CAPITALIZING ON THE SECURITY
POTENTIAL OF SOFTWARE DEFINED
NETWORKING BY PROVIDING A NETWORK
CONFIDENCE ASSESSMENT

by

JOSHUA ANDREW ALCORN
B.S. CS, United States Military Academy at West Point, 1999

A dissertation submitted to the Graduate Faculty of the
University of Colorado Colorado Springs
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
Department of Computer Science
2018
This dissertation for the Doctor of Philosophy degree by
Joshua Andrew Alcorn
has been approved for the
Department of Computer Science
By

C. Edward Chow, Chair
Terrance Boult
Sang-Yoon Chang
Jean Blair
Rita Doerr

Date: 30 April 2018
Alcorn, Joshua Andrew (Ph.D., Engineering)

Capitalizing on the Security Potential of Software Defined Networking by Providing a Network Confidence Assessment

Dissertation directed by Professor C. Edward Chow

ABSTRACT

The unauthorized access or theft of sensitive, personal information is becoming a weekly news item. The illegal dissemination of proprietary information to media outlets or competitors costs industry untold millions in remediation costs and losses every year. The 2013 data breach at Target, Inc. that impacted 70 million customers is estimated to cost upwards of one billion dollars. Stolen information is also being used to damage political figures and adversely influence foreign and domestic policy. The author offers techniques for better understanding the health and security of our networks. This understanding will help professionals to identify network behavior, anomalies and other latent, systematic issues in their networks.

An emerging field of research, Software Defined Networks (SDN) promises to change the landscape of traditional network topology and management. Options are limited for researchers and early adopters in need of adequate SDN testing facilities for their experiments. Industry is responding slowly with embedded support for SDN in their enterprise grade network hardware, but it is cost prohibitive for many test environments with a single SDN switch costing thousands of dollars. There are a few emerging community SDN test networks that are fantastic for testing large topologies with production grade traffic; however, there is a cost associated with membership and some
controlled experiments are difficult. A free and indispensable alternative to a dedicated hardware SDN is to use network emulation tools. The author provides a collection of simulation and small-scale testbed tools for use in future research.

These software tools provide an amazingly precise representation of physical network nodes and behavior, but are inherently limited by their aggregation with other virtual devices on the same compute node. However, for research requiring a higher precision than software emulation can provide there are few options. The author provides a portable, low-cost, reliable, repeatable solution for this research dilemma.
ACKNOWLEDGEMENTS

In great appreciation for the patience, direction, and understanding of my family, friends, colleagues, and my Advisory Committee. You provided me the time, support, and resources necessary to succeed.

A special thank you to Dr. C. Edward Chow for his help and guidance for the last several years. I came to him for advice prior to joining the PhD program when he reassured me that my undergraduate degree, although old, would provide me the necessary skills and knowledge to succeed in the program. Dr. Chow continued to mentor and educate me throughout my time including opportunities to publish research, present at trade shows, explore massive open online learning, and participate in the grant writing process.

Many thanks to my advisory committee members: Dr. Boult, Dr. Chang, Dr. Blair and Dr. Doerr. Thank you for your suggestions, guidance, and thoughtfulness. Thank you as well to Ali Langfels who helped me manage all the administrative details. Thanks also to Scott Melton who served as a research partner and friend.

I would especially like to thank my wife, Christel, and daughter, Annabelle. They sacrificed a great deal of time and probably put up with a good bit of my stress the past few years. Thank you for your encouragement from the beginning; helping me prepare for my first day as a PhD graduate student and seeing me through the last few days completing my dissertation.

I wish to dedicate this body of research to Christel; without you none of this would be possible.
# TABLE OF CONTENTS

## CHAPTER

1. **INTRODUCTION** ........................................................................................................ 1
   1.1 Introduction .......................................................................................................... 1
   1.2 Motivation ............................................................................................................ 2
   1.3 Contributions ...................................................................................................... 4
   1.4 Dissertation Outline........................................................................................... 5

2. **OVERVIEW OF SOFTWARE DEFINED NETWORKING** .................. 7
   2.1 Contributions....................................................................................................... 7
   2.2 Networking Prior to 2007 .................................................................................. 7
   2.3 SDN/OpenFlow – How It Works ........................................................................ 9

3. **SDN/OPENFLOW SECURITY ENHANCEMENT POTENTIAL** ........ 12
   3.1 Contributions....................................................................................................... 12
   3.2 Current State of the Art SDN Security .............................................................. 12
   3.3 Potential Advantages and Disadvantages of SDN ............................................ 16
   3.4 Weighing the Benefits .................................................................................... 19

4. **EXPLORATORY SDN/OPENFLOW SECURITY RESEARCH** ........ 21
   4.1 Contributions....................................................................................................... 21
   4.2 Small-Scale Network Modelling and Simulation .............................................. 22
   4.3 Defining A Large-Scale Network ....................................................................... 22
   4.4 Currently Available Testing and Attacks Tools ................................................. 24
   4.5 Selection of SDN/Open Modelling and Simulation Tool ................................... 27
   4.6 Controller Applications ................................................................................... 28
   4.7 DoS/DDoS Attack Modelling and Simulation .................................................... 29
4.8 Experiments and Evaluation ........................................................................ 31

5 FRAMEWORK FOR SDN DATA PATH CONFIDENCE ANALYSIS ........ 36
5.1 Contributions ............................................................................................ 36
5.2 The Data Path Security Problem .............................................................. 37
5.3 Existing Research on SDN Transport Analysis ........................................... 40
5.4 Network Confidence Framework ............................................................... 42
5.5 SDN Metric Evaluation and Experiments .................................................. 51
5.6 Conclusion .................................................................................................. 63

6 SDN SECURITY RESEARCH MODELS AND SIMULATION ............. 64
6.1 Contribution ................................................................................................ 64
6.2 Large-Scale Modelling and Simulation ...................................................... 64
6.3 University Testbed .................................................................................... 68
6.4 PlanetLab Research Network Investigation .............................................. 70

7 SDN ON-THE-GO PHYSICAL TESTBED ........................................... 81
7.1 Contributions ............................................................................................ 81
7.2 Introduction ................................................................................................ 82
7.3 Background and Current SDN Testbed Research ..................................... 84
7.4 SDN On-the-Go Testbed Architecture ...................................................... 87
7.5 Testbed Implementation ............................................................................ 93
7.6 Testbed Performance Evaluation ............................................................... 98
7.7 Conclusion ................................................................................................ 100

8 SDN NETWORK ANALYSIS TOOL IMPLEMENTATION ................ 102
8.1 Contributions ............................................................................................ 102
8.2 Implementation Overview .......................................................................... 102
### LIST OF TABLES

**TABLE**

4-1: Sampling of OpenFlow Controllers

5-1: SMV Rating Scale

5-2: Example of Base Group

5-3: Network Complexity Measure

5-4: Example of Environment Group

5-5: Example of Final SDN Confidence Assessment

5-6: Flow Duration Confidence Matrix

5-7: Packet Size Confidence Matrix

5-8: Combined Confidence Analysis

8-1: Configurable Impact Value

8-2: Configurable Collateral Damage Potential

8-3: Confidence Rating
## LIST OF FIGURES

**FIGURE**


2-1: Traditional Switch Structure ....................................................................................... 8

2-2: Traditional vs. SDN Structure [9] ............................................................................... 9

2-3: Example Flow Table .................................................................................................. 11

3-1: Detection Loop Operation [15] ................................................................................. 13

3-2: Packet Backtrace [18] ............................................................................................. 14

3-3: Sample Malware BotHunter Event [19] .................................................................... 15

4-1: BoNeSi and EstiNet DoS Simulation ......................................................................... 27

4-2: OpenFlow Controller Performance Comparison ..................................................... 29

4-3: Ping Flooding DoS Attack ....................................................................................... 32

4-4: Ping Flooding Mitigation using SDN/OpenFlow ...................................................... 33

4-5: EstiNet Modelling and Simulation Topology ............................................................ 35

5-1: Traditional Network Secure Access Approach ......................................................... 38

5-2: Framework for SDN Confidence Assessment ........................................................... 39

5-3: Overall Data Transfer Process ................................................................................... 43

5-4: Example SDN Topology ........................................................................................... 47

5-5: Example Impact of SMV Rating Changes ................................................................. 55

5-6: Impact of MitM on the Flow Duration Confidence Score ....................................... 62

5-7: Impact of MitM on Overall Confidence Score ......................................................... 62

6-1: Traditional Datacenter Networking [61] .................................................................. 65

