NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

INNOVATING TACTICAL NETWORKS:
A SOFTWARE-DEFINED NETWORK APPROACH

by

John P. Weitzel

June 2018

Thesis Advisor: Geoffrey G. Xie
Second Reader: Dennis M. Volpano

THIS PAGE INTENTIONALLY LEFT BLANK

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

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

4. TITLE AND SUBTITLE INNOVATING TACTICAL NETWORKS: A SOFTWARE-DEFINED NETWORK APPROACH	**5. FUNDING NUMBERS**
6. AUTHOR(S) John P. Weitzel	
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)

The current traditional network paradigm used in the Marine Corps has a demonstrated pattern of success in decades of austere environment operations. However, the adoption of a new idea to reduce time, effort, and manpower related to the installation, operation, and maintenance of tactical networks that simultaneously improves current network capability and capacity is attractive. That new idea is software-defined networks. Three primary keys will encourage the Marine Corps to embrace this new network technology: it must work with current network inventory, it must be capable of maintaining current configuration standards, and it must have the potential to greatly improve upon the current network model. On a physical test bed, this work seeks to evaluate all three requirements. Working topologies that consist of both software-defined and traditional network elements seamlessly connect to create a hybrid solution with near-term relevance. Then, the effective inclusion of virtual local area networks and access control lists on software-defined switches validate the ability to maintain current configuration standards. Lastly, instances of scripted configuration, resident topology constructs, and local central configuration tools show the automation potential of software-defined networks.

14. SUBJECT TERMS software-defined network, tactical network, innovation, Marine Corps, network, automation	**15. NUMBER OF PAGES** 105
	16. PRICE CODE

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

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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

INNOVATING TACTICAL NETWORKS:
A SOFTWARE-DEFINED NETWORK APPROACH

John P. Weitzel
Major, United States Marine Corps
BA, Martin Luther College, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2018

Approved by: Geoffrey G. Xie
Advisor

Dennis M. Volpano
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The current traditional network paradigm used in the Marine Corps has a demonstrated pattern of success in decades of austere environment operations. However, the adoption of a new idea to reduce time, effort, and manpower related to the installation, operation, and maintenance of tactical networks that simultaneously improves current network capability and capacity is attractive. That new idea is software-defined networks. Three primary keys will encourage the Marine Corps to embrace this new network technology: it must work with current network inventory, it must be capable of maintaining current configuration standards, and it must have the potential to greatly improve upon the current network model. On a physical test bed, this work seeks to evaluate all three requirements. Working topologies that consist of both software-defined and traditional network elements seamlessly connect to create a hybrid solution with near-term relevance. Then, the effective inclusion of virtual local area networks and access control lists on software-defined switches validate the ability to maintain current configuration standards. Lastly, instances of scripted configuration, resident topology constructs, and local central configuration tools show the automation potential of software-defined networks.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

DoD	Department of Defense
MASINT	measurement and signature intelligence
NPS	Naval Postgraduate School
SRWBR	short range wide band radio
TCP	Transmission Control Protocol
USN	U.S. Navy
UDP	User Datagram Protocol
USG	United States government
USMC	United States Marine Corps
RND	Rapid Network Design
SDN	Software-Defined Network
MOS	Military Occupational Specialty
C2	Command and Control
CUCM	Cisco Unified Call Manager
COC	Combat Operations Center
OVS	Open vSwitch
NIC	network interface card
VLAN	virtual local area network
ID	identifier

IP	internet protocol
ICMP	internet control message protocol
EIGRP	Enhanced Interior Gateway Routing Protocol
OF	OpenFlow
DOD	Department of Defense
SSH	Secure Shell
Telnet	Terminal Network
PIOM	plan, install, operate, and maintain
IOS	Internetwork Operating System
LTS	long term support
DARPA	Defense Advanced Research Projects Agency
LTS	long term support
telnetd	Defense Advanced Research Projects Agency (DARPA) telnet protocol server
OS	operating system
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
RFC	Request for Comments
IRTF	Internet Research Task Force
IETF	Internet Engineering Task Force
VM	virtual machine
MCDP	Marine Corps Doctrinal Publication

MCWP	Marine Corps Warfighting Publication
HP	Hewlett-Packard
REST	representational state transfer
API	application program interface
ACL	access control list
USMC	United States Marine Corps
GUI	graphical user interface
USB	universal serial bus
ANW2	adaptive networking wideband waveform
ONOS	Open Network Operating System
VSAT	very small aperture terminal
NFV	Network Function Virtualization
ARP	Address Resolution Protocol
AIC	Availability, Integrity, Confidentiality
MAC	media access control
SPMAGTF	Special Purpose Marine Air-Ground Task Force
SYSCON	systems control
MEF	Marine Expeditionary Force
HTTP	Hypertext Transfer Protocol
COTS	commercial-off-the-shelf
DISN	Defense Information Systems Network
STRAPEX	bootstrap exercise

COMMEX	communications exercise
CLI	command line interface
ATM	asynchronous transfer mode
ONF	Open Networking Foundation
QoS	quality of service
ISL	Inter-Switch Link
ONOS	Open Network Operating System
OSI	Open System Interconnection
UI	User Interface

Acknowledgments

I must express my deep gratitude to those people who were instrumental in the completion of this work.

My wife, Hazel, always stood alongside me with her full support and encouragement. She never wavered in her belief that I had the ability to succeed and the endurance to finish. Her faith in me was a continuous source of motivation.

My father patiently waited for me to complete this race and constantly cheered for me as I continued toward my goal. I will miss him. My mother has always shown tireless confidence in me. I am a fortunate son.

My advisor, Dr. Geoffrey Xie, was an excellent mentor. I was privileged to have such an engaged and accommodating professor to guide me along the way. He was awesome. My second reader, Dr. Dennis Volpano, provided valuable, constructive feedback. I am indebted to him as well.

Lastly, I take full responsibility for any inaccuracies included in this work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Beginning

The Marine Corps is the best at what it does: fighting and winning battles for its nation. This excellence is built upon a foundation of fundamental warfighting principles, which includes the warfighting function of Command and Control (C2). The Marine Commander, who is often the first to fight in an austere and dangerous environment, has a critical need for a communications infrastructure which will enable the commanding of forces while receiving essential real-time feedback from which to assess and subsequently task further battlefield execution. As the battlefield develops and the mission progresses, this C2 infrastructure reveals additional complementary capability to the commander as the framework that supports all the other functions in warfighting. In short, the Marine commander can achieve a decisive battlefield advantage with a relevant, capable C2 architecture.

The communications officer is the primary staff member responsible to the Commander for the realization of such a C2 architecture. To effectively prepare, install, operate, displace, and maintain such an architecture which typically includes varying degrees of data systems, telecommunications systems, radio systems, and computer systems, the communications officer measures the capabilities and limitations of resources available and properly aligns them with mission requirements [1], [2]. The orchestration of such an endeavor takes acumen in planning, a robust collection of communications assets, and experienced personnel to configure and operate the equipment.

1.1 Problem Statement

The Marine Corps is an exceptional organization and must continue to seek improvement and propel itself forward to continue to serve with excellence at the tactical edge. While the nature of war remains constant, the technology used within warfare continuously evolves [3]. As such, there is always an appetite in the service to do more and do better. While the Marine Corps continues to demonstrate success in every mission throughout the world, verification of time spent and resources used are often subjects for much discussion. Therefore, innovation is of large interest across the leadership throughout the Corps as the adoption of an idea that can potentially reduce cost in terms of time, effort, or manpower while simultaneously

improving capability and capacity would be well-received. In the communications network space, such an idea may lie within Software-Defined Network (SDN)s.

1.2 Research Questions

After understanding the research previously conducted in the area of SDNs, and armed with an understanding of the challenges associated with the implementation of tactical Marine Corps networks in an austere environment, this research will conduct an empirical analysis of SDN as a potential innovation solution for tactical Marine Corps networks. The research questions that shaped the focus of this work are as follows:

- What is the ability of SDN to operate in a hybrid network environment to include current traditional network routers and switches with an SDN controller and SDN capable switches?
- Does SDN have the ability to maintain current network capability and usability? How will virtual local area network (VLAN)s and access control list (ACL)s, two challenging yet essential network constructs in a tactical network, interoperate in hybrid network?
- What is the potential of SDN to innovate tactical networks? Are there automation solutions to simplify network configuration and management? What is the current programmability potential and current functionality of network management that resides within SDN controllers?

1.3 Organization

The rest of the thesis is organized as follows:

Chapter 2 provides the context surrounding the current tactical network paradigm as well as a look into SDNs. The fundamentals of network planning and the subsequent phases of installation, operation, and maintenance are reviewed. Afterward, an exploration into SDN provides understanding within a historical context, while differentiating SDN from a traditional network. The benefits of SDN are examined as well as related work. Chapter 3 provides a detailed review of the methodology to validate the applicability of SDN as a network solution for the Marine Corps. A hybrid network environment is constructed using a real SDN test bed. It integrates VLANs and ACLs to show how these can coexist in a real

SDN environment. Chapter 4 records the results of the experiments performed. Chapter 5 provides a summation of results, discusses limitations of SDN in tactical networks, and provides recommendations for future work to enhance the framework provided with this work and further elaborate on the relevance of SDN to innovate Marine Corps tactical networks.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

In Chapter 1, a forward look into the Marine Corps tactical network paradigm led to the hypothesis of SDN as a potential solution. This work assumes little to no thorough knowledge of either tactical Marine Corps networking or the computer science sub-discipline of networking. Therefore, both domains will receive some attention in key focus areas aimed toward enriching the reader understanding of the recommended innovation approach. At the conclusion of this chapter, this richer understanding of both the tactical network space as well as the potential of SDN within that space will provide the necessary framework to recognize the importance of the subsequent experimentation.

To provide this foundation, the first section will dive into the domain of tactical Marine Corps networks. The next section will then delve into the computer science sub-discipline of networking as it relates to SDN. Then, the last major section of this chapter will consider previous work related to this research.

2.1 Marine Corps Tactical Networks

A Marine Corps tactical network has the single primary purpose of enabling the operational function of C2 for the commander. In the Marine Corps, C2 has a different meaning from the traditional view; *command* is the exercise of authority from the commander and *control* is feedback concerning the effects of the resulting actions back to the commander [4]. Marines in the communications field seek to prepare, install, operate, and maintain tactical networks to achieve this goal. While much is written on the subject of United States Marine Corps (USMC) communications, the basic characteristics of every tactical network implementation must consider reliability, security, timeliness, flexibility, interoperablity, and survivability [5].

Tactical networks can have quite a lot of moving parts. Legacy systems are incorporated with leading edge technology which requires an integrated, highly-trained communications team. This is essential if all of this technology is to work to enable C2 for the commander. For example, within the Headquarters Battalion of the First Marine Division, the Commu-

nications Company recently consisted of six primary platoons with a specific function: a radio platoon, a wire platoon, a data platoon, a multiplexing platoon, a satellite platoon, and a maintenance platoon. While a tactical network will routinely incorporate all these functions into a rich communications landscape for the commander, the focus of this research is the subset of gear that provides the routing and switching network backbone for a tactical network C2 architecture.

2.1.1 Network Planning

The Marine Corps planning process is a well-documented, deliberate methodology for planning a wide-range of military operations to include the planning of C2 enabling infrastructure. Furthermore, it is properly nested within the overall joint planning process construct currently in use within the Department of Defense (DOD). However, the plethora of material available on military planning is an exhaustive publication list that rivals other large topic areas of study. Fortunately, work exists that has satisfactorily distilled planning concepts succinctly and even focus on planning from a C2 enabling, communcations perspective. This is seen in the work of Garcia as related to his concept of Rapid Network Design (RND) for the Marine Corps [6]. His points of primary interest for this work include:

- Communications planning assesses mission requirements and develops a network design capable of enabling C2 for those requirements.
- Network design tasks are challenging and often include network segmentation and security policy enforcement. The first challenge is addressed through the implementation of VLANs. The second challenge is addressed via ACLs.
- A communications network is a top-down, multi-layer hierarchical design. Top down design starts with a requirements list and develops an infrastructure to support the requirements.
- Network planning in the Marine Corps makes heavy use of network templates (i.e., manually created plans from past exercises or previous operations).

The work of Garcia provides a much more thorough explanation of network design and even analyzes the general Marine Corps planning process from a communications perspective [6].

Once the network plan is approved, the next major efforts at the tail end of the network planning surrounds both allocation and coordination. High levels of detailed planning are

critical in these efforts. The appropriate amount of network equipment and communications Marines must be resourced to support the installation, operation, and maintenance of the planned network. Coordination with external agencies to provide requirements related to power, climate control, transportation, and cryptographic fill include only a portion of the additional synchronization needed to ensure all the elements are ready to begin the subsequent phases of a tactical network. After all this, a network architecture is not yet ready to be installed to support a large-scale field exercise or an upcoming real-world deployment. Communications Marines still have much more ahead before they are ready to declare C2 enabled for a commander. After all, a network plan, however well-articulated and thought out, is still merely in abstract form and will not enable anything, let alone C2, without the next phases of a network which include installation, operation, and maintenance.

2.1.2 Network Installation

If a network design is the representation of a planned network [6], then the subsequent phases which include installation, operation, and maintenance is the realization of that plan. However, to minimize risk and as a precursor to the installation phase, the Marines will conduct several rehearsals, also known as communications specific exercises, prior to a tactical evolution installation. At least two communications rehearsals will take place in this interim phase. These are often communications specific exercises but entail much of the same administration and logistics associated with other tactical elements. During these rehearsals, a tremendous amount of time is spent in coordinating, configuring, and testing the planned communications architecture. For example, a rehearsal may simply be a verbal walk-through of the plan; however, a rehearsal may entail the build-out of the entire C2 architecture to include a live network gateway from which Defense Information Systems Network (DISN) services are validated. While typically the latter is pursued for most evolutions, especially in readiness for large exercises or in preparation for an upcoming deployment, the former is at times necessitated. The first exercise is known as a bootstrap exercise (STRAPEX). The second exercise is often referred to as a communications exercise (COMMEX).

The Communications STRAPEX

A STRAPEX, as the name suggest, is focused on readying an architecture through local configuration. Typically as a system of systems, a communications network is part of much larger enabling C2 structure to include satellite, radio, telephony, maintenance, multiplexing, and cyber security elements. But a STRAPEX is focused at the individual element level. For example, while integration with satellite systems or telephony is important, during a STRAPEX the data section is focused predominantly on networking switches, routers, servers, and a small number of hosts. During this evolution in a division-level communications company, the different sections typically work out of their respective bays, and while inter-platoon coordination and testing can occur, the focus is on getting local configurations correct at the platoon level.

In a larger example, such as a Marine Expeditionary Force (MEF) level exercises, all the participating communications units, to include the communications company, will consolidate at the Communications Battalion, the MEF level communications unit, to ensure all the network gear can talk among all the units. Different units will tie together here with Ethernet cables to abstract the transmission architecture as the idea is to get the network settings correct, such as router configurations, before introducing other elements of network complexity like multiplexers and encryption devices. Throughout a STRAPEX, which typically runs from one to two weeks in length, face-to-face coordination and manual configuration is the standard operating procedure. Once a STRAPEX achieves the desired objectives outlined, a communications section is ready to move into the next rehearsal stage.

The Communications COMMEX

The COMMEX, which follows after all the desired objectives of a STRAPEX are met, is focused on network-wide service validation. For a division communications company, it is the amalgamation of all the elements of a communications plan and brings all sections together to include radio, telephony, data, multiplexing, security, satellite, and maintenance. Within this communications system of systems, this is where all the elements are integrated together and basic network services are validated such as phone connectivity, internet connectivity, email, and access to shared resource locations locally as well as via the web. This happens for the communications company internally as well as externally with its subordinate communications entities. In a larger context, again such as for a MEF,

validation of connectivity to adjacent units as well as higher entities would occur as well. In either case, another important entity that is brought up during a COMMEX is the systems control (SYSCON).

The SYSCON is the hub of all information as well as the execute arm of communications for the commander. The SYSCON leads communications Marines in the installation, operation, and maintenance of the architecture. The ability to monitor the network progress within the SYSCON is almost always a manual endeavor as is the case with network configuration. Security and availability take center stage in bringing a network to an operational status; therefore, even something as helpful as dynamic topology viewing is a late addition to a tactical network. After network-wide service validation is complete, the SYSCON will call an end to the exercise. Afterward, a unit can now ready itself to prepare for the installation phase of a network architecture for the upcoming exercise or upcoming unit deployment.

Because of these two chief rehearsals, the Marines are better prepared to install the planned architecture in a tactical setting, and they do so with the more precision and increased efficiency. However, installation follows the same paradigm in that it is largely manually configured and continues to require large amounts of personal interaction throughout all the units participating. Additionally, while network management software exists and is available for use to implement a network topology monitoring solution to enhance network management, the significant learning time they require make a manual process much preferable for watch officers within the SYSCON.

2.1.3 Network Operation

Once the network hits the operation phase, the focus shifts from network configuration to user support. The network is still monitored; however, to enable C2, the Marine communicators must often shift their focus from a sole network mindset to a customer and network mindset. Network users have a whole range of requests and needs that must be logged, resolved, and closed. This necessary function in support of network users requires a large apportionment of personnel and resources to ensure it is done correctly and efficiently. And like most other processes in networking in austere environments, manual processes again dominate much of this support. It is the typical paradigm for many exercises and even tactical deployments.

2.1.4 Network Maintenance

Maintenance can be an ever present reality for a tactical network, in part due to operating commercial-off-the-shelf (COTS) equipment that was not initially designed to enable robust networks in harsh, austere conditions. Surprisingly, due to the foresight and experience of the Marines, network equipment, which is relatively plentiful as compared to a satellite system as an example, is not normally the limiting factor when it comes to a network related issue. While this does occur and can impede the network, configuration related changes are often the most challenging issues for network operators to troubleshoot and resolve.

The explanation provided is merely a cursory overview of the network phases to give a general context of how a Marine communications section, specifically a communications company, might implement a planned architecture in a tactical environment. While far from a comprehensive survey, it illustrates the many areas where networks are still a manual endeavor. While these manual practices were previously seen as necessary requirements, an enhanced understanding of current network development has changed this understanding. SDN is one such network development that will improve the installation, operation, and maintenance of Marine Corps tactical networks in a remarkable way.

2.2 Unwrapping Software Defined Networks

Even for those with extensive experience in networking, the concept of SDN can be confusing. In part, this is because the name SDN does not immediately differentiate itself from the current networking paradigm. Initially, the first portion of its name "Software Defined" might generate a conclusion that traditional networks are not software defined themselves. However, this is not the case. A Cisco 2951 Router is a piece of hardware that requires both firmware and software in order to function as a proper routing device. Firmware is the semi-permanent software that is closely tied to the hardware [7]; it essentially functions as the intermediary between the Cisco operating system or Internetwork Operating System (IOS). The operating system enables many network related functions to include the Cisco proprietary command line interface (CLI), the well-known application that enables manual network router configuration.

2.2.1 The History

While idea behind SDN and its often associated protocol OpenFlow (OF) are viewed as new developments in networking, these concepts are not modern [8]. Network Control Point was a control and data plane signaling separation that was introduced into a telephone network in the 1980s [9]. Open Signaling began in 1995 with the idea of open, programmable network interfaces [8]. Devolved Control of asynchronous transfer mode (ATM) Networks was yet another effort during that same timeframe aimed at separating control and management functions from ATMs [8]. Active networking was a Defense Advanced Research Projects Agency (DARPA) project in the 1990s with the idea that network traffic could carry customized programs that would run in network equipment [10]. 4D and Ethane, viewed as forerunners to SDN, were projects that focused on managing network policy and network security via a central controller [8], [11]. Although only a portion of the comprehensive body within this area of study, the above examples outline the many historical efforts undertaken to reform computer networks.

