MinMinCut$ and flow rules every time the topology changes (something that will be minimized in future work). Secondly, the topology change methodology only does the $MinMinCut$ on the current s and t. The ability to dynamically change s and t will greatly improve the program's ability to react to threats from different parts of the network as necessary.

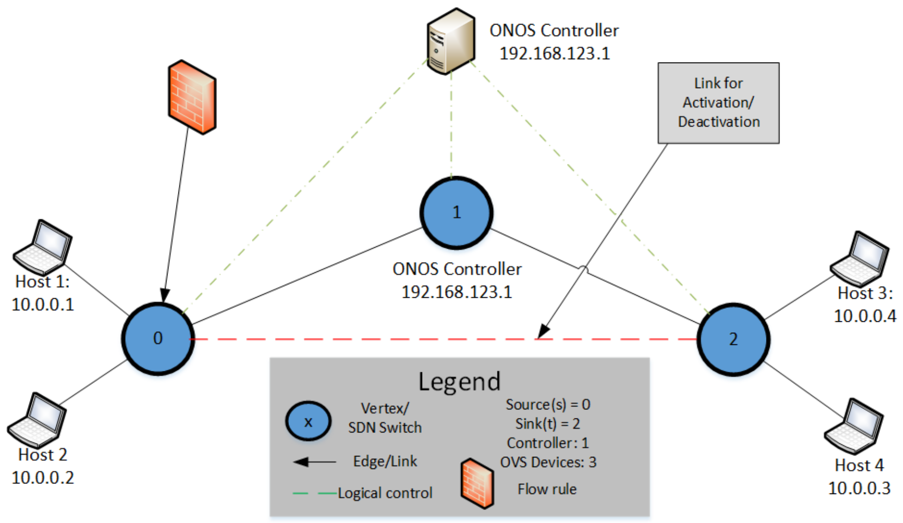


Figure 4.10. Experiment 1: Triangle Topology for Link Change Detection

4.2 Experiment 2: Infantry Regiment Testing

The purpose of experiment 2 was to evaluate the effect of the distributed firewall program under realistic network conditions. The topology of the network simulated a Marine infantry regiment (see Figure 3.11). The core links constitute legitimate links between a regimental headquarters and its subordinate links while the outer links are used to evaluate the distributed firewall program under changing topologies. Link speeds and latency emulated line of sight and beyond line of sight links that infantry battalions and regiments could potentially use. Overall, four tests were conducted to further validate the speed and functionality of the distributed firewall program under normal, dynamic, and malicious conditions to determine how effective the distributed firewall program is under realistic network conditions. An example of a simple point to point intent using the ONOS REST can be seen in Figure 4.11 dictating flow between two mac addresses. Intents can be strung together to form routes throughout the network.

These tests take advantage of the intent framework available in the ONOS architecture. Intents provide the high-level ability to create flows between devices without defining specific sub flows along a network path. This allows an administrator the ability to holistically define flows between devices at the network level (management and control plane) without having to spend time defining flows at the device level (data plane). ONOS intents give the

NOS the reins and force traffic between devices. Intents are used as a form of insurance to ensure ONOS properly establishes links between desired end points. The intents are juxtaposed against the distributed firewall program as a means of evaluating the distributed firewall program's effectiveness.

```

curl -X POST --user onos:rocks --header 'Content-Type:
    application/json' --header 'Accept: application/json' -d '{
    "type": "HostToHostIntent",
    "appId": "org.onosproject.ovsdb",
    "priority": 55,
    "one": "00:00:00:00:00:01/-1",
    "two": "00:00:00:00:00:04/-1"
 }' 'http://192.168.123.1:8181/onos/v1/intents'
```

Figure 4.11. ONOS Point to Point Intent cURL

4.2.1 Experiment 2 — Baseline Test

The baseline test was used to validate the functionality of the distributed firewall program. The following describes the fleshed out methodology of the test (repeated twenty-five times) using a Linux shell script to automate the process (Shell Scripts in section A.8):

1. Launch ONOS & Mininet
2. Setup Mesh Regiment Topology
3. Disestablish all but six core links between the regiment and battalions
4. Establish an intent from host 1 to host 6 using the ONOS REST API. Host 1 to host 6 constitutes allowed traffic on the network with respect to the flow rules the distributed firewall program uses.
5. Establish an intent from host 2 to host 5 using the ONOS REST API. Any traffic with a destination IP address of host 5 is not allowed on the network with respect to flow rules the distributed firewall program uses.
6. Start an iperf UDP server at Host 6 capturing throughput every second
7. Start an iperf UDP client at Host 1 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting

the result to a text file.

8. Start an iperf UDP server at Host 5 capturing throughput every second
9. Start an iperf UDP client at Host 2 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
10. Wait 10 seconds
11. Activate the distributed firewall program to begin ***Min Min Cut*** and rule installation for a total of 100 rules within the $MinMinCut$
12. Close ONOS and Mininet

To analyze that data, we simply iterated through every iperf client and server output file to generate the graphed data. Using regular expressions we pulled the relevant values from the iperf output file and graphed them. Regular expressions allowed us to parse though the data quickly while selectively pulling the sending and receiving rate of the client and server respectively. This method was applied to each of the following tests.

Figure 4.1 shows that hosts 1 and 2 maintain a constant stream of data at 4 Mbits/s. The server hosts, 5 and 6, show very different outcomes than their client counterparts. At approximately 11.36 seconds traffic to host 5 is cutoff by the distributed firewall program while the traffic arriving at host 6 is maintained at a throughput equivalent to host 1's output (0.00029 Mbits/s less than host 1's output) which aligns with the results from Experiment 1. Figure 4.12 shows that 100 flow rules have little effect on throughput. Figure 4.13 depicts the blocking of traffic destined for host 5 at approximately 11.36 seconds (receiving rate average over 25 runs). Table 4.1 depicts the raw statistics for this experiment. We entered the value of 0 Mbit/s where iperf had zero data to reflect the change more dramatically on the graph. There is error in the test. The iperf program's time is not synchronized with the shell scripts's timing; therefore, it is impossible to measure the exact time that the distributed firewall's program's rules took effect. From a coarse grain view, installation of 100 rules takes place approximately within the bounds discovered in experiment 1. The difference between experiment 1 and experiment 2 is that experiment 2 has much larger latency between devices.

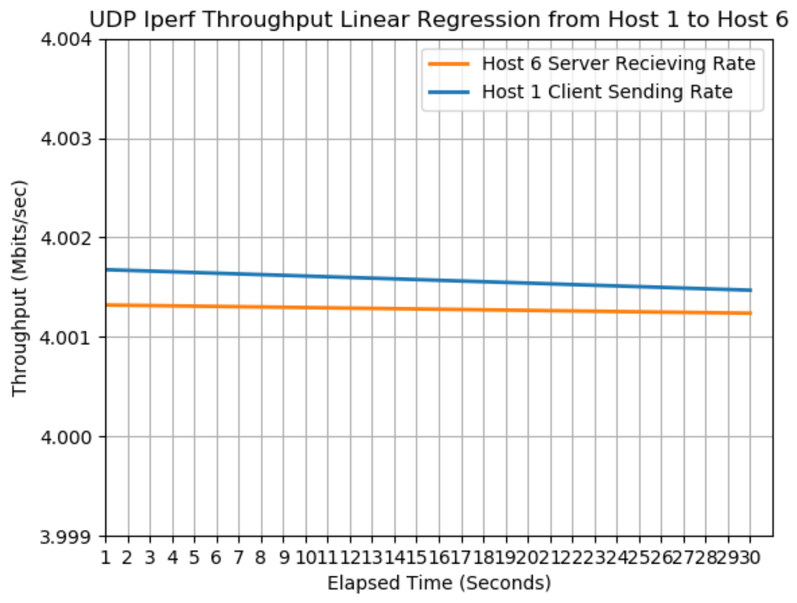


Figure 4.12. Experiment 2: Baseline Test: Host 1 to Host 6 Regression Data. Host 1 reflects iperf client sending rate while Host 6 reflects iperf server receiving rate

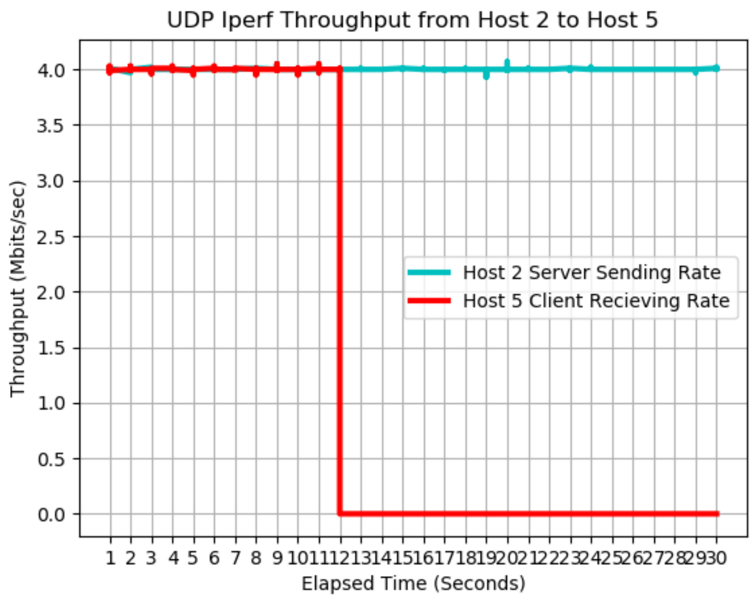


Figure 4.13. Experiment 2: Baseline Test: Host 2 to Host 5 Data. Host 2 reflects iperf client throughput while Host 5 reflects iperf server arrival rate

Baseline	Host1	Host 6	Host 2	Host 5
Sample Size (n)	25	25	25	25
Data Points	750	750	750	299
Session Active (Seconds)	30	30	30	11.36
Mean Rate (Mbps)	4.00157	4.00128	4.00157	3.7872
Median Rate (Mbps)	4.0	4.0	4.0	4.0
Standard Deviation	0.0048912	0.0077089	0.00640244	0.90057

Table 4.1. Experiment 2: Baseline Test Using a Marine Infantry Regiment Topology and 100 Flow Rules

4.2.2 Experiment 2 — Random Adjacent Link Establishment Test

The purpose of the random adjacent link establishment test was to evaluate the distributed firewall program's ability to block illegitimate traffic outside of the normal core links. This test simulated the establishment of adjacent links to enable mission accomplishment (consistent with the author's experience in Afghanistan where units established or disestablished links without permission). Intents were created between two sets of hosts on the network (H1 –> H6 and H2 –> H5) to reflect allowed and blocked traffic respectively. Additionally, establishment of a random adjacent link forced ONOS to recalculate intent flows (shortest paths) as well as the $MinMinCut$. The following describes the fleshed out methodology of the test (repeated twenty-five times) using a linux shell script to automate the process:

1. Launch ONOS & Mininet
2. Setup Mesh Regiment Topology
3. Disestablish all but six core links between the regiment and battalions
4. Activate the distributed firewall program to begin ***Min Min Cut*** and rule installation for a total of 1000 rules within the $MinMinCut$
5. Establish an intent from host 1 to host 6 using the ONOS REST API. Host 1 to host 6 constitutes allowed traffic on the network with respect to the flow rules the distributed firewall program uses.
6. Establish an intent from host 2 to host 5 using the ONOS REST API. Any traffic with a destination IP address of host 5 is not allowed on the network with respect to flow rules the distributed firewall program uses.

7. Start an iperf UDP server at Host 6 capturing throughput every second
8. Start an iperf UDP client at Host 1 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
9. Start an iperf UDP server at Host 5 capturing throughput every second
10. Start an iperf UDP client at Host 2 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
11. Wait 10 seconds
12. Re-Establish a random adjacent link from host 1/2's network to host 5/6's network.
13. Close ONOS and Mininet

The results of test 2 shows that no illegitimate traffic was able to traverse the network. Table 4.2 shows the raw statistics for this test. Moreover, we observe an expected drop in throughput (approximately 0.05 Mbits/s) due to flow rule installation (1000 rules). We chose to increase the number of installed rule from 100 in the baseline test to 1000 in this test so as to observe more obvious changes in throughput. Keep in mind that we 100 rule incurred almost no delay on the network from experiment 1 whereas at 1000 rules we begin to see more visible changes in bandwidth. No data was collected for the UDP server at Host 5 - hence the null result in the table 4.2. Zero Mbits/s was entered into the graphical depiction of the results (4.15) to illustrate the effect of the distributed firewall program. After closer inspection of the results, it was determined that this test was a rudimentary examination of the distributed firewall program as the intent framework wasn't fully leveraged against it. The intents' shortest path had not changed despite the addition of the adjacent link because the intent architecture was unaware of the distributed firewall program's flow rules. The results of this test validate this as no traffic destined for 10.0.0.5 arrived. The new adjacent link was not the shortest path and therefore the intents pushed traffic towards flow rule enforcing switches as shown by Figure 4.15. The results indicate that a more sophisticated intent is required to circumvent the distributed firewall program either by rerouting traffic around filters or elevating traffic priority levels beyond that of traffic filters. Of note, there is an overall decrease in throughput from Host 1 over time (0.01 Mbits/s - Figure 4.14) where the expected result was a constant transmission rate (as was the case in Figure 4.15).

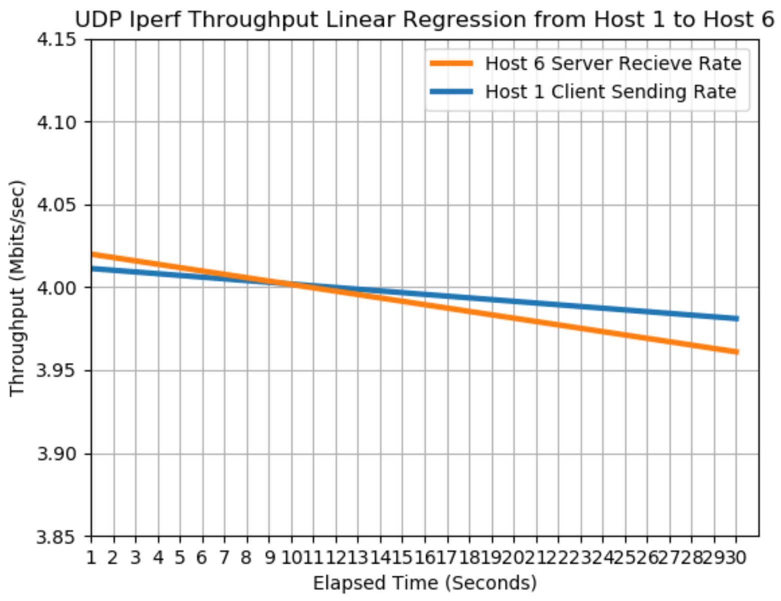


Figure 4.14. Experiment 2: Random Adjacent Link Establishment Test: Host 1 to Host 6 Regression Data

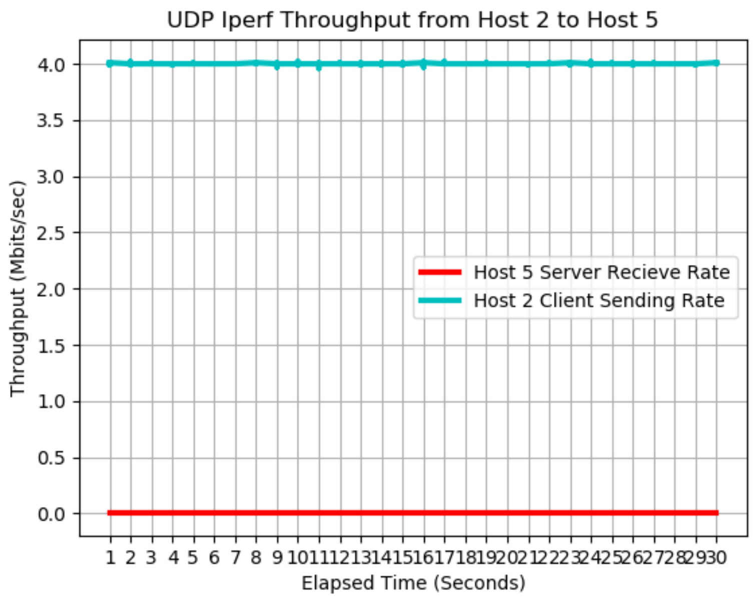


Figure 4.15. Experiment 2: Random Adjacent Link Establishment Test: Host 2 to Host 5 Data

Random Outer Link Up

Random Outer Link Up	Host1	Host 6	Host 2	Host 5
Sample Size (n)	25	25	25	25
Data Points	750	750	750	0
Average Transmit/ Receive Time	30	30	30	Null
Mean	3.99619	3.990506	4.001493	Null
Median	4.0	4.0	4.0	Null
Standard Deviation	0.14610	0.20657	0.0049969	Null

Table 4.2. Experiment 2: Random Outer Link Up Using a Marine Infantry Regiment Topology and 1000 Flow Rules

4.2.3 Experiment 2 — Random Core Link Loss

The random core link test was used to evaluate the effect of core link loss on throughput in the network utilizing the distributed firewall program. The test shows very similar results to the random adjacent link test. The key difference is that the sending host (host 1) maintains a constant transmission rate (see Figure 4.16) of 4 Mbits/s in the core link loss test while in there is a drop in transmission rate in the random adjacent link test (Figure 4.14). As was done with the random adjacent link test, the null values depicted in table 4.3 were depicted as zeros in the Figure 4.17 to illustrate zero transmissions between hosts 1 and 5. The following describes the fleshed out methodology of the test (repeated twenty-five times) using a linux shell script to automate the process:

1. Launch ONOS & Mininet
2. Setup Mesh Regiment Topology
3. Disestablish all but six core links between the regiment and battalions
4. Activate the distributed firewall program to begin ***Min Min Cut*** and rule installation for a total of 1000 rules within the $MinMinCut$
5. Establish an intent from host 1 to host 6 using the ONOS REST API. Host 1 to host 6 constitutes allowed traffic on the network with respect to the flow rules the distributed firewall program uses.
6. Establish an intent from host 2 to host 5 using the ONOS REST API. Any traffic with a destination IP address of host 5 is not allowed on the network with respect to flow

rules the distributed firewall program uses.