6-2: Simple Hierarchical Design ....................................................................................... 66
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-3</td>
<td>Multi-Tenancy Example [64]</td>
<td>67</td>
</tr>
<tr>
<td>6-4</td>
<td>Simple Virtualized Testbed</td>
<td>69</td>
</tr>
<tr>
<td>6-5</td>
<td>Nested Hypervisors [66]</td>
<td>69</td>
</tr>
<tr>
<td>6-6</td>
<td>PlanetLab – How It Works [69]</td>
<td>72</td>
</tr>
<tr>
<td>7-1</td>
<td>SDN On-The-Go a low cost, portable, standalone SDN testbed</td>
<td>84</td>
</tr>
<tr>
<td>7-2</td>
<td>SDN OTG ODL Controller Model [16]</td>
<td>89</td>
</tr>
<tr>
<td>7-3</td>
<td>4 Switch, 2 Host Partial Mesh</td>
<td>91</td>
</tr>
<tr>
<td>7-4</td>
<td>4 Switch, 4 Host with Direct WAN Access</td>
<td>91</td>
</tr>
<tr>
<td>7-5</td>
<td>4 Switch, 4 Host Hybrid Fat-Tree with Direct WAN Access</td>
<td>92</td>
</tr>
<tr>
<td>7-6</td>
<td>SDN OTG Travel Configuration</td>
<td>94</td>
</tr>
<tr>
<td>7-7</td>
<td>Default Configuration of SDN OTG</td>
<td>95</td>
</tr>
<tr>
<td>7-8</td>
<td>Zodiac FX Switch Board</td>
<td>96</td>
</tr>
<tr>
<td>7-9</td>
<td>OTG Host Board</td>
<td>97</td>
</tr>
<tr>
<td>7-10</td>
<td>Routers with Wi-Fi</td>
<td>98</td>
</tr>
<tr>
<td>7-11</td>
<td>Mininet vs. Physical Testbed</td>
<td>100</td>
</tr>
<tr>
<td>8-1</td>
<td>Floodlight Programming Model [92]</td>
<td>103</td>
</tr>
<tr>
<td>8-2</td>
<td>SDN Controller Architecture [92]</td>
<td>104</td>
</tr>
<tr>
<td>8-3</td>
<td>Application Components [93]</td>
<td>105</td>
</tr>
<tr>
<td>8-4</td>
<td>Floodlight Controller Functionality [92]</td>
<td>109</td>
</tr>
<tr>
<td>8-5</td>
<td>Floodlight Module Development Process [92]</td>
<td>110</td>
</tr>
<tr>
<td>8-6</td>
<td>Network Analysis Workflow</td>
<td>112</td>
</tr>
<tr>
<td>8-7</td>
<td>Network Analysis Process</td>
<td>113</td>
</tr>
<tr>
<td>8-8</td>
<td>Select Source and Destination</td>
<td>113</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Introduction

The explosion of mobile devices and content, server virtualization, and advent of cloud services are among the trends driving the networking industry to reexamine traditional network architectures [1]. Denial of Service (DoS)/Distributed Denial of Service (DDoS) attacks only continue to grow in complexity and volume. This is further compounded by the fact that unauthorized access of personal or proprietary information is becoming seemingly routine. Clearly new methods of network security are needed. A promising approach is software-defined networking (SDN) via the OpenFlow protocol.

OpenFlow was designed as a new networking paradigm which allows dynamic software-based control over packet routing. With OpenFlow and SDN in place, switches and routers will no longer be isolated, expensive, and proprietary hardware. OpenFlow separates the control plane from the data plane and connects them through a secure open interface, thus, allowing network administrators the ability to modify the control plane remotely via software [2].

A better understanding the data paths within our networks can help us identify network anomalies that indicate malicious activity and other latent, systemic issues. SDN enables the collection of network operational and configuration data that are not readily available, if available at all, from traditional networks. By accumulating and analyzing network metrics along with operational and configuration data my research can assess the dependability and security of the network path or segment being used.
SDN promises to change the landscape of traditional network topology and management. Researchers and early adopters need adequate SDN testing facilities. Industry is slow in responding to the cost sensitive research environment. An alternative is to use software simulated network topology tools that arguably do not emulate real-world behavior. My solution is a series of modeling and simulation (M&S) research frameworks and the construction of a low cost, portable, standalone SDN testbed.

My research goals are to (1) assess the security potential SDN/OpenFlow provides, (2) through exploratory research develop a framework to help professionals identify network behavior, anomalies and other latent, systemic issues related to specific data transmissions, (3) create SDN environment for research testing, and (4) apply these to the implementation and evaluation of a SDN confidence analysis tool.

1.2 Motivation

Both governments and corporations continue to seek better means of attack prevention, detection, and trace-back. A brief sampling of recent DoS/DDoS attacks illustrates the volatility and threat they posed from Google and Twitter in 2009 to FBI, CIA, Sony, VISA, PayPal, Amazon, MasterCard in 2011-2012 [3].
Another troubling example is the February 2014 network time protocol (NTP) Amplification DDoS attack against CloudFlare. The attack peaked just shy of 400Gbps utilizing 4,529 servers on 1,298 networks [5]. Not only did this attack affect CloudFlare, but due to its size and volume, it also impacted servers and networks with close routing proximity. Network attacks, including the 2016 Mirai Botnet, are increasing in frequency and volume. The Mirai DDoS attack utilized Internet of Things (IoT) devices against well-known security expert Brian Krebs to exceed 620 Gbps of traffic volume in September 2016 [6]. Mirai was later used to DDoS DynDNS, temporally disrupting internet in the Eastern United States for several hours.

Denial of service is not the only vulnerability making headlines. This past year numerous Man in the Middle (MitM) attacks and vulnerabilities were identified. Will Dormann’s research into the top 58 business hypertext transfer protocol secure (HTTPS) inspection products found a range of flaws that failed to properly validate certificates and carried out HTTPS and secure sockets layer (SSL) inspections which put users at a greater
risk of MitM [7]. Then in October 2017, Nick Freeman reported a flaw with Windows DNS client which exposed millions of users to MitM [8]. According to Freeman, for an attacker to exploit this issue, they need to be between you and the DNS server you're using. For example, if you're using coffee shop Wi-Fi and someone is tampering with it, or if they've hacked your cable router – they can modify DNS responses that your computer receives.

My research in this area focuses on maximizing the network awareness of SDN/OpenFlow to mitigate network disturbances with long-term applications for (1) internet service provider’s (ISP’s) outbound filtering and (2) SDN-wide packet target monitoring.

The centralization of network control via the abstraction of the control plane from the data plane will allow for great visibility of the network. Combining the holistic network view and the OpenFlow protocol, SDN offers the potential for tracking or tracing network traffic in near-real-time. Exploring this capability for the purposes of validating traffic flow patterns and message source origination is an area of key interest. I believe this information is beneficial in enhancing secure access and intrusion detection.

1.3 Contributions

Throughout this dissertation, I provide multiple contributions in various areas of SDN security. I discuss the potential network security enhancements that SDN provides researchers. I provide a survey of simulation and traffic/attack generation tools which I utilize in future phases of my research. I offer my approach to securing sensitive data transmission across a network by analyzing data path metrics. I then develop a series of modeling, simulation, and testbeds which are used in experiments and evaluation of my
research. Next, I build a portable testbed which can be reused for research or as a teaching tool. Lastly, I implement a Network Analysis Tool to enhance data path security.

### 1.4 Dissertation Outline

This dissertation follows the development and progress of my research. In Chapter 2, I present an overview of SDN and its capabilities. This chapter focuses on the foundations of SDN and the potential it offers to enhance network security protocols. This leads into Chapters 3-8, where I explore various aspects of improving network security through modeling and analytic tool development.

Chapter 3 provides insight on specific security advantages and disadvantages of SDN. It is important to assess the strengths and weaknesses of SDN before developing new network applications. Chapter 3 identifies several potential areas for security research.

Chapter 4 is an in-depth exploration in SDN simulation and modeling for security research. It provides a survey of available tools and tests several traffic simulations utilities for use in later experiments and evaluations.

Chapter 5 contains the SDN security analysis framework – Data Path Confidence Analysis. Here I offer an in-depth explanation of the metrics collected, how the analysis was conducted, and the level of success of the services.

In Chapter 6, I expand upon Chapter 4 and develop a series of modeling, simulation, and testbeds for use in the development and testing of the Data Path Confidence Analysis framework described in Chapter 5. Chapter 7 takes the work in Chapter 6 one step further with the construction of a portable SDN physical testbed. This testbed can be utilized for not only SDN research, but also in the classroom to illustrate networking concepts.
After having laid this foundation, in Chapters 2-7, Chapter 8 is the culmination of my research with an implementation of a functional SDN Network Analysis Tool and associated experiments within the modeling, simulation, and testbeds previously developed.

Finally, I provide a conclusion and discussion on future research in Chapter 9. First, known issues and how they were dealt with are considered. Then I will provide a self-evaluation of the success criteria and research contributions, as well as lessons learned. The first criterion is to identify the ability of SDN to enhance network security, then create a new framework capitalizing on these capabilities. The second criterion is to provide a series of repeatable modeling, simulation, and testbeds for assessing the success of the proposed framework. The last criterion is the implementation of an SDN Network Analysis Tool and experimentation. This is followed by a section on how this research might be furthered. Finally, a summary of my research contributions and the results achieved is given. Of special note for the reader, Appendix K provides a list of acronyms used throughout this dissertation.
CHAPTER 2

OVERVIEW OF SOFTWARE DEFINED NETWORKING

In today’s network security environment, it is necessary for the network security engineer to understand all aspects of SDN.