2.2.2 The Taxonomy

To encourage clarity and a working understanding of frequently used terms within this work, the following definitions are provided as they relate to SDN:

- SDN: A programmable, centralized approach for managing internet protocol (IP) related traffic across IP related paths or routes. Distinct separation between a controller entity on a control plane and a controlled entity on a forwarding plane [12].
- Network Device: Performs tasks related to the routing and switching of IP related traffic across IP related paths or routes [12].
- Control Plane: A collection of functions that provide prescriptive direction to SDN network devices [12].
- Forwarding Plane: A collection of SDN network devices that request and receive prescriptive direction for routing network traffic [12].
- Controller: The network device that exists at the control plane of a SDN. Provides instructions to controlled devices within the forwarding plane on how to process and forward network traffic [12].
- SDN Switch: A network device that exists at the forwarding plane of a SDN. Requests and receives instructions from a SDN controller on how to process and forward

network traffic [12]. While flow tables and group tables make up its core to enable first a lookup within the table scheme and then a subsequent forwarding of the packet, the device must also contain at least one OF controller channel [13].

- Middlebox: As an intermediate network device, performs functions other than standard functions of a traditional IP router on network traffic between a source and destination [14].
- OF Protocol: A Stanford University development that is still being revised through the Open Networking Foundation (ONF) [12]. Request for Comments (RFC) 7426 continues to describe the protocol as a communication protocol that enables a centralized controller to manage a switch within its apportioned network. Contains flow tables with flow entries to match network traffic headers and subsequently apply appropriate action [8].

2.2.3 The Difference from a Traditional IP Network

The routing process is a network-wide operation to determine the path that network traffic will take from a source to a destination [15]. The distinction between a traditional network and a SDN is the calculation of this path and how it is subsequently applied to the routing devices. In a traditional network, this routing process is distributed. In a SDN, this routing process is centralized.

Routing in a traditional network is done via forwarding tables. As network traffic flows into the router, the traffic contains field headers that are used to match it with stored values in the forwarding table. Once matched with a forwarding port, the traffic is pushed onward to the next router in the path. This will continue until the traffic finally reaches its destination. A routing algorithm outlines the contents of the table. The routers within a common network will use the same algorithm and then communicate results amongst other routers of a network via a routing protocol. In this manner, a path for network traffic is created and a source host can push data to a destination host [15].

Routing in a SDN is also done via forwarding tables. However, this process is now managed through the implementation of a SDN controller. Now, as network traffic flows into a SDN router, more aptly termed a SDN switch, a request for a forwarding table entry will be made to the SDN controller. The SDN, which has oversight over the entire SDN topology,

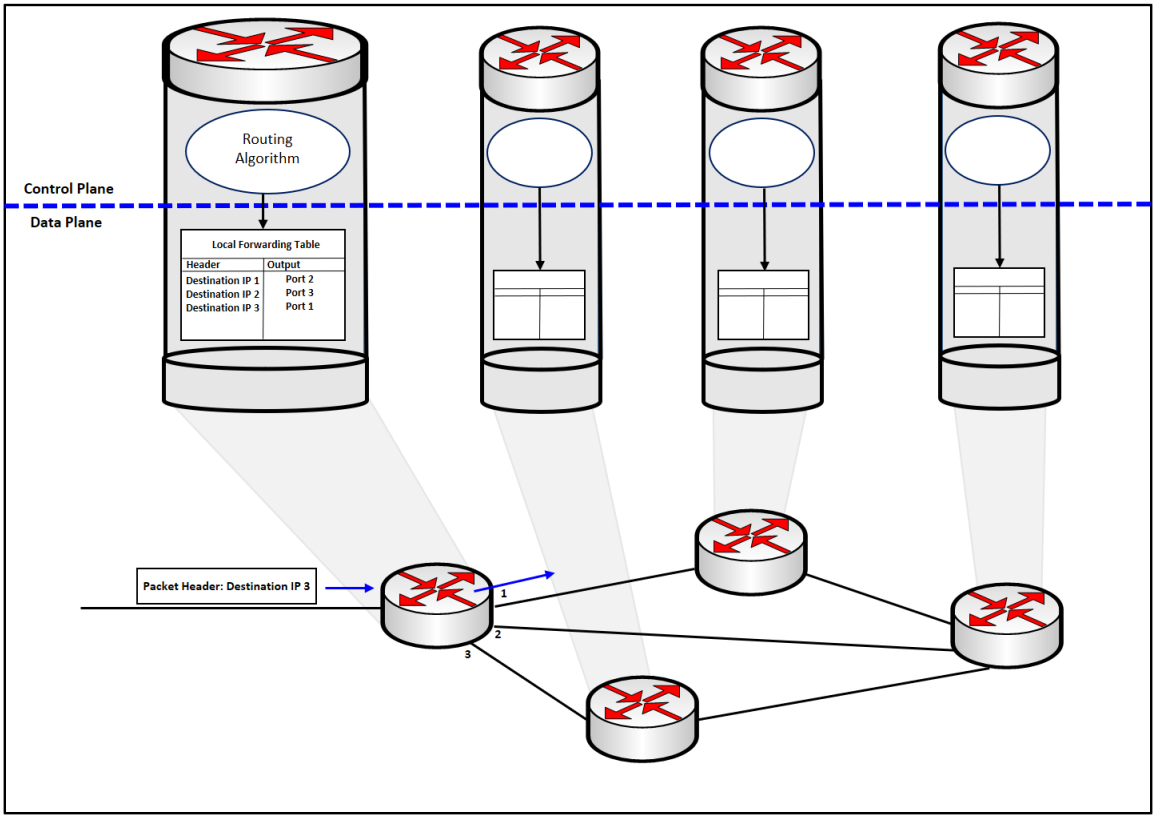

Figure 2.1. Traditional Routing. Source: [15].

In traditional routing, a routing algorithm provides the routing determination.

will provide the appropriate forwarding table information to the requesting SDN switch as well as push an entry for the network traffic to each SDN along the path. The SDN switches will include the new entry in their respective forwarding table, and the network traffic will continue along the path the SDN controller created until the traffic reaches its destination [15].

While Kurose and Ross provide a much more thorough explanation of the distinction between traditional networking and a SDN [15], the above clarity provides the key insight to understand the fundamental difference between the two network approaches. There do exist other alternative explanations that try to explain the difference. For example, SDN is often explained as a separation of the control plane and forwarding plane; however, this is not a sufficient explanation as routers have assumed this separation for many years [16]. Likewise, more succinct explanations may also exist. In one such example, SDN is understood as a

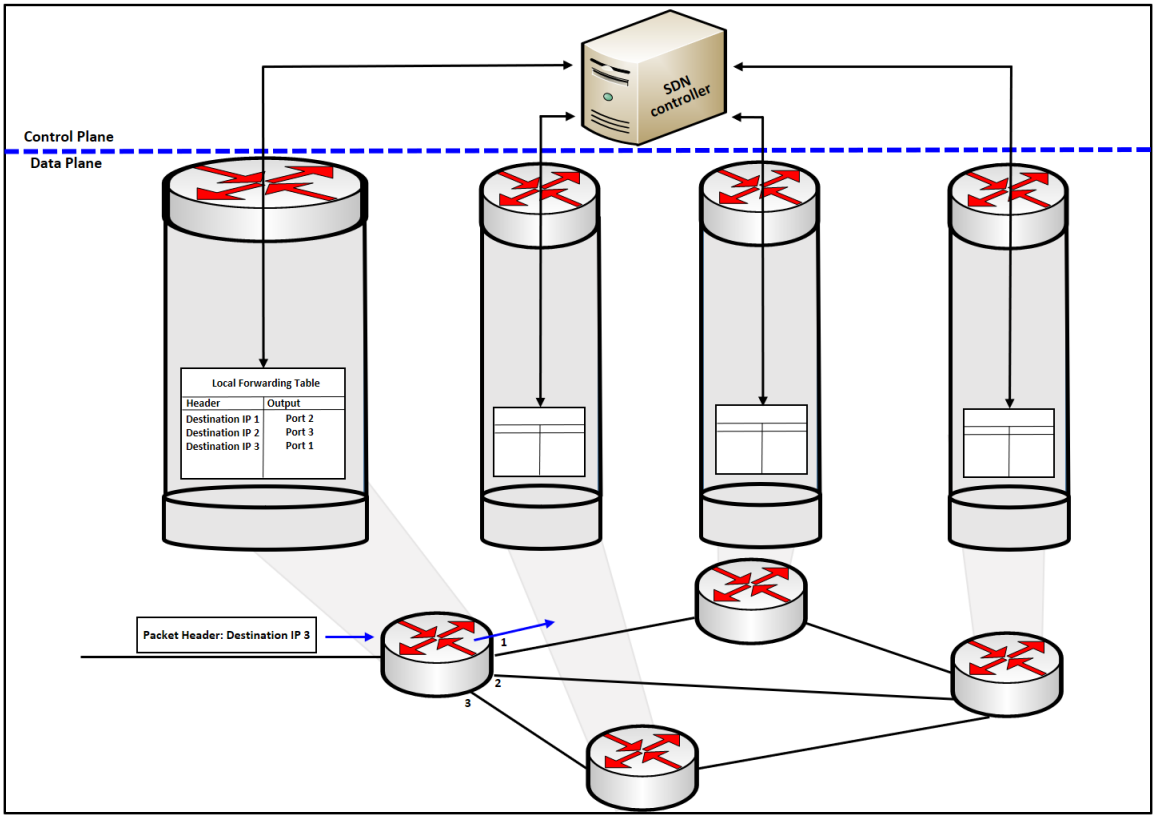

Figure 2.2. SDN Routing. Source: [15].

In SDN routing, the controller provides the routing determination.

separation between a controller entity and a controlled entity [12]. In short, the literature surrounding SDN is abundant and wide-spread. Not only can it enable an understanding of SDN fundamentals, this same literature can provide insight into how SDN can benefit a tactical network.

2.2.4 The Benefits

SDN may not be an entirely new networking approach, but it can have new impact on tactical networks that have been entrenched in a traditional networking model for a number of years. Even ideas previously existing in theory may now be achievable as technology progresses over time and research into programmable networks continues to build upon the efforts of previous trailblazers.

Simple and Programmable

SDN was developed to promote an innovative approach to networking through simplicity and programmability. Simplicity is achieved as network devices become predominantly forwarding devices apart from the SDN controller. Programmability is achieved through open application program interface (API) software which allows for network functionality consolidation, straightforward implementation of new protocols and applications, and less complicated network management and visualization capability [8]. Such programmability encourages adaptability and allows for customization in networks to address unpredictable future requirements [17].

An Integrated Approach

Modern SDN approaches are able to coexist within a traditional Ethernet framework to enable an imcremental adoption. Because SDN protocols, such as OF do not significantly adjust the current methodology of network device forwarding, they are implementable within real-world networking [9].

Currently Available Software

Recent work has focused on SDN standardization and adoption. To this end, capable software is already available to enable a shift in the network paradigm. This software is purposely built for deployment on non-proprietary, non-vendor specific network devices [9].

Abstract Control via the Controller

Centralizing network control can ease and even automate network management tasks. With an SDN controller, lower level configuration and instruction sets can be made transparent through the use of common functionality similar in function to current computer operating systems. In this way, higher level network applications can be developed that makes use of network state, network topology, device discovery, and configuration distribution which the controller can provide [9].

Fault Tolerant Controller Applications

While designed for centralized control, certain network implementations may necessitate the use of multiple network controllers. SDN controllers can be distributed to operate

individual network segments or may even be clustered [9].

Enhanced Configuration

From recent experience, tactical networks are a heterogeneous mix of capable network equipment that require precision in configuration to ensure a smooth running network. Because this is observed as mostly a manual endeavor, it can both time-consuming and error-prone, even with seasoned operators. In SDN, a controller that is in charge of all network devices can ease configuration from a solitary location while simultaneously addressing the traditional network challenges of automatic and dynamic configuration and reconfiguration [17].

Performance Improvement

Traditional network approaches to optimization focus on improving a subset of the network or a portion of the network services which is difficult. Because SDN is a centralized model, optimization can be spanned across a network. Therefore, centralized algorithms could better address global performance optimization in traffic scheduling, congestion control, load-balancing, and quality of service (QoS) [17].

2.3 Related Work

Prior work in tactical networks and SDN focused on increasing agility and reducing operator burden [18]. As Spencer *et al.* focused within the tactical domain, the primary contribution of their work considered SDN applicability in a tactical network environment through quantitative analysis using the Mininet emulation platform. This work will contrast in that its focus is on hybrid networking with an implementation of current network configurations on a real test bed environment.

Work has also been completed using SDN with VLANs [19]. Nguyen and Kim conducted work in graphical user interface (GUI) development for VLAN network management and validated the implementation of hybrid support on a physical test bed [19]. This work differs in that hybrid VLAN integration will be explored using non-OF capable switches, such as a Cisco Catalyst 1900 Switch.

2.4 Putting It All Together

Can SDN really achieve all the benefits previously outlined and provide a foundation from which to enhance tactical networks? How can SDN be integrated into the current network paradigm? Can current capability be maintained in network installation, operation, and maintenance in an austere environment? Is SDN more than an academic technology? Is it relevant today? If SDN can positively respond to all the above questions, then maybe it is time to shift the Marine Corps tactical network paradigm. This work will seek to determine if SDN can indeed be a useful innovation tool in a tactical network implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Tactical Network Experimentation

While Marines are able to prepare, install, operate, and maintain tactical networks as the tactical communications C2 backbone for the commander, there still exists a tremendous opportunity for innovation. SDN is posited as the innovation solution. However, while an exploration into SDN enhances overall understanding of the technology, empirical data will provide concrete evidence in the viability of SDN to enhance tactical network deployments. The quantitative methodology of experimentation is the appropriate avenue to yield the empirical data desired.

3.1 Test Network Configuration

A thorough description of the test network is outlined, primarily as an aid toward reproducible results but also to serve as the framework for the experiments conducted. While the network described could definitively serve as a template for a hybrid network architecture replication, it is recommended that future work capitalize on improvements in both hardware and software that will undoubtedly arise. At the time of this writing, SDN is still a relatively emergent technology as evidenced in Internet Research Task Force (IRTF) RFC 7426 or Internet Engineering Task Force (IETF) RFC 7149 which are informational and not final version standards [12], [16].

3.1.1 The Equipment Set

While there exist whole simulation or emulation environments that enable viable data from experimentation, a physical test bed was preferred. Physical test beds remove layers of abstraction which can better clarify a topology design, especially to those unaccustomed to working with virtual technology or emulated environments. In addition, simple physical elements, which are often times the chief culprits related to network connectivity issues, were deemed important to configure. In addition to the appropriate physical cable connections required to link all the physical devices together, the following devices were used:

- (1) Cisco 2951s with IOS version 15.2(4)M2. This router will serve as the base of

the traditional layer 3 routing architecture.

- (1) Cisco Catalyst 1900 Switch with 2820 Enterprise Edition Software version V9.00.04. Apart from an IP assign for network management, this device will enable layer 2 switching functionality in its default configuration.
- (2) Hewlett-Packard (HP) 2920-24G switches. These dedicated hardware switches can function as traditional switches and are also capable of serving as OF switches.
- (2) Dell OptiPlex 9020 Desktop computers with Ubuntu 16.04 long term support (LTS). These will serve as the Open vSwitch (OVS) switches for the SDN network.
- (1) Dell OptiPlex 5040 Desktop computer with Ubuntu 16.04 LTS. This will serve as the middlebox from which to run the SDN Controller.
- (3) Dell OptiPlex 5040 Desktop computers with Windows 7 Professional. These will serve as clients within the SDN network.
- (1) Toshiba Satellite C855D with Ubuntu 16.04 LTS. This will serve as a client on the traditional network.
- (1) Alienware 17 R4 with Windows 10 Home. This will serve as the configuration laptop and the main computer from which to develop experiments in emulated environments prior to test network builds.

While the entire equipment set is not required for every experiment line or its parts, a subset of this equipment will provide all the hardware necessary to conduct each of the experiments. The equipment set was purposefully scaled in such a manner to focus experimentation on key goals, while also demonstrating the flexibility that SDN can bring to a tactical network.

3.1.2 The Software Suite

In addition to the software packages resident with the aforementioned equipment, additional software was required to enable the experiments. While this software was not included as standard in any of the above listed equipment, it is open-source software that is freely available:

- DARPA telnet protocol server (telnetd) on Linux-based systems. While Terminal Network (Telnet) is the solution for the machines running Windows or Cisco, it must be enabled as it is disabled by default. In Linux environments, such as the Ubuntu operating system (OS), telnetd allows for remote connection and configuration.

- VMWare Workstation 12 Player version 12.5.7 build-5813279. A hypervisor platform that enables the emulation of virtual computer instances on a computer.
- Oracle virtual machine (VM) VirtualBox Version 5.2.4r119785 (Qt5.6.2). A hypervisor platform that enables the emulation of virtual computer instances on a computer.
- The Ryu SDN Controller. A software suite to transform a middlebox to function as the control plane flow manager for the SDN portion of the hybrid network. Of note, additional Python programming language libraries that are not included in a Python installation are also required and are annotated on the Ryu web page.
- OVS version 2.5.4. A virtual switch software to transform a middlebox to function as a multi-layer SDN switch.
- Putty (release 0.70 for 64-bit Windows) with clang 5.0.0 compiler. A terminal emulator that enables various remote connection (e.g., Telnet, Secure Shell (SSH), serial) to access and configure network devices either manually or remotely.
- Wireshark version 2.4.6 (v2.4.6-0-ge2f395aa12). A network protocol analyzer to enable troubleshooting and understanding of network traffic at a granular level.

Cross-platform support of the SDN software enables its implementation on a wide variety of devices. As a result, the installation of the software on the available middleboxes was a straightforward and relatively simple process. Additionally, because the Wireshark version used already included support for the OF protocol, it was immediately ready for any troubleshooting and analysis of a hybrid network environment.

3.2 Overall Experimentation Design

The primary focus of Chapter 3 centers around three lines of experimentation to test the potential relevance of SDN within a tactical network construct. The three lines of experimentation include hybrid network validation, network capability maintainability, and innovation potential.

The innovation potential of SDN is unquestionably the most interesting aspect of moving into a different type of networking construct. However, these interesting aspects will be deprived of their strength if SDN is first unable to integrate into the current network paradigm and then maintain capability that currently exists in a traditional network environment.

3.3 Experiment 1: Hybrid Architecture Validation

As referenced in Chapter 2, the ability to integrate SDN into a traditional network is an important consideration. Validation of a functional, hybrid network means that SDN can be a near-term implementation vice a deep-horizon consideration. It also means that the cost of adoption can be a gradual process. Lastly, it provides an incremental approach to building expertise within an organization largely unaware of SDN technology. To demonstrate a hybrid network, a realistic Jump or Forward Combat Operations Center (COC) architecture will serve as the physical test bed for this experiment.

A Jump or Forward COC is typically a spoke of the Main COC. Because of this, network packets typically proceed through the transmission equipment stack (e.g., a very small aperture terminal (VSAT) terminal) and back to the MainCOC. While the physical separation could potentially span across countries, the link would still be considered as an internal node to the unit and thus be handled in such manner. Multiple networks that are based on different classification levels are also a part of a Jump or ForwardCOC; however, for the purpose of this work, the focus is one network without any particular classification. While aggregation of these different networks occur prior to the transmission equipment stack, they are logically separate, unable to cross-communicate, and do not affect the ability of SDN to communicate with a traditional network.