7. Start an iperf UDP server at Host 6 capturing throughput every second
8. Start an iperf UDP client at Host 1 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
9. Start an iperf UDP server at Host 5 capturing throughput every second
10. Start an iperf UDP client at Host 2 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
11. Wait 10 seconds
12. Deactivate a random core link from host 1/2's network to host 5/6's network.
13. Close ONOS and Mininet

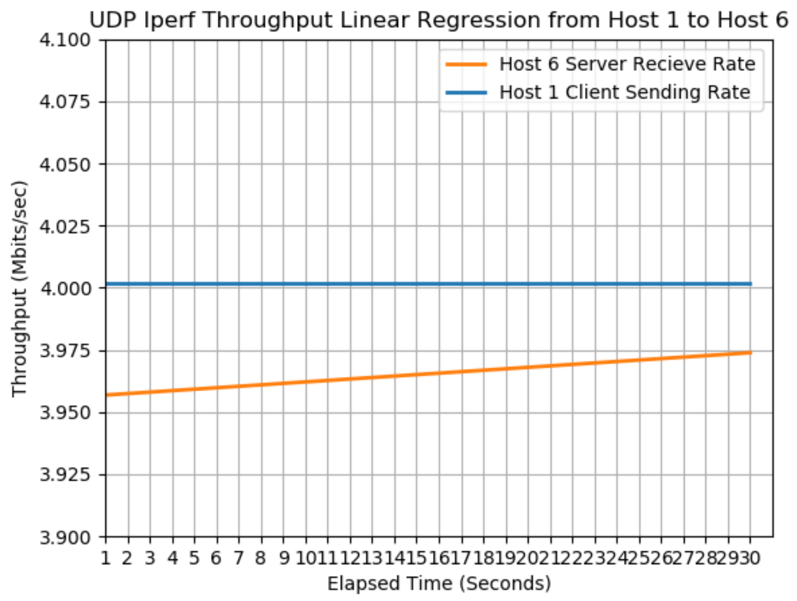


Figure 4.16. Experiment 2: Random Core Link Loss Test: Host 1 to Host 6 Regression Data

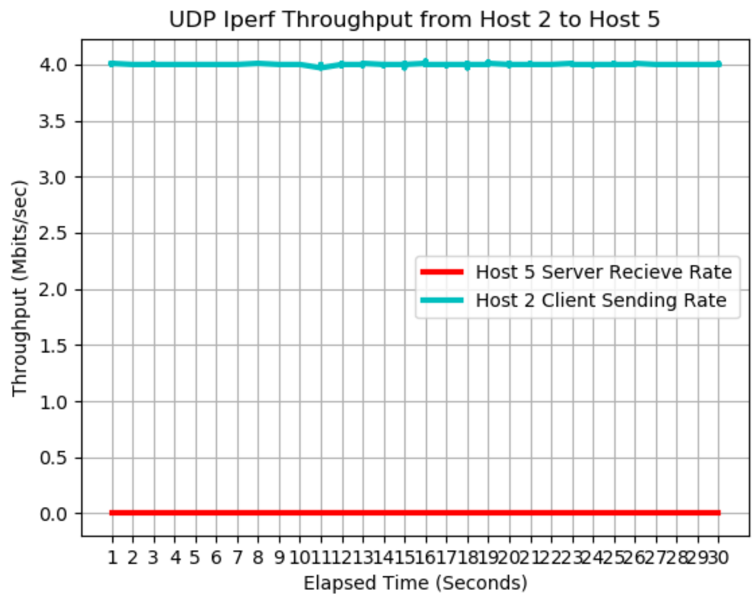


Figure 4.17. Experiment 2: Random Core Link Loss Test : Host 2 to Host 5 Data

Random Core Link Down	Host1	Host 6	Host 2	Host 5
Sample Size (n)	23	23	23	23
Data Points	690	685	690	Null
Average Transmit/ Receive Time	30	30	30	Null
Mean	4.0015507	3.96534	4.0014202	Null
Median	4.0	4.0	4.0	Null
Standard Deviation	0.0047914	0.35155	0.0048166	Null

Table 4.3. Experiment 2: Random Core Link Down Using a Marine Infantry Regiment Topology and 1000 Flow Rules

4.2.4 Experiment 2 — Intent Versus Distributed Firewall Shortest Path

The intent versus distributed firewall shortest path test evaluated the speed of the distributed firewall program versus the speed of an ONOS point to point intent. This test was devised after the first three tests results were examined. It was determined that the ONOS intent framework utilized shortest a path algorithm and had no method of automatically rerouting around DF program rules. Therefore, the methodology enabled the ONOS point to point intent to send traffic down the shortest path while the distributed firewall program raced to enforce security policy on the new data path. This test also simulates the case of malicious link establishment. The results of the experiment are shown in Figures (4.18, 4.19, 4.20, 4.21, 4.22) and Table 4.4. The following describes the fleshed out methodology of the test (repeated twenty-five times) using a linux shell script to automate the process:

1. Launch ONOS & Mininet
2. Setup Mesh Regiment Topology
3. Disestablish all but six core links between the regiment and battalions
4. Activate the distributed firewall program to begin ***Min Min Cut*** and rule installation for a total of 1000 rules within the $MinMinCut$
5. Deactivate All core links from host 1/2's network to host 5/6's network
6. Establish an intent from host 1 to host 6 using the ONOS REST API. Host 1 to host 6 constitutes allowed traffic on the network with respect to the flow rules the distributed firewall program uses.
7. Establish an intent from host 2 to host 5 using the ONOS REST API. Any traffic with a destination IP address of host 5 is not allowed on the network with respect to flow rules the distributed firewall program uses.
8. Start an iperf UDP server at Host 6 capturing throughput every second
9. Start an iperf UDP client at Host 1 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
10. Start an iperf UDP server at Host 5 capturing throughput every second
11. Start an iperf UDP client at Host 2 for 30 seconds with a maximum throughput of 4 Mbits/s (bottleneck link speed) capturing throughput data every second and outputting the result to a text file.
12. Wait 10 seconds
13. Re-Establish a random adjacent link from host 1/2's network to host 5/6's network.
14. Close ONOS and Mininet

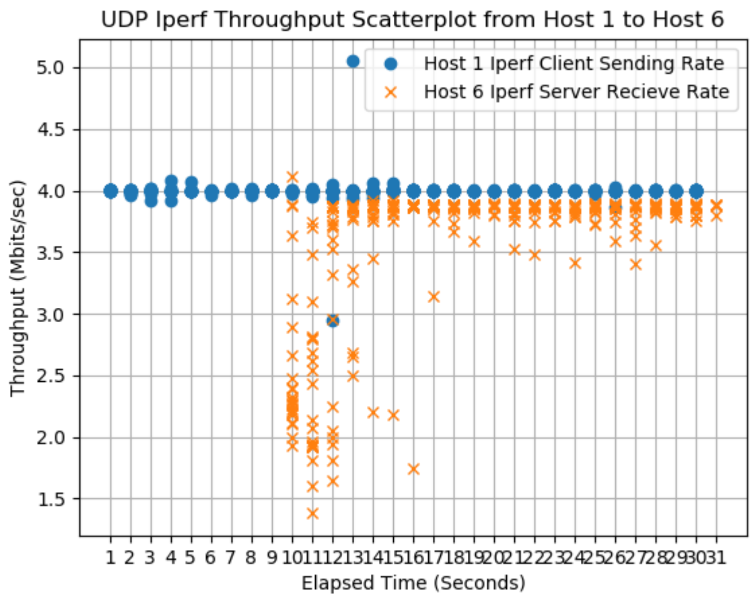


Figure 4.18. Experiment 2: Intent and Distributed Firewall Shortest Path : Host 1 to Host 6 Scatter-plot.

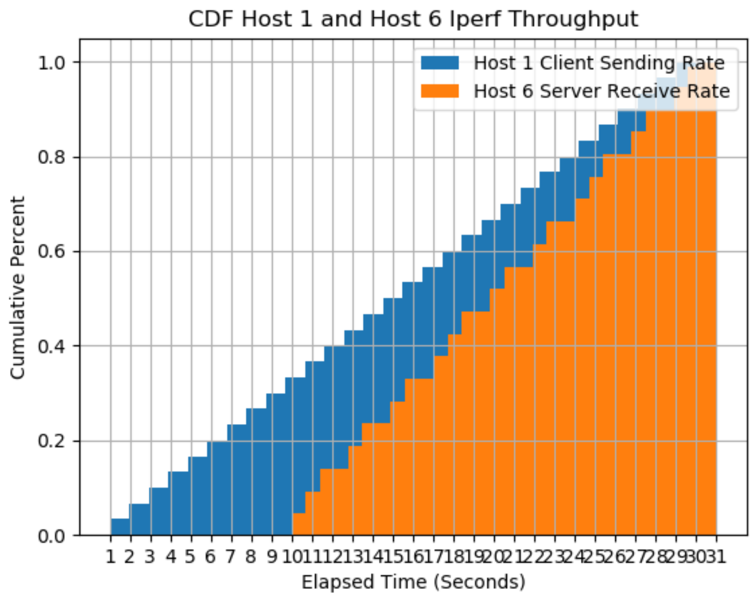


Figure 4.19. Experiment 2: Intent and Distributed Firewall Shortest Path : Host 1 to Host 6 CDF.

Figure 4.18 displays the scatter-plot of the iperf data captured from host 1 (UDP iperf client) and host 6 (UDP iperf server). Throughput at host 6 starts off with a significant amount of variation then increasing in throughput consistency. Table 4.4 shows that host 6 has an average of 3.68 Mbits/s and a median of 3.88 Mbits/s which is well within expected tolerances given the variability of the throughput at the beginning of the capture. Figure 4.19 (Host 1 to Host 6 Cumulative Distribution Function) shows that there is a consistency in the throughput from host 1 to host 6 which is what we expected for allowed traffic.

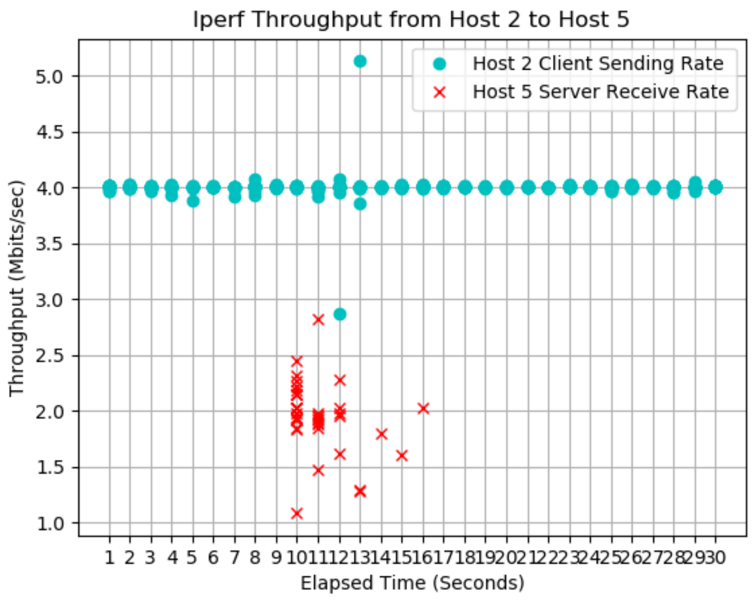


Figure 4.20. Experiment 2: Intent and Distributed Firewall Shortest Path : Host 2 to Host 5 Scatter-plot.

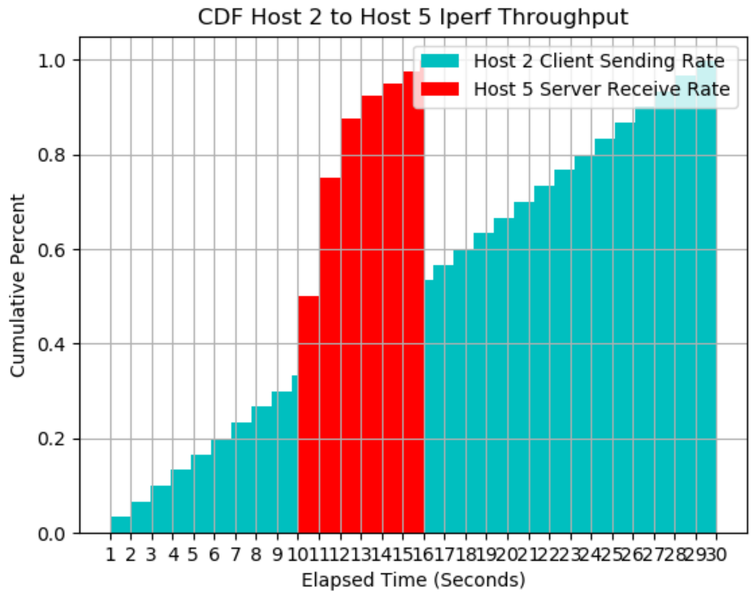


Figure 4.21. Experiment 2: Intent and Distributed Firewall Shortest Path : Host 2 to Host 5 CDF.

The data from host 2 to host 5 was the most revealing. Figure 4.20 shows that while the distributed firewall program was able to adapt to the new link and block, traffic destined for host 5 the majority of times, the speed of deployment varied. The CDF (Figure 4.21) of host 2 and host 5 showed that 100% of traffic is blocked within 6 seconds with approximately 76% of all traffic blocked within two seconds. The histograms of the host 2 and host 5 data reinforce the CDF data and show that number of times traffic reached host 5 was small. Given this information, further testing is needed to decrease the block time to less than one second.

Intent vs DF Program	Host1	Host 6	Host 2	Host 5
Sample Size (n)	25	25	25	19
Data Points	720	506	720	40
Average Transmit/ Receive Time	30	20.086	30	1.025
Mean	4.000805	3.684604	4.00056944	1.94699
Median	4.0	3.88	4.0	1.95
Standard Deviation	0.0565971	0.511490	0.0607781	0.308108

Table 4.4. Experiment 2: The Intent Versus Distributed Firewall Shortest Path Using a Marine Infantry Regiment Topology and 1000 Flow Rules

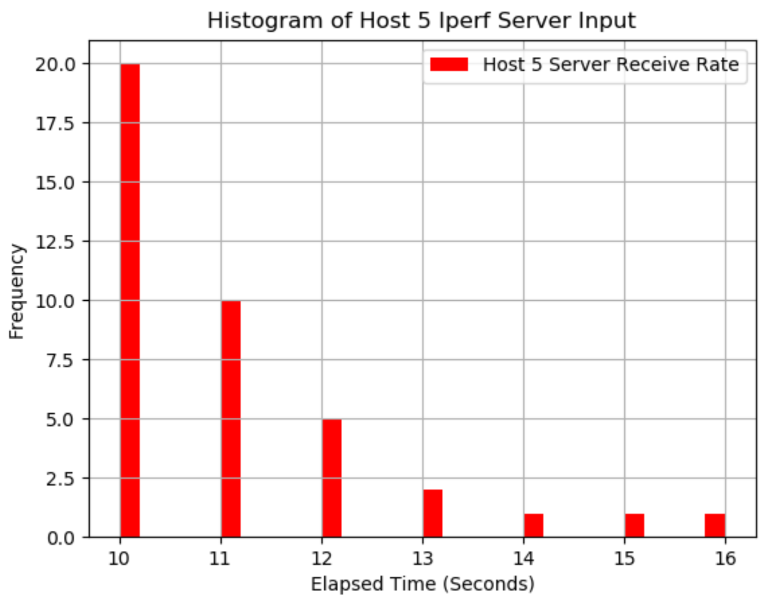


Figure 4.22. Experiment 2: The intent versus distributed firewall shortest path : Host 5 Histogram. The frequency in this graph equates to the number of times, out of twenty-five, that traffic arrived at the iperf server (Host 5)

4.3 Experiment 3: Hybrid Network Evaluation

Hybrid networks are a necessity for any network looking to transition from a traditional network to an SDN setup. Fully outfitting any network is a costly and difficult process from the perspective of maintaining services for the user. In this experiment we outline the basic procedures as well as the pros and cons of various strategies for potential USMC deployment of hybrid SDN networks. Every network in the Marine Corps is different and is built to suit the designer's needs; however, for the purposes of proscribing a general heuristic for purchasing of equipment we will assume there is a baseline structure for Marine Infantry Battalions (see Figure 4.24). The baseline topology was created to represent a possible layout for a Marine infantry battalion. The battalion's logical topology was derived from previously used physical topologies that provided network services to the battalion's different commodity sections (called 'S' shops where the number of the shop indicates its function). Each battalion runs combat operations out of a "Combat Operations Center", run by the operations section (S-3), and has supporting sections that provide intelligence (S-2), administrative (S-1), logistics (S-4), and communications (telecommunications / S-6) services to the battalion. Figure 4.23 shows the generic battalion setup with switches supporting the various commodity sections.