2.1 Contributions

In this chapter, I provide a brief overview of SDN and how it differentiates itself from traditional networking. Additionally, I provide some practical and simple examples of SDN flow tables in action. The following overview will enhance the learning of new users (or those new to SDN).

2.2 Networking Prior to 2007

Gaining an understanding of the benefits and challenges of SDN/OpenFlow is essential to developing a solid M&S framework and analysis tool. A traditional computer network is accomplished through the interconnection of computers with a combination of cable and wireless media and networking hardware. Regardless of the network topology or scale, the network hardware consists of three key components: application, control plane, and data plane. The control plane maintains information that can be used to change data used by the data plane. Maintaining this information requires handling complex signaling protocols. The data plane is a subsystem of a network node that receives and sends packets from an interface, processes them as required by the applicable protocol, and delivers, drops, or forwards them as appropriate, see Figure 2-1 [9] [10]. The control plane, also a
subsystem, is responsible for providing the routing logic to the data plane. The quality and efficiency of this logic was, until 2007, the proprietary domain of the device manufacturer.

Traditional network switches and routers were comprised of millions of lines of unique code and hundreds (if not thousands) of different implementations and protocols. All of which was functionality “baked” into the network hardware. This limited the ability to customize networks and created numerous issues including: difficulty to change hardware brands, long-term support commitments, restricted user controls, and many more.

SDN is triggering a major change in the way network administrators think about their networks. With SDN, the data and control planes can be separated enabling more programmable and flexible networks; see the Figure 2-2. One of the primary technologies behind SDN is the open source protocol - OpenFlow [11]. In addition to abstraction of the control plane, an additional transport layer security/secure sockets layer (TLS/SSL) secure channel is established to support out of band communication directly between the

**Figure 2-1: Traditional Switch Structure**
controller and the switches. These things and more cause network administrators to rethink their strategy for providing network control and security.

OpenFlow is an open standardized interface for approaching the SDN architecture. Through the Layer 2 communications protocol OpenFlow is given access to the forwarding plane of a network switch/router [12]. The basic governing concept behind OpenFlow is to deliver a shared data and forwarding plane and a slice, user-managed control plane at Layer 2.

![Figure 2-2: Traditional vs. SDN Structure [9]](image)

### 2.3 SDN/OpenFlow – How It Works

SDN is an approach to building data networking equipment and software that separates and abstracts elements of these systems. The abstraction of the control plane from the physical switch or router in a virtual SDN Controller, located on a system administrator server, creates an opportunity to manage network flow decisions in near-real-time.
The data path of an OpenFlow switch consists of a Flow Table, and an action associated with each flow entry. The set of actions supported by an OpenFlow switch is extensible; below I describe a minimum requirement for all switches. To support high-performance and low-cost routing, the data path must have a carefully prescribed degree of flexibility. This means foregoing the ability to specify arbitrary handling of each packet by seeking a more limited, but still useful, range of actions [2].

Each flow-entry has a simple action associated with it; the three basic ones (which all OpenFlow switches must support) are:

1. Forward this flow’s packets to a given port (or ports). This allows packets to be routed through the network. In most switches this is expected to take place at line-rate.

2. Encapsulate and forward this flow’s packets to a controller. The packet is delivered to Secure Channel, where it is encapsulated and sent to a controller. Typically used for the first packet in a new flow, so a controller can decide if the flow should be added to the Flow Table. Or in some experiments, it could be used to forward all packets to a controller for processing, such as malware pattern matching.

3. Drop this flow’s packets. This can be used for security, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from end-hosts [2].

An entry in the Flow-Table has three fields: (1) a packet header that defines the flow, (2) the action, which defines how the packets should be processed, and (3) statistics, which
keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow (to help with the removal of inactive flows) [2].

The Flow Table example listed in the Figure 2-3 illustrates four key functionalities of SDN via the OpenFlow protocol [9]. Entries one and two are standard switch routing (which port to utilize) based upon a given media access control (MAC) or internet protocol (IP) destination address. In addition, OpenFlow provides the ability to count or track statistics for an entry in the Flow Table. This can be particularly useful for prioritization, load balancing, and identification of potential network attacks. Entry three shows how OpenFlow can serve as a switch-based firewall; drop all transmission control protocol (TCP) from port 25. Flow Table entry number four shows the switch’s ability to retain its own local logic (e.g., not everything has to be OpenFlow directed). In this case, the switch knows traffic for IP address 192.x.x.x should utilize local switch logic. Lastly, entry five depicts a new flow which the switch has not previously encountered; therefore, the switch should forward the packet to the SDN Controller for further information.

![Figure 2-3: Example Flow Table](image)
CHAPTER 3

SDN/OPENFLOW SECURITY ENHANCEMENT POTENTIAL

There are many thoughts what exactly SDN is and means to the future of networking. The main theme is a flexible centrally managed network that can quickly change to meet new business requirements [13].

3.1 Contributions

In this chapter, I provide an overview of the many possible benefits that centralized network control (i.e., SDN) offers to enhance overall network security. First, I will cover some of the current state of the art SDN security implementation. This is followed by a list of potential advantages and disadvantages of utilizing SDN. This list itself could offer future researchers ideas for new security enhancements. Lastly, I provide some thoughts on weighing the benefits of SDN prior to use and security augmentation.

3.2 Current State of the Art SDN Security

Several recent papers and commercial applications attempt to take advantage of the SDN/OpenFlow. These works provided an excellent starting point for my SDN security research. Publications from several major journal and conferences offer sound examples which may be applicable to the creation of a large-scale OpenFlow M&S, network analysis, and DoS/DDoS security.

Researchers from the University of Wurzburg provided an early modeling and performance evaluation architecture for OpenFlow. Their work delivers a basic model for
estimating packet sojourn time and the probability of lost packets, both of which are necessary to measure the effectiveness of OpenFlow against existing routing schema [14].

An OpenFlow study from Universidade Federal de Amazonas presented a method for DDoS attack detection based on traffic flow features, in which the extraction of such information is made with very low overhead. Compared with traditional approaches, it provides some insight into basic networking simulation and testing [15]. Figure 3-1 illustrates their approach which utilizes a controller module to collect, extract, and classify a flow.

![Figure 3-1: Detection Loop Operation](image-url)
Understanding recent work from the University of Memphis and its utilization of game theory-based defense mechanisms against DDoS attacks provides a contemporary approach to the DDoS threat and M&S [16].

Research from Bell Laboratories offers some insight to utilization of SDN visibility to, “enable applications to leverage a network without exposing the network provider’s internal details or policies” [17]. Additionally, work from Stanford University offers a packet backtrace, though not in real-time, which shows the sequence of forwarding actions seen by that packet [18]. Figure 3-2 illustrates a packet backtrace for a bug on a chain topology.

![Figure 3-2: Packet Backtrace [18]](image)

Commercial research also helps with DDoS attack simulation and detection techniques as well. For example, a project on BotNet detection by the Computer Science Laboratory at SRI International [1] [2], is one of a few early OpenFlow specific intrusion detection
system-style (IDS) approaches to DDoS. Figure 3-3 depicts how BotHunter and a security specific SDN controller, FortNOX, can be used within an SDN domain to isolate and limit malicious activity.

Figure 3-3: Sample Malware BotHunter Event [19]

Finally, a review of attack simulation research helped to identify methods for eventual security testing. The current networking architecture attack applications [20] [3] [21] are helpful in the generation of attack message traffic for an SDN/OpenFlow network. I explore these applications further in Chapters 3, 5, and 8.
3.3 Potential Advantages and Disadvantages of SDN

As with any emerging technology advancement, there are associated costs and benefits with SDN. Early adopters of SDN would have found little industry hardware support and even less for software. A thorough review of the SDN concept and the OpenFlow protocol identifies a series of strengths and weaknesses.

3.3.1 Security Advantages

1. *Better Knowledge/Control of the Complete Data Path*: Ability to know the actual data path and therefore being able to discount everything not in the path when troubleshooting. In traditional networks, every switch and router is an island and you rarely know the complete end-to-end path without tracing it hop by hop [22].

2. *Abstraction of Control Plane*: OpenFlow can take control of how traffic flows through a network out of the hands of the infrastructure -- the switches and routers -- and put it in the hands of the network owner, individual users or individual applications. … proponents say it is particularly useful for: load balancing, flow control and virtual networking in data centers and private clouds and campus local area networks (LANs) where devices are multiplying and straining network topologies like Spanning Tree, which can take tens of seconds to reconverge after a topology change [23].

3. *Similar Benefits As Hardware Virtualization*: Advances in virtualization and automation has helped make people ready to automate and control the networking side of things to gain the same type of benefits: scale, control and the massive cost savings that go with it [22].
4. **Multivendor Environment:** Enables multivendor environments. It is always a good thing to keep the option of having several vendors in your network or of switching vendor later down the road if needed [24]. Furthermore, OpenFlow gives the network the ability to retro-fit hardware.