3.3.1 Experiment 1: Design Rationale

Aside from interoperability, other factors exist in the value of SDN automation in a hybrid network configuration. First, SDN migration will incur a monetary cost. While middleboxes may not be as expensive as proprietary, commercial equipment, there will still be a cost associated with the procurement of devices upon which SDN networks can be built. Current assets that exist within communications sections cannot simply be repurposed; these devices are inventory that serve as either components of other systems or as end-devices required for personnel. For example, laptops could act as switches; however, laptops are meant to serve as workstations for unit personnel. Servers within a COC equipment set are typically resourced appropriately to support network functions related to domain and network services functions (e.g., domain controllers and Microsoft Exchange). Second, the procurement process toward SDN can be incremental. Existing equipment used in tactical networks can continue in its lifecycle and as legacy equipment is replaced, a gradual move toward

SDN can occur. Moreover, this incremental approach can apply to units within a military component (i.e., the Marine Corps) or even between components within the DOD. Third, SDN troubleshooting may prove the most challenging aspect in network management as is the case in traditional networks of today. While the ability to determine and resolve issues locally is largely facilitated through physically co-located and direct communication, nodes separated in space and force encounter the bulk of network fault tracing and correction time. SDN can serve at the network edge, but the flexibility to enable a hybrid architecture may alleviate any potential fixation of SDN devices as the culprit of connection issues regardless of its past, proven performance. Since the baseline of network expertise in tactical communications currently resides in traditional networking, the capability to rapidly connect more familiar technology while simultaneously developing competency in SDN prevents the incremental adoption of SDN at the cost of tactical boundary connection issues. These considerations formed the basis for the overall design of the first experiment.

3.3.2 Experiment 1: Description

Figure 3.1 graphically depicts the network topology used as the foundational architecture. While relatively simple in its design, this topology could easily support the network requirements within tactical Jump or Forward COC where network requirements are much more minimal and often reliant on reach back capability to the Main COC of a unit. Regardless, this network design will focus on core networking equipment and includes devices typically resident within a tactical network while also including SDN capable equipment.

There will be three tests in the first experiment. The first test will validate connectivity of a hybrid network. The second test will then replace the HP switches with OVS switches. The third test will validate functionality of a hybrid SDN configuration within the overall hybrid network.

Purpose of Test 1

This first test will validate communication between a traditional IP routed network and a SDN network. Validation of both internet control message protocol (ICMP), a typical network protocol, and Wireshark, a typical network analyzer to operate in an SDN environment is foundational to the progression of this experiment as well as the other two lines of experimentation.

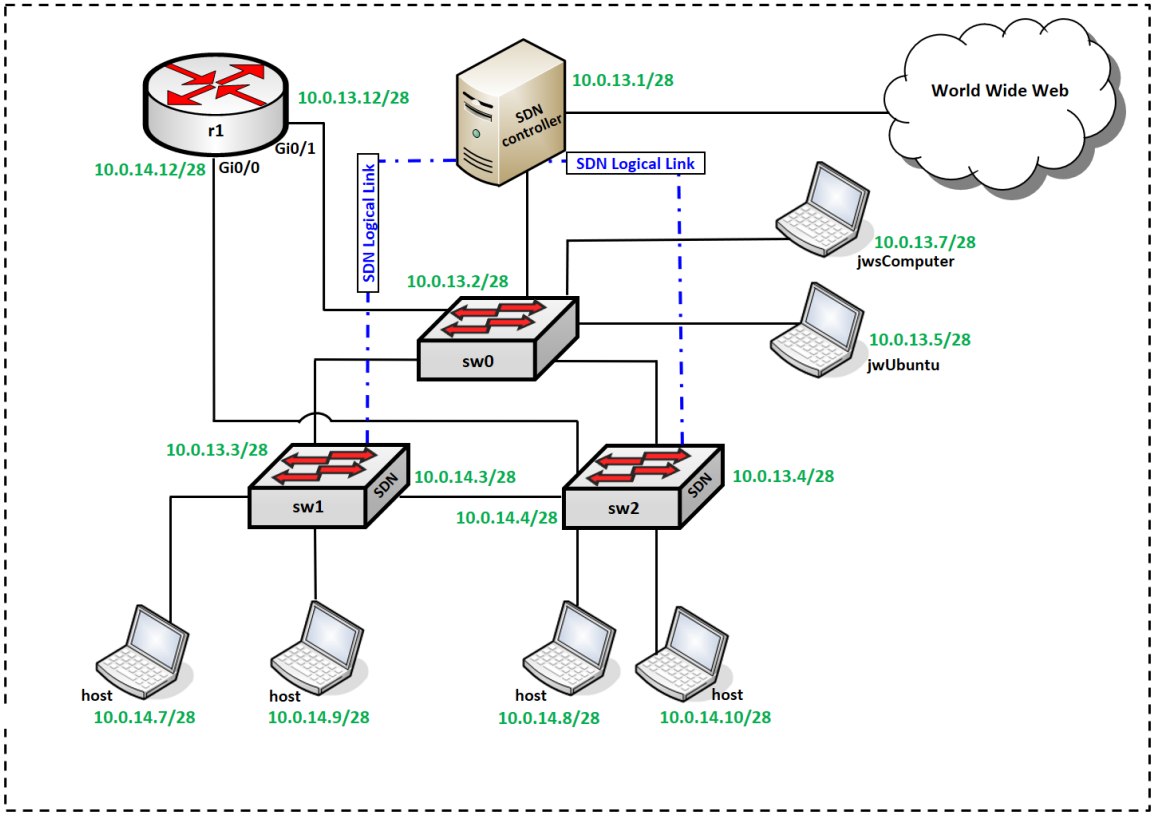

Figure 3.1. Experiment 1 Diagram.

This is a simple hybrid network architecture that demonstrates the potential configuration that could be implemented in a remote tactical network node such as a Jump or Forward COC.

Hypothesis for Test 1

Network traffic generated from a host within a SDN network will be able to traverse from the SDN network through a traditional network and reach a destination host on the traditional network side. The converse will also hold true.

Procedure for Test 1

The following is a list of equipment that will be used in the creation of the initial hybrid network:

- One Cisco Router.
- A Ryu Controller.

- Two HP SDN capable switches.
- Network Hosts (as required).

With the equipment set identified, the installation approach will proceed as follows:

1. Wired, physical connections will join all the network devices outlined in Figure 3.1.
2. Manual configuration of the traditional portion of the architecture will commence. As there is only one router within the architecture, no routing protocol will be required.
3. The Ryu Controller will be manually loaded to run *simple_switch.py*. Afterward, the controller will begin listening on its assigned IP address on its default port for SDN related request traffic.
4. The SDN capable switches will be manually configured.
5. ICMP will be enabled amongst all devices to provide an avenue for connectivity validation between hosts on the various networks.
6. Wireshark will be opened and run to conduct live packet captures to include OF related traffic.

The Ryu Controller and its provided programs will not be modified. The included script from the Ryu Controller download *simple_switch.py* has all the functionality required for flow creation in an SDN environment. These flows will then get installed into the flow tables of the respective HP switches which will enable the forwarding of packets to and from the SDN hosts. Lastly, while no network protocol is required at router as there is only one router implemented, a layer-3 device is still required to enable connectivity between the SDN network and the traditional network.

Data Collection for Test 1

The data collected will determine if such a topology can indeed be implemented. ICMP messages as well as Wireshark captures will collectively determine a successful base functionality of SDN in a traditional network. A list of configurations used is provided in Appendix B.1.

Purpose of Test 2

This second test will replace the HP hardware with middleboxes configured to run as OVS switches. While a hardware solution is a potential avenue into SDN technology, a benefit of

SDN is the Network Function Virtualization (NFV) flexibility it provides to enable general purpose devices to be integrated into a network as required to meet the C2 needs that often change during the course of the tactical network implementation.

Hypothesis for Test 2

Middleboxes serving as SDN switches will interoperate with a traditional architecture as with dedicated OF capable devices. Communication between a SDN host and a traditional networked host will be able to communicate in a full-duplex capacity.

Procedure for Test 2

The two HP switches will be replaced with two middleboxes serving as SDN switches via the OVS software. After a swap of devices, the following procedure will commence:

1. The Ryu Controller will be properly shutdown and then manually reloaded to run *simple_switch.py*. Afterward, the controller will begin listening on its assigned IP address on its default port for SDN related request traffic now from OVS enabled devices.
2. The SDN capable switches will be manually configured. First, the requisite quantity of ports will be enabled on each of the middleboxes. Then, manual configuration of the switches will ensure that the OVS devices point to the controller for flow programming to enable packet forwarding.
3. ICMP will be enabled amongst all devices to provide an avenue for connectivity validation between hosts on the various networks.
4. Wireshark will be opened and run to conduct live packet captures to include OF related traffic.

No changes will be made to the SDN host network configurations. No modifications will be implemented on the traditional network architecture.

Data Collection for Test 2

A similar collection of data as gathered from Test 1 using both the ICMP protocol as well as Wireshark captures will verify the results observed. The configuration template used for OVS programming is available in Appendix B.1.1.

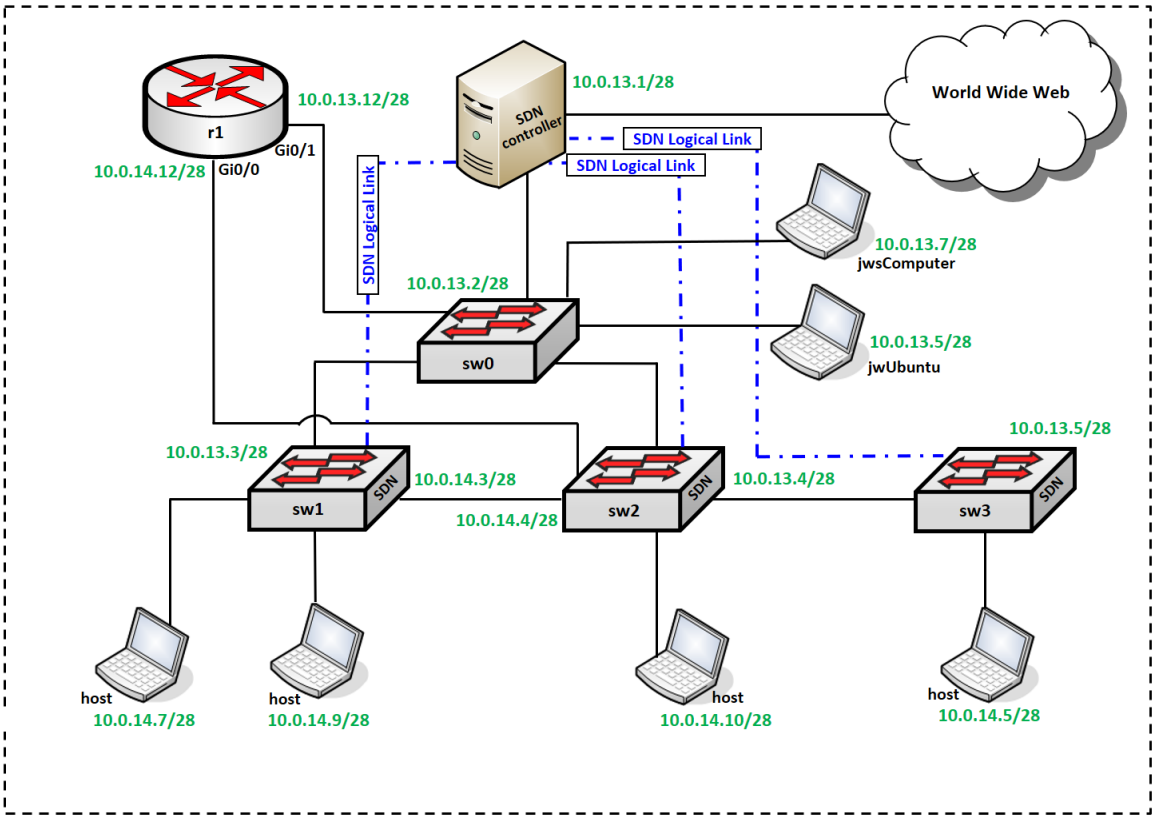

Figure 3.2. Experiment 1: Test 3 Diagram.

Growth of tactical network is at times necessitated as a situation develops. The previous Jump or Forward COC network is augmented with another SDN device.

Purpose of Test 3

The third test will create a hybrid topology within SDN via the reintroduction of a HP switch. Such a nested hybrid network implementation will demonstrate the compatibility of different SDN capable devices to coexist to provide SDN network connectivity. Additionally, network growth in a tactical setting may necessitate the need for an additional switch physically located in another part or section of the forward operating area. Figure 3.2 outlines the addition in the hybrid network topology.

Hypothesis for Test 3

The SDN controller will dynamically handle different SDN implementations to install flow rules that will network connectivity to each of the SDN hosts behind their respective switches. Packets will be able to traverse freely to any host within the SDN and traditional

network environment.

Procedure for Test 3

One HP switches will supplement the previously depicted network architecture. After the reintroduction of the HP switch, the following procedure will commence:

1. The Ryu Controller will be properly shutdown and then manually reloaded to run *simple_switch.py*. Afterward, the controller will begin listening on its assigned IP address on its default port for SDN related request traffic now from OVS enabled devices.
2. The HP switch will be manually configured. Previously generated configurations for the HP switch will be retained, albeit with different IP addressing as appropriate.
3. An additional host will be added to the HP switch and assigned the appropriate Internet Protocol version 4 (IPv4) configurations.
4. ICMP will be enabled among all devices to provide an avenue for connectivity validation between hosts on the various networks.
5. Wireshark will be opened and run to conduct live packet captures to include OF related traffic.

No changes will to any of the other devices within the hybrid network architecture.

Data Collection for Test 3

A similar collection of data as gathered from both Test 1 and Test 2 using both the ICMP protocol as well as Wireshark captures will verify the results observed.

3.4 Experiment 2: Maintain Network Capability

As a logical progression from the validation of a hybrid network topology, a SDN must also achieve traditional network functions typically configured in a tactical network. A demonstration of VLANs and ACLs within a SDN architecture provides substantive proof of SDN to achieve two required network functions essential in tactical network configurations.

3.4.1 Experiment 2: Design Rationale

Two critical challenges for network engineers involve network segmentation and security policy enforcement. The standard solution to the first challenge is the application of VLANs in the network. The typical response to the second challenge is enacting packet filters, otherwise known as ACLs [6], [20].

While a more thorough exploration of VLANs and ACLs are better outlined elsewhere, the essence of each is captured here. VLANs, a configuration primarily in switches, enable network traffic isolation which can greatly reduce network broadcast traffic, enhance the efficient use of switches as one switch can support many segments, and alleviate challenges with users needing access of various network segments. ACLs, an implementation of a traditional packet filter firewalls normally installed in routers, examine and subsequently allow or deny network traffic the ability continue along a forwarding path based on a set of configured rules [6], [15], [20]. For these reasons, these two elements are heavily used in tactical Marine Corps networks. If a near-term SDN implementation is desired, there is benefit in revealing an identical ability exists therein as well.

SDN could even propel change in these typical implementations over time. With the rise of SDN, the relevance of VLANs and even ACLs on networks has received some attention as networks grow more wireless and distributed with application level security seen as more effective in data protection [21], [22]. In an example with respect to VLANs, Address Resolution Protocol (ARP) broadcast traffic, a fundamental protocol that enables network connectivity within a VLAN, could be minimized as a central SDN controller now manages all network flow creation. In another example that addresses ACL functionality, security of servers sharing a network could be enabled with less difficulty than can be accomplished in a traditional network through the use of a bridging firewall. Regardless, both VLANs and ACLs are part of the defense-in-depth defense strategy that communications Marines will continue to integrate into tactical network for the foreseeable future. And since SDN switches, particularly OVS specific switches, have default functionality to act as a traditional layer-2 device if unable to reach a controller on the network, a SDN switch should be able to accomplish these two elements which are important in tactical networks.

As a result, the capacity to prove maintained network capability while simultaneously enabling the enhancement to both network performance and network security are important

areas of interest. The second experiment was designed around these considerations.

3.4.2 Experiment 2: Description

This linear network switch topology could satisfy an access layer network requirement at a higher echelon Forward COC, such as a deployed Special Purpose Marine Air-Ground Task Force (SPMAGTF). A security posture may dictate a physical separation of personnel and equipment, and the introduction of a switch in the middle of trunk can act as a repeater for a signal that attenuates beyond 100 meters. A single VLAN might initially be installed as a management VLAN for segregating administration resources from general network use.

As a network topology subset from the previous experimentation, this experiment will focus initially on VLAN configuration and then conclude with tests in ACL implementation. Of the five planned tests, the first three will center around VLANs while the last two tests will focus on ACLs. Of note, the switch trunking protocol of an OVS switch uses the open standard 802.1Q [23], but the Cisco Catalyst 1900 only supports Cisco Inter-Switch Link (ISL) encapsulation [24]. As a result, no trunking between the traditional switch and the OVS switches was planned in the second experiment.

Purpose of Test 1

The first test will validate the ability of the hybrid switch topology to communicate within a single VLAN. The hybrid network will be achieved via a Cisco switch between two OVS middleboxes. A host will be connected to each switch in the linear topology.

Hypothesis for Test 1

One VLAN configured at the Cisco switch which includes the ports of both SDN switches will keep all hosts within one VLAN segmented from another configured VLAN on the Cisco switch. Although the SDN switches do not have access to a network controller, they will be able to communicate at the layer-2 level via ARP.

Procedure for Test 1

Each of the switches will be cleaned of any configuration information and start in an initial, default configuration. Figure 3.3 outlines the VLAN topology for this test. Afterward, the following will outline the installation procedure:

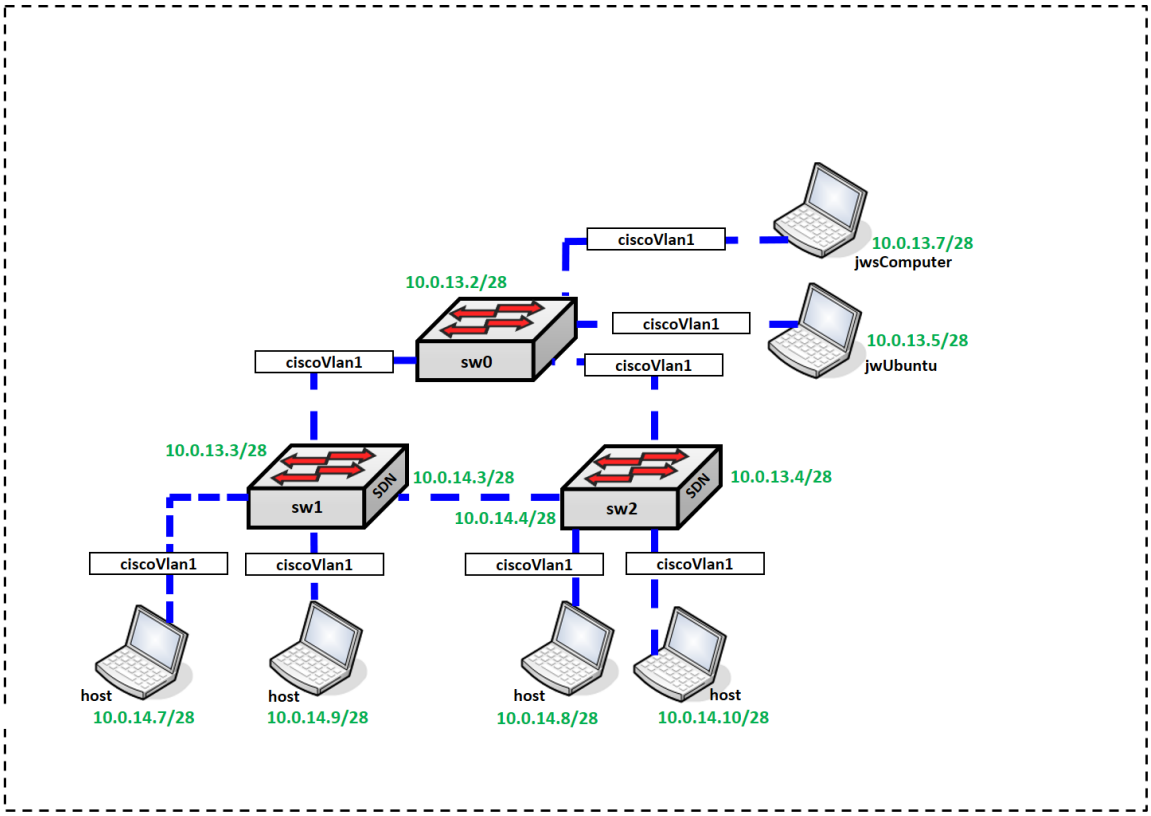

Figure 3.3. Experiment 2: Test 1 Switch VLAN Topology

1. The OVS switches will be configured with two ports. One port will serve as the switch trunk, while the other will serve as an access port for the connected host.
2. The Cisco switch will be configured with two VLANs. VLAN 100 will be the network segment inclusive of the SDN network, while VLAN 200 will only encapsulate two clients off the Cisco switch.
3. Each host will be configured on the appropriate network segment with includes an IP address as well as a broadcast address to resolve layer-2 communication via ARP.
4. Tcpdump, a packet analyzer, will be an initialized process on each of the network hosts which will enable visual confirmation of all ARP messaging on the network.
5. ICMP will be enabled amongst all devices to provide an avenue for connectivity validation between hosts.