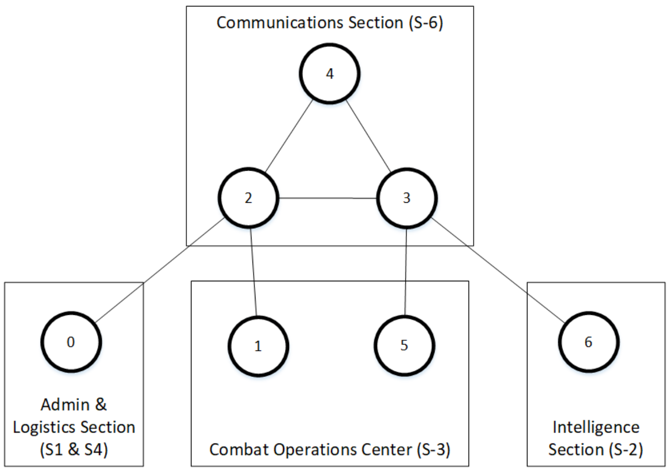

Communications Section (S-6)
4
2
3
0
1
5
6
Admin & Logistics Section (S1 & S4)
Combat Operations Center (S-3)
Intelligence Section (S-2)

Figure 4.23. Experiment 3: Example Infantry Battalion Physical Layout

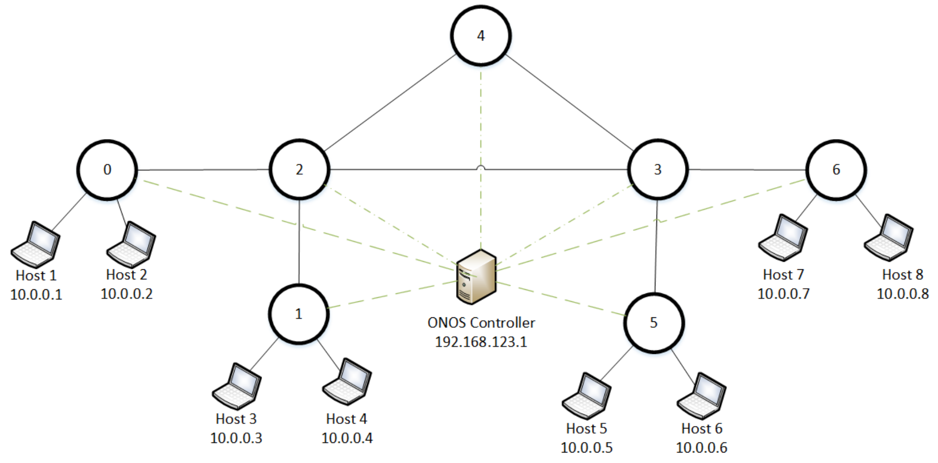


Figure 4.24. Experiment 3: Example Homogeneous Battalion SDN Structure

If we examine Figure 4.24, we assume this to be the final generic structure of a Marine Infantry battalion network with a total of 7 SDN switches or 7 Legacy Network Devices. SDN switches are all generically the same while legacy networking devices can vary greatly depending on their use. The only similarities between them on a physical level is the number of interfaces they have. This device total does not account for devices left in reserve or the total amount of devices that a battalion should utilize based on average annual maintenance requirements. The seven-node topology provides enough flexibility to physically separate different commodity sections while also providing enough logical control to the operator. Moreover, this number of devices can provide some semblance of fault tolerance to the operator.

Ideally, an Infantry battalion would be fielded a full complement of SDN devices. More SDN devices provide more control of network flows and break the battalion into boundaries of control. Every SDN device can explicitly control traffic to and from itself and serve as gatekeepers to other parts of the network. Legacy devices by default do not provide such control and are therefore are prone to be the locations where illegitimate traffic flows can begin and end. Take Figure 4.25 for example, illegitimate flows can start and begin from nodes 0, 1, 6, 5. This can be a problem for different commodity sections trying to segregate network traffic from each other.

Weitzel's research into hybrid networks in [33] showed that hybrid networks can maintain basic network segmentation mechanisms like VLANs and ACLs in heterogeneous setups.

This suggests that legacy devices could be maintained at the access layer as there are mechanisms to control network traffic that work in hybrid setups. The primary negatives of this setup are a knowledgeable inside threat and an unknowing operator. We believe this risk is acceptable so long as the operator has properly enabled network controls such as MAC address filtering, VLANs, and access control lists. Moreover, any connection from legacy device to legacy device will be visible to an SDN Controller as it will result in a topology change from the lens of an SDN to SDN connection. For example, if node 0 and node 6 are connected together then another logical connection is formed between nodes 2 and 3 (SDN switches - see Figure 4.26). The topological change will be visible to network operators. With respect to the distributed firewall program, the $MinMinCut$ would be ran again and deployed; however, security policy will not be upheld on this link unless additional manual configuration is conducted for the two legacy devices. Figure 4.26 shows the SDN logical view of this setup.

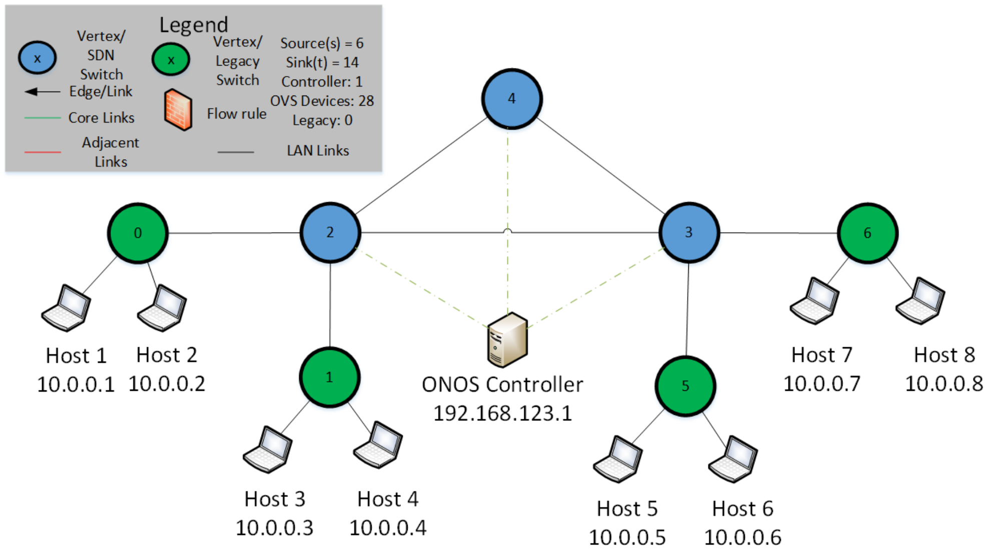


Figure 4.25. Experiment 3: Heterogeneous Battalion SDN Structure

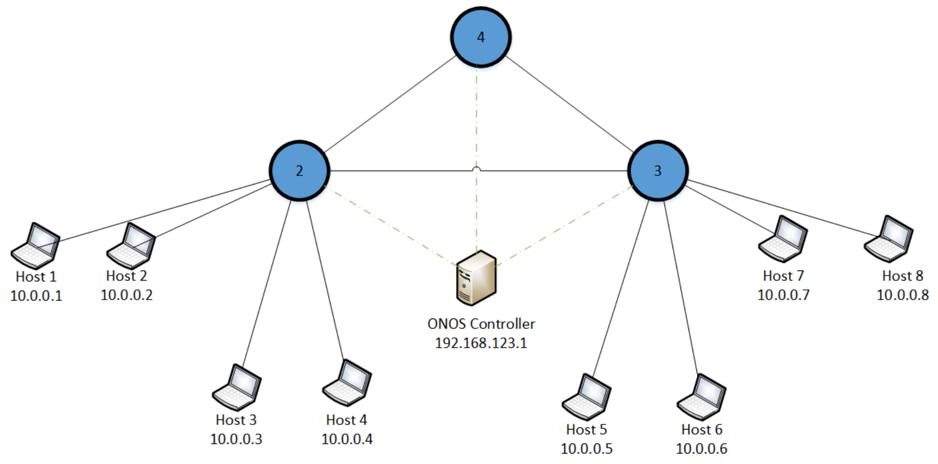


Figure 4.26. Experiment 3: Logical Heterogeneous Battalion SDN Structure

Figure 4.27 shows another possible hybrid battalion configuration where all but the core switch are legacy devices. This setup logically appears in Figure 4.28 and provides absolute control over the traffic within the battalion but performs poorly in context of the larger network. Core device best practice are to let those devices perform routing and filtering functions while access layer devices are best left to switching user traffic. In this case, one device handles all local area network switching and routing to and from other networks. This setup places too much traffic load on node 4 and is a central point of failure. This should be the absolute worst case for use and fielding we highly recommend against it.

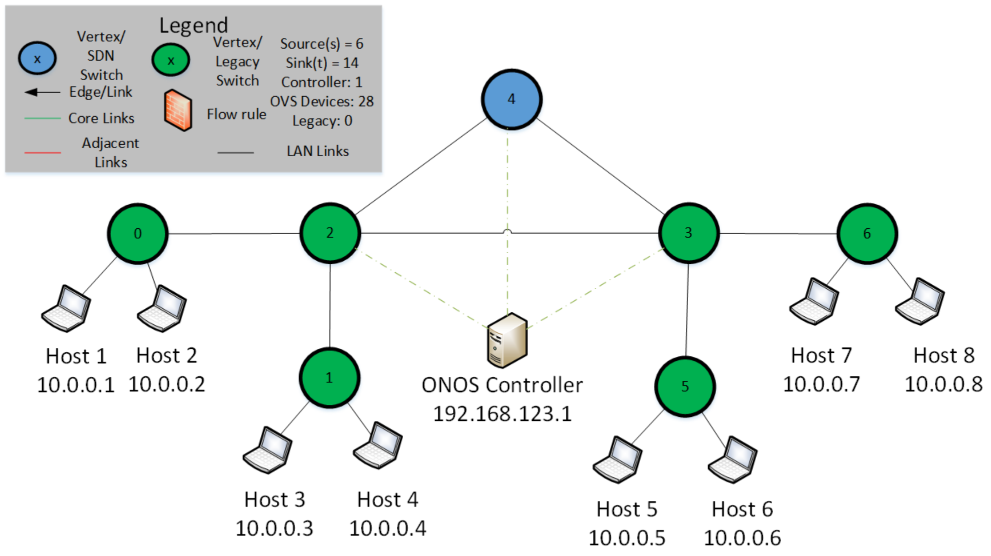


Figure 4.27. Experiment 3: Heterogeneous Battalion SDN Structure 2

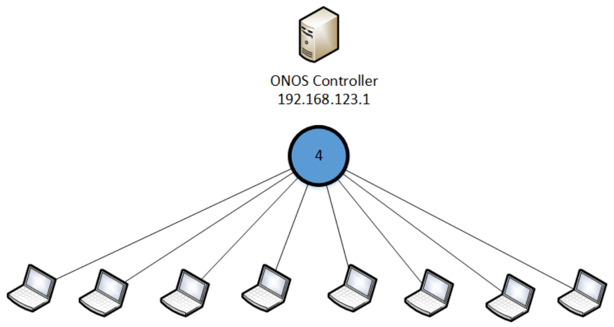


Figure 4.28. Experiment 3: Logical Heterogeneous Battalion SDN Structure 2

Another methodology for deployment of SDN switches is to take the SDN deployment to the access layer providing the most control over user traffic (Figures 4.29 and 4.30). From a flow control perspective this setup is best because it provides total control over user traffic and makes it unlikely that users will circumvent network controls. Users would have a very large barrier to overcome to make illicit connections on the battalion network. From the inside looking out, you would have four separate SDN devices needing flows outside of the network. Flows outside of the network would be unwieldy and would be the drawback of this setup. Moreover, when examining this from the firewall perspective, this is not best practice. It is easier and less error prone to configure one edge device to handle external flows than four separate ones. From a logical perspective (Figure 4.28) this constitutes a poor use flows as a full mesh architecture is generally very wasteful from a bandwidth and flow control perspective.

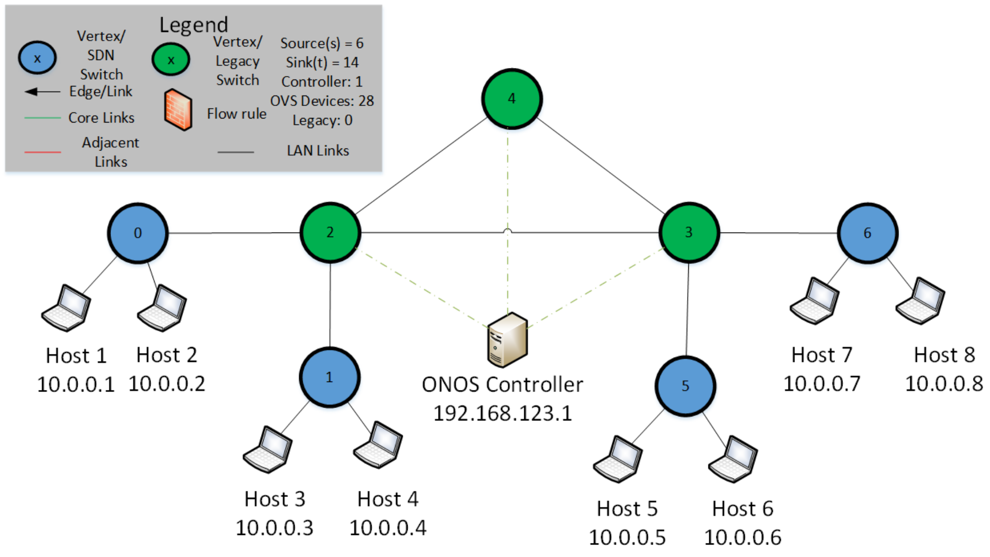


Figure 4.29. Experiment 3: Heterogeneous Battalion SDN Structure 3

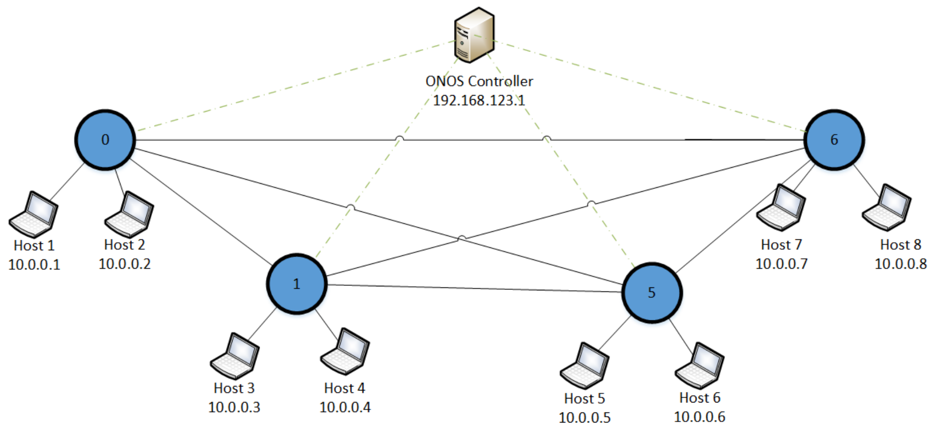


Figure 4.30. Experiment 3: Logical Heterogeneous Battalion SDN Structure 3

Having examined some example architectures and their benefits and drawbacks we can formulate a general heuristic for heterogeneous SDN architectures. We want to use the least amount of SDN switches to conserve cost but also provide enough switches to maintain network ingress filtering, logical network segmentation and flow control at the user level. Moreover, the heuristic must be applicable to any hybrid SDN setup regardless of configuration or device count. At a minimum, a network administrator must adhere to the principles in [28] to transition to SDN; however, our recommendation is to utilize the setup in Figure 4.25 where the core and distribution switches are SDN devices. This provides ingress traffic control and distribution layer traffic control without creating a single point of failure. The general heuristic should be to place an SDN switch at the network edge (where connections are made to external organizations) and at places in the distribution layer such that there is an SDN device between every source and destination path within the intranet. In the case of our example topology (Figure 4.25), this means 3 out of 7 switches (43%) should be SDN devices.

4.4 Summary

Through a series of experiments, we examined Open vSwitch, a distributed firewall program, and hybrid Marine Corps topologies in an effort to demonstrate the programmability and control of Software Defined Networking and its advantages. Moreover, we have provided more evidence of Mininet's flexibility as a network testing platform and the capability of Open vSwitch as virtual switching platform. We also examined the distributed firewall program in depth looking at its capabilities and shortfalls. Lastly, we have also shown a primitive heuristic for hybrid SDN topologies that can be applied towards setup but also incremental procurement of SDN switches.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5: Conclusion and Future Work

We're behind the curve. And there's a shame in that, in that, we *literally* invented the Internet. And yet we're the example that other nations point to of, like, "The Americans, don't be a victim like the Americans, don't let what happened to the Americans happen to us." That's the discourse in Sweden, Estonia, France.

—P.W. Singer, Author of *Ghost Fleet* and *Like War* [69]

5.1 Conclusions

During this research we set out to answer two primary research questions:

1. ***Question 1:*** Using the greedy heuristic algorithm for Access Control List (ACL) placement shown in [27], what are the effects of deploying an SDN distributed firewall application in battalion and regimental size USMC tactical networks? What are the effects in terms of security, throughput, and resilience in comparison to a traditional USMC network?
2. ***Question 2:*** Given the work done in [28], how can the USMC begin incrementally purchasing and fielding SDN equipment to maximize immediate networking gains and minimize fiscal cost for the organization?

We examined software defined networking and software defined networking tools from multiple angles producing results that not only help answer these questions but also raise a few new questions that deserve further research. The ability to create and refine an application with the capability of the distributed firewall program demonstrates the power of software defined networking's programmability. Additionally, the distributed firewall program is a practical means of dealing with ubiquitous networking and zero trust environments where segmentation and automated reachability control are necessary. In testing the distributed firewall program, we utilized open source software including Open Networking Operating

System (ONOS) network operating system, Mininet network emulation tool, and Open vSwitch distributed virtual multi-layer switch. We found these software suites more than capable of handling emulation and stress testing tasks, making them suitable alternatives for networking use in tactical Marine Corps Networks.

The distributed firewall program is an effective tool for segmenting networks and filtering traffic automatically between end points. The program automatically detects topology changes and installs flow rules accordingly and could serve as a replacement for real firewalls and are capable of handling a large amounts of flows. Given physical SDN hardware, switching and memory performance should improve. Moreover, if the distributed firewall program is iteratively run between endpoints for which traffic filtering is necessary, only allowed traffic will traverse the network as traffic will immediately be filtered at the first hop that is an SDN capable switch.