5. **Separates Application Development From Hardware:** “It also offers a nice horizontal architecture for your network to evolve separating the hardware platform from the applications” [24].

6. **Virtual Switch Support:** OpenFlow supports in their physical devices and virtual switches such as OpenvSwitch and Cisco Nexus 1000V [25] [26].

7. **Self**” Defending Network:** Ability to leverage telemetry and flow tables in the controllers “centrally” and then “dispatch” the necessary security response on an as-needed basis to the network location that needs it [27].

8. **Reduced Downtime:** By eliminating manual intervention, SDNs enable users to reduce configuration and deployment errors that can impact the network.

9. **Flexibility:** SDNs create flexibility in how the network can be used and operated. Users can write their own network services using standard development tools for their own unique environment and needs. For example, a firewall application may need a high-performance, low-latency, low-complexity data exchange, while a monitoring application might only need to read flows as they pass [27]. SDN gives the ability to “slice” the network traffic based upon protocol, payload, or addressing for greater quality of service.

10. **Better Management/Planning:** A single viewpoint and toolset to manage virtual networking, computing and storage resources. Better visibility into network,
computing, and storage resources means users can also plan IT strategies more effectively for their customers.

11. *Infrastructure Savings:* Separating route/switching intelligence from packet forwarding reduces hardware prices as routers and switches must compete on price-performance features. As a standard way of conveying flow-table information to the network devices, it fosters open, multi-vendor markets. Network services can be packaged for application owners, freeing up the network administrators [28].

### 3.3.2 Security Disadvantages

1. *OpenFlow Not Mature Enough:* This is becoming less and less of an issue as time passes because of more vendor (Big Switch Networks announced two applications along with its OpenFlow controller. HP has announced applications that plan to be available in 2013 [25]), research, and commercial use. Many existing switches/routers have firmware upgrades that support OpenFlow so there is no need for large hardware upgrades.

2. *Lack Of Killer Applications:* OpenFlow shouldn’t be dismissed just because it’s not very useful on one network at one point in time, but it needs a better industrial grade application that capitalizes on its value [29].

3. *Learning Curve To Implement:* Deployment of a new network will take more time due to the time to learn OpenFlow and develop solid controller rules (although project like Project Floodlight do offer some good free controller solutions) [30].

4. *Controller is Single Processor/Bottleneck:* Concerns that the OpenFlow controllers are a single point of computation in the network and are designed to re-establish interrupted flows by creating a tunnel through all the network paths. There has
been some industry discussion that computations involved in creating those tunnels on the fly may overwhelm the controller [23].

3.4 Weighing the Benefits

The security benefits of SDN/OpenFlow are often seen as a double-edged sword. For example, the centralization of control means a single point of failure. Further Open Source control modules which allow for customization, create the potential for malicious and conflicting rules sets.

As we reach the end of the first decade with SDN, we see it is transitioning from academia/research to production environments. Several major vendors like VMware, with NSX and software defined datacenters, and Cisco, with its SDN/OpenFlow platform onePK/OpFlex, are clearly embracing the SDN.

In an interview with Google principle engineer, Amin Vahdat expressed the biggest advantage of SDN/OpenFlow is being able to get better utilization of existing lines. The traditional networking state-of-the-art is to run your lines at 30% to 40% utilization, and Google can run their wide-area lines at close to 100% utilization, just through careful traffic engineering and prioritization [31].

The advantages and disadvantages are becoming more specific and focused on user requirements. Looking at SDN strictly form a security prospective there appear to be positive and negative impacts. SDN enables more automation and better network visibility. Automation means less “human” manual involvement in the network, which means less potential for mistakes. It also means greater consistency across the network. The provided visibility will allow more accurate and timely decision-making about performance and potential threats.
That said, network automation comes from the SDN application, which if flawed will open networks to breaches. Software was proprietary under the traditional networking model. The closed nature of the software and cost of hardware helped limit malicious experimentation. With SDN, attackers can begin to focus on open-source controllers and application programming interfaces (APIs) to identify vulnerabilities which can be implemented across numerous SDN networks at little or no cost. According to an Open Networking Foundation (ONF) whitepaper, “Network services (residing in the Control Layer) expose the SDN communication services through a series of northbound APIs and directly control the forwarding behavior of the underlying network devices that reside in the Infrastructure Layer (through OpenFlow). Business applications are vulnerable to potential threats because of the powerful SDN programming model” [32]. Thus, the threat might not be one of SDN developer omission, but rather as users implement multiple SDN services that interference between routing rules could compromise security policy. Northbound APIs makes the control information of the network available to higher instance abstractions which allows for services and applications, such as firewalls, load balancers, or cloud orchestration, to access the network. Southbound APIs allows for physical and virtual switches to exchange control information with the SDN controller platform, an example of this is OpenFlow [13].

The benefits of SDN outweigh the potential costs and offer a great deal of areas for potential academic research and contribution. The remainder of this dissertation focuses specifically on enhancing security by leveraging to SDN/OpenFlow.
CHAPTER 4

EXPLORATORY SDN/OPENFLOW SECURITY RESEARCH

More than a survey of the technology is necessary to understand SDN/OpenFlow and its potential to improve security. Exploratory research into the capabilities of SDN/OpenFlow, methods of M&S, and some validation of its usefulness is needed to help identify areas for future research. The goal of this area of research is to develop a baseline M&S framework for attacks on SDN/OpenFlow networks. To support this goal, I explored available M&S tools, explore traffic/attack generation approaches, deployed both a small-scale and large-scale M&S experiments, and developed criteria to evaluate their performance. I presented, as first author, the initial finding of this research at the International Conference on Computer Communications and Networks (ICCCN) [33].

Successful deployment of a large-scale M&S framework begins with learning from small-scale M&S. Small-scale M&S requires fewer resources in terms of processing power, time to learn the M&S construct, and general ease of use.

4.1 Contributions

In this Chapter, I make the following contributions: (1) briefly outline small-scale modelling, (2) define a large-scale network, (3) identify several techniques for simulating a network attack, (4) analyze current available SDN modelling and simulation tools, (5) review open-source SDN controllers, (6) examinate attack modeling and simulation, and (7) provide exploratory experimentations of such attacks.
4.2 Small-Scale Network Modelling and Simulation

Building a simple model is necessary for understanding the domain of an OpenFlow driven network. From this simple simulation, I gathered valuable SDN/OpenFlow experience.

I used Mininet to conduct SDN/OpenFlow research without the expense of OpenFlow switches and the need for high volume network traffic. Mininet is an open source, network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet’s simple command-line interface allows users to quickly setup and run a realistic virtual network.

The Mininet.org consortium has provided us with an excellent simple OpenFlow network modeling tool [34]. Online tutorials for Mininet as well as the OpenFlow protocol are essential for understanding how to setup and structure a basic network.

I define a small-scale network as a Mininet model comprised of three network hosts, one switch, and one OpenFlow enabled controller. I created this network to, ping hosts within the network, limit traffic via the controller, and monitor traffic to successfully deploy this network model. Below I will cover the findings of DoS/DDoS attack testing against this basic small-scale SDN/OpenFlow model.

4.3 Defining A Large-Scale Network

Defining what constitutes “large-scale” was an early challenge. As networks evolve and technology changes, the make-up of a network changes as well. Additionally, one researcher’s large-scale network may include more LANs or Wi-Fi than another researcher’s. The network hierarchy is probably the single largest variable in defining and
differentiating between various large-scale networks. From a M&S standpoint it is best to consider the number of events versus the number of nodes when attempting large-scale M&S [35] because the events in a simulation are a function of the number of nodes plus:

- Traffic profile
  - Offered load (insertion rate)
  - Traffic type (mix of voice, video, and data)
  - Delivery mechanism (unicast or multi-cast)

- Degree of “wireless” operations and type

- Fidelity of the protocol models

- Amount of mobility

- Multi-tenancy and controller distribution

I intend to grow the scale and makeup of my model; see Chapter 6 for more information. Below is a brief survey of available open source large-scale network M&S tools which can emulate OpenFlow:

1) **NS-3** - A discrete-event network simulator, targeted primarily for research and educational use. NS-3 is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use [36]. NS-3 is one of the mostly widely utilized and oldest available open source M&S tools; however, it is a text-based tool.

2) **EstiNet** - Version 8.0 is an SDN/OpenFlow network simulator and emulator that can simulate thousands of Version 1.3.2 and Version 1.0.0 OpenFlow switches and run the real-world NOX, POX, Floodlight, and Ryu controllers without any
modification to control these switches during simulation. Its performance simulation results are realistic, accurate, and repeatable [37].

3) **OMNeT++** - An extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators [38]. It is free for academic and non-profit use, and it is a widely used platform in the global scientific community.