Data Collection for Test 1

The data collection here will focus on ensuring that traffic is only received in the appropriate network segments. To the end, captures from Tcpdump on hosts assigned to different VLANs will serve as the appropriate verification. Configurations for the switches are available in Appendix B.2.

Purpose of Test 2

This second test will build upon the previous configuration of the first test in this set. An initial architecture with one VLAN at the distant ends of a network will not normally achieve the endstate desired of a network design. Therefore, an additional VLAN will be added to each endpoint.

Hypothesis for Test 2

Multiple VLANs can be configured at each of the OVS switches. Hosts in each of the VLANs can communicate to each other while communication outside of the VLAN assigned will be restricted. Figure 3.4 outlines the VLAN topology for this test.

Procedure for Test 2

The following procedure will occur to ready the modified switch topology for testing:

1. Each of the hosts on the Cisco switch will be moved to the OVS switches.
2. The Cisco switch configurations will be removed. The Cisco switch will function in its default configuration.
3. The OVS switches will be configured with one additional port.
4. The appropriate OVS configurations to enable VLAN tagging and trunking will be implemented.
5. Tcpdump will be initialized on each of the hosts.
6. ICMP requests will provide the network traffic to monitor the implementation of the modified VLAN architecture.

Data Collection for Test 2

Data collection will occur in the same manner as with test 1. The revised configurations for the OVS switches are available in Appendix appendix:configs2.

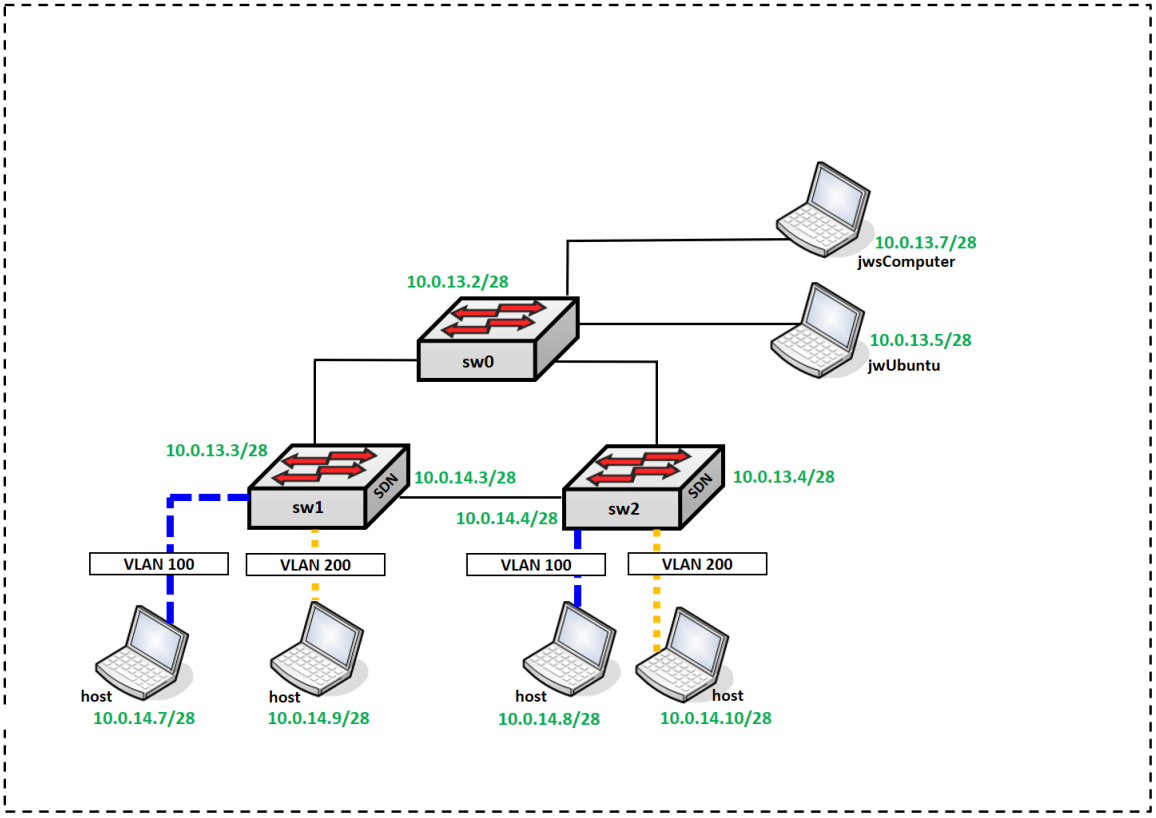

Figure 3.4. Experiment 2: Test 2 Switch VLAN Topology

Purpose of Test 3

Test 3 will conclude the final VLAN validation with this experiment. The hybrid switch topology will be configured differently from the previous two tests to demonstrate the full functionality of VLANs in a hybrid architecture. Figure 3.5 outlines the VLAN topology for this test.

Hypothesis for Test 3

As before, hosts in each of the VLANs will receive and send traffic to other hosts within their network segment.

Procedure for Test 3

For this reconfiguration, each of the three switches will be cleaned of any configuration information and start in an initial, default configuration. Then, the following installed will

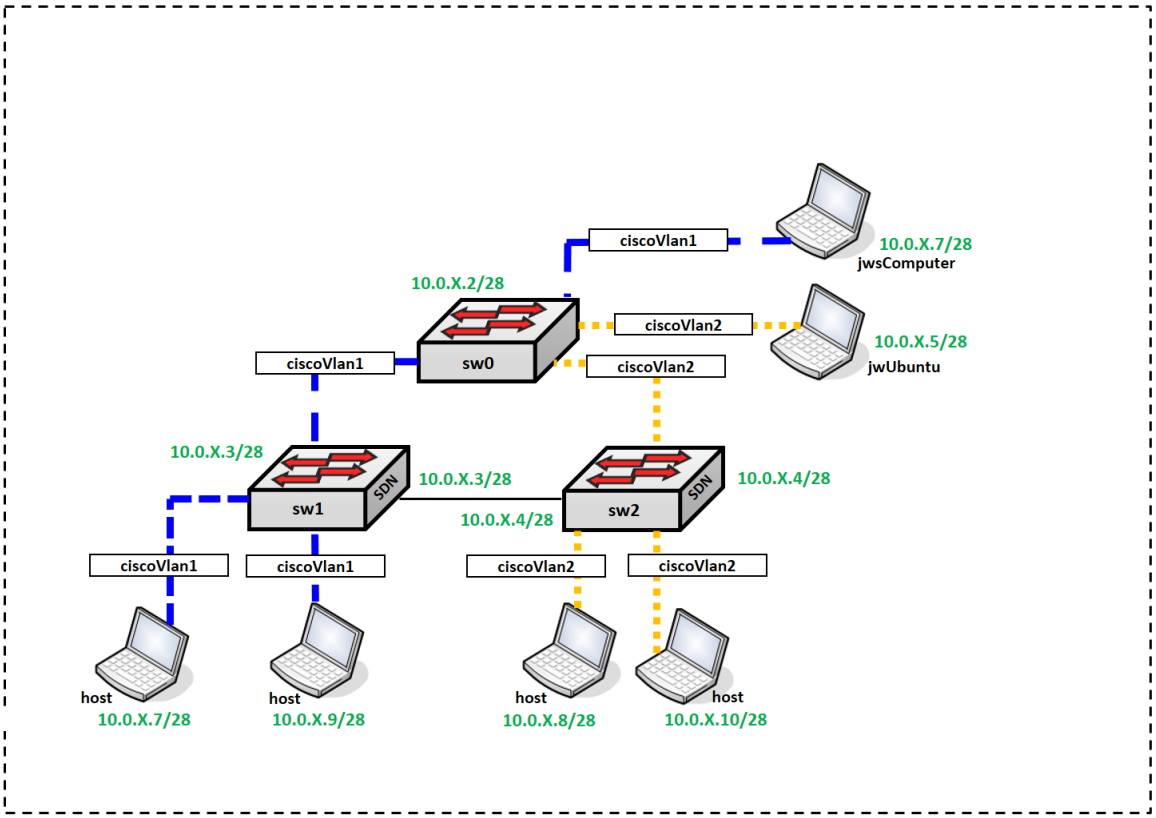

Figure 3.5. Experiment 2: Test 3 VLAN Topology.

follow:

1. One host will be assigned to each of the OVS switches. Two hosts will be attached to the Cisco switch.
2. The Cisco switch will be configured with two VLANs.
3. The OVS switches will be configured to enable access for each host onto the respectiveVLAN.
4. Tcpdump will be initialized on each of the hosts.
5. ICMP requests will provide the network traffic to monitor the implementation of the modified VLAN architecture.

Data Collection for Test 3

Tcpdump will display network traffic each host on the network either sends or receives. The updated configurations for the switches are available in Appendix appendix:configs2.

Purpose of Test 4

The fourth test will move into the realm of ACLs with SDN. The first test will validate the ability of a flow rule to act as a packet filter. While flows rules can match on several fields above the configuration ability of that within a traditional architecture, flow rules will match currently used fields within a traditional network environment.

Hypothesis for Test 4

A flow rule can be implemented in a SDN switch to enable traffic to reach a destination or prevent traffic from reaching a destination. The flow rules can be configured in various ways available in a traditional network paradigm.

Procedure for Test 4

For this test, the architecture used in test 2 will serve as the topology. The following procedure will be used to ready the environment:

1. The Cisco switch will be removed of all of its configurations and restarted in a clean-slate, default configuration.
2. The OVS switches with flow rules with matching fields typically used in a traditional network to allow or deny traffic.
3. ICMP requests will provide the network traffic to monitor the implementation of the ACLs configured.

Data Collection for Test 4

Simple ping activity will be used to validate connectivity to and from both source and destination hosts.

Purpose of Test 5

As a progression from Test 4, the fifth test will delve into the realm of OVS switch tables to expound on some of the additional functionality that SDN enables for a network engineer. An ACL list will be configured and traffic will move through the flow tables with the OVS switch to ultimately be allowed to reach the destination host.

Hypothesis for Test 5

An ACL list will be configured on a flow table in much that same way as is currently practiced in a tactical network environment. Additionally, the flow rules can be configured in various ways unavailable in traditional networks to include timeout fields to collapse flows after certain time requirements are met.

Procedure for Test 5

The architecture will remain same as previously configured with the exception of the removal of any lingering flows. The following procedure will be used to modify the architecture:

1. All flow tables within the SDN switches will be removed.
2. The OVS flow table will be configured in a similar fashion as a traditional ACL. However, additional tables will be used to better capture the functionality of SDN in pushing traffic through the flow table as appropriate to allow or deny traffic between hosts on a network.
3. ICMP requests will provide the network traffic to monitor the implementation of the ACLs configured.

Data Collection for Test 5

While simple ping activity will once again provide the validation of a successful ACL implementation, Wireshark captures will also verify that traffic within a network is traversing through the flow tables as appropriate.

3.5 Experiment 3: Innovation Potential

Successfully testing a hybrid architecture and demonstrating a maintenance of current network capability might show SDN as a relevant network solution, but to adopt a new solution, particularly in a large organization accustomed to the functionality of a current construct, something more is required. The ability of SDN to reform tactical networks is the primary objective of this work; therefore, the final experiment will focus on the potential of SDN to enhance the current Marine Corps network environment.

3.5.1 Experiment 3: Design Rationale

The experimentation of ACLs already began to hint at the additional power that SDN can have in a tactical network. In another example, the SDN controller, OVS switches, and network hosts are interchangeable and enable much more flexibility with a more generic equipment set than is currently achievable in a tactical network environment with dedicated network devices. Still, more is achievable within SDN. One such immediate capability within SDN is automation.

Automation has always been a goal of computer scientists [25]. And while automation is achievable in a traditional network environment, it is not taught or utilized currently within the Marine Corps. Even today, tactical networks are predominantly configured manually via the command line interface. This is primarily due to unrealized ability in network automation and not willful neglect. However, if automation can be enabled through SDN, this would not reduce the role of Marines in the development of a C2 architecture, but it can enhance their performance. In line with the doctrinal tenants outlined in Marine Corps Doctrinal Publication (MCDP) 6, automation, less error prone versus manual configuration, can decrease the number of personnel potentially required in the installation and configuration of a tactical networks through basic automation practices that are both user-friendly and easy to understand. Because of the decreased time in the implementation of these routine processes within networking, Marines would regain that time to focus on high-level tasks that require judgment and intuition, a stated technological goal in Marine Corps doctrine [4]. Validated automation processes remove a large amount of potential human error often associated with the network installation and management while at the same time improving uptime. Any substantial proof of SDN to enhance the communications Marines ability to accomplish objective of enabling C2 for the commander would make rapid adoption of SDN an almost rhetorical imperative. This contemplation served as the foundation for the third experiment to be grounded in SDN capable automation.

3.5.2 Experiment 3: Description

Three predominantly identifiable avenues of SDN automation exist within network configuration, network observation, and representational state transfer (REST) API integration. First, automation in network configuration is achievable through programming; a generated script can be pushed to network devices to configure them in a remote environment. Network

observation, typically a development once a network topology is first operationally capable, is an existing library within SDN controllers, such as the Ryu controller, offer immediate topology viewing even in the infancy stage of network setup. REST, as outlined in RFC 7231, uses the Hypertext Transfer Protocol (HTTP) protocol to push or pull information to and from network resources [26]. While HTTP is a primary world wide web protocol to enable web applications, it can also be implemented within an SDN API to carry out instructions for networked devices as it can access data and functions of respective devices in a more abstract way to achieved a desired programmed environment endstate [27]. While none of these concepts are particularly tied to SDN, they are natural complements. The testing within the third experiment will focus on these three automation capabilities within SDN.

Three tests will occur within this third line of experimentation. The first test will achieve network automation of a hybrid network through a program that is pushed to devices via a single configuration laptop. The second test will explore the implementation of the built-in topology feature that is already available within the Ryu SDN controller. The final experiment will dive into the REST API of the Ryu Controller to both gather data and subsequently push configurations.

Purpose of Test 1

Figure 3.1, the topology for the first experiment, will provide the equipment set for test 1. This test will validate the ability of a script to automate the configuration of all the network devices which can enable inefficiencies in installation and greatly reduce setup time.

Hypothesis for Test 1

Having validated a working hybrid architecture, automation of both the traditional and SDN portions of the network will be achieved via scripts run from a configuration laptop.

Procedure for Test 1

The list of equipment used for this test will mirror what was used in the first experiment. All of the devices will be removed of any configuration information and start in an initial, default configuration. Afterward, the following will outline the installation for this test:

1. Validation of the physical connections between the devices will ensure.
2. A basic configuration will be implemented on each network device. A single port will need to be configured to enable remote access.
3. The Telnet protocol will serve as the communication medium to push of configuration files to automate the configuration of the network devices. The router, controller, and switches are the focus of the automation script.
4. Connectivity via ICMP will test that all network devices are reachable from a single configuration laptop.
5. The python script adnAndTraditionalAutomate.py will execute on the network. The script will automatically configure all the network devices in the same configuration as previously implemented to include the Ryu controller.
6. ICMP will then validate connectivity between all the hosts on the network.

Data Collection for Test 1

A successfully run script will be the first determination that the network devices were successfully configured. ICMP messages as well as Wireshark captures will collectively definitively determine a successfully executed script. The program files used for this automation test is provided in Appendix B.3.

Purpose of Test 2

Throughout the course of a network installation, operation, and maintenance, a tremendous amount of resources are expended in systems control. The overall purpose of systems control is to align understanding of the current network state and to guide efforts toward a desired endstate. The entity within a tactical communications unit responsible for the management of this process is called SYSCON and is an exercised function the entire duration of a network implementation. A SYSCON section will normally stand up in the installation phase of a network and continue through the operation and maintenance phases. In the installation phase, the primary objective is a fully operational network. In a maintenance phase, this might entail a reconfiguration of a network device or implementation of a new network resource which then needs to be validated as functional within the allotted maintenance window. Regardless of the phase a network topology provides the framework for understanding the network state. Early in a network implementation, this is chiefly a manual endeavor. While software exists to enable network topology state observation

and management, this is typically functionality reserved for a more mature network and be challenging to implement. The ability to have immediate topology observation would have immediate benefit in efficiency and provide enhanced awareness for the SYSCON in driving operations toward different network endstates.

This second test will enable the functionality of the topology resources within the Ryu Controller and validate its ability to present an current operational picture of the SDN environment. Of note, other controllers have more robust and capable topology functionality, such as the Open Network Operating System (ONOS) controller. The functionality within a topology view is outside the scope of this test; however, further development of the basic Ryu example could enhance its ability to function similarly to an ONOS controller, if Ryu was adopted to serve as the SDN controller for USMC tactical networks.

Hypothesis for Test 2

The included topology functionality provided in the Ryu Controller will provide a viewable topology of the SDN network.

Procedure for Test 2

The end state of the previous test will serve as the beginning state for this test. From there, the follow installation procedure will bring a viewable topology to the network manager:

1. The Ryu Controller will be initialized with two programs. *gui_topology.py* and *simple_switch.py*
2. ICMP will then validate connectivity between all the hosts on the network.
3. From the SDN controller, access to the topology view will occur after opening the web browser and pointing to the location of the page. In its default configuration, the page will be viewable at http://0.0.0.0:8080.

Data Collection for Test 2

After successfully running the topology program, a message will be listed within the notification messages of the Ryu Controller. A screenshot capture will reveal the viewable topology created in Ryu which will determine if the functionality is operating correctly.

Purpose of Test 3

The REST API provides an additional avenue into the management and configuration of an SDN architecture. While vary implementations may occur in different SDN controllers, the REST API within Ryu will provide the requisite demonstration of such an application to enhance the installation, operation, and maintenance of a tactical network.

Hypothesis for Test 3

REST API can be integrated into a program to provide the rest functionality. As a result, instructions can be push from the controller to SDN switches to gather data or configure SDN devices.

Procedure for Test 3

Without any changes to the topology used in the previous two tests, the following procedure will ready the Ryu Controller with REST API capability:

1. The previously used program simple_switch.py will be modified to include the REST API library.
2. The Ryu Controller will be initialized with the modified program.
3. REST commands will then be executed to first conduct data collection on the SDN switches and then to push configuration information across the SDN.