Open vSwitch has a high-performance capacity for a distributed virtual multi-layer switch. Given the experimental constraints, topology, and devices, the upper bound for flow rules for an individual switch is greater than or equal to 75,000 and less than 100,000 flow rules — which is larger than typical Marine Corp network device access control list sizes. Flow rule deployment speed is high for rule sets of size smaller than 1000 (less than a second); however, flow rule set deployment speed is low for rule set sizes larger than 1000 (up to 60 seconds for 75,000 flow rules). Rule installation speed is a lower bound because of the low latency inherent with the testing environment and the high bandwidth of experiment 1. More research is needed to scale the distributed firewall program to larger equipment sets, and networks with higher latency and lower bandwidth.

There is a large body of work evaluating and testing Mininet as an effective network emulation tool and SDN prototyping platform [39], [40], [53]. Additionally, there is an easier technical hurtle to overcome using Mininet than using network simulators like ns-3 which are programmed in C; however, testing in Mininet can be slow in comparison to simulators like ns-3 because it requires better knowledge of shell scripting and UNIX command line tools to extract data. Mininet also runs Open vSwitch which enables SDN testing since Open vSwitch is SDN capable by default. Lastly, Mininet is integrated with ONOS, both of which are actively supported by the Open Networking Foundation. [65].

ONOS comes with Mininet support and a Python program called onos.py which integrates the Mininet command line and the ONOS command line into a singular interface. The drawback of using the onos.py is that it can take a long time to setup an environment depending on the size of the topology [70]. With that in consideration, ONOS is a well documented, supported, and actively updated SDN NOS. Learning ONOS can be relatively straightforward because of the large user base, support structure, and active development community [56].

5.2 Limitations and Future Work

This research is not exhaustive and has technical limitations that need to be explored in future work.

1. This work only examined single controller environments. Multi-controller environments are currently in use and are an active area of research [71] in order to alleviate single points of failure and balance processing load for network control. Future research should investigate enterprise networks using multiple SDN controllers to efficiently control flow rule installation and monitor reachability control at the core, distribution, and access layers of the network.
2. The distributed firewall program only filters from a single source s to a single sink t controlled from the ONOS command line. The program needs to be expanded to simultaneously include multiple sources and sinks in one execution so that the operator does not need to run the program multiple times.
3. The distributed firewall program is stateless and therefore wasteful with respect to flow rule installation. A check is not done to see whether or not the same flow rules have already been installed on a switch and sends flow modification packets regardless of the switch's state. It would be faster to only install flow rules on the set of switches that need them.
4. Flow rule batching is not done and typically a set of flow rule modifications and enforcement packets are sent for every flow rule. Batching flow rules into sets can lead to more efficient network bandwidth.
5. A potential method of increasing reachability control reaction time is pre-loading flow rules onto inactive flow tables and activating them when necessary. This would eliminate the need to load rules in real time, decrease switch CPU load until flow rule activation is required, and increase reaction time to the time it takes to send an

activation message from controller to switch.

6. Algorithm optimization can be implemented to improve program reaction time and improve scalability. One example of a start would be to utilize an adjacency list vice a matrix to represent the network topology.
7. An in depth quantitative study of SDN's cyber security risks and vulnerabilities needs to be conducted beyond qualitative supposition to provide methods and means of defending SDN's from active and passive threats.

Our research also identifies several important questions in the areas of Operations Research, Defense Systems Analysis, and Information Technology Management. Given the basic fielding heuristic discussed in Section 4.3, what SDN-capable hardware should the Marine Corps invest in that mimics the switching speed, port density, and memory of the Marine Corps' current networking inventory? Secondly, how should the Marine Corps train and educate its networking and cyber security workforce on SDN? Lastly, besides Marine infantry battalions, how can SDN optimally be phased into Marine Corps data centers in Kansas City and base, posts, and stations?

5.3 Closing Remarks

Software Defined Networking is a networking technology that enables innovation, fine grain control, and automation. It is a technology that can improve the Marine Corps' ability to "orient" and "decide" and "act" in the information age where sorting through avalanches of data are a daily occurrence. This research has shown that correct automated reachability control (through flow rules) is possible without the burden of middleware, third party software, or vendor proprietary software. Fine grain control of network flows are an inherent benefit of Software Defined Networking but SDN's programmability could be a limitless means of innovation. Automation, machine learning, delay tolerant networking, and dynamic resource control are possible with SDN as a platform. Moreover, Software Defined Networks can be created utilizing a small amount of resources. We have suggested an approach for device placement and procurement that could save the Marine Corps money and improve its information capabilities.

APPENDIX: Literary Review, SDN Software, and Source Code

Learning about Software Defined Networking can be overwhelming because of the sheer amount of resources that are available to the uninitiated. The first step in learning about Software Defined Networks is a good background in the fundamentals of traditional networking. If you do not possess that, then studying SDN will be a difficult journey.

A.1 SDN Literary Review

The foundation of learning SDN is conducting a literary review to provide historical, theoretical, and technical context for and about the subject. The book that introduced the author to the subject matter was Kurose and Ross's *Computer Networking: A Top Down Approach 7th Edition* [36]. From there, A History of SDNs covered by *The Road to SDN* by Feamster et al [29] and *Software-defined networking: A comprehensive survey* by Kreutz et al [14] provide a thorough analysis of the history and technical underpinnings of the subject.

A.1.1 Openflow

Openflow and SDN are mutually supporting and intertwined subject matters. While Openflow is not SDN and SDN is not Openflow, you cannot have a complete understanding of SDN without understanding the protocol that enables it. The foundational paper about Openflow [32]

A.2 Open Network Operating System, Mininet, and Open vSwitch

Open Network Operating System is an open-source carrier grade SDN Network Operating System supported by the Open Networking Foundation. Tutorials in its used can be found at [72]. ONOS offers a pre-built virtual machine on which to learn the system. Conversely, you can build ONOS from the source code and create a development platform.

A.2.1 Mininet and Open vSwitch

Another method of learning Software Defined Networking is through using Open vSwitch and Mininet. Open vSwitch is a virtual switching platform that supports the openflow protocol. Using the Open vSwitch commands you can install and configure SDN switches on your computer. Going a step further, mininet can do all of this manual labor for you through instantiating multiple Open vSwitch devices in a topology and configuration of your choice. Mininet tutorials can be found at [39] while Open vSwitch documentation is available at [41].