4) **Mininet** - Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine virtual machine (VM), cloud or native), in seconds, with a single command [34]. It is widely used by the academic research community and supported by a robust community.

### 4.4 Currently Available Testing and Attacks Tools

#### 4.4.1 Traffic Generation Tools and Approaches

There are several Linux-based tools and approaches for the creation of network traffic. The most simplistic approach is using `ping` from one host to another. This will work for basic SDN security, but to make more realistic assessment of security solutions I have identified two additional options.

The first traffic generation tool is `hping3`. It is a network tool able to send custom TCP/IP packets and to display target replies like the ping program does with internet control message protocol (ICMP) replies. The tool `hping3` handles fragmentation, handles arbitrary packet body and size, and can be used to transfer files encapsulated under supported protocols. Using `hping3` you can perform at least the following [39]:

- Test firewall rules.
- Advanced port scanning.
• Test net performance using different protocols, packet size, type of service (ToS) and fragmentation.

• Path maximum transmission unit (MTU) discovery.

• Transferring files between even very strict firewall rules.

• Traceroute-like under different protocols.

• Firewalk-like usage.

• Remote operating system (OS) fingerprinting.

• TCP/IP stack auditing.

A more modern approach is Ostinato. It is often referred to “Wireshark in Reverse” [40]. Ostinato is a packet crafter, network traffic generator and analyzer with a friendly graphical user interface (GUI). Ostinato allows users to craft and send packets of several streams with different protocols at different rates. A limitation of Ostinato is that it cannot be used to generate fake traffic to a website. For general SDN security research and traffic testing, Ostinato is a very useful tool.

4.4.2 Denial of Service Simulation Tools

Like the traffic generation tools listed above, there are both simplistic and complex solutions for creating a DoS/DDoS simulation. Below I utilized the simple form, a ping flooding attack. Ping flooding is where the attacker overwhelms the victim with ICMP "echo request" (ping) packets.

A more complete and complex tool is DDOSIM, which can be used in a laboratory environment to simulate a DDoS attack against a target server. The test shows the capacity of the server to handle application-specific DDoS attacks [20].
It simulates several zombie hosts (having random IP addresses) which create full TCP connections to the target server. After completing the connection, DDOSIM starts the conversation with the listening application (e.g., HTTP server).

An even more complex tool is BoNeSi which simulates Botnet Traffic in a testbed environment on the wire [21]. It is designed to study the effect of DDoS attacks. BoNeSi generates ICMP, User Datagram Protocol (UDP) and TCP (HTTP) flooding attacks from a defined botnet size (different IP addresses). BoNeSi is highly configurable and rates, data volume, source IP addresses, uniform resource locators (URLs) and other parameters can be configured.

There are plenty of other tools out there to spoof IP addresses with UDP and ICMP, but for TCP spoofing, there are few solutions. BoNeSi is one of the first tools to simulate HTTP-GET floods from large-scale bot networks. BoNeSi also tries to avoid generating packets with easily identifiable patterns (which can be filtered out easily). BoNeSi sniffs for TCP packets on the network interface and responds to all packets to establish TCP connections.

I collaborated with engineers from EstiNet to test BoNeSi using their M&S tool, see Figure 4-1. We created a step-by-step user guide to execute such an attack, which is available in Appendix A. The attack tool proved to be very powerful and easy to utilize.
4.5 Selection of SDN/Open Modelling and Simulation Tool

After conducting a survey of existing tools and attempting to setup an SDN/OpenFlow environment like the small-scale Mininet simulation described above, I selected EstiNet as the environment for my exploratory M&S research. I made this selection based upon the following criteria (1) support for latest OpenFlow releases, (2) simulation on “real-world” Linux kernel versus simulation engine code, (3) scalability for large-scale networks, (4) ability to emulate “real-world” devices into simulation, and (5) repeatability of results.

- NS-3 does not run on Linux kernel and does not support OpenFlow newer than v0.8.9. Furthermore, support from the academic community to advance OpenFlow in NS-3 has stalled indefinitely.
- EstiNet meets all the above criteria for an SDN/OpenFlow large-scale M&S tool.
- OMNeT++ does not support the latest release of OpenFlow and its SDN support is experimental in nature requiring significant modification [41].
Mininet does not provide scalability (suffering major lag with M&S above 50 nodes) nor does it offer repeatability (cannot guarantee packet scheduling or performance) [42].

### 4.6 Controller Applications

To implement the OpenFlow protocol, a server-side software-based controller is necessary. There are numerous controllers in multiple languages (C, C++, Java, Python and Ruby most notably). Table 4-1 lists some of the more common controllers, their language, and a brief description.

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOX</td>
<td>C++</td>
<td>Open sourced by Nicira Networks</td>
</tr>
<tr>
<td>POX</td>
<td>Python</td>
<td>Targeted largely at research and education</td>
</tr>
<tr>
<td>Project Floodlight</td>
<td>Java</td>
<td>Enterprise-class; Apache-licensed; Supported by Big Switch Networks</td>
</tr>
<tr>
<td>NOX D</td>
<td>C++</td>
<td>Multi-threaded, highly optimized implementation of NOX</td>
</tr>
<tr>
<td>Beacon</td>
<td>Java</td>
<td>Predecessor of Project Floodlight; Originated at Stanford</td>
</tr>
<tr>
<td>Maestro</td>
<td>Java</td>
<td>Originated at Rice University</td>
</tr>
<tr>
<td>FortNOX</td>
<td>C++</td>
<td>Secure kernel version of NOX</td>
</tr>
<tr>
<td>Trema</td>
<td>Ruby</td>
<td>Full-stack, easy-to-use framework for developing in Ruby and C</td>
</tr>
<tr>
<td>Open Daylight</td>
<td>Java</td>
<td>Open platform for network programmability to enable SDN and create a solid foundation for NFV for networks at any size and scale</td>
</tr>
</tbody>
</table>
Limited performance comparisons have been conducted using these implementations. Figure 4-2 shows the results of a May 2011 study conducted by OpenFlow.Org.

![32 Switch Emulated Throughput](image)

Figure 4-2: OpenFlow Controller Performance Comparison

### 4.7 DoS/DDoS Attack Modelling and Simulation

Understanding the types of DoS/DDoS attacks, potential network vulnerabilities, and how these attacks are implement is a crucial first step to realistic attack M&S. The ability to monitor any such attack traffic within this simple network is a necessity for repeatability and verification.

A DoS/DDoS attack can be perpetrated in many ways. The five basic types of attack are:
1) Consumption of computational resources, such as bandwidth, disk space, or processor time.

2) Disruption of configuration information, such as routing information.

3) Disruption of state information, such as unsolicited resetting of TCP sessions.

4) Disruption of physical network components.

5) Obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

In addition to the ping flooding technique utilized in these experiments, I also wish to highlight the SYN flooding method. It takes advantage of the TCP three-way handshake model.

1) The attacker creates packets that contain spoofed IP addresses, every packet having a SYN flag set meaning it would like to open a new connection to the Server.

2) The Server receives the spoofed packets and is sending back ACK packets to the spoofed IP addresses. The Server waits for an ACK coming back from the attacker, but since the IP addresses were spoofed it never gets any packets back.

3) During this process the server connection table is full, and all new connections are ignored. This affects all users who want to make a connection to the Server.

4) After the attacker had its share of fun and stops flooding the Server, the Server normally goes back to its normal state.
4.8 Experiments and Evaluation

Following a thorough review of M&S tools and a variety of DoS/DDoS attack vectors, I (1) created a simple M&S for future OpenFlow security research and (2) validated an attack method. The goal of the large-scale M&S is to identify the best practices for the M&S tool and to provide support for future OpenFlow security research. All experiments were conducted under VMware vSphere 5.1 as virtual machines with 4 x 2.66GHz CPUs, 4GB RAM, and 300GB of storage assigned.

4.8.1 Small-Scale Testing

The small-scale network for my experiments was defined above during my initial discussion of Mininet. My primary goal for this segment of the larger M&S problem is to setup an SDN/OpenFlow network, to generate DoS attacks, and to mitigate the attacks using the OpenFlow controller. For this small-scale model, I utilized Mininet in a three host, one switch, and one controller configuration with one host attacking another single host. My attack methodology uses the existing ping flooding command present in any Linux distribution. My defensive approach is to limit/filter message traffic from a particular IP address.
A small-scale Mininet M&S without any IDS or firewall rules was easily saturated by the ping flooding DoS attack. Figure 4-3 illustrates my results and shows the exponential rise attack volume. Although this attack is less effective today due to network bandwidth, increased processing power, and the fact that it’s from a single source, the concept of this type of attack against and SDN/OpenFlow network remains valid. It is not difficult to image a more complex attack strategy utilizing multiple computers via botnet and employing IP spoofing. The key takeaway is the rapid increase of traffic and eventual overflow of either switch or OpenFlow controller resources.