Data Collection for Test 3

Screen captures will validate data collection to the controller. Controller message configuration data pushed to the switches will be captured via wireshark and through visual observation within both the controller as well as the reconfigured switches.

3.6 Summary

Far from a comprehensive exploration into the merits of SDN, the above experimentation was focused first to demonstrate interoperability with current traditional networking that exists in Marine Corps networks, then to validate maintain network capability, and finally to highlight the potential and near-term applicability of SDN in a tactical network environment.

It is the ultimate intent of this work to set the foundation for further exploration into SDN as a viable innovation solution for tactical Marine Corps networks. Furthermore, it is believed that the results from these experiments will both capture the relevance of SDN as a near-term networking solution while simultaneously spurring future work into additional innovation potential of this accessible technology.

CHAPTER 4:
Findings

The empirical data garnered from the experiments in Chapter 3 provide the basis to substantiate a baseline claim of SDN as potential solution for networks in a tactical Marine Corps environment. The specific areas of experimental focus included network interoperability, SDN capacity in fundamental network constructs, and SDN network included hybrid architecture validation, maintenance of network capability, and SDN innovation potential.

4.1 Experiment 1 Results

The first experiment successfully enabled a hybrid network topology as depicted in Figure 3.1 and Figure 3.2. Different configurations of a SDN were constructed for each of the tests and all proved capable of enabling network connectivity amongst all the clients on the live testbed network.

4.1.1 Test 1 Results

After properly wiring all network devices in accordance as previously outlined, the traditional network portion was configured. In its default configuration, no changes were made to the traditional Cisco switch. Two Ethernet ports were configured on the router; one port was configured for the 10.0.13.0/28 network, and the second port was configured for the 10.0.14.0/28 network. As seen in Figure 4.1, no routing protocol was required in this traditional installation as the Cisco router negotiated network traffic between each the ports it controlled.

After completing configurations on the traditional network, manual configuration of the SDN network commenced. After the port configuration, the SDN controller was initialized with the program *simple_switch.py* and began listening for incoming flow requests as depicted in Figure 4.2. The next portion of configuration included manually formatting each of the two SDN capable HP switches. After configuring two ports, one for controller interface and the other for the SDN network, OF was subsequently enabled. Figure 4.3 highlights the OF configuration used while Figure 4.4 outlines the configuration for the

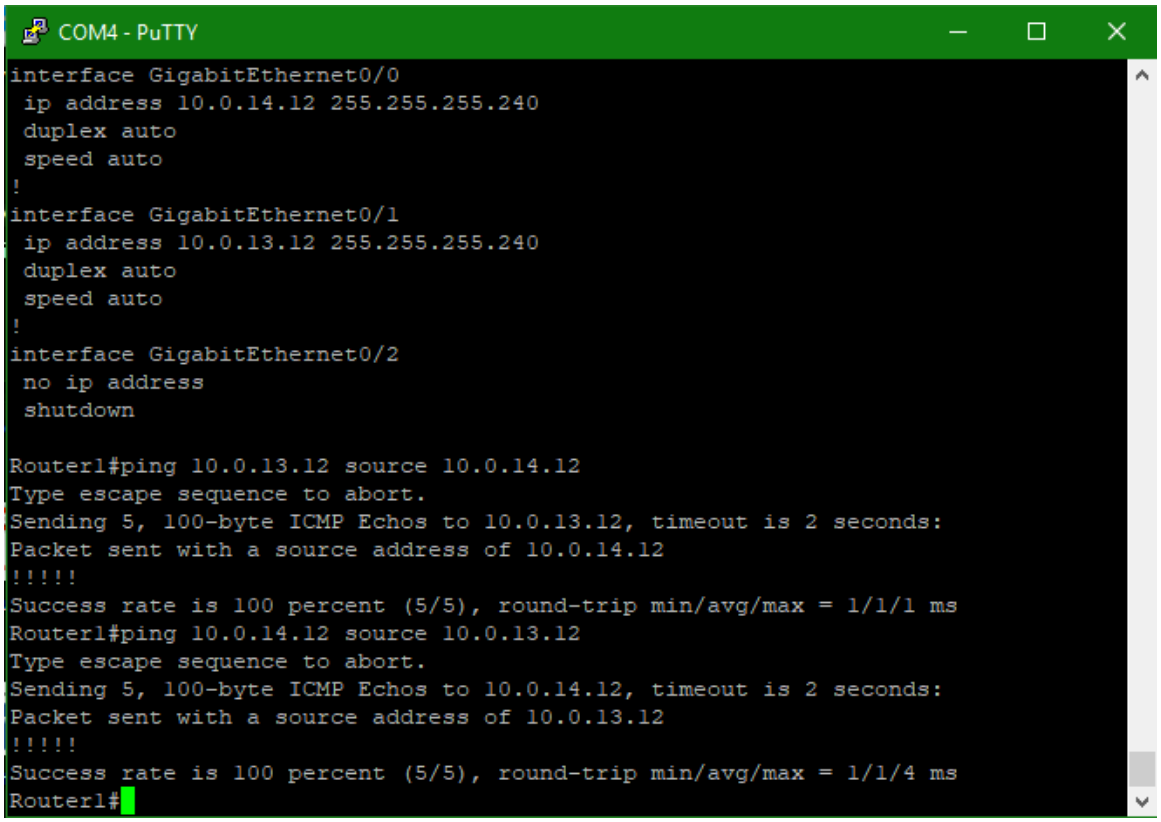

Figure 4.1. Experiment 1: Test 1 Router Connections.

ports of each of the HP devices. Finally each of the three hosts were configured on their respective networks to enable testing of the hybrid topology.

Observations during the implementation intially provided inclusive results as to the proper implementation of the SDN network. First, connectivity between hosts within the 10.0.14.0/24 network, earmarked for the SDN architecture were able to communicate without a controller present or OF enabled on the HP switches. Then, after OF was enabled on the SDN architecture, the controller was able to be removed from the network without affected the ability of the SDN hosts to send and receive network traffic. Lastly, when the SDN controller was managing the SDN, the Ethernet cable for an SDN host was moved from one port within the HP alloted OF port block to another port within the same port interface block without any updates provided to the controller which resulted in traffic no longer being able to traverse to and from the host. However, connectivity resumed after the Ethernet cable was returned to the original port within the port block.

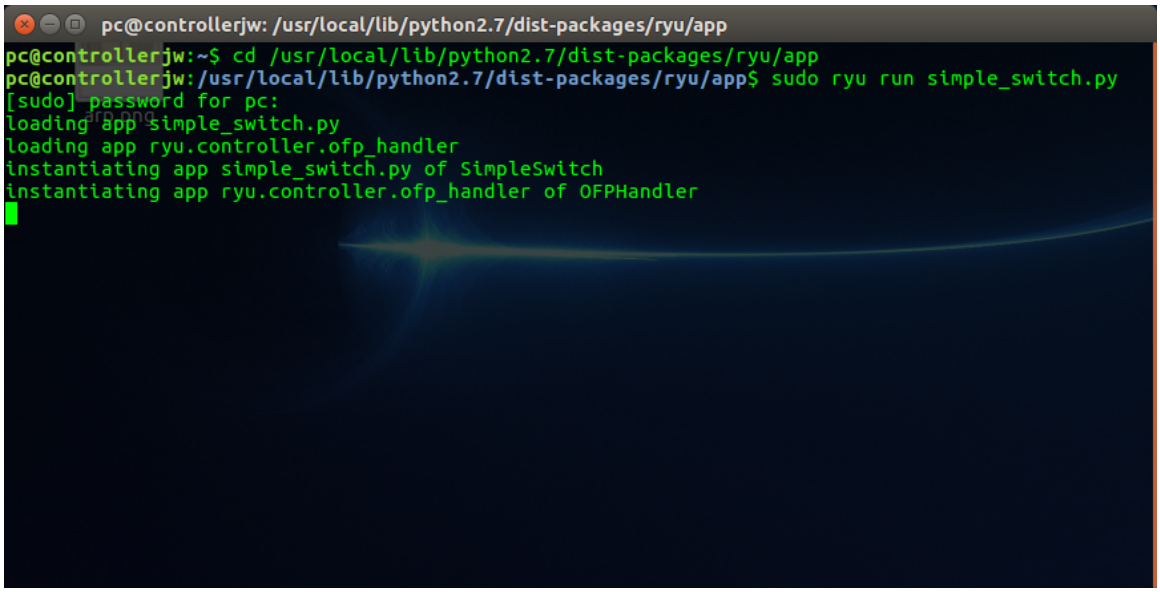

Figure 4.2. Ryu Controller Running *simple_switch.py*

For the first observation, a closer inspection of the network traffic using Wireshark revealed that ARP was enabling connectivity. This was a result of the HP switches allowing traditional network connectivity in their default configuration much in the same manner as occurs within a Cisco switch. To prevent this default configuration from automatically occurring within a SDN environment, a modification was required to prevent ARP within the OF network. The default traditional switching protocol was disabled via the injection of the additional command in the OF portion of the HP switch configuration:

```
HP-Switch1(of-inst-ofnetwork)# connection-interruption-mode fail-
    ↪ secure
```

Afterward, a true SDN via the OF protocol was validated and traffic within the SDN was fully managed via the SDN controller through the use of the OF protocol. While the additional command prevented inadvertent connectivity within the HP switches using ARP, it did not display in the configuration file for the HP switches.

For the second observation, the Ryu Controller creates flows with an idle timeout and a hard timeout both set to zero. Per the OF switch specification, this will result in a flow being created permanently within the flow table of a OF capable switch [28]. A second controller, the ONOS controller, was also tested, but a different result was observed. Because ONOS

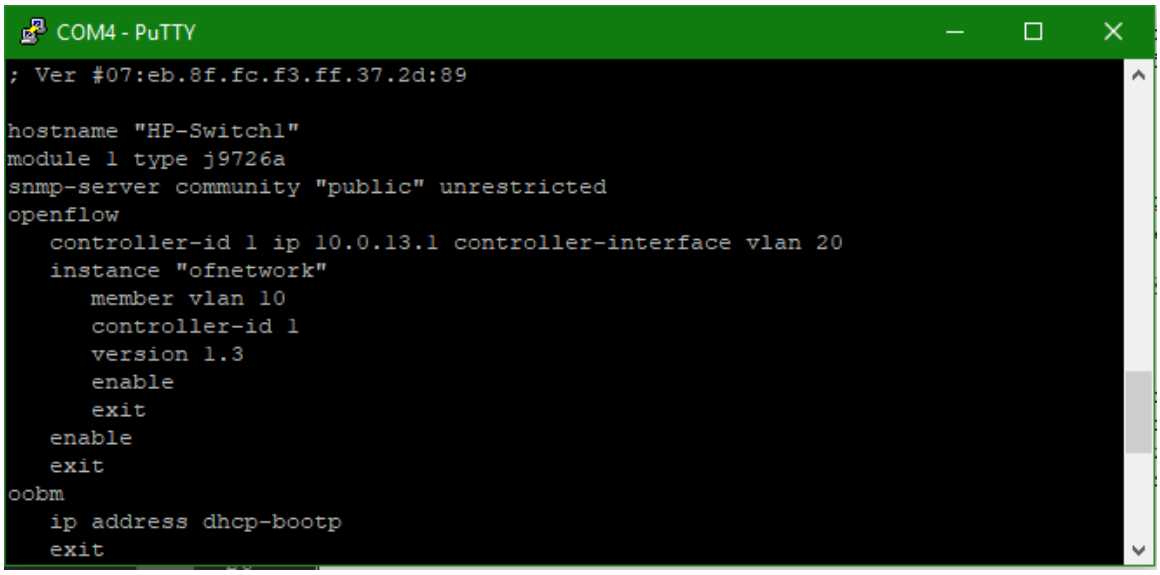

Figure 4.3. Test 1 HP OF Configuration.

The SDN configuration on a HP switch.

did not set both the idle timeout and hard timeout values to zero, flows collapsed almost immediately once the controller was removed from the SDN. In short, everything worked exactly as designed which was clarified after delving into the SDN flow tables as well as the controller code.

The third observation is similarly related to the second observation; the flows created with the Ryu Controller do not expire. In conjunction, because there is no mechanism within the HP switches to notify the controller of a link outage, there was no update to the flow table and the switch continued to push packets to a port with the disabled link. The workarounds implemented included non-permanent flow creation as well as manual removal of flow entries.

4.1.2 Test 2 Results

After shutting down the Ryu controller, the HP switches were replaced with OVS switches in the SDN environment. Afterward, the Ryu controller was restarted using the program *simple_switch.py* and began listening for incoming flow requests as was the case previously. The OVS switches were then configured and network connectivity was restored among all the hosts within the hybrid architecture.

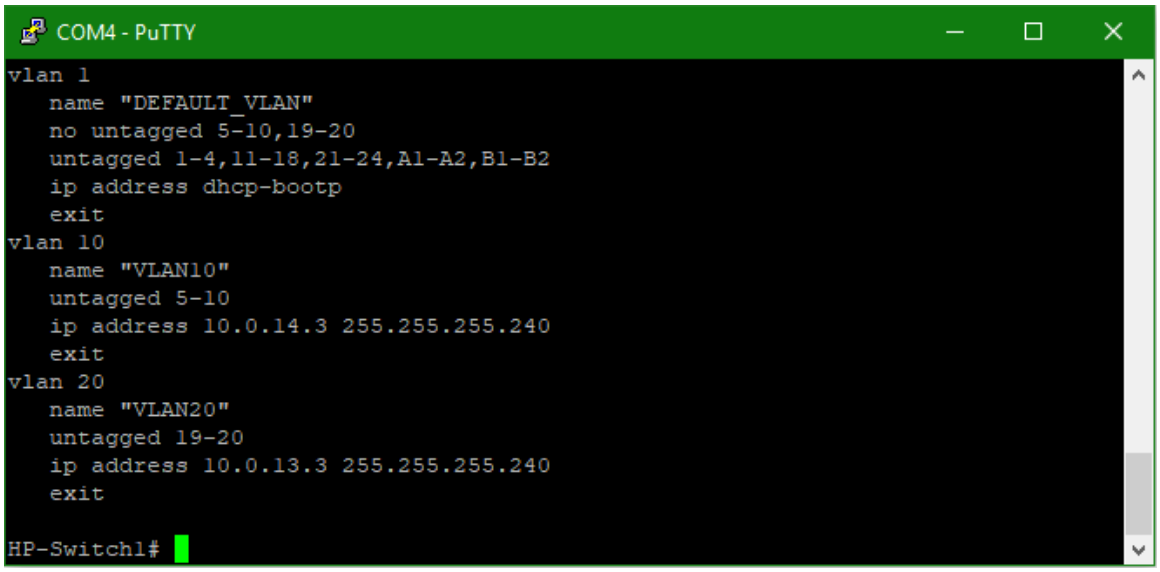

Figure 4.4. Test 1 HP Port Configuration.

The port configuration used on a HP switch.

As outlined in Figure 4.14, the OVS switch configuration displayed is slightly different than in the HP switches used. In contrast with Figure 4.3, the OVS switches displayed controller port information, if the switch was connected to the controller, and the fail mode. All of this additional information proved valuable during the testing process.

4.1.3 Test 3 Results

As with the previous test, the Ryu controller was shut down and restarted using the *simple_switch.py* program. The flow tables of the OVS switches were returned to an initial state with zero flow entries. One of the SDN hosts was transferred to the HP switch and the HP switch was configured. Network connectivity was validated among all the hosts on the network via ICMP ping commands.

4.2 Experiment 2 Results

The second experiment successfully demonstrated the ability of SDN to include the traditional network constructs of VLANs and ACLs. Each of the VLAN configurations provided the desired segmentation endstate. Packet filtering was enabled through customizeable flow

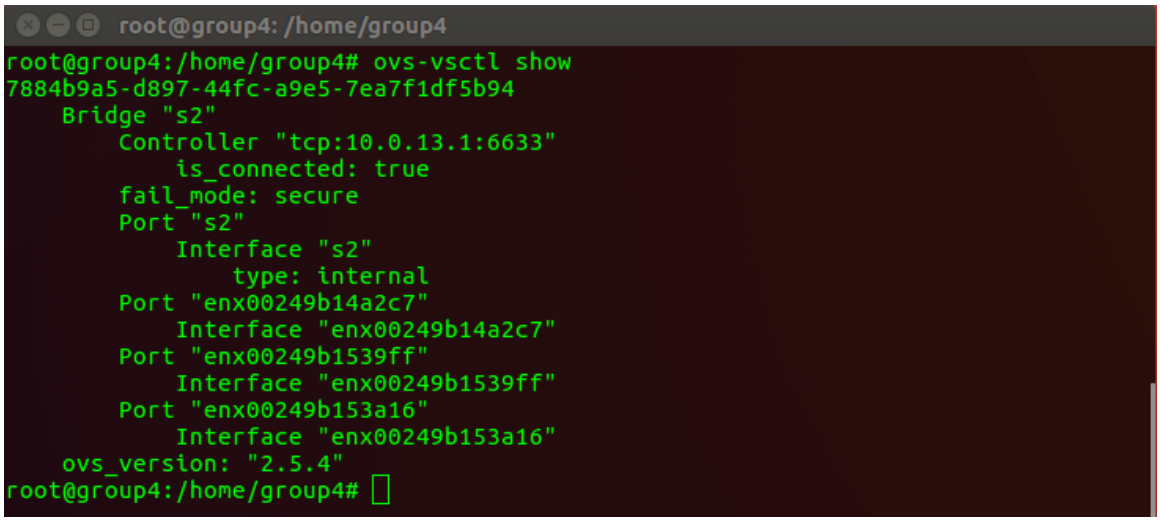

Figure 4.5. Experiment 2: Test 2 OVS Configuration.

rule creation and a traditional ACL implementation was successfully implemented in a flow table of a SDN capable switch.

4.2.1 Test 1 Results

For the first test of this experiment, the network was reverted to the original hybrid network architecture outlined in Figure 3.1 which included two SDN hosts per switch and two traditional hosts connected to the traditional switch. The traditional switch was then configured with a single VLAN, *ciscoVlan1*, and all physical Ethernet cables were moved into the *ciscoVlan1* port block. Once connectivity was validated amongst all network hosts, one of the computers on the traditional switch was moved to a port outside of the VLAN. As expected, no host within *ciscoVlan1* was able to communicate with the computer outside of *ciscoVlan1*.