A.3 Distributed Firewall Command Line Class

DF Command Line Class

```
/*
 * Copyright 2018-present Open Networking Foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.graph;

import org.apache.karaf.shell.commands.Argument;
import org.apache.karaf.shell.commands.Command;
import org.onlab.graph.DefaultEdgeWeigher;
import org.onlab.graph.Weight;
import org.onlab.packet.Ip4Prefix;
import org.onlab.packet.VlanId;
import org.onosproject.cli.AbstractShellCommand;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.DefaultApplicationId;
import org.onosproject.net.DeviceId;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.flow.*;
import org.onosproject.net.link.LinkEvent;
import org.onosproject.net.link.LinkListener;
import org.onosproject.net.link.LinkService;
import org.onosproject.net.topology.*;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
/**
 * Apache Karaf CLI command to determine the edge cut b/w 2 nodes and deploy ACLs
 */

@Command(scope = "onos", name = "kill",
         description = "Deploy Flow Rules to Min Min Edge Cut Set")

public class AppCommand extends AbstractShellCommand {

    @Argument(index = 0, name = "s", description = "source", required = true, multiValued = false)
    private int s = -1;
    @Argument(index = 1, name = "t", description = "destination", required = true, multiValued = false)
    private int t = -1;
    @Argument(index = 2, name = "r", description = "# of rules", required = true, multiValued = false)
    private int r = 1;

    @Override
    public void execute() {
        log.info("Graph Application: Started with min cut between " + s + " and " + t);
        LinkListener linkListener = new LinkListener() {
            @Override
            public void event(LinkEvent event) {
                if (event.type() != null) {
                    try {

                        int x, y = 0, rulesbefore = 0, rulesafter = 0, rulesadded = 0, totalrules=0;
                        DeviceService deviceService = get(DeviceService.class);
                        TopologyService topologyService = get(TopologyService.class);
                        Topology topo = topologyService.currentTopology();
                        TopologyGraph graph = topologyService.getGraph(topo);
                        Set<TopologyEdge> edges = graph.getEdges(); //grab the edges from current topology
                        Set<TopologyVertex> vertexes = graph.getVertexes(); // grab the vertexes from the current topologyon
                        HashMap<DeviceId, Integer> idtonum = new HashMap<>();// deviceid to int
                        HashMap<Integer, String> idtonum2 = new HashMap<>();//for rules
                        HashMap<DeviceId, List> idtostats = new HashMap<>(); //device id to edges
                        HashMap<DeviceId, List> idtoports = new HashMap<>(); // devices and their associated ports
                        DefaultEdgeWeigher edgeWeigher = new DefaultEdgeWeigher();
                        MetricLinkWeight linkWeight = new MetricLinkWeight();
                        FlowRuleService flowRuleService = get(FlowRuleService.class);
                        int devicenum = vertexes.size();
                        int[][] adjmatrix = new int[devicenum][devicenum];
                        DeviceId[] idlist = new DeviceId[devicenum];
                        Graph graph1 = new Graph(devicenum);
                        String[] result;
                        String printout = "";

                        for (TopologyVertex temp1 : vertexes) {
                            String name = String.valueOf(temp1);
                            DeviceId id = temp1.deviceId();
                            idtonum.put(id, y);
                            idtonum2.put(y, name);
                            idlist[y] = id;
                            List ports = deviceService.getPorts(id);
                            List stats = deviceService.getPortStatistics(id);
                            idtostats.put(id, stats);
                            idtoports.put(id, ports);
                            y++;
                        }
                        String linkchangetype = event.type().toString();
                        String linkchangename = event.type().name();
                        printout = printout.concat(" EVENT " + linkchangetype + "\n" + "NAME" + linkchangename + "\n");
                        printout = printout.concat(
                                "######################### device id to matrix number #########################" + "\n");
                        Set set = idtonum.entrySet();
                        Iterator iterator = set.iterator();
                        while (iterator.hasNext()) {
                            Map.Entry entry = (Map.Entry) iterator.next();
                            printout = printout.concat("Device id: " + entry.getKey() + " ; matrix number : " + entry.getValue() + "\n");
                        }
                        printout = printout.concat(
        "##########################    WEIGHTS    ########################################" + "\n");
for (TopologyEdge edgetemp : edges) {
    for (int j = 0; j < devicenum; j++) {
        String edge = String.valueOf(edgetemp);
        String src = null;
        String dst = null;
        String pattern = "((of:)([a-zA-z0-9]*))";
        Pattern p = Pattern.compile(pattern);
        Matcher matcher = p.matcher(edge);
        if (matcher.find()) {
            src = matcher.group(0);
        }
        if (matcher.find(44)) {
            dst = matcher.group(0);
        }
        DeviceId source = DeviceId.deviceId(src);
        DeviceId destination = DeviceId.deviceId(dst);
        DeviceId id = idlist[j];
        if (id.equals(source)) {
            int row = idtonum.get(id);
            int column = idtonum.get(destination);
            Weight weight = edgeWeigher.weight(edgetemp);
            Weight weight1 = linkWeight.weight(edgetemp);
            String stringweight = String.valueOf(weight);
            int lastindex = stringweight.lastIndexOf('}');
            String sw = stringweight.substring(19, lastindex);

            float floatweight = Float.valueOf(sw);
            int intweight = Math.round(floatweight);
            adjmatrix[row][column] = intweight;
            adjmatrix[column][row] = intweight;
        }
    }
}
printout = printout.concat(
        "#########################     MATRIX     #######################################" + "\n");
/*
for (int i = 0; i < devicenum; i++) {
    printout = printout.concat(
            "Row " + i + "------------------------------------------------------------ Row " + i + "\n");
    for (int j = 0; j < devicenum; j++) {
        printout = printout.concat(adjmatrix[i][j] + "\n");
    }
}
*/
printout = printout.concat(
        "######################### SHORTEST PATHS #######################################" + "\n");
ShortestPath shorty = new ShortestPath(devicenum);
int origin = s;
int[] shortydijkstra = shorty.dijkstra(adjmatrix, origin);
printout = printout.concat("Vertex    Distance from Source");
for (int i = 0; i < devicenum; i++) {
    printout = printout.concat(i + "                  " + shortydijkstra[i] + "\n");
}
printout = printout.concat(
        "#########################       CUT       #######################################" + "\n");
Vector devicelist = new Vector();
HashMap<Integer, Integer> nodedistance = new HashMap(); //Key:node Value: Distance from source
if ((s != -1) && (t != -1)) {
    result = graph1.minCut(adjmatrix, s, t);
    x = result.length;
    y = 0;

    for (int i = 0; i < x; i++) {
        if (result[i] != null) {
            String sub = result[i];
            String[] subtwo = sub.split("-");
            int device0 = Integer.parseInt(subtwo[0]);
```

```
                    devicelist.add(y, (device0));
                    int distance0 = shortydijkstra[device0];
                    nodedistance.put(device0, distance0);
                    y++;
                    int device1 = Integer.parseInt(subtwo[1]);
                    devicelist.add(y, (device1));
                    int distance1 = shortydijkstra[device1];
                    nodedistance.put(device1, distance1);
                    y++;
                    printout = printout.concat(result[i] + "\n");
                }
            }
        }
        int devicelistsize = devicelist.size();
        for (int i = 0; i < devicelistsize; i++) {
            Object tempobject = devicelist.get(i);
            String tempstring = tempobject.toString();
            int tempint = Integer.valueOf(tempstring);
            printout = printout.concat(" Device ID: " + idtonum2.get(tempint) + "  Matrix id: " + tempint + "\n");
        }
        printout = printout.concat(
                "######################## MIN MIN CUT     ######################################" + "\n");
        int i = 0;
        result = graph1.minCut(adjmatrix, s, t);
        HashMap<Integer, Integer> finalnodeddistance = new HashMap();
        while (result[i] != null) {
            String sub = result[i];
            printout = printout.concat(result[i] + "\n");
            String[] edgenodes = sub.split("-");
            int node1 = Integer.parseInt(edgenodes[0]);
            int node2 = Integer.parseInt(edgenodes[1]);
            int dist1 = nodedistance.get(node1); //get node 1 distance
            int dist2 = nodedistance.get(node2); //get node 2 distance
            if (dist1 < dist2) {
                //nodedistance.remove(node2);
                finalnodeddistance.put(node1, dist1);
            } else if (dist1 > dist2) {
                //nodedistance.remove(node1);
                finalnodeddistance.put(node2, dist2);
            } else if (dist1 == dist2) {
                finalnodeddistance.put(node1, dist1);
                finalnodeddistance.put(node2, dist2);
            }
            i++;
        }
        Set finalset = finalnodeddistance.keySet();
        Iterator finalnodedistanceiterator = finalset.iterator();
        while (finalnodedistanceiterator.hasNext()) {
            Object obj = finalnodedistanceiterator.next();
            Integer finalcutint = Integer.valueOf(obj.toString());
            String finalcutnode = idtonum2.get(finalcutint);
            printout = printout.concat(" final cut node: " + finalcutnode + "\n");
        }
        printout = printout.concat(
                "######################## ACL/FLOW RULES ######################################" + "\n");
        // get rid of duplicates in the list
        try {
            final DefaultFlowRule.Builder flowrulebuilder = DefaultFlowRule.builder();

            print("The total number of flow rules is : ");
            printout = printout.concat("The total number of flow rules is : ");
            rulesbefore = flowRuleService.getFlowRuleCount();
            print(String.valueOf(rulesbefore));
            printout = printout.concat(rulesbefore + "\n");

            print("#####################Reading Input Rules#################################");
            printout = printout.concat("#####################Reading Input Rules#################################" + "\n");
```

```
Set finale = finalnodeddistance.keySet();
Iterator finaleiterator = finale.iterator();


while (finaleiterator.hasNext()) {
    // create # of rules equal to the # in r
    Object finalobj = finaleiterator.next();
    Integer finaleint = Integer.valueOf(finalobj.toString());
    BufferedReader bufferedReader = new BufferedReader
            (new FileReader("/home/brent/inputrules/rules.txt"));
    String line;
    String rule;
    while ((line = bufferedReader.readLine()) != null) {
        rule = line;
        print(rule);
        String denypattern = "(\\w+_\\w+_?\\w+?)\\s(\\d+.\\d+.\\d+.\\d+)";
        String vlanpatten = "((\\w+_\\w+_?\\w+?)\\s(\\d+)_(\\d+))";

        Pattern deny = Pattern.compile(denypattern);
        Pattern vlan = Pattern.compile(vlanpatten);

        Matcher denymatcher = deny.matcher(rule);
        Matcher vlanmatcher = vlan.matcher(rule);

        if( denymatcher.find()) {
            print("DENY RULE");
            String denyipaddress = denymatcher.group(2);
            TrafficSelector.Builder selectorbuilder = DefaultTrafficSelector.builder();
            TrafficTreatment.Builder treatmentbuilder = DefaultTrafficTreatment.builder();
            //build rule one
            ApplicationId applicationId = new DefaultApplicationId(158, "org.onosproject.graph");
            //DeviceId deviceId = DeviceId.deviceId("of:0000000000000001");
            short type = 0x800;
            int priority = 40000;
            int timeout = 10000;

            Ip4Prefix ip4Prefixdst1 = Ip4Prefix.valueOf(denyipaddress + "/32");
            printout = printout.concat(rule + "\n");

            String tempname = idtonum2.get(finaleint);
            //print(" rule place onto " + tempname);
            printout = printout.concat(" rule place onto " + tempname + "\n");
            DeviceId deviceId = DeviceId.deviceId(tempname);
            TrafficSelector selector = selectorbuilder.
                    matchIPDst(ip4Prefixdst1).
                    matchEthType(type).
                    build();
            TrafficTreatment treatment = treatmentbuilder.
                    drop().
                    build();
            FlowRule rule1 = flowrulebuilder.
                    withSelector(selector).
                    withTreatment(treatment).
                    makePermanent().
                    forDevice(deviceId).
                    fromApp(applicationId).
                    withPriority(priority).
                    withHardTimeout(timeout).
                    build();

            flowRuleService.applyFlowRules(rule1);
            rulesadded = rulesadded + 1;

        } else if (vlanmatcher.find()){
            print("VLAN RULE");
            Short vlanfrom = Short.parseShort(vlanmatcher.group(3));
            Short vlanto = Short.parseShort(vlanmatcher.group(4));
            VlanId vlanIdto = VlanId.vlanId(vlanto);
            VlanId vlanIdfrom = VlanId.vlanId(vlanfrom);

            TrafficSelector.Builder selectorbuilder = DefaultTrafficSelector.builder();
            TrafficTreatment.Builder treatmentbuilder = DefaultTrafficTreatment.builder();
            //build rule one
            ApplicationId applicationId = new DefaultApplicationId(158, "org.onosproject.graph");
            //DeviceId deviceId = DeviceId.deviceId("of:0000000000000001");
            short type = 0x800;
            int priority = 40000;
            int timeout = 10000;
            printout = printout.concat(rule + "\n");
            String tempname = idtonum2.get(finaleint);
            //print(" rule place onto " + tempname);
            printout = printout.concat(" rule place onto " + tempname + "\n");
            DeviceId deviceId = DeviceId.deviceId(tempname);
            TrafficSelector selector = selectorbuilder.
                    matchVlanId(vlanIdfrom).
                    matchEthType(type).
                    build();
            TrafficTreatment treatment = treatmentbuilder.
                    setVlanId(vlanIdto)
                    .build();
            FlowRule rule1 = flowrulebuilder.
                    withSelector(selector).
                    withTreatment(treatment).
                    makePermanent().
                    forDevice(deviceId).
                    fromApp(applicationId).
                    withPriority(priority).
                    withHardTimeout(timeout).
                    build();

            flowRuleService.applyFlowRules(rule1);
            rulesadded = rulesadded + 1;

        } else {
            print("no bueno");
        }
    }
}

rulesafter = flowRuleService.getFlowRuleCount();
print("Original Rulecount =  " + rulesbefore + "   Rulecount After = " + rulesafter
        + " Rules Added #  " + rulesadded);
printout = printout.concat(
        "Original Rulecount =  " + rulesbefore + "   Rulecount After = " + rulesafter
                + "Rules Added #  " + rulesadded + "\n");
 print("##############################Printing Results################################" + "\n");
try {
    String rulescreated = String.valueOf(rulesadded);
    String rulesrequested = String.valueOf(r);
    String rules = "";
    ApplicationId applicationId = new DefaultApplicationId(158, "org.onosproject.graph");
    Iterable iterable = flowRuleService.getFlowEntriesById(applicationId);
    Integer counted = 1;
    for (Object s : iterable) {
        rules = rules.concat(" Rule " + counted + " : " + s.toString() + "\n");
        counted = counted + 1;
    }
    String fileName = new SimpleDateFormat("yyyyMMddHHmmssSS'.txt'").format(new Date());
    FileWriter fileWriter = new FileWriter("/home/brent/onostopologychange/LINK_CHANGE_DETECTEDRESULTS" + fileName + '
    fileWriter.write(fileName + "Rules created : " + rulescreated + "\n" +
            "Rules requested : " + rulesrequested + "\n" + "Rules added: " + rules + "\n" + printout);
    fileWriter.close();
} catch (IOException e) {
    String notification = e.toString();
    FileWriter fileWriter = new FileWriter(
                                "/home/brent/onostopologychange/topologychangedexception1.txt");
                        fileWriter.write(notification);
                        fileWriter.close();
                    }
                } catch (Exception e) {
                    String notification = e.toString();
                    FileWriter fileWriter = new FileWriter(
                            "/home/brent/onostopologychange/topologychangedexception2.txt");
                    fileWriter.write(notification);
                    fileWriter.close();
                }

            } catch (Exception e) {
                try {
                    String notification = e.toString();
                    FileWriter fileWriter = new FileWriter(
                            "/home/brent/onostopologychange/topologychangedexception3.txt");
                    fileWriter.write(notification);
                    fileWriter.close();
                } catch (IOException i) {
                }
            }
        }
    }
};
/*
TopologyListener topologyListener = new TopologyListener() {
    @Override
    public void event(TopologyEvent event) {
        if(event.type() == TopologyEvent.Type.TOPOLOGY_CHANGED){

            Integer presult=null , counter = 0 , limit = 20;
            String[] output = new String[20];
            try{
                String[] mycommand = new String[]{"sh onos kill "+s + " "+ t + " " + r};
                Process p = Runtime.getRuntime().exec(mycommand);
                presult = p.waitFor();

                BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
                String readline;
                while ((readline = reader.readLine()) != null & counter < limit ){
                    output[counter] = (readline);
                    counter = counter + 1;
                }
                p.destroy();
                reader.close();

            } catch (Exception e){
                log.info("Exception " + e.toString());
                try{
                    String notification = e.toString();
                    FileWriter fileWriter = new FileWriter(
                            "/home/brent/onostopologychange/topologychangedexception.txt");
                    fileWriter.write(notification);
                    fileWriter.close();
                }
                catch (IOException i){

                }
            }

            try {
                String date = new SimpleDateFormat("yyyyMMddHHmm").format(new Date());
                String notification = "1" + event.toString()
                        + " event time" + event.time()
                        + " CPU time : " + date
                        + " process exit code: " + presult
                        + " output " + Arrays.toString(output);
                String fileName = "topologyinfo";
                FileWriter fileWriter = new FileWriter(
                        "/home/brent/onostopologychange/"+fileName +".txt");
                fileWriter.write(notification);
                fileWriter.close();
            }
            catch (IOException e){
              log.info("caught exception " + e.toString());
            }
        }
    }
};
*/
//topologyService.addListener(topologyListener);
LinkService linkService = get(LinkService.class);
linkService.addListener(linkListener);
int x, y = 0, rulesbefore = 0, rulesafter = 0, rulesadded = 0;
DeviceService deviceService = get(DeviceService.class);
TopologyService topologyService = get(TopologyService.class);
Topology topo = topologyService.currentTopology();
//listenerRegistry.addListener(tl);
TopologyGraph graph = topologyService.getGraph(topo);
Set<TopologyEdge> edges = graph.getEdges(); //grab the edges from current topology
Set<TopologyVertex> vertexes = graph.getVertexes(); // grab the vertexes from the current topologyon
HashMap<DeviceId, Integer> idtonum = new HashMap<>();// deviceid to int
HashMap<Integer, String> idtonum2 = new HashMap<>();//for rules
HashMap<DeviceId, List> idtostats = new HashMap<>(); //device id to edges
HashMap<DeviceId, List> idtoports = new HashMap<>(); // devices and their associated ports
DefaultEdgeWeigher edgeWeigher = new DefaultEdgeWeigher();
MetricLinkWeight linkWeight = new MetricLinkWeight();
FlowRuleService flowRuleService = get(FlowRuleService.class);
int devicenum = vertexes.size();
int[][] adjmatrix = new int[devicenum][devicenum];
DeviceId[] idlist = new DeviceId[devicenum];
Graph graph1 = new Graph(devicenum);
String[] result;
String printout = "";

for (TopologyVertex temp1 : vertexes) {
    String name = String.valueOf(temp1);
    DeviceId id = temp1.deviceId();
    idtonum.put(id, y);
    idtonum2.put(y, name);
    idlist[y] = id;
    List ports = deviceService.getPorts(id);
    List stats = deviceService.getPortStatistics(id);
    idtostats.put(id, stats);
    idtoports.put(id, ports);
    y++;
}
print("######################### device id to matrix number #########################");
printout = printout.concat(
        "######################### device id to matrix number #########################" + "\n");
//get a set of entries
Set set = idtonum.entrySet();
//get an iterator
Iterator iterator = set.iterator();
while (iterator.hasNext()) {
    Map.Entry entry = (Map.Entry) iterator.next();
    print("Device id: " + entry.getKey() + " ; matrix number : " + entry.getValue());
    printout = printout.concat("Device id: " + entry.getKey() + " ; matrix number : " + entry.getValue() + "\n");
}
print("#########################   WEIGHTS   ########################################");
printout = printout.concat(
        "#########################   WEIGHTS   ########################################" + "\n");
for (TopologyEdge edgetemp : edges) {
    for (int j = 0; j < devicenum; j++) {
        //DefaultTopologyEdge{src=of:0000000000000001, dst=of:0000000000000002}
        String edge = String.valueOf(edgetemp);
        String src = null;
        String dst = null;
        String pattern = "((of:)([a-zA-z0-9]*))";
        Pattern p = Pattern.compile(pattern);
        Matcher matcher = p.matcher(edge);
        if (matcher.find()) {
            src = matcher.group(0);
        }
        if (matcher.find(44)) {
            dst = matcher.group(0);
        }
        DeviceId source = DeviceId.deviceId(src);
        DeviceId destination = DeviceId.deviceId(dst);
        DeviceId id = idlist[j];
        if (id.equals(source)) {
            int row = idtonum.get(id);
            int column = idtonum.get(destination);
            Weight weight = edgeWeigher.weight(edgetemp);
            Weight weight1 = linkWeight.weight(edgetemp);
            String stringweight = String.valueOf(weight);
            int lastindex = stringweight.lastIndexOf('}');
            String sw = stringweight.substring(19, lastindex);
            float floatweight = Float.valueOf(sw);
            int intweight = Math.round(floatweight);
            adjmatrix[row][column] = intweight;
            adjmatrix[column][row] = intweight;
        }
    }
}
print("#########################     MATRIX     #######################################");
printout = printout.concat(
        "#########################     MATRIX     #######################################" + "\n");
/*
for (int i = 0; i < devicenum; i++) {
    print("Row " + i + "--------------------------------------------------------- Row " + i);
    printout = printout.concat(
            "Row " + i + "--------------------------------------------------------- Row " + i + "\n");
    for (int j = 0; j < devicenum; j++) {
        print(String.valueOf(adjmatrix[i][j]));
        printout = printout.concat(adjmatrix[i][j] + "\n");
    }
}
*/
print("######################## SHORTEST PATHS #######################################");
printout = printout.concat(
        "######################## SHORTEST PATHS #######################################" + "\n");
ShortestPath shorty = new ShortestPath(devicenum);
int origin = s;
int[] shortydijkstra = shorty.dijkstra(adjmatrix, origin);
print("Vertex    Distance from Source");
printout = printout.concat("Vertex    Distance from Source" + "\n");
for (int i = 0; i < devicenum; i++) {
    print(i + "                    " + shortydijkstra[i]);
    printout = printout.concat(i + "                    " + shortydijkstra[i] + "\n");
}
print("#########################       CUT      #######################################");
printout = printout.concat(
        "#########################       CUT      #######################################" + "\n");
Vector devicelist = new Vector();
HashMap<Integer, Integer> nodedistance = new HashMap(); //Key:node Value: Distance from source
if ((s != -1) && (t != -1)) {
    result = graph1.minCut(adjmatrix, s, t);
    x = result.length;
    y = 0;

    for (int i = 0; i < x; i++) {
        if (result[i] != null) {
            String sub = result[i];
            String[] subtwo = sub.split("-");
            int device0 = Integer.parseInt(subtwo[0]);
            devicelist.add(y, (device0));
            int distance0 = shortydijkstra[device0];
            nodedistance.put(device0, distance0);
            y++;
            int device1 = Integer.parseInt(subtwo[1]);
            devicelist.add(y, (device1));
            int distance1 = shortydijkstra[device1];
            nodedistance.put(device1, distance1);
            y++;
            print(String.valueOf(result[i]));
            printout = printout.concat(result[i] + "\n");

        }
    }
}
int devicelistsize = devicelist.size();
for (int i = 0; i < devicelistsize; i++) {
    Object tempobject = devicelist.get(i);
    String tempstring = tempobject.toString();
    int tempint = Integer.valueOf(tempstring);
    //intarray[i] = tempint;
    print(" Device ID: " + idtonum2.get(tempint) + "  Matrix id: " + tempint);
    printout = printout.concat(" Device ID: " + idtonum2.get(tempint) + "  Matrix id: " + tempint + "\n");
}
print("######################## MIN MIN CUT    #####################################");
printout = printout.concat("######################## MIN MIN CUT    #####################################" + "\n");
result = graph1.minCut(adjmatrix, s, t);
HashMap<Integer, Integer> finalnodeddistance = new HashMap<>();
int i = 0;
while (result[i] != null) {
    String sub = result[i];
    print(result[i]);
    printout = printout.concat(result[i] + "\n");
    String[] edgenodes = sub.split("-");
    int node1 = Integer.parseInt(edgenodes[0]);
    int node2 = Integer.parseInt(edgenodes[1]);
    int dist1 = nodedistance.get(node1); //get node 1 distance
    int dist2 = nodedistance.get(node2); //get node 2 distance
    if (dist1 < dist2) {
        //nodedistance.remove(node2);
        finalnodeddistance.put(node1, dist1);
    } else if (dist1 > dist2) {
        //nodedistance.remove(node1);
        finalnodeddistance.put(node2, dist2);
    } else if (dist1 == dist2) {
        finalnodeddistance.put(node1, dist1);
        finalnodeddistance.put(node2, dist2);
    }
    i++;
}
Set finalset = finalnodeddistance.keySet();
Iterator finalnodedistanceiterator = finalset.iterator();
while (finalnodedistanceiterator.hasNext()) {
    Object obj = finalnodedistanceiterator.next();
    Integer finalcutint = Integer.valueOf(obj.toString());
    String finalcutnode = idtonum2.get(finalcutint);
    print(" final cut node: " + finalcutnode);
    printout = printout.concat(" final cut node: " + finalcutnode + "\n");
}
print("######################## ACL/FLOW RULES #####################################");
printout = printout.concat(
        "######################## ACL/FLOW RULES #####################################" + "\n");
// get rid of duplicates in the list
try {
    final DefaultFlowRule.Builder flowrulebuilder = DefaultFlowRule.builder();
```

```
print("The total number of flow rules is : ");
printout = printout.concat("The total number of flow rules is : ");
rulesbefore = flowRuleService.getFlowRuleCount();
print(String.valueOf(rulesbefore));
printout = printout.concat(rulesbefore + "\n");

print("#####################Reading Input Rules#################################");
printout = printout.concat("#####################Reading Input Rules#################################" + "\n");

Set finale = finalnodeddistance.keySet();
Iterator finaleiterator = finale.iterator();

while (finaleiterator.hasNext()) {
    // create # of rules equal to the # in r
    Object finalobj = finaleiterator.next();
    Integer finaleint = Integer.valueOf(finalobj.toString());
    BufferedReader bufferedReader = new BufferedReader
            (new FileReader("/home/brent/inputrules/rules.txt"));
    String line;
    String rule;
    Integer barrier = 1000;

    Integer slowroll = 100;
    Integer rulebatch = 10;
    //ArrayList<FlowRule> rulelist = new ArrayList<FlowRule>();
    // start sleeping 200ms every 100 rules after 1000 rules are applied
    while ((line = bufferedReader.readLine()) != null) {
        rule = line;
        try {
            if (rulesadded > barrier) {
                if ((rulesadded % slowroll) == 0) {
                    //System.gc();
                    Thread.sleep(75);

                }
            }

        }catch (Exception e){
            print("sleep error");
        }
        //print(String.valueOf("rules added " + rulesadded));
        //if rules

        String denypattern = "(\\w+_\\w+)\\s(\\d+\\.\\d+\\.\\d+\\.\\d+)";
        String vlanpatten = "((\\w+_\\w+_?\\w+?)\\s(\\d+)_(\\d+))";

        Pattern deny = Pattern.compile(denypattern);
        Pattern vlan = Pattern.compile(vlanpatten);

        Matcher denymatcher = deny.matcher(rule);
        Matcher vlanmatcher = vlan.matcher(rule);

        if( denymatcher.find()) {
            //print("DENY RULE");
            String denyipaddress = denymatcher.group(2);
            //print (denyipaddress);
            TrafficSelector.Builder selectorbuilder = DefaultTrafficSelector.builder();
            TrafficTreatment.Builder treatmentbuilder = DefaultTrafficTreatment.builder();
            //build rule one
            ApplicationId applicationId = new DefaultApplicationId(158, "org.onosproject.graph");
            //DeviceId deviceId = DeviceId.deviceId("of:0000000000000001");
            short type = 0x800;
            int priority = 40000;
            int timeout = 10000;

            Ip4Prefix ip4Prefixdst1 = Ip4Prefix.valueOf(denyipaddress + "/32");
            //printout = printout.concat(rule + "\n");
    String tempname = idtonum2.get(finaleint);
    //print(" rule place onto " + tempname);
    //printout = printout.concat(" rule place onto " + tempname + "\n");
    DeviceId deviceId = DeviceId.deviceId(tempname);
    TrafficSelector selector = selectorbuilder.
            matchIPDst(ip4Prefixdst1).
            matchEthType(type).
            build();
    TrafficTreatment treatment = treatmentbuilder.
            drop().
            build();
    FlowRule rule1 = flowrulebuilder.
            withSelector(selector).
            withTreatment(treatment).
            makePermanent().
            forDevice(deviceId).
            fromApp(applicationId).
            withPriority(priority).
            withHardTimeout(timeout).
            build();

    flowRuleService.applyFlowRules(rule1);
    //rulelist.add(rule1);
    rulesadded = rulesadded + 1;

} else if (vlanmatcher.find()){
    //print("VLAN RULE");
    Short vlanfrom = Short.parseShort(vlanmatcher.group(3));
    Short vlanto = Short.parseShort(vlanmatcher.group(4));

    VlanId vlanIdto = VlanId.vlanId(vlanto);
    VlanId vlanIdfrom = VlanId.vlanId(vlanfrom);

    TrafficSelector.Builder selectorbuilder = DefaultTrafficSelector.builder();
    TrafficTreatment.Builder treatmentbuilder = DefaultTrafficTreatment.builder();
    //build rule one
    ApplicationId applicationId = new DefaultApplicationId(158, "org.onosproject.graph");
    //DeviceId deviceId = DeviceId.deviceId("of:0000000000000001");
    short type = 0x800;
    int priority = 40000;
    int timeout = 10000;
    //printout = printout.concat(rule + "\n");
    String tempname = idtonum2.get(finaleint);
    //print(" rule place onto " + tempname);
    //printout = printout.concat(" rule place onto " + tempname + "\n");
    DeviceId deviceId = DeviceId.deviceId(tempname);
    TrafficSelector selector = selectorbuilder.
            matchVlanId(vlanIdfrom).
            matchEthType(type).
            build();
    TrafficTreatment treatment = treatmentbuilder.
            setVlanId(vlanIdto)
            .build();
    FlowRule rule1 = flowrulebuilder.
            withSelector(selector).
            withTreatment(treatment).
            makePermanent().
            forDevice(deviceId).
            fromApp(applicationId).
            withPriority(priority).
            withHardTimeout(timeout).
            build();

    flowRuleService.applyFlowRules(rule1);
    //rulelist.add(rule1);
    rulesadded = rulesadded + 1;
```

```
            } else {
                print("no bueno");
            }
            //
            /*
            if ((r % rulebatch != 0 )){
                if (rulesadded > 0) {
                    if ((rulesadded % rulebatch) == 0) {
                        //FlowRuleOperations flowRuleOperations = flowruleopsbuilder.build();
                        flowRuleService.applyFlowRules(
                                rulelist.get(0),
                                rulelist.get(1),
                                rulelist.get(2),
                                rulelist.get(3),
                                rulelist.get(4),
                                rulelist.get(5),
                                rulelist.get(6),
                                rulelist.get(7),
                                rulelist.get(8),
                                rulelist.get(9)
                        );
                        rulelist.clear();
                    } else {
                        for (int j = 0; j < rulelist.size(); j++) {
                            flowRuleService.applyFlowRules(rulelist.get(j));
                        }
                        rulelist.clear();
                    }
                }

            }else if ((r % rulebatch) == 0) {
                //print("10 mod rules");
                if (rulesadded > 0) {
                    //print("rules added > 0");
                    if ((rulesadded % rulebatch) == 0) {
                        //print ("rules added mod 10");
                        //FlowRuleOperations flowRuleOperations = flowruleopsbuilder.build();
                        flowRuleService.applyFlowRules(
                                rulelist.get(0),
                                rulelist.get(1),
                                rulelist.get(2),
                                rulelist.get(3),
                                rulelist.get(4),
                                rulelist.get(5),
                                rulelist.get(6),
                                rulelist.get(7),
                                rulelist.get(8),
                                rulelist.get(9)
                        );
                        rulelist.clear();
                        //print(String.valueOf("size " + rulelist.size()));
                    }
                }
            }
                */
            //
        }
    }

        //rulesafter = flowRuleService.getFlowRuleCount();
        //print("Original Rulecount =  " + rulesbefore + "   Rulecount After = " + rulesafter
        //        + " Rules Added #  " + rulesadded);
        //printout = printout.concat(
        //         "Original Rulecount =  " + rulesbefore + "   Rulecount After = " + rulesafter
        //                 + "Rules Added #  " + rulesadded + "\n");
        print("############################Printing Results################################");
        try {
            String rulescreated = String.valueOf(rulesadded);
```

```
                String rulesrequested = String.valueOf(r);
                String rules = "";
                //ApplicationId applicationId = new DefaultApplicationId(158, "org.onosproject.graph");
                //Iterable iterable = flowRuleService.getFlowEntriesById(applicationId);
                //Integer counter = 1;
                //for (Object s : iterable) {
                //    rules = rules.concat("Rule " + counter + " :" + s.toString() + "\n");
                //    counter = counter + 1;
                //}
                String fileName = new SimpleDateFormat("yyyyMMddHHmmssSS'.txt'").format(new Date());
                FileWriter fileWriter = new FileWriter("/home/brent/captures/OUTCOME" + fileName + ".txt");
                fileWriter.write("rules created : " + rulescreated + "\n" +
                        " Rules requested : " + rulesrequested + "\n" + " Rules added: " + rules + "\n" + printout);
                fileWriter.close();
            } catch (IOException e) {
                print(e.toString());
            }
        } catch (IOException e) {
        print(e.toString());
        }
    }
}
```