I mitigated this DoS attack by filtering traffic by IP address using the OpenFlow controller. Figure 4-4 shows the enhanced network performance. Conceptually this approach helps us understand the impact an OpenFlow-based network can have against DoS/DDoS attacks.
After selecting EstiNet as my M&S tool for this exploratory research, I compared it against our small-scale model. Figure 4-3 illustrates the results of multiple runs of the same attack profile. It shows similar results to that of the Mininet simulation. This is to be expected on a small-scale M&S. Likewise, I found the mitigation script to perform relatively close to that of Mininet small-scale M&S shown in Figure 4-4. Since the number of switches in these experiments was small, these observations are in-line with simulation comparisons published by Shie-Yuan Wang in IEEE Communications Magazine [40]. However, the deviation of error between test-runs was far greater utilizing Mininet versus EstiNet.

4.8.2 Large-Scale Framework

If the continued use of EstiNet was desired, then a long-term test evaluation process was necessary. I evaluated each EstiNet M&S (small or large-scale) against the following:
• Repeatability: What happens when I run the M&S multiple times and do I receive similar results? If not, why not? Repeatability of my experiments is one of the key reasons I selected EstiNet going forward with my long-term research.

• Attack Success: Was the DoS/DDoS attack simulation successful in the given scenario? If not, was it a problem with the network M&S or the attack simulator? The emulation functionality built in to EstiNet is another reason it was selected. This should offer a wider range of attack tools access to the M&S.

• Time to Setup and Execute Experiment: This includes building the simulation environment, connecting the attack tool(s), emulating any additionally resources, and initializing the attack. This metric was used to provide future researchers a baseline for M&S reuse.

• Network Traffic: Ability to capture traffic between attacker(s), target(s), and other legitimate traffic. Further I analyzed the OpenFlow “overhead” traffic between the switches and controller. Recent research [43] has shown the lines of communication between switches and controller to be a potential SDN/OpenFlow attack vector.

With these criteria defined, I established the network topology, shown in Figure 4-5, to simulate a larger scale environment; hosts 7-9 can be substituted with networks of aggregated nodes. This topology supports DoS and DDoS attacks and legitimate additional traffic. It also provides feedback on the potential benefits of large OpenFlow-enabled infrastructure.
4.8.3 Final Thoughts on EstiNet

Throughout my initial exploratory research, EstiNet has shown to be the best M&S tool available for realistic testing. However, in the long run it proved to be cost prohibitive for academic use. I completed my initial work with a trial license with the understanding that I would publish my results (which I did in 2014). After doing so, EstiNet informed me their future licensing model required large annual fees which I could not accommodate. As a result, all future research was completed using either Mininet or a physical testbed.
CHAPTER 5

FRAMEWORK FOR SDN DATA PATH
CONFIDENCE ANALYSIS

Building upon this understanding of SDN and modelling, I am now focused on enhancing network security through data path analysis. The initial findings of this research were presented at the International Conference on Dependable Systems and Networks (DSN) [44]. I presented, as first author, the final results of this research at the IEEE Conference on Dependable and Secure Computing (DSC) in 2017 [45].

5.1 Contributions

My research provides a framework for a broad range of capabilities for administrators to use as well as for automated protection services. To narrow the scope of the research, this chapter focuses on a subset of those capabilities as they apply to the analysis of a specific network path at the time of use or inspection. I developed a service framework that inspects a network path before and after sending sensitive information and compares this inspection to our known behavior and security patterns to provide a user with a dependability assessment of that specific network path. This dependability assessment allows users to decide whether a network path is secure enough for sending their information. My research proposes techniques for a network path analysis service that can be used to identify latent systemic network problems, facilitate a more dependable network and to prevent the theft of information by malicious actors.
5.2 The Data Path Security Problem

5.2.1 Data Path Security Introduction

As we continue to transmitting greater amounts of sensitive information via the Internet, the risk of the information being viewed by unintended or malicious actors increases [46]. Whether sending an email, sending a batch of medical records, or allowing a direct download of proprietary software, the ability to further analyze the security of the data path in near-real-time is a priority. Data spillage, accidental or purposeful, costs industry millions of dollars in remediation costs and lost revenue [47].

I explore techniques to identify the current health of the data path and detect malicious activity by providing a confidence analysis of the data transport across the network path using SDN metrics. These techniques can be used by individuals to assess the risk of transferring sensitive information or be inserted into an existing security framework for autonomous network behavior modification.

I propose techniques for accumulating and analyzing SDN metrics to provide a security confidence level or score of the network data path before, during and after sensitive data transmission. In addition to providing this layer of security, my techniques can also be used to assess the health and confidence of network infrastructure.

Public and private networks alike depend on some of the same shared network infrastructure which amplifies the need for greater data path security tools. The ability to isolate and secure traffic is key to the correct behavior of this infrastructure [48]. A few examples of increasing data transport security include the following instances. Many business and regulatory requirements exist to keep sensitive customer data isolated from
other traffic. Further restrictions may also include the countries in which the data resides in or travels through. Many government agencies restrict different levels of sensitive information from traveling across certain devices or network segments. At the host-level, datacenters must ensure that traffic does not flow across the devices of other customers including virtual devices. Secure and reliable transport requires the correct behavior of shared network infrastructure. My techniques provide a better understanding of that infrastructure and behavior.

### 5.2.2 Data Path Security Concept Description

Today, if you want to send a secure message from one person to another, the sender, Alice, first encrypts the message. Then Alice binds the message in an authenticated wrapper. Next, Alice sends the message through/over the network to the receiver, Bob. Bob first verifies the authenticity of the message (is it really from Alice or is it tampered malware?). Once verified, Bob then decrypts the message (this may also include Bob verifying his identity to an authentication server – prior to receiving the decryption key).

![Figure 5-1: Traditional Network Secure Access Approach](image)
At no point in the scenario above, see Figure 5-1, has any analysis of the transport medium been assessed. I propose utilizing the power of SDN to do exactly this – verify, validate, and assess the data path, see Figure 5-2.

I designed and implemented a framework for SDN network security confidence analysis. To enhance traditional analysis, I utilized SDN in two key areas: (1) route and destination verification and (2) switch metrics analysis. Referring to Figure 5-2, this framework allows SDN authentication applications to validate and verify the routing and destination of data as well as assess the network devices for unexpected behavior (i.e., data compromise, MitM attacks, etc.) This framework provides a confidence analysis of network security elements.

Verify: What route did the message travel?

Validate: Was it the correct/desired path?

Assess: Based on SDN switch metrics, what dependability of the data path?

Figure 5-2: Framework for SDN Confidence Assessment
5.3 Existing Research on SDN Transport Analysis

Before beginning any new research, it is essential to review existing work and the current state of the art. Below is an assessment of existing SDN data transport research which I reviewed prior to conducting this area of research.

5.3.1 Pathlet Routing

A team from Berkeley researched a new protocol for tracing the data path. They offered a new method, pathlet routing, in which networks advertise fragments of end-to-end paths from which a consumer can assemble an end-to-end route. Their work provides a solid foundation in traceability, although this technique does not focus on SDN or the potential benefits of network awareness/control [49]. They propose this technique to emulate network policy such as BGP, source routing and multipath routing.

5.3.2 Pathlet Tracer

NEC Labs developed Pathlet Tracer, a traceroute-like method that provides limited route step verification and path validation [50]. Pathlet Tracer was designed to detect mistranslations between high level policy and the Layer 2 forwarding plane behavior. Their concept is for an SDN controller that generates a “codebook” of flow IDs and routing is an interesting approach, which could be employed to enhance data path analysis.

5.3.3 SDN Traceroute

IBM researchers propose a tool, SDN Traceroute, which can query the current path taken by any packet through an SDN-enabled network [51]. The key to this approach is that the path is traced by using the actual forwarding mechanism on each SDN-enabled device. This method adds a highest priority forwarding rule to each device and for each device incurs the cost of a round trip exchange with the controller. Like Pathlet Tracer,
this tool detects differences between the high-level policy and low level, Layer 2, forwarding behavior. My work focuses on this framework implementation of data path tracking.

5.3.4 ICING and Path Verification Mechanism

Using Path Verification Mechanism (PVM), ICING provides two mechanisms for policy enforcement, Path Consent and Path Compliance [52]. It offers a method for determining if a packet followed an approved path, but requires specialized hardware at twice the cost of today’s hardware, which would be a significant burden to network operations. Furthermore, this method involves modifying the packet header, which is an undesirable (and potentially dangerous) approach.

5.3.5 FlowMon: Detecting Malicious Switches in Software-Defined Networks

The authors examine two algorithms for detecting packet dropping and packet swapping to detect a compromised switch [53]. This use-case demonstrates utility that can be derived from a network confidence analysis.

5.3.6 Anomaly Detection: A Survey

This work provides a structured and comprehensive analysis of anomaly detection [54]. In addition to reviewing their approach, I employed the common terminology developed for anomalies in my research.

5.3.7 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) provides an open framework for communicating the characteristics and impacts of IT vulnerabilities [55]. CVSS focuses on exploits, but its methodology is applicable to my data path metric analysis. I have
specifically chosen to employ the CVSS methodology based on its lengthy review by both
industry and academic researchers.