4.2.2 Test 2 Results

In this test, the traditional switch remained configured as before with the SDN switches and all network hosts connected within the same VLAN, *ciscoVlan1*. Then, controller was removed from the network. Afterward, the OVS switches were configured to function in standalone mode. Lastly, two VLANs were created on each OVS switch as depicted in Figure 3.4 with one SDN host assigned to each OVS VLAN. Figure 4.6 depicts the updated

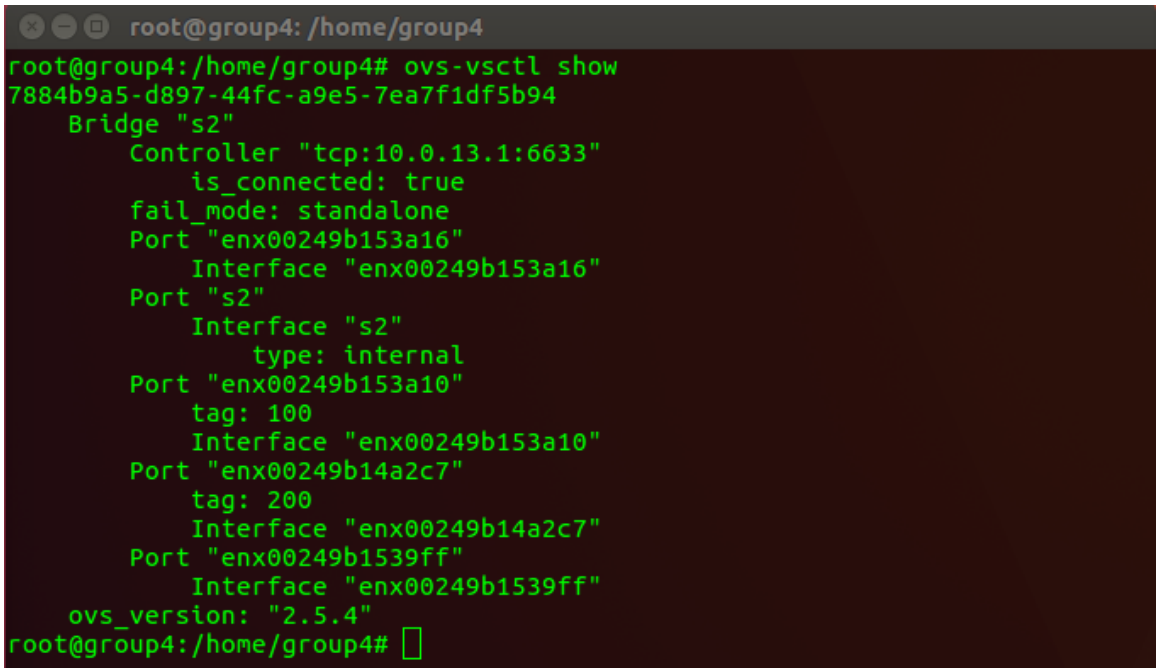

Figure 4.6. Experiment 2, Test 2 updated OVS configuration

configuration used for each of the SDN switches which now include the VLAN, or tag, information.

Once the controller was removed from the topology, loss of connectivity was visible across all of the SDN nodes. After one noticeable timed out request ping request, the hosts on the same VLAN then resumed communication with each other while the other two separated nodes were now unreachable. Lastly, the two remaining hosts connected to the traditional router were not included either VLAN with the SDN hosts. As a result, while they were able to still communicate with each other, they were properly segmented from both SDN VLANs and no network traffic was visible from either traditionally networked end device.

Because the controller was left in its current default configuration state, it was necessary to remove it from the SDN network; otherwise, the SDN controller would negotiate between the VLANs, much like a traditional router would, and traffic would still flow amongst all the hosts on the network. This was observed in both the Ryu Controller as well as the ONOS Controller. While the OVS switches were purposefully configured in secure mode to ensure traffic seen on each SDN host was only OF related, it was necessary to configure the switches in standalone mode. In this way, the OVS switch will set up flows directly

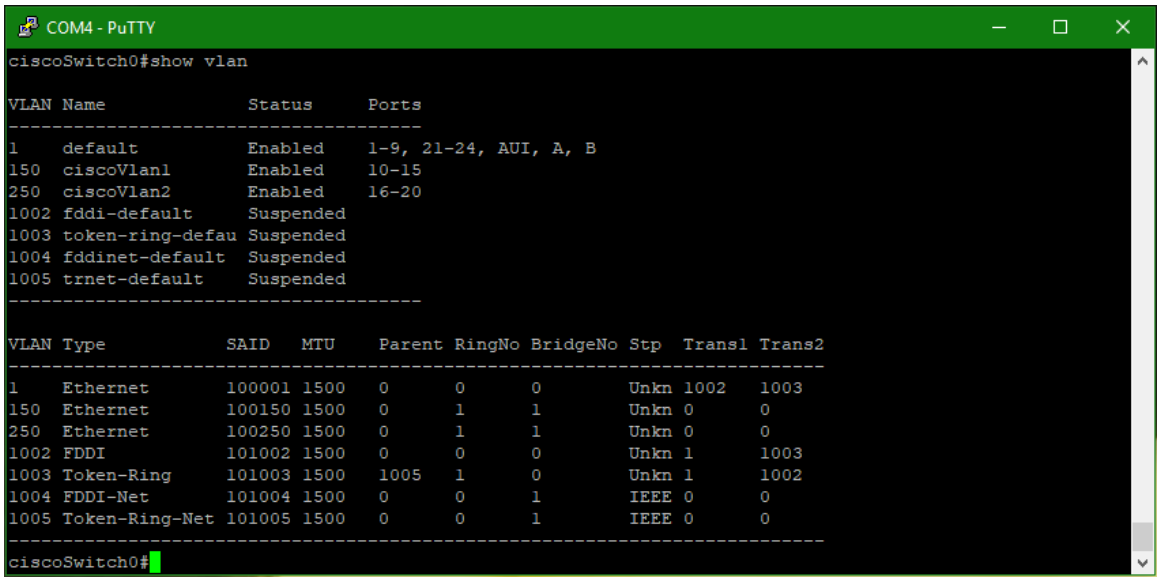

Figure 4.7. Experiment 2, Test 2 Updated Cisco Switch VLAN Configuration.

after three attempts to connect to a SDN controller fails, much in the same manner that a traditional switch would negotiate connectivity between hosts within the same VLAN [29]. Of note, as the OVS manual further states, this is the default functionality built into the OVS software and will automatically discontinue this standalone mode behavior once the controller is again available. These observations were witnessed during the course of the experimentation.

4.2.3 Test 3 Results

For the third test, a hybrid VLAN implementation was constructed. As outlined in Figure 3.5, traditional hosts were included in the VLAN configuration. In addition to the configuration changes from the second test, an additional VLAN, *ciscoVlan2*, was added to the traditional switch as shown in Figure 4.7. After moving a traditional host as well as one SDN switch into ciscoVlan2, the network was properly segmented and traffic was only able to pass amongst the devices associated with each respective VLAN.

4.2.4 Test 4 Results

In the fourth test, the network was reverted back to the topology from Experiment 1, Test 2. Connectivity was established across the network. Continous ICMP traffic was established

to all the end hosts on the network from host 10.0.14.7. A rule to filter packets in the flow table of OVS switch *s1* was installed with the following command:

```
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 priority
    ↪ =65535,hard_timeout=25,icmp,actions=drop
```

Immediately, traffic was unable to reach any other end host from host 10.0.14.7. After the 25 second timer expired, the flow rule was removed and ICMP traffic was re-established to all the end hosts on the network.

4.2.5 Test 5 Results

With the existing topology from the previous test and host 10.0.14.7 continuing to reach all hosts throughout the network via ICMP ping commands, the controller was removed from the network. Afterward, the flow table in OVS switch *s1* was delete to completely clean the table of any permanent flow entries remaining. This resulted in all connectivity lost to any host within the network. An ACL was implemented at the layer 2 level to only allow connectivity between host 10.0.14.7 and host 10.0.14.8. The following three lines of code were used:

```
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=500,in_port=2,dl_src=18:66:da:3b:b6:13,dl_dst
    ↪ =18:66:da:3b:b6:09,actions=output:1
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=500,in_port=1,dl_src=18:66:da:3b:b6:09,dl_dst
    ↪ =18:66:da:3b:b6:13,actions=output:2
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=600,arp,actions=normal
```

Once the flow table in *s1* included the aforementioned commands, connectivity resumed between host 10.0.14.7 and host 10.0.14.9. All other host traffic was properly filtered.

While packet filtering is normally administered at the layer 3 level of the network stack in a tactical network, SDN is able to implement filtering at multiple levels. While this test focused on filtering at the layer 2 level, ACLs can be administered in other layers. Appendix A.3 outlines other levels of matching validated during the course of this experiment.

4.3 Experiment 3 Results

The third experiment successfully revealed the potential of SDN to improve aspects of network configuration and network management currently underutilized within Marine Corps tactical network installation, operation, and maintenance phases. Automation via script implementation remotely configured network devices throughout the hybrid architecture. A network observation capability was immediately available via the current functionality residing within the SDN controller. REST API functionality was injected into an included SDN controller program to demonstrate a current ability of SDN to collect network statistics to inform network analysis as well as conduct configuration change in SDN topology using a less intensive, non-scripting approach.

4.3.1 Test 1 Results

With the topology from Figure 3.1 in Chapter 3 in place and all connectivity amongst all the devices validated, each of the network devices to include the router, traditional switch, the controller, and HP switches were erased of all their configuration files. No additional configuration of the traditional switch was required as the Cisco switch in its default mode enabled layer 2 connectivity across the network devices it connected. To run the script from a configuration laptop, each of the network devices were basically configured to enable a remote session. After configuring one port for each of the devices and applying password information necessary to remotely access the Cisco router. The configuration script was run. After conclusion of the configuration script, connectivity amongst all the hosts on the network was validated via ICMP ping commands. Figure 4.8 provides an example of the completed configuration script.

The Python script used ran the configuration via a list of parallel threads. If the controller was manually initialized, each of the threads completed their independent runs and the script completed successfully. However, to automate the controller initialization as a part of the automation, an artificial timer was required between the thread executing the controller configuration and the rest of the threads configuring the SDN devices. This ensured enough time for the controller to be initialized and listening for SDN traffic prior to the HP switches being configured. Without the artificial timer, the threads running the HP configurations hung as the switch configurations relied on a running SDN controller to complete the OF network install. Lastly, the script output listed all the configurations used for each of the

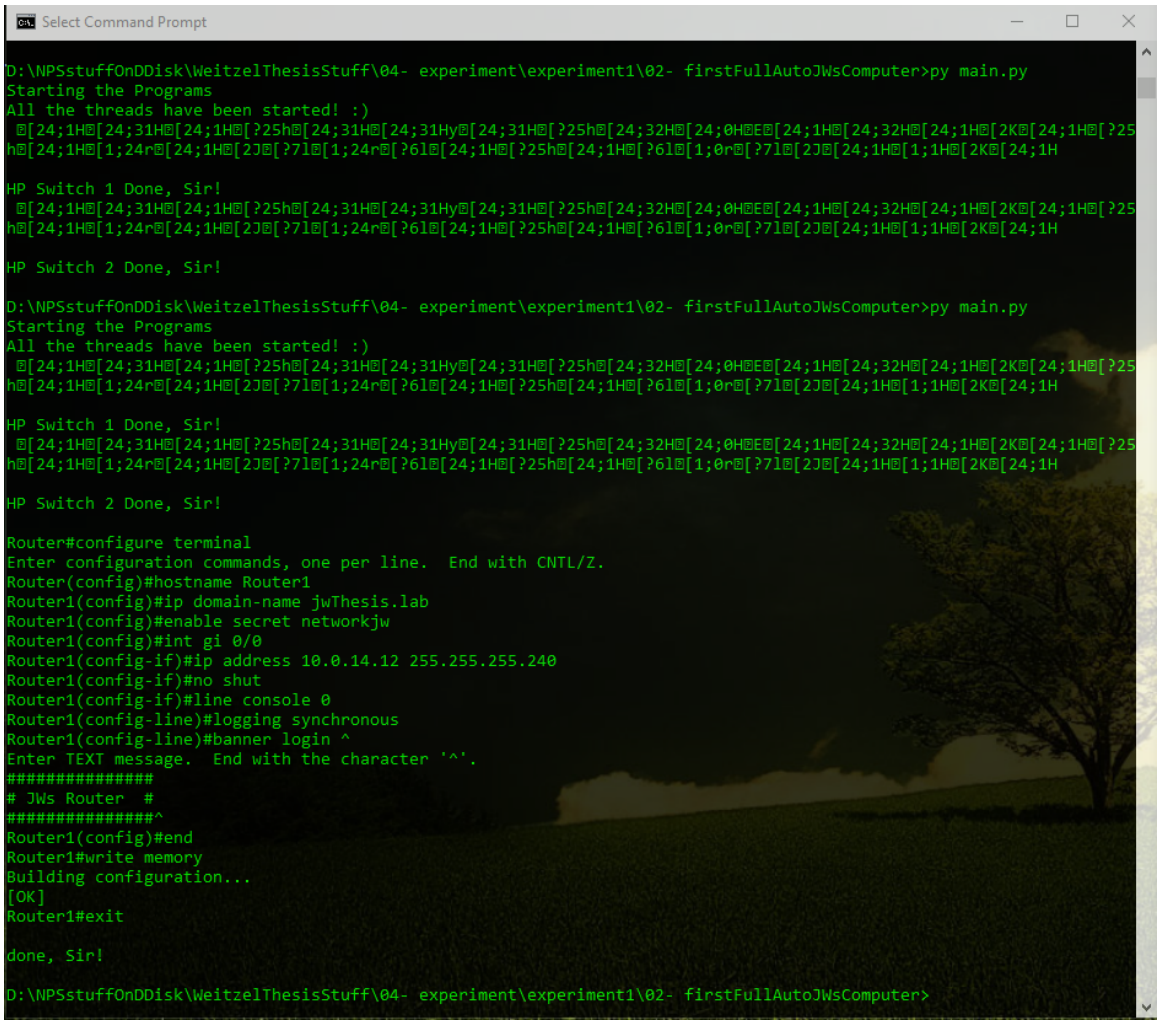

Figure 4.8. Experiment 3, Test 1 Automated Script Output.

devices. While the Cisco output was readable, the output from the HP devices was not readable. As seen in Figure 4.9, the automation of the controller yielded log information in its output.

4.3.2 Test 2 Results

The topology from Figure 3.1 also served as the foundation for this test; however, the HP switches were replaced with correctly configured OVS switches. The Ryu controller was shutdown and then re-initialized to run both *gui_topology.py* and *simple_switch.py* with the following command:

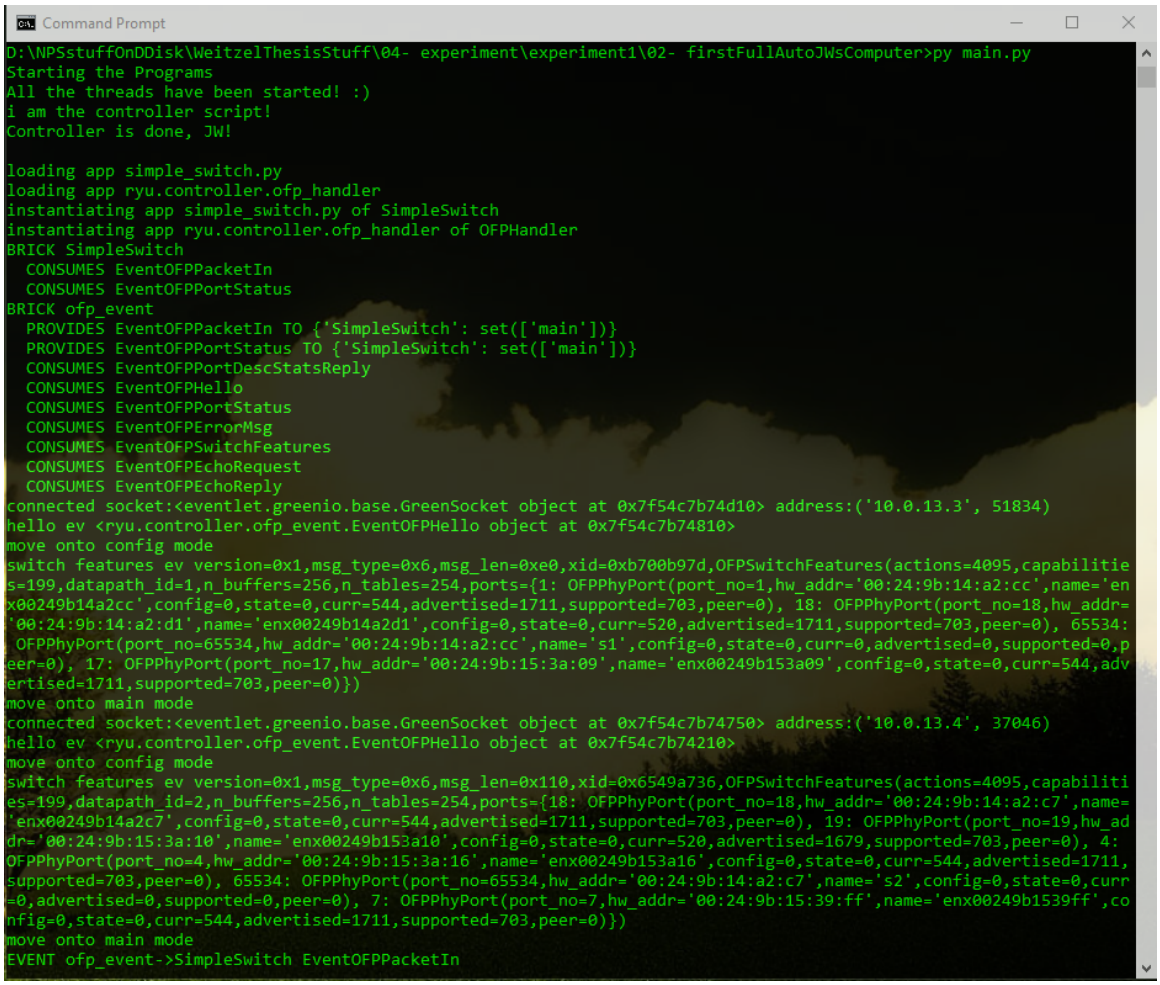

Figure 4.9. Experiment 3, Test 1 Automated Controller Script Output.

```
pc@controllerjw:/usr/local/lib/python2.7/dist-packages/ryu/app$
    ↪ ryu run gui_topology/gui_topology.py simple_switch.py --
    ↪ verbose --observe-links
```

Then, a web browser was opened to http://10.0.13.1:8080 and the Ryu Topology Viewer displayed the connected OVS switches as seen in Figure 4.10. To contrast the differences in the controller GUIs, the Ryu controller was then shutdown and the ONOS controller was brought online via the following command:

```
pc@controllerjw:~/onos$ buck run onos-local
```

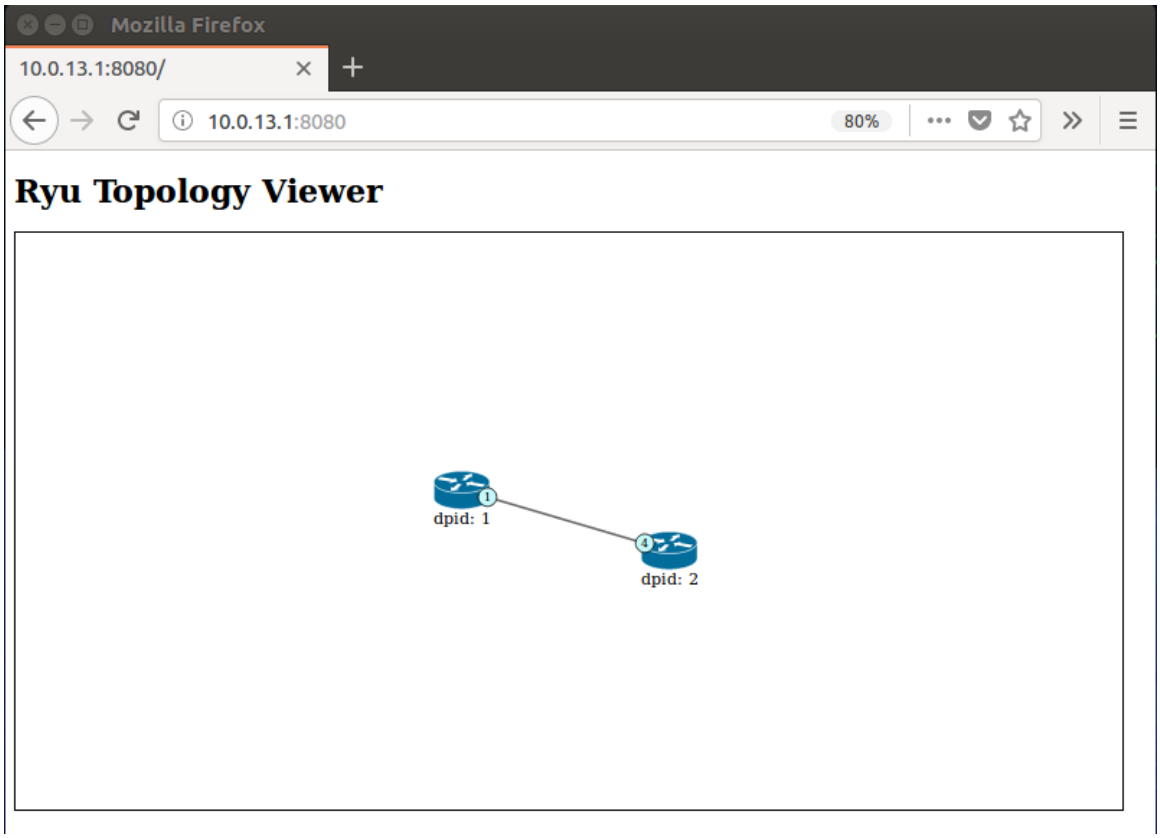

Figure 4.10. Experiment 3, Test 2 Ryu Topology Viewer.