A.4 Distributed Firewall Graph Class

The graph class was used to gather the graph representation of the ONOS topology. This code was taken from the Geeks for Geeks website at [59].

Graph Class

```
package org.graph;

import java.util.LinkedList;
import java.util.Queue;

public class Graph {

        /*
        Returns true if there is a path
        from source 's' to sink 't' in residual
        graph. Also fills parent[] to store the path
        code pulled from
        https://www.geeksforgeeks.org/minimum-cut-in-a-directed-graph/
        and modified to for use.
        */

        private String[] result = null;

        public Graph(Integer indy){
            this.result = new String[indy];
        }

        public static boolean bfs(int[][] rGraph, int s,
                                  int t, int[] parent) {

            // Create a visited array and mark
            // all vertices as not visited
            boolean[] visited = new boolean[rGraph.length];

            // Create a queue, enqueue source vertex
            // and mark source vertex as visited
            Queue<Integer> q = new LinkedList<Integer>();
```

```
    q.add(s);
    visited[s] = true;
    parent[s] = -1;

    // Standard BFS Loop
    while (!q.isEmpty()) {
        int v = q.poll();
        for (int i = 0; i < rGraph.length; i++) {
            if (rGraph[v][i] > 0 && !visited[i]) {
                q.offer(i);
                visited[i] = true;
                parent[i] = v;
            }
        }
    }

    // If we reached sink in BFS starting
    // from source, then return true, else false
    return (visited[t] == true);
}

// A DFS based function to find all reachable
// vertices from s. The function marks visited[i]
// as true if i is reachable from s. The initial
// values in visited[] must be false. We can also
// use BFS to find reachable vertices
public static void dfs(int[][] rGraph, int s,
                        boolean[] visited) {
    visited[s] = true;
    for (int i = 0; i < rGraph.length; i++) {
        if (rGraph[s][i] > 0 && !visited[i]) {
            dfs(rGraph, i, visited);
        }
    }
}

// Prints the minimum s-t cut
//private static void minCut(int[][] graph, int s, int t)
public String[] minCut(int[][] graph, int s, int t) {
    int u, v;
    int w = 0;

    // Create a residual graph and fill the residual
    // graph with given capacities in the original
    // graph as residual capacities in residual graph
    // rGraph[i][j] indicates residual capacity of edge i-j
    int[][] rGraph = new int[graph.length][graph.length];
    for (int i = 0; i < graph.length; i++) {
        for (int j = 0; j < graph.length; j++) {
            rGraph[i][j] = graph[i][j];
        }
    }

    // This array is filled by BFS and to store path
    int[] parent = new int[graph.length];

    // Augment the flow while tere is path from source to sink
    while (bfs(rGraph, s, t, parent)) {

        // Find minimum residual capacity of the edhes
        // along the path filled by BFS. Or we can say
        // find the maximum flow through the path found.
        int pathFlow = Integer.MAX_VALUE;
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            pathFlow = Math.min(pathFlow, rGraph[u][v]);
        }
```

```
            // update residual capacities of the edges and
            // reverse edges along the path
            for (v = t; v != s; v = parent[v]) {
                u = parent[v];
                rGraph[u][v] = rGraph[u][v] - pathFlow;
                rGraph[v][u] = rGraph[v][u] + pathFlow;
            }
        }

        // Flow is maximum now, find vertices reachable from s
        boolean[] isVisited = new boolean[graph.length];
        dfs(rGraph, s, isVisited);

        // Print all edges that are from a reachable vertex to
        // non-reachable vertex in the original graph
        for (int i = 0; i < graph.length; i++) {
            for (int j = 0; j < graph.length; j++) {
                if (graph[i][j] > 0 && isVisited[i] && !isVisited[j]) {
                    result[w] = (i + "-" + j);
                    w++;
                    //System.out.println(i + " - " + j);
                }
            }
        }
        return result;
    }
}
```

A.5 Distributed Firewall Dijkstra Class

This code was used to find the shortest paths from s to each node in the network to prune the farthest node from the *MinMinCut*. This code was taken from the Geeks for Geeks website at [73].

Djikstra Class

```
package org.graph;

// A Java program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

class ShortestPath {
    // A utility function to find the vertex with minimum distance value,
    // from the set of vertices not yet included in shortest path tree
    // adapted from https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
    private static int V = 0;

    //private int[][] adjmatrix = null;
    //constructor to ingest the adjmatrix from appcommand
    public ShortestPath(int indy){
        V = indy;
    }

    int minDistance(int[] dist, Boolean[] sptSet) {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }

        return min_index;
    }

    // A utility function to print the constructed distance array
    /*
    void printSolution(int dist[], int n) {
        System.out.println("Vertex   Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i + " tt " + dist[i]);
    }
    */
    // Funtion that implements Dijkstra's single source shortest path
    // algorithm for a graph represented using adjacency matrix
    // representation
    public int[] dijkstra(int[][] graph, int src) {
        int[] dist = new int[V]; // The output array. dist[i] will hold
        // the shortest distance from src to i

        // sptSet[i] will true if vertex i is included in shortest
        // path tree or shortest distance from src to i is finalized
        Boolean[] sptSet = new Boolean[V];

        // Initialize all distances as INFINITE and stpSet[] as false
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Distance of source vertex from itself is always 0
        dist[src] = 0;

        // Find shortest path for all vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick the minimum distance vertex from the set of vertices
            // not yet processed. u is always equal to src in first
            // iteration.
            int u = minDistance(dist, sptSet);

            // Mark the picked vertex as processed
            sptSet[u] = true;

            // Update dist value of the adjacent vertices of the
            // picked vertex.
            for (int v = 0; v < V; v++)

                // Update dist[v] only if is not in sptSet, there is an
                // edge from u to v, and total weight of path from src to
                // v through u is smaller than current value of dist[v]
                if (!sptSet[v] && graph[u][v] != 0 &&
                        dist[u] != Integer.MAX_VALUE &&
                        dist[u] + graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
        }

        // print the constructed distance array
        //printSolution(dist, V);
        return (dist);
    }
}
```

A.6 Access Control List Rule Generator

Below is the code used to randomly generate a list of IP addresses to block and VLANs to change traffic to. This was used to create large lists of rules to replicate ACLs. This code was generated with the help of Terry Kvitchko.

ACL Rule Generator

```
#!/usr/bin/python3

import sys
import ipaddress

if len(sys.argv)<2:
        print("Needs command line argument: number of rules.")
        exit()

rule_types = [
        "ip_deny",
        "vlan_id_change"
        ]

num_rules = int(sys.argv[1])
loop_count = num_rules // len(rule_types)
leftover_rules = num_rules%len(rule_types)

currIP = ipaddress.ip_address(u'10.0.0.10')

rulefile  = open("rulegen.txt", "w")

for i in range(0, loop_count):

        for j in range(0, len(rule_types)):
                rulefile.write(rule_types[j]+" " + str(currIP)+"\n")

        currIP+=1
        if (i!=0 and (i+10)%254 == 0):  #skip broadcast and network addresses
                currIP+=2

for j in range(0, leftover_rules):
        rulefile.write(rule_types[j]+" " + str(currIP)+"\n")

rulefile.close()
print("Wrote to rulegen.txt")
```

A.7 Mininet Custom Topologies

Below is the Python Code used to generate custom topologies for use in Mininet that were included in the ONOS.py file that comes with ONOS.

A.7.1 Regimental Topology

Regimental Topology

```
class Regiment( Topo ):
    "Hybrid Battalion Topology"
```

```
def __init__( self ):
    "Create custom battalion topo."

    # Initialize topology
    Topo.__init__( self )

    s14 = self.addSwitch('s14')
    s3 = self.addSwitch('s3')
    s23 = self.addSwitch('s23')
    s25 = self.addSwitch('s25')
    s28 = self.addSwitch('s28')
    s12 = self.addSwitch('s12')
    s21 = self.addSwitch('s21')
    s26 = self.addSwitch('s26')
    s6 = self.addSwitch('s6')
    s1 = self.addSwitch('s1')
    s18 = self.addSwitch('s18')
    s8 = self.addSwitch('s8')
    s15 = self.addSwitch('s15')
    s17 = self.addSwitch('s17')
    s24 = self.addSwitch('s24')
    s19 = self.addSwitch('s19')
    s20 = self.addSwitch('s20')
    s11 = self.addSwitch('s11')
    s5 = self.addSwitch('s5')
    s13 = self.addSwitch('s13')
    s10 = self.addSwitch('s10')
    s4 = self.addSwitch('s4')
    s16 = self.addSwitch('s16')
    s9 = self.addSwitch('s9')
    s7 = self.addSwitch('s7')
    s2 = self.addSwitch('s2')
    s22 = self.addSwitch('s22')
    s27 = self.addSwitch('s27')

    info( '*** Add hosts\n')
    h2 = self.addHost('h2')
    h5 = self.addHost('h5')
    h6 = self.addHost('h6')
    h1 = self.addHost('h1')

    info( '*** Add links\n')
    self.addLink(s16, s23, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s18, s25, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s17, s2, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s21, s4, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s3, s10, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s7, s14, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s9, s24, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s11, s28, cls=TCLink, bw=4, delay='10ms', loss=0, use_htb=True)
    self.addLink(s22, s15, cls=TCLink, bw=45, delay='10ms', loss=0, use_htb=True)
    self.addLink(s15, s1, cls=TCLink, bw=45, delay='10ms', loss=0, use_htb=True)
    self.addLink(s1, s8, cls=TCLink, bw=45, delay='10ms', loss=0, use_htb=True)
    self.addLink(s8, s15, cls=TCLink, bw=45, delay='10ms', loss=0, use_htb=True)
    self.addLink(s22, s8, cls=TCLink, bw=45, delay='10ms', loss=0, use_htb=True)
    self.addLink(s22, s1, cls=TCLink, bw=45, delay='10ms', loss=0, use_htb=True)
    self.addLink(s22, s24, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s23, s22, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s24, s23, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s23, s26, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s25, s23, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s24, s27, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s24, s28, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s10, s13, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s10, s8, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s10, s14, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
    self.addLink(s9, s10, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
```

```
        self.addLink(s9, s12, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s9, s11, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s8, s9, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s1, s3, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s3, s2, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s2, s4, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s2, s5, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s2, s1, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s3, s6, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s3, s7, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s15, s16, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s15, s17, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s17, s20, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s17, s21, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s16, s19, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s16, s18, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s16, s17, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(h1, s19, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(h2, s20, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(s13, h6, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
        self.addLink(h5, s12, cls=TCLink, bw=100, delay='1ms', loss=0, use_htb=True)
```

A.7.2 Battalion Topology

Battalion Topology

```
class Battalion( Topo ):
    "Battalion Topology"

    def __init__( self ):
        "Create custom battalion topo."

        # Initialize topology
        Topo.__init__( self )

        s2 = self.addSwitch('s2')
        s5 = self.addSwitch('s5')
        s3 = self.addSwitch('s3')
        s1 = self.addSwitch('s1')
        s6 = self.addSwitch('s6')
        s4 = self.addSwitch('s4')
        s7 = self.addSwitch('s7')

        h6 = self.addHost('h6')
        h4 = self.addHost('h4')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h7 = self.addHost('h7')
        h5 = self.addHost('h5')
        h3 = self.addHost('h3')
        h8 = self.addHost('h8')

        self.addLink(s3, h2)
        self.addLink(h3, s4)
        self.addLink(s4, h4)
        self.addLink(h5, s5)
        self.addLink(s2, s1)
        self.addLink(s3, s1)
        self.addLink(s1, s4)
        self.addLink(s5, s2)
        self.addLink(s2, s6)
        self.addLink(s1, s7)
        self.addLink(s7, s2)
        self.addLink(s5, h6)
        self.addLink(s6, h8)
        self.addLink(h7, s6)
        self.addLink(h1, s3)
        #lower link is for add
```

A.7.3 Link Change Testing Topology

Link Change Tester

```
class Linkchange( Topo ):
    "Tree for Topology change testing."

    def __init__( self ):
        "Tree for testing."

        # Initialize topology
        Topo.__init__( self )

        s2 = self.addSwitch('s2')
        s1 = self.addSwitch('s1')
        s3 = self.addSwitch('s3')

        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )


        self.addLink(s2, s1, cls=TCLink, bw=15, delay='3ms', loss=0, use_htb=True)
        self.addLink(s2, s3, cls=TCLink, bw=5, delay='2ms', loss=0, use_htb=True)
        self.addLink(s3, h3)
        self.addLink(s3, h4)
        self.addLink(s1, h1)
        self.addLink(s1, h2)
        self.addLink
```

A.7.4 S-T Cut Topology

S-T Cut Topology

```
class Stcut( Topo ):
    "Battalion Topology"

    def __init__( self ):
        "Create custom s-t cut topo."

        # Initialize topology
        Topo.__init__( self )

        s3 = self.addSwitch('s3')
        s1 = self.addSwitch('s1')
        s6 = self.addSwitch('s6')
        s2 = self.addSwitch('s2')
        s5 = self.addSwitch('s5')
        s4 = self.addSwitch('s4')

        info( '*** Add hosts\n')
        h2 = self.addHost('h2')
        h5 = self.addHost('h5')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h1 = self.addHost('h1')
        h6 = self.addHost('h6')

        info( '*** Add links\n')
        self.addLink(s4, s3)
        self.addLink(s3, s1)
        self.addLink(s1, s2)
        self.addLink(s2, s3)
        self.addLink(s3, s5)
        self.addLink(s5, s4)
        self.addLink(s4, s1)
        self.addLink(s5, s6)
        self.addLink(s6, s4)
        self.addLink(h1, s2)
        self.addLink(s1, h3)
        self.addLink(h4, s4)
        self.addLink(s3, h2)
        self.addLink(s5, h5)
        self.addLink(s6, h6)
```

A.8 Experiment Automation Shell Scripts

These are simple shell scripts to automate experimentation utilizing multiple Linux command line tools, the ONOS.py file, and the merged ONOS-Mininet command line. This streamlined experimentation and allowed us to rapidly manipulate topologies and output results into text files that we could parse through. Once again, I'd like to thank Terry Kvitchko for helping me in generating and testing this code.