5.4 Network Confidence Framework

I developed techniques that when combined provide users with an insight into the
health and confidence of their network. While the analysis is not limited to SDN metrics,
this combination is in the form of an SDN northbound framework that is SDN controller
agnostic with a design that is applicable to any SDN framework. In addition to this
overview of the framework, the source code is available in Appendices B and C. Figure
5-3 depicts the overall data transfer process from start to finish.

First the endpoints of the transmission are identified. If a messaging tool was utilized,
then this information could be retrieved from the messaging API. Once the transfer begins,
metrics are collected and monitored via the SDN/OpenFlow protocol. Upon completion
of the transfer, calculations and analysis of the metrics are performed and a confidence
score is generated. In the overall process, after the score is provided either a user,
administrator or automated process can make a decision whether or not to release the
encryption keys. This process assumes a key control server/infrastructure is in place.
An implementation of metric collection is outside the scope of this framework; I focused on a northbound application that analyzes network metrics. One possible source of metric collection is those gathered in the OpenDaylight (ODL) Time Series Data Repository (TSDR) project [56] [57]. I am a contributor to the TSDR project which collects several different classes of metrics and exposes them to northbound applications via a restful interface and a yang generated API. TSDR collects SDN controller metrics, OpenFlow statistics, netflow statistics, sFlow statistics, snmp data, restconf data and syslog data. TSDR also provides an external collector for applications to store custom data. I leverage this feature to store my analytics.
My data path analysis employs a proven vulnerability scoring methodology (derived from historical time series data and probe results) against an array of network metrics to establish a confidence level.

5.4.1 Metric Collection Overview

My data path assessment compares the flow metrics from three flows over the target route: pre-flow, actual-flow and post-flow. Pre and post flows are small in nature to limit overhead, but provide the necessary SDN route and flow metrics for comparison with my historic time series flow patterns.

5.4.1.1 Pre-Traffic

Once the source identifies a destination for traffic, the SDN confidence service initiates the traffic flow sequence. First, a pre-traffic flow is sent from the source to the destination to baseline device metrics. A short series of pre-traffic packets with predetermined attributes and known, historical behavior provides a baseline for the actual traffic.

5.4.1.2 Actual Traffic

Next, the actual traffic flows. The overhead of this approach is known and extracted from the measurements. This approach has minimal with out-of-band controller communication.

5.4.1.3 Post-Traffic

Lastly, a post-traffic flow is sent from the source to the destination to provide a post path flow metrics. Like the pre-traffic flow, this has a minimum impact on the network.

5.4.2 Current SDN/OpenFlow Device Metrics

The current OpenFlow protocol supports metrics collection. Traditional metrics are gathered using the netflow and sFlow capabilities in ODL TSDR [1].
5.4.2.1 OpenFlow 1.0 Flow Entry [1]

- Per Flow Table:
  Active Entries, Packet Lookups, Packet Matches

- Per Flow:
  Received Packets, Received Bytes, Duration (seconds), Duration (Nano seconds)

- Per Port:
  Received Packets, Transmitted Packets, Received Bytes, Transmitted Bytes, Receive Drops, Transmit Drops, Receive Errors, Transmit Errors, Receive Frame Errors, Receive Overrun Errors, Receive Cyclic Redundancy Check (CRC) Errors, Collisions

- Per Queue:
  Transmit Packets, Transmit Bytes, Transmit Overrun Errors

5.4.2.2 OpenFlow 1.3.x Flow Entry

- Per Flow Table:
  Reference Count (active entries, required), Packet Lookups, Packet Matches

- Per Flow:
  Received Packets, Received Bytes, Duration (seconds, required), Duration (Nano seconds)

- Per Port:
  Received Packets (required), Transmitted Packets(required), Received Bytes, Transmitted Bytes, Receive Drops, Transmit Drops, Receive Errors, Transmit Errors,
Receive Frame Alignment Errors, Receive Overrun Errors, Receive CRC Errors, Collisions, Duration (seconds, required), *Duration (Nano seconds)

- Per Queue:
  Transmit Packets (required), Transmit Bytes, Transmit Overrun Errors, Duration (seconds, required), Duration (Nano seconds)

- Per Group:
  Reference Count (flow entries), Packet Count, Byte Count, Duration (seconds, required), Duration (Nano seconds)

- Per Meter:
  Flow Count, Input Packet Count, Input Byte Count, Duration (seconds, required), Duration (Nano seconds)

- Per Meter Band:
  In Band Packet Count, In Band Byte Count.

5.4.3 Potential Metrics Measurements for SDN Confidence Analysis

To better highlight the metrics collected and how they may be analyzed, reference the example network topology in Figure 5-4 and assume that host A is sending a sensitive message to host C. For the remainder of this chapter I utilize the example network and scenario in Figure 5-4 and the common variables:

\[ P = \text{packet} \quad Cr = \text{Controller} \]
\[ F = \text{flow} \quad T = \text{time} \]
\[ x, y, z = \text{transmission identification} \quad n = \text{size/bytes} \]
The following depicts the data path of the scenario described above:

\[ \text{Path: } P_x \rightarrow \text{Host A}, S1, Cr, S1, S2, S4, \text{Host C} \]

### 5.4.3.1 Packet Arrival Time to Controller

One of the most basic elements of SDN metrics, at a minimum, could be used to compare a sampling of packets from within a given flow to determine similarities or discrepancies. If traffic were intended for multiple recipients, then the arrival time of similar segments of the routing could be compared as well. Assessing the median arrival time from pre/post and actual traffic provides a baseline metric for subsequent transmissions along the same route.

![Figure 5-4: Example SDN Topology](image)

#### 5.4.3.2 Packet/Flow Size

Measuring the size of the first and last packets, then multiply it by the total number within a flow. Packet size can then be compared as the data flows from switch to switch
and from each of the three elements of message traffic (pre/post and actual). The formula in Equation 1, which I created specifically for this framework, illustrates the comparison of packet size from transmissions (x, y, and z) as it is measured at the switches within the data path (switch 1, 2, and 4).

\[
\frac{(P_{x1} + P_{x2} + P_{x3})}{(P_{y1} + P_{y2} + P_{y3})} \cdot \frac{(P_{z1} + P_{z3} + P_{z4})}{(P_{z2} + P_{z4} + P_{z5})} \cdot \frac{(P_{z3} + P_{z4} + P_{z5})}{(P_{z2} + P_{z4} + P_{z5})} \quad \text{Eq. 1}
\]

5.4.3.3 Packet/Flow Duration

This analysis uses metrics about a flow from when the first packet enters the switch plane at the first switch until the last packet has exited the switch plane at the last switch. In other words, it is the amount of time from when the first packet enters the flow exits the switch. Using these metrics along with flow sizes and types/protocols, I compare traffic flows with performance tests and standardized metrics to assess and create expectations for transport time and routing. Equation 2, which I created specifically for this framework, is the duration of time a flow spends in the switch plane, specifically from switch to switch helps determine if a high volume of malicious data is utilizing the same flow table entry (i.e., many flows, but few packets).

\[
\text{Total Flow Time per Switch: } (T_{F_i} - T_{P_i}) \quad \text{Eq. 2}
\]

5.4.3.4 Hop Count

This is a simple metric and is available conventionally; however, the SDN controller could have a much more accurate estimation of the hop count per recipient. This estimation helps validate the path, eliminate routing to devices well outside of the network/system control, and has limited overhead. It is important to note that hop count can be changed by the controller/installed Flow Entry Action. Therefore, this would likely be a minor contributor to the overall metric analysis.
5.4.3.5 **Switch/Device Location**

This refers to the geospatial or at least time zone location of a device. This is used with varying degrees of trust to assess the strength of a partial route. Some larger level (ISP) or company internal switches would have a higher level of trust versus the open internet. Further, in a closed classified/sensitive network, the location could have more trust. Location data should be largely static, so the overhead of calculating the data and assessing a level of trust from proposed path versus the actual path should be minimal. Location data such as an authorized IP range can identify a switch/network owner, combined with an external entity like Internet Assigned Numbers Authority (IANA) or a business information technology (IT) department.

5.4.3.6 **Switch/Device Characteristics**

Knowing the type of switch, level within the LAN/wide area network (WAN) hierarchy, and switch owner is utilized to develop an algorithm for trust with the network. Physical vs. virtual switches – typically a physical switch would have more trust as it is harder to spoof. The level within the LAN/WAN hierarchy could be inferred by the controller’s view of the network topology.