Subsequently, a web browser was opened to http://10.0.13.1:8181/onos/ui and after logging into ONOS, the topology as seen in Figure 4.11 was observed.

The Ryu controller required an internet connection before it would properly display the SDN network. In contrast, the ONOS controller was able to bring up the SDN topology without an internet connection.

The stark contrast between the two viewers is readily apparent. While the interface provided with Ryu is simple in nature, it provided a basic accurate depiction of the SDN topology. The interface of ONOS is much more robust with a myriad of configuration preferences available. It provided a more comprehensive network metric as compared to the Ryu interface; however, it incorrectly listed the Cisco router with the IP of 10.0.14.12 as a network host and not as a traditional router.

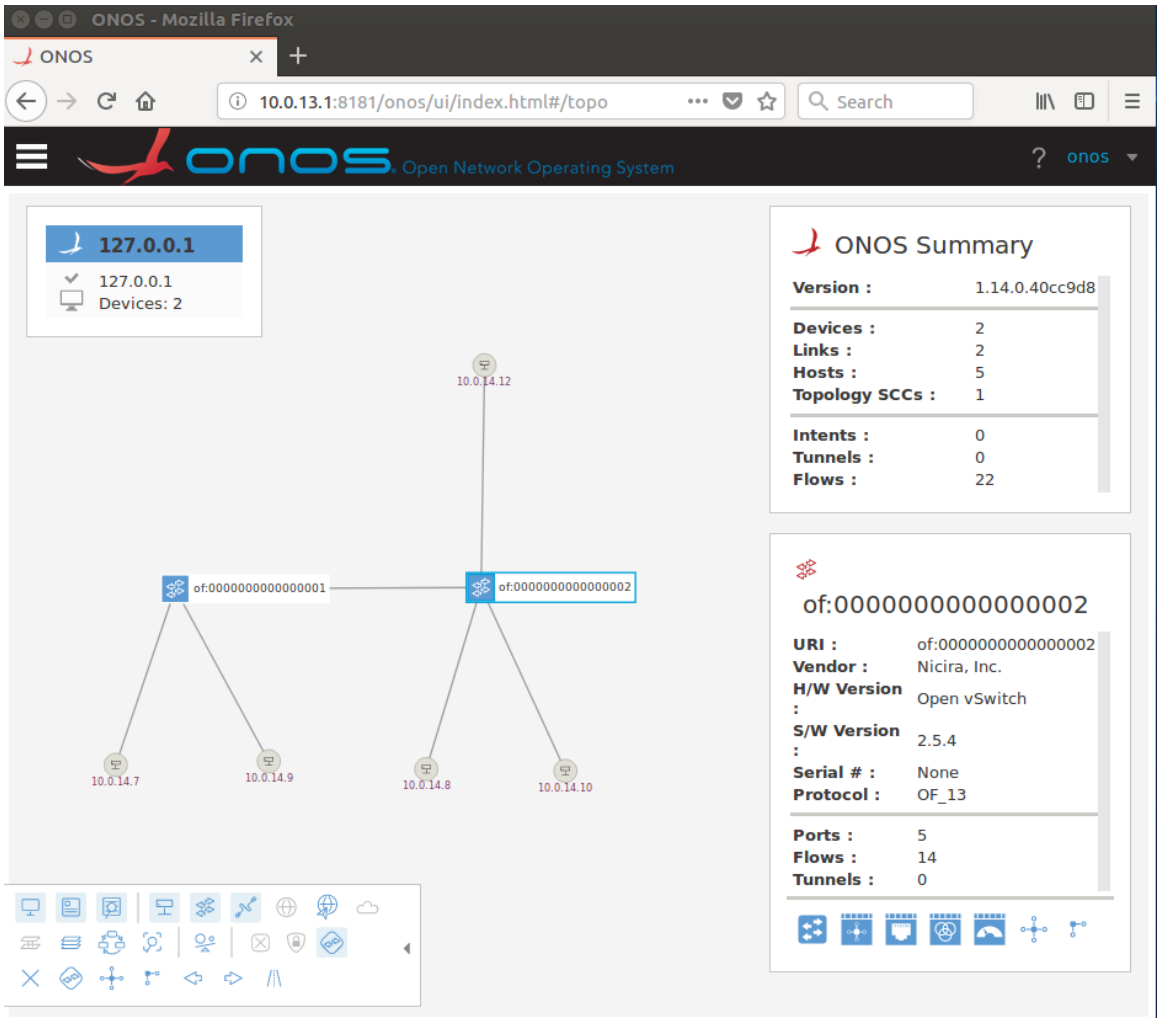

Figure 4.11. Experiment 3, Test 2 ONOS Topology UI.

4.3.3 Test 3 Results

The first part of this test gathered SDN statistics. With the existing topology from Figure 3.1, the Ryu Controller was initialized with the following command to enable REST API functionality:

```
pc@controllerjw:/usr/local/lib/python2.7/dist-packages/ryu/app$
    ↪ ryu run simple_switch.py ofctl_rest.py
```

With a new terminal open from the Ryu controller, the following command provided a list of SDN switches online:

```
pc@controllerjw:~$ curl -X GET http://10.0.13.1:8080/stats/
   ↪ switches
```

An additional GET command was used to gather flow data from OVS switch *s1*:

```
pc@controllerjw:~$ curl -X GET http://10.0.13.1:8080/stats/flow/1
```

After the validation of the first portion, the next part of the test would modify the flow table of OVS switch *s1*. Specifically, flows would be removed via curl commands from the controller. After analyzing the 10 flows in the flow table, match criteria against the *in_port* value was decided on and curl commands were implemented on flows matching the *in_port* values of 17 and 18. Afterward, the flow table was again observed which showed the removal of all flows matching the previously mentioned criteria. The command below was the curl command to remove flow entries matching *in_port* 17:

```
curl -X POST -d '{
    "dpid": 1,
    "table_id": 0,
    "priority": 1,
    "match":{
        "in_port":17,
    },
    "actions":[
        {
            "type":"OUTPUT",
            "port": 2
        }
    ]
 }' http://localhost:8080/stats/flowentry/delete
```

As seen in Figure 4.12 and Figure 4.13, each of the curl commands were accepted and the flow table was modified.

While merely a fraction of the capability already resident within the Ryu REST API was tested [30], both elements of test three validated an alternative to configuration not currently exercised within tactical communications networks.

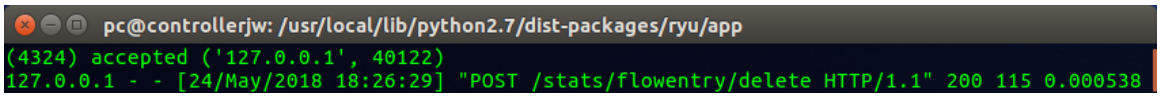

Figure 4.12. Experiment 3, Test 3 Ryu POST Acknowledgment.

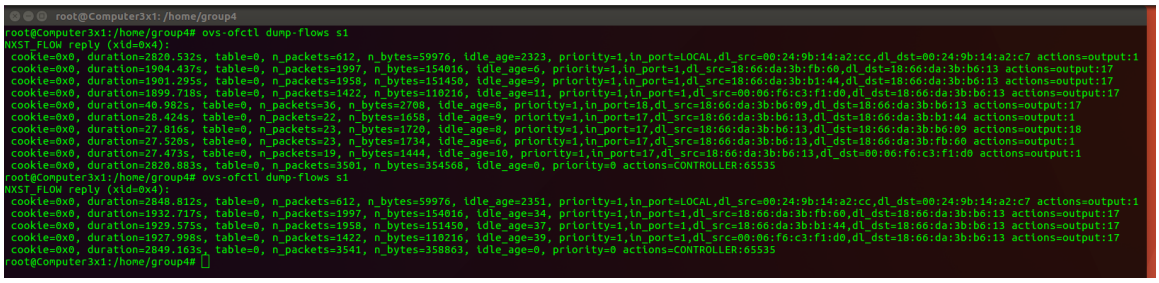

Figure 4.13. Experiment 3, Test 3 Modified Flow Table for Switch *s1*.

4.4 Summary

Throughout all the experiments the primary controller used was the Ryu controller. In addition to the *simple_switch.py* program included in the Ryu installation, another program *simple_monitor_13.py* was also successfully used throughout the experiment. While *simple_switch.py* provides all the required functionality required to enable a SDN network complete with basic log information, *simple_monitor_13.py* captured more network metrics that would be of value to communications Marines as shown in Figure 4.14. In addition to the Ryu controller, the ONOS controller was implemented throughout the experiments as well. ONOS provided a much more comprehensive log file, a robust CLI into the controller, and a much more full-featured GUI topology. However, this functionality makes ONOS much more complex than Ryu. Installation of the ONOS controller proved much more challenging, and access to the source code was obfuscated.

The results of the experiments serve as substantive proof in the foundational focus areas that SDN has the ability to impact the tactical networking landscape in a very positive and demonstrable way. This research outlined the network fundamentals necessary to propel this technology into view of the top leadership within the Marine Corps. However, further efforts to validate SDN technology in additional aspects will provide a more comprehensive set of data to empower the decision-makers of the Corps to confidently pursue a technology that could revolutionize the networking community.

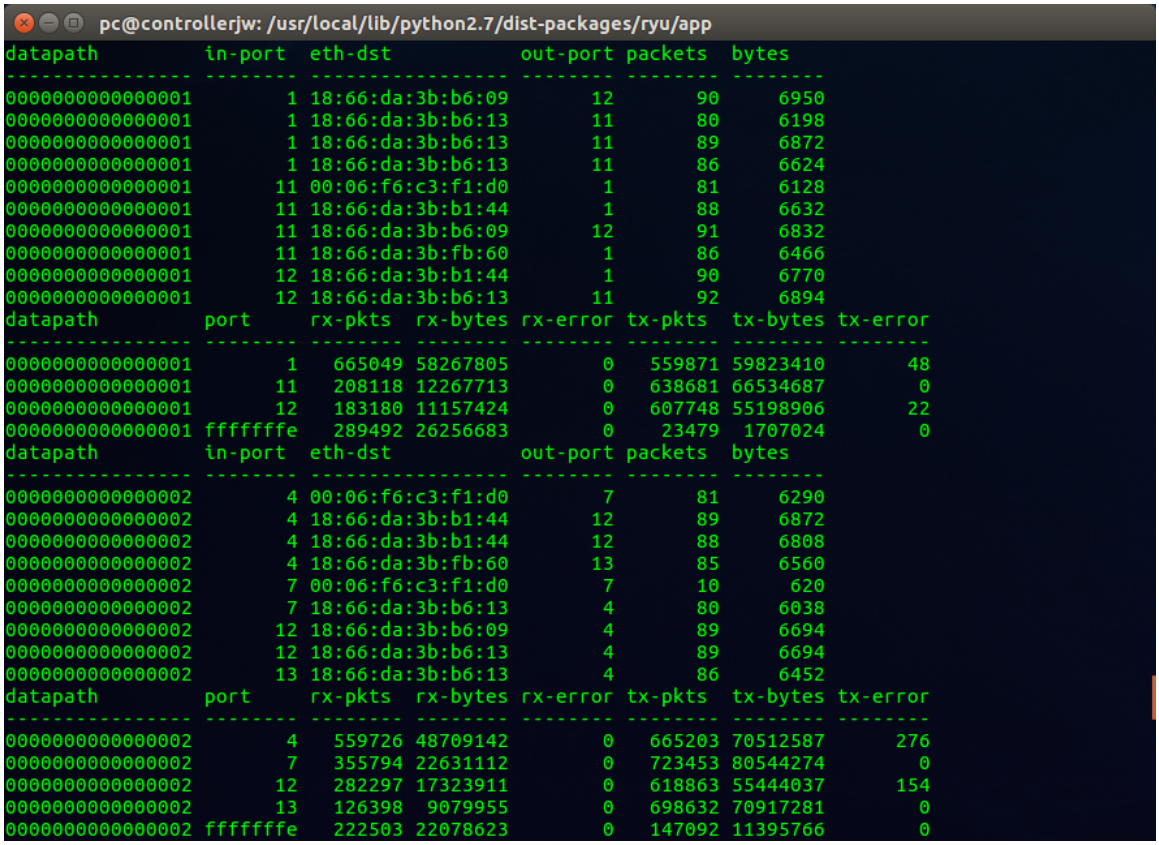

Figure 4.14. Ryu Controller Running *simple_monitor_13.py*.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions and Future Work

An investigation ensued into the innovation of tactical networks through a SDN approach with three main goals. The first goal worked toward an automated, hybrid solution in a valid tactical network setup using both traditional and SDN equipment. The second goal was focused on ensuring current network capability could be maintained with an implementation of a SDN hybrid network. The final goal was to assess the innovation potential of SDN in a tactical network environment.

The experimentation performed was simultaneously focused on achieving these three goals. The introduction of SDN into a traditional network via a typical use case such as a Jump or Forward COC network package was successfully automated. Additionally, SDN maintained capabilities critical in traditional tactical networks to include VLAN and ACL implementation. Lastly, the successes achieved within the experimentation revealed the potential of SDN technology to enable innovation in tactical Marine Corps networks.

5.1 Conclusions

Even within the focused experimentation, the empirical data collected validated benefits cited in the literature previously discussed in Chapter 2. Furthermore, the results showed SDN as a promising technology capable of achieving all six characteristics outlined in Marine Corps Warfighting Publication (MCWP) 6-22 to include reliability, security, timeliness, flexibility, interoperablity, and survivability [5]. While the conclusions of SDN from the work conducted are at times interleaving, distinct points were observed and cataloged.

- Operationally Functional. SDN works. The test bed was purposefully set up to remove SDN from an emulated environment and validate the technology in a more practical setting.
- Hybrid Network Interoperability. The success of the first experiment demonstrated an ability to amalgamate SDN technology with a traditional network.
- Automation Capable. Automation was an essential part of this work and was demonstrated throughout each of the three experiments performed.

- Maintained Capability. The second and third experiments both focused on typical configurations required on Marine Corps networks. Both VLANs and ACLs are able to be configured in an SDN environment.
- Backwards Compatibility. As observed in the first and second experiments, both the SDN capable HP switches as well as the OVSs were able to function as traditional layer-2 devices in the absence of an SDN controller.
- Flexible Network Configuration. A customized controller program in the third experiment created custom flows of OF network packets in a demonstration of the network configuration potential that SDN provides.
- Network Scalability. Basic computers were able to be purposed as different network devices within an SDN environment. The controller in the first experiment was repurposed to run as an OVS in the second experiment. Similarly, a host in the first experiment as repurposed to run as an OVS in the second experiment.
- Centralized Network Management. The third experiment enabled REST API implementation which served as an alternate avenue for SDN programmability.
- Network Awareness. In the first experiment, a topology of the network was a built in function of the Ryu Controller.
- Standard Protocol Implementation. While the second experiment highlighted a challenge to trunk layer-2 traffic between the Cisco switch and the OVS, this was the result of a conflict in trunking protocol, not in a capability gap of either of the two devices.

These conclusions were drawn from the recorded observations and substantiate the continued potential of SDN in tactical Marine Corps communications networks. The results demonstrated a technology that is able to both preserve all the current network functionality required in a tactical communications scenario while providing opportunity to enhance that same network beyond the current paradigm.

5.2 Limitations

In addition to the conclusions previously presented, some limitations were also cataloged. Some elements identify features that were not available in the SDN software utilized. A few aspects were outside the planned scope of this work.

- IP Addressing. A minimum requirement prior to enabling remote configuration was

the manual configuration of IP address assignments for all hosts on the network prior to configuration scripts being executed.

- Port Availability. Dedicated network switch hardware typically contains a large quantity of Ethernet ports to enable network connectivity. The basic computers used had one Ethernet port and required universal serial bus (USB)-to-Ethernet adapters to enable Ethernet port connectivity as required.
- Programming Proficiency. The SDN functionality enabled required competency in various aspects of computer programming and a familiarization with both HP and OVS configuration commands.
- Dynamic Flow Re-Routing. By default, the Ryu controller created flows that did not expire. While flow timeout values could be adjusted, no dynamic capability or protocol was implemented to enable validation of created flows or request flow updates to collapse stagnate, invalid flow entries.
- IPv4. No Internet Protocol version 6 (IPv6) functionality was tested in any of the experimentation.
- GUI. All configurations and network functionality of the SDN software was executed in terminal windows. While the Ryu Controller does not provide a lot of immediate GUI functionality, other SDN controllers, such as the ONOS controller, have much more developed, interactive GUI applications.

5.3 Recommendations for Future Work

As an exciting technology, SDN provides continued opportunities for further research to demonstrate its applicability as a Marine Corps tactical network solution. The recommendations for future work into SDN toward this end state include the following:

- Incorporation of SDN Technology into a Communications Exercise. Communications units throughout the Marine Corps conduct numerous communications-only field events throughout the calendar year. Validation of SDN in one such exercise would further legitimize the technology while providing an opportunity to gain insights from Marine operators employing SDN at the tactical level.
- Route Control Development. Enabling dynamic re-routing of flows that are no longer usable or stagnate is an important consideration. In addition, route control implementations that can utilize all configured network paths, to include adaptive networking

wideband waveform (ANW2) radio links will maximize available bandwidth and ensure more robust and resilient C2 architecture schemes.

- SDN Vulnerability Assessment and Risk Mitigation. There is concern in the ability of SDN to serve as a secure network solution. An exploration into the network vulnerabilities existent within SDN, the mitigation available, as well as any potential to improve a network security posture would be of value.
- Determine SDN Performance Metrics. While the primary constraint of tactical network throughput and delay often reside at the radio or satellite terminal, the ability of SDN performance to meet or even exceed that of a traditional network in other aspects could demonstrate its potential as a valid networking solution in the garrison environment. Of particular interest would be the comparison of traffic overhead in an SDN network to the overhead in a traditional network architecture.
- Only one controller was used in this work. Other controllers may prove more suitable for the tactical network needs of the Marine Corps and could be explored. Additionally, the ability to implement multiple controllers on an SDN network could be explored to enhance fault tolerant properties of a network or even sub-divide network responsbilities or even segment the network further, much like VLANs in a switched architecture.

5.4 Final Thought

In short, this work highlights SDN as a near-term, relevant technology for the Marine Corps. Because SDN adoption will yield the innovative spark long overdue for networks at the tactical edge, this technology must make it into the hands of communications Marines at the tactical level. Enabled with SDN, the Marines will undoubtedly develop creative applications and profound solutions to enhance the tactical network in enabling C2 for the commander.