A.8.1 Experiment 1 Automation

Experiment1 Automation

```
#!/bin/sh
# launch with "sudo bash ./mn_shell.sh"

if [ $USER != "root" ]; then
        echo "This script must be run as root. (Try 'sudo bash ./SCRIPT_NAME')"
        exit 1
fi
# go into onos directory
cd ~/onos

counter=1

numRules=$(wc -l /home/brent/inputrules/rules.txt | grep -oP "([0-9]*) " | xargs )
TIME=$(date +%s)

while [ $counter -le 25 ]
do
        mn -c
        #clean up mininet
        echo ""
        echo "link s1 s3 down" >> /tmp/mn_shell_temp
        echo "sh echo 'starting tshark capture (60 secs)'" > /tmp/mn_shell_temp
        echo "sh sudo -u brent tshark -i any -d 'tcp.port==6633,
        openflow' -j 'openflow_v4.type==14' -a duration:60 -w
        ~/capturesresults/${TIME}_${numRules}_run_$
        {counter}_pcap.pcap &" >> /tmp/mn_shell_temp
        echo "sh sudo -u brent tshark -i any -f 'tcp port 6633' -a duration:60 -w ~/capturesresults/${TIME}_${numRules}_run_$
        {counter}_pcap.pcap &" >> /tmp/mn_shell_temp
        echo "sh sudo -u brent tshark -i any -f 'port 6653' -a duration:220 -w ~/capturesresults/${TIME}_${numRules}_run_$
        {counter}_pcap.pcap &" >> /tmp/mn_shell_temp
        echo "onos app activate graph" >> /tmp/mn_shell_temp
        echo "sh sleep 2s" >> /tmp/mn_shell_temp
        echo "onos kill 0 2 50000" >> /tmp/mn_shell_temp
        # 0 & 2 represents the S & T while the last
        number reflects the number of rules in the ACL file its reading from
        HOST_A=3
        HOST_B=1
        SWITCH_A=1
        SWITCH_B=2
        SWITCH_C=3

        echo "sh echo 'starting to sleep'" >> /tmp/mn_shell_temp
        echo "sh sleep 30s" >> /tmp/mn_shell_temp #sleep time adjustable as neccesary
        echo "sh echo 'done sleeping'" >> /tmp/mn_shell_temp

        echo "h${HOST_A} ifconfig > ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IFCONFIG_SERVER_h${HOST_A}test &" >> /tmp/mn_shell_temp
        echo "h${HOST_A} iperf -s -p 5566 -i 2 >> ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IFCONFIG_SERVER_h${HOST_A}test &" >> /tmp/mn_shell_temp

        echo "h${HOST_B} ifconfig > ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IFCONFIG_CLIENT_h${HOST_B}test &" >> /tmp/mn_shell_temp
        echo "h${HOST_B} iperf -c 10.0.0.${HOST_A} -p 5566 -t 15 >> ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IFCONFIG_CLIENT_h${HOST_B}test &" >> /tmp/mn_shell_temp

        echo "s${SWITCH_A} iostat > ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IOSTAT_s${SWITCH_A}iostat &" >> /tmp/mn_shell_temp
        echo "s${SWITCH_B} iostat >> ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IOSTAT_s${SWITCH_A}iostat &" >> /tmp/mn_shell_temp
        echo "s${SWITCH_C} iostat >> ~/capturesresults/${TIME}_${numRules}_run_${counter}
        IOSTAT_s${SWITCH_A}iostat &" >> /tmp/mn_shell_temp

        echo "sh echo 'starting to sleep'" >> /tmp/mn_shell_temp
        echo "sh sleep 30s" >> /tmp/mn_shell_temp
        echo "sh echo 'done sleeping'" >> /tmp/mn_shell_temp

        cp /tmp/mn_shell_temp ~/capturesresults/${TIME}_${numRules}_run_${counter}_mininet_src.txt


        #sudo -u brent tshark -i any -a duration:60 -w ~/captures/capture1.pcap &

        echo " "
        echo " "
        echo " "
        echo " "

        #launch minionos
        cd ~/onos/tools/dev/mininet
        echo "source /tmp/mn_shell_temp" | mn --custom onos.py
        --controller onos,1 --topo brenttree #--link tc,delay=1ms,bw=1000 and change topology as needed

        #cleanup
        rm /tmp/mn_shell_temp
        echo "Output to ~/capturesresults/${TIME}_${numRules}_run_${counter}"
        ((counter++))
done
```

A.8.2 Experiment 2 Automation

Experiment 2 had four tests. We used different variations of the code below to run the baseline test, random outer link addition, random core link loss, and the race condition between the ONOS intent and our DF program. You simply manipulate which links you want up and down when collecting iperf data.

Experiment2 Automation

```
#!/bin/sh
# launch with "sudo bash ./mn_shell.sh"

if [ $USER != "root" ]; then
        echo "This script must be run as root. (Try 'sudo bash ./SCRIPT_NAME')"
        exit 1
fi
# go into onos directory
cd ~/onos
#first while loop counter
counter=1
#time and date for file naming
TIME=$(date +%s)

#variables for intent cURL request build
DATA1="'{ "\"type\"": "\"HostToHostIntent\"",
"\"appId\"": "\"org.onosproject.ovsdb\"", "\"priority\"": 55, "\"one\"": "\"00:00:00:00:00:01/-1\"","\"two\"": "\"00:00:00:00:00:04/-1\"" }'"
DATA2="'{ "\"type\"": "\"HostToHostIntent\"",
"\"appId\"": "\"org.onosproject.ovsdb\"", "\"priority\"": 55, "\"one\"": "\"00:00:00:00:00:02/-1\"","\"two\"": "\"00:00:00:00:00:03/-1\"" }'"
URL="'http://192.168.123.1:8181/onos/v1/intents'"
HEADER="'Content-Type: application/json'"
HEADER2="'Accept: application/json'"
#variables for link command building
corelinks=("s15 s8" "s22,s1" "s15 s22" "s22 s8" "s1 s8" "s1 s15")
outerlinks=("s18 s25" "s16 s23" "s17 s2" "s21 s4" "s24 s9"
"s28 s11" "s3 s10" "s7 s14")
h1h2sidelinks=("s16 s23" "s18 s25" "s17 s2" "s21 s4")
corethree=("s15 s22" "s15 s1" "s15 s8")
LINK="link "
DOWN=" down"
UP=" up"
while [ $counter -le 1 ]
do
        #clean up mininet
        echo " Starting........................................"
        echo " "
        echo " "
        echo " "
        mn -c
        #create legimitate traffic to install flows and intents
        echo "sh sleep 5s" >> /tmp/mn_shell_temp
        echo "pingallfull" >> /tmp/mn_shell_temp
        echo "sh sleep 5s" >> /tmp/mn_shell_temp
        #take down unnecessary outer Links (base line | rand core link cd| rand outer link)
        cntr=0
        while [ $cntr -lt 8 ]
        do
                echo "${LINK}${outerlinks[$cntr]}${DOWN}" >> /tmp/mn_shell_temp #~/capturesresults/file1.txt
                ((cntr++))
        done
        #activate
        echo "onos app activate graph" >> /tmp/mn_shell_temp
        echo "sh sleep 2s" >> /tmp/mn_shell_temp
        echo "onos kill 5 24 1000" >> /tmp/mn_shell_temp
        #take down core links (race condition only)
```

```
echo "sh sleep 8s" >> /tmp/mn_shell_temp
num=0
while [ $num -lt 3 ]
do
        echo "${LINK}${corethree[$num]}${DOWN}" >>
        /tmp/mn_shell_temp #~/capturesresults/file1.txt
        ((num++))
done
#install intents
echo "sh curl -X POST --user onos:rocks --header ${HEADER} --header ${HEADER2} -d ${DATA1} ${URL}" >>  /tmp/mn_shell_temp
echo "sh curl -X POST --user onos:rocks --header ${HEADER} --header ${HEADER2} -d ${DATA2} ${URL}" >>  /tmp/mn_shell_temp
# start iperf tests H1 -> H6
echo "sh sleep 5s" >> /tmp/mn_shell_temp

echo "sh echo "$(date +%T.%N)" >  ~/capturesresults/experiment2/${TIME}_h6iperfserver_run$
{counter} &" >> /tmp/mn_shell_temp
echo "h6 ifconfig >> ~/capturesresults/experiment2/$
{TIME}_h6iperfserver_run${counter} &" >> /tmp/mn_shell_temp
echo "h6 iperf -s -u -i 1 >> ~/capturesresults/experiment2/${TIME}_h6iperfserver_run$
{counter} &" >> /tmp/mn_shell_temp
echo "sh echo "$(date +%T.%N)" >  ~/capturesresults/experiment2/${TIME}_h1iperfclient_run${
counter} &" >> /tmp/mn_shell_temp
echo "h1 ifconfig >> ~/capturesresults/experiment2/$
{TIME}_h1iperfclient_run${counter} &" >> /tmp/mn_shell_temp
echo "h1 iperf -c 10.0.0.4 -u -b 4m -i 1 -t 30 >> ~/capturesresults/experiment2/${TIME}_h1iperfclient_run$
{counter} &" >> /tmp/mn_shell_temp
#start iperf tests H2 -> H5
echo "sh echo "$(date +%T.%N)" >  ~/capturesresults/experiment2/${TIME}_h5iperfserver_run$
{counter} &" >> /tmp/mn_shell_temp
echo "h5 ifconfig >> ~/capturesresults/experiment2/$
{TIME}_h5iperfserver_run${counter} &" >> /tmp/mn_shell_temp
echo "h5 iperf -s -u -i 1 >> ~/capturesresults/experiment2/${TIME}_h5iperfserver_run$
{counter} &" >> /tmp/mn_shell_temp
echo "sh echo "$(date +%T.%N)" >  ~/capturesresults/experiment2/${TIME}_h2iperfclient_run$
{counter} &" >> /tmp/mn_shell_temp
echo "h2 ifconfig >> ~/capturesresults/experiment2/$
{TIME}_h2iperfclient_run${counter} &" >> /tmp/mn_shell_temp
echo "h2 iperf -c 10.0.0.3 -u -b 4m -i 1 -t 30 >> ~/capturesresults/experiment2/${TIME}_h2iperfclient_run$
{counter} &" >> /tmp/mn_shell_temp
# pause
echo "sh sleep 10s" >> /tmp/mn_shell_temp
#link down
#echo "${LINK}${corelinks[(($RANDOM%6))]}${DOWN}" >> /tmp/mn_shell_temp
#random side link up
echo "${LINK}${h1h2sidelinks[(($RANDOM%4))]}${UP}" >> /tmp/mn_shell_temp
echo "sh echo "$(date +%T.%N)" >>  ~/capturesresults/experiment2/${TIME}_h6iperfserver_run$
{counter} &" >> /tmp/mn_shell_temp
echo "sh echo "$(date +%T.%N)" >>  ~/capturesresults/experiment2/${TIME}_h5iperfserver_run$
{counter} &" >> /tmp/mn_shell_temp
echo "sh echo "$(date +%T.%N)" >>  ~/capturesresults/experiment2/${TIME}_h2iperfclient_run$
{counter} &" >> /tmp/mn_shell_temp
echo "sh echo "$(date +%T.%N)" >>  ~/capturesresults/experiment2/${TIME}_h1iperfclient_run$
{counter} &" >> /tmp/mn_shell_temp


#outer links down
#echo "${LINK}${outerlinks[(($RANDOM%8))]}${DOWN}" >> /tmp/mn_shell_temp #~/capturesresults/file1.txt

echo "sh sleep 50s" >> /tmp/mn_shell_temp

cd ~/onos/tools/dev/mininet
echo "source /tmp/mn_shell_temp" | mn --custom onos.py --controller onos,1 --topo regiment --mac
echo " "
echo " "
echo " "
echo "Ending..................................................  "
cp /tmp/mn_shell_temp ~/capturesresults/experiment2/${TIME}_inputs_run${counter}.txt
rm /tmp/mn_shell_temp
```

```
        #echo "Output to ~/capturesresults/experiment2/${TIME}_${bandwidth}mbs_run_${counter}"
        ((counter++))
done

        #echo "${LINK}${outerlinks[(($RANDOM%8))]}${DOWN}" >> /tmp/mn_shell_temp #~/capturesresults/file1.txt
```

A.9 Experiment Analysis Code

Below is the code that was used during Experiment 1 to analyze rule installation time, iperf, and iostat output files. The following generates a csv from which you can use for analysis or graph creation. Terry Kvitchoko created this code from ground up to analyze xml.