APPENDIX A:
Learning SDN

The learning process for SDN can be a straightforward endeavor. Prior knowledge of traditional networking is not necessarily required; however, it is needed especially if the hybridization portions of this work are being replicated or improved.

A.1 Learning SDN

This work recommends a layered approach to learning SDN. Initially, network fundamentals are abstracted away via emulation so that core SDN concepts are grasped. Next, a remote controller is then included alongside the emulation environment to provide a more realistic architecture. Then, once validation of a controller occurs, VMs with OVS software serve as an excellent break-away from the emulation environment to bring even more realism to the SDN. Finally, after a successful implementation of an SDN controller and VMs, a physical topology completely removes any of the abstractness which results in a real, functioning SDN architecture.

A.1.1 Mininet

A virtual network environment originally that Lantz and Heller originally created [31]. Getting started with Mininet is as simple as accessing the webpage http://mininet.org/download/ and following the listed instruction set. After working through the installation and Mininet tutorials, the next step is to work with SDN controllers outside of the mininet environment.

A.1.2 Remote SDN Controllers

Many controllers exist which can be used to further SDN learning; however, the recommendation in this work is that the Ryu controller should serve as the first controller. Accessing the webpage https://osrg.github.io/ryu/ is the first step in getting started. Of note, this controller is based in the Python programming language and will require additional dependencies in Python that what is provided within a typical Linux OS distribution installation. A tutorial link with the Ryu controller is also available in the previously mentioned link.

A.2 OVS Emulation

Once the fundamentals and concepts of a remote SDN controller coupled with different Mininet topologies have are completed, a simple topology of one controller and two OVS switches, each on a separate VM is the last step before a real test bed setup. The webpage http://docs.openvswitch.org/en/latest/intro/ provides links for installation as well as follow-on tutorials.

A.3 Flow Rules: A Layered Approach

Flow rules can match on different levels of the protocol stack. This section will provide four different examples from layer 1 through layer 4. The Open System Interconnection (OSI) model is one of the two prevalent layered network models typically referenced in networking theory. This work will follow the OSI model.

To match on layer 1, the physical layer, the following OVS configuration example is provided:

```
root@Computer3x1:/home/group4# ovs-ofctl add-flow s1 table=0,
    ↪ priority=7,in_port=1,actions=output:LOCAL
root@Computer3x1:/home/group4# ovs-ofctl add-flow s1 table=0,
    ↪ priority=13,in_port=LOCAL,actions=output:1
```

To match on layer 2, the data link layer, the following OVS configuration example is provided:

```
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=7,in_port=2,dl_src=18:66:da:3b:b6:13,dl_dst=18:66:
    ↪ da:3b:b6:09,actions=output:1
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=7,in_port=1,dl_src=18:66:da:3b:b6:09,dl_dst=18:66:
    ↪ da:3b:b6:13,actions=output:2
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=13,arp,actions=normal
```

To match on layer 3, the network layer, the following OVS configuration example is provided:

```
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=7,dl_type=0x800,nw_src=10.0.4.0/24,nw_dst
    ↪ =10.0.14.0/24,actions=normal
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=13,dl_type=0x806,nw_dst=10.0.14.0/24,actions=flood
```

To match on layer 4, the transport layer, the following OVS configuration example is provided:

```
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=7,ip,tcp,nw_protoc=23,nw_src=10.0.4.0/24,nw_dst
    ↪ =10.0.14.0/24,actions=normal
root@Computer3x1:/home/group4# ovs-vsctl add-flow s1 table=0,
    ↪ priority=13,arp,actions=normal
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
Configurations Used

Each section provides the bulk of the configuration files used for this work. Some variation may be necessary for each of the specific tests, but the essence of each of the files are provided in the following sections.

B.1 Experiment 1 Configurations

The router configuration file is programmed as follows:

```
enable
hostname RouterTemplate2
ip domain-name jwThesis.lab
enable secret ciscoPassword
ing gi 0/0
ip address 10.0.13.13 255.255.255.240
int gi 0/1
ip address 10.0.14.13 255.255.255.240
no shut
line console 0
logging synchronous
password ciscoPassword
login
line vty 0
password ciscoPassword
login
banner login ^
###############
# JWs Router  #
###############
end
write memory
exit
```

The HP switch configuration file is programmed as follows:

```
configure
vlan 20
ip address 10.0.13.8 255.255.255.240
untag 19-20
vlan 10
untag 5-10
ip address 10.0.13.8 255.255.255240
openflow
controller-id 1 ip 10.0.13.1 controller-interface vlan 20
instance ofnetwork
member vlan 10
controller-id 1
enable
exit
enable
exit
```

B.1.1 OVS Configuration Template

The OVS configuration file is programmed as follows:

```
ovs-vsctl add-br s1
ovs-vsctl add-port enp0s8
ovs-vsctl set-controller s1 tcp:192.168.154.129:6633
ovs-vsctl set-fail-mode s1 secure
sudo ip addr flush dev eth0
sudo ip addr add 192.168.154.129/24 dev s1
sudo ip link set s1 up
```

B.2 Experiment 2 Configurations

The OVS configuration file to tag a port is as follows:

```
ovs-vsctl add-port enp0s8 tag=100
```

B.3 Experiment 3 Configurations

The following file is the main Python program:

```
"""-----------------------------------------
# Project      : Computer Science Thesis Project
# File Name    : autoScriptCiscoFinal.py
# Class        : Thesis Project
# Date         : 03/28/2018
# Version      : v1.0
# Environment  : Wing IDE 101 (Version 5.1) with Python 2.7
# Copyright    : This work was done by Major John Weitzel
#                and is in the public domain.
# Description  : This program automates programming for a hybrid
   ↪ network.
#
# Limitations :
#     1. All physical connections must be in place (console
   ↪ cable as well
#         as ethernet connectivity from computer with program to
   ↪ network device)
#     2. Start with a clean slate.
#     3. Basically configuration prior to automating
   ↪ configuration.
#     4. Validate connectivity to the router via the ping
   ↪ command.
#     5. Execute the following program...
#-----------------------------------------"""

from multiprocessing import Process
from controller import *
from hp1File import *
from hp2File import *
from routerFile import *

def controllerFunction():
    controller()
```

```
def hpSwitchFunction1():
    hp1()

def hpSwitchFunction2():
    hp2()

def routerFunction():
    router()

def main():
    print "Starting the Programs"

    p1 = Process(target = controllerFunction)
    p1.start()

    time.sleep(30) #put a delay on the script to give the
       ↪ controller time to get ready for switches

    p2 = Process(target = hpSwitchFunction1)
    p2.start()
    p3 = Process(target = hpSwitchFunction2)
    p3.start()

    p4 = Process(target = routerFunction)
    p4.start()

    print "All the threads have been started! :)"

if __name__ == "__main__":
    main()
```

The following file is the controller configuration program:

```
import getpass
import sys
```

```
import telnetlib
import time

def controller():

    print "i am the controller script!"

    HOST = "10.0.13.1" #destination middlebox to configure

    #user = raw_input("Enter your username: ") #for this computer
       ↪  it is:
    user = "pc"
    #password = getpass.getpass() #for this computer it is:
    password = "password"

    tn = telnetlib.Telnet(HOST)

    tn.read_until("controllerjw login: ") #specific prompt for
       ↪ this computer
    tn.write(user + "\n")

    tn.read_until("Password: ") #specific prompt for this
       ↪ computer
    tn.write(password + "\n")

    """
    This section is where we will create the script to be run in
       ↪ Ubuntu machine
    """
    tn.write("cd Desktop\n") #move to Desktop... create file here
       ↪  and run from here
    tn.write("touch newControllerProgram\n") #create new file
    tn.write("chmod 777 newControllerProgram\n") #make the new
       ↪ file an executable in bash
```

```
"""
Now we will add commands in our newly created script
"""

tn.write("echo '#! /bin/bash' >> newControllerProgram\n") #
    ↪ enables script to be run from terminal
tn.write("echo ' ' >> newControllerProgram\n") #just adding a
    ↪  spacer...
tn.write("echo cd /usr/local/lib/python2.7/dist-packages/ryu/
    ↪ app >> newControllerProgram\n")
tn.write("echo ryu run simple_switch.py --verbose >>
    ↪ newControllerProgram\n")
tn.write("echo ' ' >> newControllerProgram\n") #just adding a
    ↪  spacer...
tn.write("echo echo script done in here! now back to you JW
    ↪ ... >> newControllerProgram\n")

"""
This section will allow us to run our newly created script
"""
tn.write("/usr/bin/sudo ./newControllerProgram\n") #allows us
    ↪  to run our program as root
tn.read_until("[sudo] password for pc: ")
tn.write(password + "\n")

time.sleep(30) #put a delay on the script to give the
    ↪ controller time to negotiate with switches

tn.write("\x03") #how to write Control+C terminal command in
    ↪ Ubuntu

"""
Now that we have run the program, let's clean up the Desktop
    ↪ by removing the file
"""
```

```
#     tn.write("rm newProgram\n") #remove file that was created on
    ↪  Desktop

     print "Controller is done, JW!"

     tn.write("exit\n")

     print tn.read_all()

if __name__ == "__main__":
     main()
```

The following file is the HP switch configuration program:

```
"""-----------------------------------------
# Project      : Computer Science Thesis Project
# File Name    : autoScriptCiscoFinal.py
# Class        : Thesis Project
# Date         : 03/28/2018
# Version      : v1.0
# Environment  : Wing IDE 101 (Version 5.1) with Python 2.7
# Copyright    : This work was done by Major John Weitzel
#                and is in the public domain.
# Description  : This program automates programming for a HP
    ↪ 2920-24G Switch
#
# Limitations :
#      1.  All physical connections must be in place (console
    ↪ cable as well
#          as ethernet connectivity from computer with program to
    ↪ network device)
#      2.  Start with a clean slate:
#          (Warning: this will also delete any username and
    ↪ password info):
#          (with the switch running press both 'reset' and 'clear'
    ↪ simultaneously
```

```
#         then release 'reset' but continue to hold down 'clear'
   ↪ until the
#         Test LED begins to blink)
#     3.  Basically configuration prior to automating
   ↪ configuration:
#         (quickly hit enter twice and then once more to get into
   ↪  configuration
#         mode):
#         enable -> configure -> vlan 20 ->
#         ip address 10.0.13.8 255.255.255.240 -> untag 20 ->
   ↪ exit -> exit ->
#         exit -> exit -> (then type 'y' at the 'log out' prompt)
#     4.  Validate connectivity to the switch via the ping
   ↪ command.
#     5.  Execute the following program...
#----------------------------------------"""

import getpass
import sys
import telnetlib

HOST = "10.0.13.8"

tn = telnetlib.Telnet(HOST)

tn.write("\n") #two taps on enter will get to start screen
tn.write("\n")

tn.write("\n") #another continue is required to begin configuring

tn.write("enable\n") #sequence to enable configuration of HP
   ↪ Switch
tn.write("configure terminal\n")

tn.write("hostname HP-SwitchTemplate\n") #change default name of
```

```
    ↪ switch

tn.write("vlan 10\n") #this is the vlan that will be data plan
    ↪ for controller
tn.write("untag 5-10\n")
tn.write("ip address 10.0.14.8 255.255.255.240\n")

tn.write("vlan 20\n") #also vlan allowing telneting; stays
    ↪ outside controller influence
tn.write("untag 19-20\n")
tn.write("ip address 10.0.13.8 255.255.255.240\n")

tn.write("openflow\n") #configure the openflow portion
tn.write("controller-id 1 ip 10.0.13.1 controller-interface vlan
    ↪ 20\n")
tn.write("instance ofnetwork\n")
tn.write("member vlan 10\n")
tn.write("controller-id 1\n")
tn.write("version 1.3\n")
tn.write("enable\n")
tn.write("exit\n") #now out of openflow configuration mode
tn.write("enable\n")

tn.write("exit\n") #now out of configuration mode

tn.write("write memory\n") #save the configuration to device

tn.write("exit\n") #now out of configuration mode
tn.write("exit\n") #now out of priveleged exec mode
tn.write("exit\n") #now out of user exec mode

tn.read_until("Do you want to log out (y/n)?") #final logout
    ↪ banner
tn.write("y\n")
```

```
print tn.read_all() #maybe HP not configured to work with Python
   ↪ telnet function?

print "done, Sir!"
```

The following file is the router configuration program:

```
"""-----------------------------------------
# Project      : Computer Science Thesis Project
# File Name    : autoScriptCiscoFinal.py
# Class        : Thesis Project
# Date         : 03/28/2018
# Version      : v1.0
# Environment  : Wing IDE 101 (Version 5.1) with Python 2.7
# Copyright    : This work was done by Major John Weitzel
#                and is in the public domain.
# Description  : This program automates programming for a 2951
   ↪ Cisco Router
#
# Limitations :
#     1.  All physical connections must be in place (console
   ↪ cable as well
#         as ethernet connectivity from computer with program to
   ↪ network device)
#     2.  Start with a clean slate:
#         (Warning: this will also delete any username and
   ↪ password info from
#         User Exec [line console 0] and Privileged Exec mode [
   ↪ enable]):
#         enable -> erase startup-config (hit enter to the
   ↪ following prompt) ->
#         (power cycle the router)
#     3.  Basically configuration prior to automating
   ↪ configuration:
#         (type 'n' and press enter to the 'initial configuration
   ↪  dialog'):
```

```
#         enable -> configure terminal -> interface gi0/0 ->
#         ip address 10.0.13.13 255.255.255.240 -> no shutdown ->
#         line console 0 -> password password -> login -> line
   ↪ vty 0 ->
#         password password -> login -> exit -> exit -> write
   ↪ memory -> exit
#     4.  Validate connectivity to the router via the ping
   ↪ command.
#     5.  Execute the following program...
#-------------------------------------------"""

import getpass
import sys
import telnetlib

HOST = "10.0.13.13" #input arguments to customize behavior, ip
   ↪ scheme, passwords,
#user = raw_input("Enter your username: ") #don't use a user name
   ↪  to login, so don't require this line
password = getpass.getpass()

tn = telnetlib.Telnet(HOST)

#tn.read_until("login: ") #don't use a user name to login, so don
   ↪ 't require this line
#tn.write(user + "\n") #don't use a user name to login, so don't
   ↪ require this line

if password: #cisco requires a password for Telnet sessions this
   ↪ for User Exec mode
    tn.read_until("Password: ")
    tn.write(password + "\n")

tn.write("enable\n")
```

```
if password: #this password line is for Priveleged Exec mode (if
   ↪ required)
    tn.read_until("Password: ")
    tn.write(password + "\n")

tn.write("configure terminal\n")

tn.write("hostname RouterTemplate2\n") #provides an opportunity
   ↪ to rename router
tn.write("ip domain-name jwThesis.lab\n") #domain name
   ↪ information

tn.write("enable secret password\n") #password for privileged
   ↪ exec mode (ie enable)

tn.write("int gi 0/1\n") #set up for an additional interface
tn.write("ip address 10.0.14.13 255.255.255.240\n")
tn.write("no shut\n")

tn.write("line console 0\n") #may also want to enable a user
   ↪ login name in future
tn.write("logging synchronous\n") #enables continous CLI work
   ↪ during messaging

tn.write("banner login ^\n") #Warning banners exist on government
   ↪  systems
tn.write("################\n")
tn.write("# JWs Router  #\n")
tn.write("################^\n")

tn.write("end\n")

tn.write("write memory\n") #saves configuration made from script

tn.write("exit\n")
```

```
print tn.read_all() #provides detail(s) of the session as an
   ↪ output

print "done, Sir!"
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] Commandant of the Marine Corps, *NAVMC 3500.56A Communications (COMM) Training and Readiness (T&R) Manual*. Washington, D.C.: Department of the Navy, 2011.

[2] Commandant of the Marine Corps, *Marine Corps Order 1200.17E Military Occupational Specialties Manual (Short Title: MOS Manual)*. Washington, D.C.: Department of the Navy, 2013.

[3] Commandant of the Marine Corps, *Warfighting*. Washington, D.C.: U.S. Government Printing Office, 1997.

[4] Commandant of the Marine Corps, *Command and Control*. Washington, D.C.: U.S. Government Printing Office, 1996.

[5] Commandant of the Marine Corps, *MAGTF Communications System*. Albany, GA: Marine Corps Logistics Base, 2010.

[6] T. Garcia, "Rapid network design," M.S. thesis, Dept. Computer Science, Naval Postgraduate School, Monterey, California, 2013.

[7] The Tech Terms Computer Dictionary. (n.d.). TechTerms. [Online]. Available: https://techterms.com/. Accessed May. 7, 2018.

[8] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[9] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, 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, Jan 2015.

[10] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1955–1980, Fourthquarter 2014.

[11] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[12] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (SDN): Layers and architecture terminology," RFC 7426, Jan. 2015.

[13] OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06). (2015). Open Network Foundation. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. Accessed Dec. 3, 2017.

[14] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," RFC 3234, 2002.

[15] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Hoboken, NJ: Pearson Education, Inc., 2017, ch. 4, sec. 4.4, p. 357.

[16] M. Boucadair and C. Jacquenet, "Software-defined networking: A perspective from within a service provider environment," RFC 7149, Mar. 2014.

[17] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 27–51, Firstquarter 2015.

[18] J. Spencer, O. Worthington, R. Hancock, and E. Hepworth, "Towards a tactical software defined network," in *2016 International Conference on Military Communications and Information Systems (ICMCIS)*, May 2016, pp. 1–7.

[19] V. Nguyen and Y. Kim, "Sdn-based enterprise and campus networks: A case of vlan management," *Journal of Information Processing Systems*, vol. 12, pp. 511–524, Sep. 2016.

[20] Y.-W. E. Sung, X. Sun, S. G. Rao, G. G. Xie, and D. A. Maltz, "Towards systematic design of enterprise networks," *IEEE/ACM Trans. Netw.*, vol. 19, no. 3, pp. 695–708, June 2011. Available: http://dx.doi.org/10.1109/TNET.2010.2089640

[21] SDN & Breaking The VLAN Contract. (2014). NETWORKComputing. [Online]. Available: https://www.networkcomputing.com/networking/sdn-breaking-vlan-contract/1849833481. Accessed May. 1, 2018.

[22] Aruba: SDN tools to change wireless network models. (2013). ComputerWeekly.com. [Online]. Available: https://www.computerweekly.com/news/2240185264/Aruba-SDN-tools-to-change-wireless-network-models. Accessed May. 1, 2018.

[23] Open vSwitch Release 2.9.90. Open vSwitch. [Online]. Available: https://media.readthedocs.org/pdf/openvswitch/latest/openvswitch.pdf. Accessed May 23, 2018.

[24] Configuring Trunking Between a Catalyst 1900 and any Switch Running CatOS Software. Cisco. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/lan-switching/inter-switch-link-isl/24066-174.html. Accessed May 23, 2018.

[25] W. Isaacson, *The innovators. How a group of hackers, geniuses and geeks created the digital revolution*. London: Simon & Schuster, 2014.

[26] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," RFC 7231, 2014.

[27] M. Masse, *REST API Design Rulebook*. Sebastopol: O'Reilly Media, 2011. Available: https://cds.cern.ch/record/1416939

[28] OpenFlow Switch Specification. Open Network Foundation. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf#page=17. Accessed May 23, 2018.

[29] Open vSwitch Manual. Open vSwitch. [Online]. Available: http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt. Accessed May 23, 2018.

[30] ryu Documentation Release 4.0. (2016). ryu development team. [Online]. Available: https://media.readthedocs.org/pdf/ryu-takahashi/test/ryu-takahashi.pdf. Accessed May 24, 2018.

[31] Mininet Credits/History. (n.d.). Mininet. [Online]. Available: http://mininet.org/credits/. Accessed May 28, 2018.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California