Analysis.py for Experiment 1

```
# NOTE: make sure to run this from inside the relevant folder of captures

# ################################################################

import glob
import re
import sys
from collections import defaultdict
import os

tcp_port = 6633        # change if openflow port update files!

def iostat_scrape():
        files=[]
        for name in glob.glob('*iostat'):
            files.append(name)
        if not files:
            print "no iostat files found!"
            return []

        results = defaultdict(list)
        cpuRE = re.compile(" ([0123456789\.]+) ")
        numRulesRE = re.compile("([0123456789]+)_run")

        numIostats = len(files)
        i = 1
        for file in files:

            try:

                printStr = "iostat: Working with file " +
                str(i) + " of " + str(numIostats) + "\r"
                print printStr,
                sys.stdout.flush()
                i+=1

                with open(file, 'rb') as iofile:
                    line = ""
                    while "avg-cpu" not in line:
                            line = next(iofile)
                    line = next(iofile)    # actual line with data
                    m = cpuRE.search(line)  # get cpu num
                    n = numRulesRE.search(file)  # get number of rules
                    results[n.group(1)].append(m.group(1))
                    #results[num rules] += list entry for this cpu num

            except Exception as e:
                print str(e) + " error in file " + file + ", skipping"
                continue

        return results

def bandwidth_scrape():
        files=[]
        for name in glob.glob('*CLIENT*'):
                files.append(name)
        if not files:
                print "no ifconfig_client files found!"
                return []

        results = defaultdict(list)
        bandwidthRE = re.compile("([0123456789\.]+) Gbits")
        numRulesRE = re.compile("([0123456789]+)_run")

        numIostats = len(files)
        i = 1
        for file in files:

                try:

                        printStr = "bandwidth: Working with file " + str(i)
                        + " of " + str(numIostats) + "\r"
                        print printStr,
                        sys.stdout.flush()
                        i+=1

                        with open(file, 'rb') as iofile:
                            line = ""
                            while "Gbits" not in line:
                                    line = next(iofile)
                            m = bandwidthRE.search(line)  # get cpu num
                            n = numRulesRE.search(file)  # get number of rules
                            results[n.group(1)].append(m.group(1))
                            #results[num rules] += list entry for this cpu num

                except Exception as e:
                        print str(e) + " error in file " + file + ", skipping"
                        continue

        return results

def pcap_helper():

        requestTimes = {}
        responseTimes = {}
        delay = []
        linecount = 0
        packetcount = 0

        firstRequestTime = 0
        lastResponseTime = 0

        timestampRE = re.compile("value=\"([0123456789\.]*)\"")
        transactionRE = re.compile("Transaction ID: ([0-9]*)\"")
        ofptTypeRE = re.compile("Type: (.*?)\"")

        with open('temp.xml', 'rb') as xmlfile:
            line = ""
            try:
                while ( 1 ):
                    line = next(xmlfile)
                    linecount+=1
                    print str(linecount) + "\r",

                    if "<packet>" in line:       # keep going until new packet starts
```

```
packetcount+=1

timestamp = 0       # reset these values just in case
transactionID = 0
ofptType = 0

for i in range(0, 5):
# grab packet's timestamp (will discard later if not openflow)
    line = next(xmlfile)
    linecount+=1
timestamp = line


for i in range(0, 100):
    line = next(xmlfile)
    linecount+=1
for i in range(0, 20):
    line = next(xmlfile)
    if "openflow" in line or "</packet>" in line:
            break
    linecount+=1
if "openflow" not in line:
# skip if it's not an openflow
    continue

while "openflow_v4.type" not in line:
# grab ofpt type and transaction lines
    line = next(xmlfile)
    linecount+=1
ofptType = line
line = next(xmlfile)
line = next(xmlfile)
linecount+=2
transactionID = line

if "BARRIER_REPLY" not in ofptType and
"OFPT_FLOW_MOD" not in ofptType:
# it's ofpt, but not the right ofpt. go on to next packet
    continue

m = timestampRE.search(timestamp)        # extract fields
if m:
    timestamp = m.group(1)
else:
    print("Failed regex in timestamp line: " + str(timestamp)
    + ", relevant transaction ID line #: " + str(linecount))
    continue

if firstRequestTime == 0:
    firstRequestTime = timestamp

m = transactionRE.search(transactionID)
if m:
    transaction_test = transactionID
    transactionID = m.group(1)
else:
    print("Failed regex in transaction ID line: "
    + str(transactionID) + ", relevant transaction ID line #: " + str(linecount))
    continue

m = ofptTypeRE.search(ofptType)
if m:
        ofptType = m.group(1)
else:
    print("Failed regex in OFPT_TYPE line: " + str(ofptType)
    + ", relevant transaction ID line #: " + str(linecount))
    continue
```

```
                    if "REPLY" in ofptType:  # is it a request or response?
                        responseTimes[transactionID] = timestamp
                        lastResponseTime = timestamp
                    elif "OFPT_FLOW_MOD" in ofptType:
                        if transactionID not in requestTimes.keys():
                            requestTimes[transactionID] = timestamp
                    else:
                        print("Unexpected OFPT_TYPE: " + str(ofptType) +
                        ", relevant transaction ID line #: " + str(linecount))
                        continue

            except StopIteration as e:
                    pass

                for key in requestTimes.keys():
                        try:
                            delayVal = float(responseTimes[key]) − float(requestTimes[key])
                        except KeyError as e:
                            if str(key)!="0": # some kind of openflow error creates multiple 0s, ignore
                                    print("Failed to find response for transaction ID " + str(key))
                            continue
                        delay.append('%.9f'%delayVal)

                avgDelay = 0
                amtKeys = 0
                for value in delay:
                    amtKeys+=1
                    avgDelay+=float(value)
                avgDelay = avgDelay/float(amtKeys)

                return [avgDelay, (float(lastResponseTime) − float(firstRequestTime)) ]

def pcap_scrape():
        files=[]
        for name in glob.glob('*.pcap'):
            files.append(name)
        if not files:
            print "no pcap files found!"
            return []

        print "Working with " + str(len(files)) + " total files."

        results = defaultdict(list)
        numRulesRE = re.compile("([0123456789]+)_run")

        for file in files:
            try:

                #tsharkString = 'tshark −r ' + str(file) + ' −d "tcp.port==' + str(tcp_port) +
                ',openflow" −T pdml −V > temp.xml'
                tsharkString = 'tshark −r ' + str(file) + ' −d "tcp.port==6633,openflow"
                −d "tcp.port==6653,openflow" −T pdml −V > temp.xml'
                print "pcap: " + tsharkString + "\r",
                sys.stdout.flush()

                os.system(tsharkString)

                print "pcap: Working on xml file for " + str(file) + "
\n",

                single_file_result = pcap_helper()

                print "pcap: Done working on xml file for " + str(file) + ", found avg delay "
                + str(single_file_result[0]) + " and overall time " + str(single_file_result[1]) + "."

                n = numRulesRE.search(file)  # get number of rules
                results[n.group(1)].append(single_file_result)  #results[num rules] += list entry for this cpu num
            except Exception as e:
                print str(e) + " error in file " + file + ", skipping"
                continue

        return results

if __name__ == "__main__":

        print ""

        print "Running iostat data scrape..."
        iostats = iostat_scrape()
        print "Done.                                        "
        print "===RESULTS=========\n(num rules: list of cpu times)"
        print iostats

        print ""

        print "Running bandwidth data scrape..."
        bandwidths = bandwidth_scrape()
        print "Done.                                        "
        print "===RESULTS====\n(num rules: list of bandwidths found in clients)"
        print bandwidths

        print ""

        print "Running pcap scrape..."
        ruletimes = pcap_scrape()
        print "Done.                                        "
        print "===RESULTS========\n(num rules: tuple of average delay and overall delay)"
        print ruletimes

        print ""

        with open('output_iostat.csv', 'w') as f:
            for key in iostats.keys():
                line = str(key)
                    for entry in iostats[key]:
                        line += "," + entry
                    f.write(line + "\n")

        with open('output_bandwidth.csv', 'w') as f2:
            for key in bandwidths.keys():
                line = str(key)
                    for entry in bandwidths[key]:
                        ine += "," + entry
                    f2.write(line + "\n")

        with open('output_avgdelay.csv', 'w') as f3:
            with open('output_overalldelay.csv', 'w') as f4:
                    for key in ruletimes.keys():
                            line1 = str(key)
                            line2 = str(key)
                            for entry in ruletimes[key]:
                                line1 += "," + str(entry[0])
                                line2 += "," + str(entry[1])
                            f3.write(line1 + "\n")
                            f4.write(line2 + "\n")
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] J. M. Richardson, "Review of the FY2019 Budget Request for the U.S. Navy & Marine Corps," April 24, 2018. [Online]. Available: https://www.appropriations.senate.gov/download/042418_-richardson-testimony

[2] "War," *Merriam-Webster*. Accessed May 14, 2019. [Online]. Available: https://www.merriam-webster.com/dictionary/war

[3] USMC. *Command and Staff College AY18 8906 Coursebook: Lesson 6: Module 1: Global Security Challenges*. Marine Corps University, Command and Staff College Distance Education Program, 2017.

[4] N. Freier, C. Compton, and T. Magsig, "Gray Zone: Why we're losing the new era of national security," Defense One. June 9, 2016. [Online]. Available: https://www.defenseone.com/ideas/2016/06/gray-zone-losing-new-era-national-security-strategy/128957/

[5] R. B. Neller, "Review of the FY2019 Budget Request for the U.S. Navy & Marine Corps," April 24, 2018. [Online]. Available: https://www.appropriations.senate.gov/download/042418_-neller-testimony

[6] R. C. Molander, A. Riddle, P. A. Wilson, and S. Williamson, "Strategic information warfare: A new face of war," RAND Corp., Santa Monica, CA, USA, MR-661, 1996. [Online]. Available: https://www.rand.org/pubs/monograph_reports/MR661/index2.html

[7] J. Turrito, "Understanding warfare in the 21st century," *International Affairs Review*, vol. 18, no. 3, Winter 2010. [Online]. Available: http://www.iar-gwu.org/node/145

[8] W. Matthews, "Innovation at work: Can DoD get tech and acquisition in sync?" GovTechWorks, November 30, 2016. [Online]. Available: https://www.govtechworks.com/innovation-at-work-can-dod-get-tech-and-acquisition-in-sync/

[9] R. Kalinyak, "CNO: It's 'Imperative' the Navy speeds up to keep up," C4ISRNET, June 15, 2017. [Online]. Available: https://www.c4isrnet.com/it-networks/2017/06/15/cno-it-s-imperative-the-navy-speeds-up-to-keep-up/

[10] J. M. Richardson, "A design for maintaining maritime superiority version 2.0," December 2018. [Online]. Available: https://www.navy.mil/navydata/people/cno/Richardson/Resource/Design_2.0.pdf

[11] USMC. *Command and Staff College AY18 8906 Coursebook: Lesson 3: Module 2: Marine Operating Concept: Public Affairs Playbook (2017)*. Marine Corps University, Command and Staff College Distance Education Program, 2017. [Online][Published for USMC Command and Staff Students].

[12] *Joint Operations*, JP 3-0, Joint Chiefs of Staff, Washington, DC, 2017. [Online]. Available: https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/jp3_0ch1.pdf?ver=2018-11-27-160457-910

[13] USMC, *MAGTF Communications System*, MCWP 3-40.3, Marine Corps Logistics Base, Albany, GA, 2011, pCN:143 000042 00.

[14] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6994333

[15] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: a survey," *IEEE Access*, vol. 5, pp. 25,487–25,526, 2017. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8066287

[16] M. Knight, "Data management and the internet of things," DATAVERSITY, December 12, 2018. [Online]. Available: https://www.dataversity.net/data-management-internet-things/#

[17] L. Columbus, "10 charts that will challenge your perspective of IoT's growth," Forbes, June 6, 2018. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2018/06/06/10-charts-that-will-challenge-your-perspective-of-iots-growth/#5d24a2b23ecc

[18] D. Edwards, "Commentary: exponential growth of IoT becoming the next tech revolution," Robotics & Automation News, July 17, 2018. [Online]. Available: https://roboticsandautomationnews.com/2018/07/17/commentary-exponential-growth-of-iot-becoming-the-next-tech-revolution/18338/

[19] B. Heller, "Reproducible network research with high-fidelity emulation," Ph.D. dissertation, Stanford University, Palo Alto, CA USA, 2013. [Online]. Available: https://stacks.stanford.edu/file/druid:zk853sv3422/heller_thesis-augmented.pdf

[20] D. Yu, A. W. Moore, C. Hall, and R. Anderson, "Security: a killer app for sdn?" Indiana University at Bloomington, Bloomington, IN, Tech. Rep. AFRL-RI-RS-TP-2014-048, October 2014. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a613601.pdf

[21] J. R. Agre, K. D. Gordon, and M. S. Vassiliou, "Commercial Technology at the Tactical Edge," presented at the 18th International Command and Control Research Technology Symposium, Alexandria, VA, June 2013. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a587552.pdf

[22] N. Cvijetic, A. Tanaka, K. Kanonakis, and T. Wang, "Sdn-controlled topology-reconfigurable optical mobile fronthaul architecture for bidirectional comp and low latency inter-cell d2d in the 5g mobile era," *Optics Express*, vol. 22, no. 17, pp. 20,809–20,815, 2014. [Online]. Available: https://www.osapublishing.org/oe/fulltext.cfm?uri=oe-22-17-20809id=299642.

[23] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: Sdn for big data and big data for sdn," *IEEE network*, vol. 30, no. 1, pp. 58–65, 2016. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7389832

[24] U.S.Army, "Shaping the Army Network: 2025-2040," U.S. Army, Office of the Chief Information Officer, Mar 2016. Available: https://www.hsdl.org/?view&did=791846

[25] U.S.Navy, "A Cooperative Strategy for 21st Century Seapower," Mar 2015. [Online]. Available: https://www.navy.mil/local/maritime/150227-CS21R-Final.pdf

[26] "Expeditionary Advanced Base Operations," *U.S. Marine Corps Concepts Programs*, accessed March 5, 2019. [Online]. Available: https://www.candp.marines.mil/Concepts/Subordinate-Operating-Concepts/Expeditionary-Advanced-Base-Operations/

[27] Y.-W. E. Sung, X. Sun, S. G. Rao, G. G. Xie, and D. A. Maltz, "Towards systematic design of enterprise networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 3, pp. 695–708, 2011. [Online]. Available: delivery.acm.org/10.1145/2050000/2042979/p695-sung.pdf?ip=205.155.65.226&id=2042979&acc=ACTIVE%20SERVICE&key=B318D1722F7F4203%2E44DF46464A4B769E%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1557884650_a713f8a48b4ff85ce43ed428c779a76b

[28] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks," in *USENIX Annual Technical Conference*. USENIX Association, 2014, pp. 333–345. [Online]. Available: https://www.usenix.org/system/files/conference/atc14/atc14-paper-levin.pdf

[29] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, p. 20, 2013, [Online]. Available: http://delivery.acm.org/10.1145/

2610000/2602219/p87-feamster.pdf?ip=205.155.65.226&id=2602219&acc=ACTIVE%20SERVICE&key=B318D1722F7F4203%2E44DF46464A4B769E%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1557884977_d056b3dcba2fae4257372e150266719f

[30] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, 1997. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=568214

[31] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu, "Attacking the brain: Races in the sdn control plane," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017, pp. 451–468. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-xu-lei.pdf

[32] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008. [Online]. Available: http://ccr.sigcomm.org/online/files/p69-v38n2n-mckeown.pdf

[33] J. P. Weitzel, "Innovating tactical networks: A software defined network approach," M.S. Thesis, Dept. Computer Science, Naval Postgraduate School, Monterey, CA, 2018. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/1060105.pdf

[34] P. Goransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*. Waltham, MA: Morgan Kaufmann, 2016.

[35] K. Ingham and S. Forrest, "A history and survey of network firewalls," *University of New Mexico*, 2002. [Online]. Available: https://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf

[36] J. F. Kurose and K. Ross, *Computer networking: A top-down approach, 7th Edition*. London, United Kingdom: Pearson Education Inc., 2017.

[37] S. M. Bellovin, "Distributed firewalls," USENIX, 1999. [Online]. Available: http://static.usenix.org/publications/login/1999-11/features/firewalls.html

[38] "Network simulation and emulation," class notes for Network Modeling an Simulation, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, Winter 2019.

[39] Mininet, "Mininet: An instant virtual network on your laptop (or other pc)," accessed on January 7, 2019. [Online]. Available: http://mininet.org/

[40] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, “Using mininet for emulation and prototyping software-defined networks,” in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6860404

[41] “Production quality multilayer open virtual switch,” accessed on January 7, 2019. [Online]. Available: https://www.openvswitch.org/

[42] Title 14 U.S. Code: Coast Guard. Aug 14, 1949. *Code of Federal Regulations*. [Online]. Available: https://www.govinfo.gov/content/pkg/USCODE-2010-title14/pdf/USCODE-2010-title14.pdf

[43] Title 10 U.S. Code: Armed Forces. January 7, 2011. *Code of Federal Regulations*. [Online]. Available: https://www.govinfo.gov/content/pkg/CPRT-112HPRT67342/pdf/CPRT-112HPRT67342.pdf

[44] USMC, “United states marine corps concepts and programs 2013,” 2013. [Online]. Available: https://www.marines.mil/Portals/59/Publications/U.S.%20Marine%20Corps%20Concepts%20and%20Programs%202013_1.pdf

[45] USMC, *MCDP 1-0 (w/change 1): Marine Corps Operations*, 26 July, 2017. [Online]. Available: https://www.marines.mil/Portals/59/Publications/MCDP%201-0%20W%20CH%201.pdf?ver=2017-09-25-150919-793

[46] USMC, *NAVMC 1200.1D Military Occupational Specialty Manual*, 2018. Accessed on February 18, 2019. [Online]. Available: https://www.trngcmd.marines.mil/Portals/207/Docs/wtbn/MCCMOS/FY19%20MOS%20Manual%20NAVMC_1200.1D.PDF?ver=2018-05-16-070623-087

[47] “Basic Communications Course BCC,” Marines.mil. Accessed on January 17, 2019. [Online]. Available: https://www.trngcmd.marines.mil/Units/West/MCCES/MCCES-Schools/

[48] USMC, *Marine Corps Strategy for Assured Command and Control: Enabling C2 for Tomorrow’s Marine Corps, Today*, March 2017. [Online]. Available: https://www.hqmc.marines.mil/Portals/61/Marine_Corps_Strategy_for_Assured_Command_and_Control_March_2017.pdf?ver=2017-05-30-160731-940

[49] M. Schwartz, “Defense acquisitions: How DoD acquires weapon systems and recent efforts to reform the process,” Washington DC: CRS Report No. RL34026, Tech. Rep., 2014. [Online]. Available: https://fas.org/sgp/crs/natsec/RL34026.pdf

[50] Defense Acquisition Reform is a National Security Issue. *Grayline Group*. Accessed March 22, 2019. [Online]. Available: https://graylinegroup.com/defense-acquisition-reform-is-a-national-security-issue/

[51] J. V. Saturno, B. H. Jr., and M. S. Lynch, "The congressional appropriations process: An introduction," Washington DC: CRS Report No. R42388, Tech. Rep., November 30, 2016. [Online]. Available: https://fas.org/sgp/crs/misc/R42388.pdf

[52] V. H. Krulak, *First to fight: an inside view of the US Marine Corps*. Annapolis, MD: Naval Institute Press, 2013.

[53] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Mininet performance fidelity benchmarks," October 21, 2012. [Online]. Available: https://hci.stanford.edu/cstr/reports/2012-02.pdf

[54] B. Lantz and B. O'Connor, "A mininet-based virtual testbed for distributed sdn development," in *ACM SIGCOMM Computer Communication Review*. ACM, vol. 45, no.4, 2015, pp. 365–366. [Online]. Available: https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p365.pdf

[55] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19. [Online]. Available: conferences.sigcomm.org/hotnets/2010/papers/a19-lantz.pdf

[56] "Open networking operating system," ONOS. Accessed March 15, 2019. [Online]. Available: https://onosproject.org/

[57] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms second edition*. Cambridge, MA: The MIT Press, 2001.

[58] "Networks II," March 10, 2015. [Online]. Available: https://blogs.cornell.edu/info4220/2015/03/10/the-origin-of-the-study-of-network-flow/

[59] "Find minimum s-t cut in a flow network," Geeks for Geeks. May 2018. [Online]. Available: https://www.geeksforgeeks.org/minimum-cut-in-a-directed-graph/

[60] J. N. Davies, P. Comerford, and V. Grout, "Principles of eliminating access control lists within a domain," *Future Internet*, vol. 4, no. 2, pp. 413–429, 2012. [Online]. Available: https://pdfs.semanticscholar.org/b895/9a8084d485dac508fb237c06305e7666edbc.pdf?_ga=2.184932841.1390330342.1557893448-1634724069.1557893448

[61] A.-H. Esfahanian, “Connectivity algorithms.” Cambridge, England: Cambridge University Press, 2013. [Online]. Available: https://www.cse.msu.edu/~cse835/Papers/Graph_connectivity_revised.pdf

[62] “Minimum s-t node cut,” Network X. accessed November 21, 2018. [Online]. Available: https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.algorithms.connectivity.cuts.minimum_st_node_cut.html

[63] “Dijsktra’s algorithm,” Geeks for Geeks. Accessed November, 15, 2018. [Online]. Available: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

[64] USMC, “MCRP 1-10.1 Organization of Marine Corps Forces,” 1998. [Online]. Available: https://www.marines.mil/Portals/59/Publications/MCRP%201-10.1%20(formerly%20MCRP%205-12D).pdf?ver=2016-05-09-113114-630

[65] "SDN Technical Specifications". *Open Networking Foundation*. [Online]. Available: https://www.opennetworking.org/software-defined-standards/specifications/. *Open Networking Foundation.*

[66] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, “Software-defined networking (sdn): Layers and architecture terminology,” Tech. Rep., January 2015. [Online]. Available: https://tools.ietf.org/html/rfc7426

[67] M. Boucadair and C. Jacquenet, “Software-defined networking: A perspective from within a service provider environment: RFC7149 ISSN:2070-1721,” Tech. Rep., March 2014. [Online]. Available: https://tools.ietf.org/html/rfc7149

[68] zim7563zim7563 334 and M. 4, “The default link bandwidth in mininet,” accessed February 15, 2019. [Online]. Available: https://stackoverflow.com/questions/45907540/the-default-link-bandwidth-in-mininet

[69] A. Pasternack, “War is memes. "don’t be a victim like the americans",” Fast Company. Oct 25, 2018. [Online]. Available: https://www.fastcompany.com/90253809/war-is-memes-pw-singer-likewar

[70] “ONOS.py troubleshooting guide,” accessed February 12, 2019. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Troubleshootingonos.py

[71] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, “Multi-controller based software-defined networking: A survey,” *IEEE Access*, vol. 6, pp. 15,980–15,996, 2018. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8314783

[72] “ONOS wiki,” accessed January 8, 2019. [Online]. Available: https://wiki.onosproject.org/

[73] “Dijkstra’s algorithm,” Feb 2019. Available: https://en.wikipedia.org/wiki/Dijkstra’s_algorithm

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California