5.4.3.7 **Average of Packets per Flow**

Data transmission is a two-way street, so it is equally as important to ensure the safety of the receiving node from malicious attack. Many attacks feature source IP spoofing, which makes the task of tracing the attack’s original source very difficult. A side effect is the generation of flows with a small number of packet, given that normal traffic usually involves a higher number of packets. If I can determine a median value for this, then I can assess confidence [15].
5.4.3.8 Median Bytes per Flow

The attack payload size is often very small (for example TCP flooding attacks typically contain packets of ~100 bytes). If I can determine a median value for this, then I can assess confidence. Equation 3 is borrowed from Guo’s research in attack detection [58].

\[
\text{md}(X) = \begin{cases} 
\frac{X(n+1)}{2}, & \text{if } n \text{ is odd} \\
\frac{X(n)+X\left(\frac{n+1}{2}\right)}{2}, & \text{otherwise}
\end{cases} \quad \text{Eq. 3}
\]

5.4.3.9 Growth of Single Flows

It is necessary to verify how many pair-flows occur in the flow stream during a certain interval. Malicious activity often increases the number of single-flows into the network because they send packets with a fake IP [15].

5.4.3.10 Packet Timestamp Comparison

By employing timestamping on the first packet of a given flow, I can assess exactly when traffic enters and exits SDN hybrid-network. Although latency may be the cause of this, any deviation would at a minimum degrade my confidence in the data path. Hashing the packet header and timestamp could serve additional purposes. For example, passing this hash to an authentication server (which knows the header) would serve to validate the packet based upon the returned timestamp.

5.4.3.11 Packet/Flow Lapse Time

This is defined as the time from when a packet arrives a switch until it arrives at the next switch. Measuring the packet’s arrival at two switches provides metrics that should be validated with performance traceroute to assess speed and detect man in the middle (MitM) attacks. Currently the OpenFlow 1.3 protocol does not support this metric collection.
\[ T_{p_x} = \text{Time Packet Arrives at Switch } x \]

To calculate lapse time, I created Equation 4 specifically for this framework to help identify issues within the network, by calculating the lapse time between the switches 1 and 4.

\[ (T_{p_2} - T_{p_1}) + (T_{p_4} - T_2) \quad \text{Eq. 4} \]

5.5 SDN Metric Evaluation and Experiments

5.5.1 SDN Metric Evaluation

Gathering network device data and being able to validate the data path is not the end point for SDN Data Path Confidence Assessment. Rather it is the beginning of a greater statistical review and analysis of the data collected. As mentioned in Section 5.3, there exists considerable research and standardization for measuring individual vulnerabilities utilizing the CVSS framework. Scores are calculated based on a formula that depends on several metrics that approximate confidence of a secure data path. Scores range from 0 to 10, with 10 being the least secure.

To measure the above metrics, I employ the following double weighted analysis: Base and Environmental. The Base group, like CVSS [55], represents the intrinsic qualities of security metrics and user-defined acceptable impact scoring, and the Environmental group represents the characteristics of the user’s network/data flow environment.

5.5.1.1 Base Group Methodology

For assessing the metrics in general, I propose the following criteria (aka SMV):

- S: Spoofability – measures the ability of the metric in general to be falsified in some manner
• M: Measurability – measures the degree of exactness that SDN allows for measure (whether subjective or objective metric)

• V: Variability – measures the range of acceptable values that would be considered within bounds for a given metric.

I worked with experts from industry and academia in the selection/creation of these three measures. A key influence is understanding how a network performance metric might be susceptible to attack. A baseline measurement for each of the previously identified metrics and a weighting standard is based upon two groups of input, see Table 5-1.

• A general survey of industry/academic professionals for their assessment of SMV for each metric – the current respondents are comprised of 40% IT Industry, 24% Academia, 18% Government, 18% Other Professional. Reference Appendix D for a copy of the survey and complete information from the survey respondents.

• The authors’ review of existing security and metering research. This area is highly influenced by the CVSS standard for exploitability scoring; however, it is a counter-balance (to avoid bias) with the survey above.

The SMV rating is based on a 0.0-1.0 scale. The lower the score, the greater the metric’s ability to predictably and objectively provide a higher security confidence. Each of the three values of SMV are assessed independently. For example, the metric Hop Count is alterable, but recognizable so it is measured at 0.61. However, Hop Count is easily assessed and has a small band of acceptable values so Measurability and Variability are scored at 0.48.
I referred to CVSS for insight regarding the numeric values and their range. The CVSS committee defined the acceptable numeric ranges for each severity level, then they collaborated with Deloitte & Touche LLP to adjust formula parameters to align metric combinations to the proposed CVSS severity ratings [55]. I followed the CVSS alignment in the creation of my SMV rating scale.

Table 5-1: SMV Rating Scale

<table>
<thead>
<tr>
<th>Scoring / Criteria</th>
<th>Spoofability</th>
<th>Measurability</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.35</td>
<td>Very difficult to spoof and/or easily recognized by most</td>
<td>Easily assessed metric with limited network overhead</td>
<td>Very narrow band of acceptable values</td>
</tr>
<tr>
<td>0.48</td>
<td>Hard to spoof and/or easily recognized by some</td>
<td>Easily assessed with moderate network overhead</td>
<td>Small band of acceptable values</td>
</tr>
<tr>
<td>0.61</td>
<td>Alterable and/or recognized with some training/effort</td>
<td>Assessable metric</td>
<td>Moderate band of acceptable values</td>
</tr>
<tr>
<td>0.66</td>
<td>Alterable by many and/or hard to recognize</td>
<td>Hard to assess and may reduce performance due to overhead</td>
<td>Large array of acceptable values within a single band</td>
</tr>
<tr>
<td>0.71</td>
<td>Easily altered and/or very difficult to recognize</td>
<td>Very difficult to assess metric with considerable overhead</td>
<td>Wide array of acceptable values, potentially in different bands</td>
</tr>
</tbody>
</table>

The Base Score is a combination of the SMV-weighted metric quality and the user/admin-defined impact of a compromised data path. I utilize modified-CVSS exploitability equations for calculating the value of the SMV weight and retain the impact metrics intact. In the context of data path confidence, it is important to consider impact from the perspective of the user and the sensitivity requirements for a specific transmission-
type or data-type. The coefficient values used in my experiments are a result of the SMV analysis/survey and the existing values of CVSS.

\[
Baseline = (0.6 \times Impact + 0.4 \times MetricQuality - 1.5) \times f(Impact) \quad \text{Eq. 5}
\]

\[
MetricQuality = 20 \times Spoofability \times Measurability \times Variability \quad \text{Eq. 6}
\]

\[
Impact = 10.41 \times (1 - (1 - ConfImpact) \times (1 - IntegImpact) \times (1 - AvailImpact)) \quad \text{Eq. 7}
\]

\[
f(Impact) = \begin{cases} 
0, & \text{if } Impact = 0 \\ 
1.176, & \text{otherwise} 
\end{cases} \quad \text{Eq. 8}
\]

In Equation 5 above, the CVSS measure Exploitability was replaced with my measure MetricQuality, a combination of SMV. All other aspects of the CVSS formula remain the same (i.e., 0.6 weighting on Impact, etc.) Equation 6, MetricQuality, is a modification of the CVSS metric reflecting a combination of SMV. Equations 7 and 8 are unchanged from the CVSS methodology.

Testing of the new SMV measures produced a similar range of variance to the CVSS measures (although CVSS ranges from 0.35 to 1) and SMV is from 0.35 to 0.71. This is largely due to each measure of SMV using the same scale; however, the CVSS measures of AccessComplexity, Authentication, and AccessVector do not follow this pattern.

An analysis of the impact of any one SMV measure could influence the overall single metric score by as much as 0.72. Although this factor seems small on a 0-10 Confidence Score, it proved to be substantial under normal network conditions. The impact of a change to any one measure within SMV is average 3.4\% to the single metric score and 0.5\% to the overall score. Figure 5-5 shows an example of the impact changing an individual measure of SMV, in this case Spoofability, can have on single metric scoring, as well as on the overall score. Although the overall score is not heavily impacted, Figure 5-5 illustrates
that an assessment change of two degrees can alter the confidence score from high to moderate.

SMV provides three benefits. First, it provides a standardized security metric score. A common scoring algorithm provides for a single management policy for assessing a metric’s quality for as a security indicator. Next, it provides an open framework. The scoring of analyzed metrics is transparent and configurable by the user. Lastly, SMV enables the prioritization of metric collection/analysis. As implementations of this framework are created, the computational impact of metric collection/analysis can be compared to the quality of the metric as a data path security indicator.

**Figure 5-5: Example Impact of SMV Assessment Changes**

Impact values are scored as high, medium, low and are scored 0.660, 0.275, and 0, respectively. All three components of the Impact score have identical definitions to their
CVSS counterparts [10]. For example, an analysis of the 3 of the 13 criteria is provided.

Table 5-2 illustrates the Base Group assessment of the metrics using a SMV analysis.

<table>
<thead>
<tr>
<th>#</th>
<th>Metric Name</th>
<th>MetricQuality</th>
<th>Impact</th>
<th>BaseScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flow Duration</td>
<td>2.21</td>
<td>6.44</td>
<td>3.82</td>
</tr>
<tr>
<td>2</td>
<td>Packet Size</td>
<td>1.18</td>
<td>6.44</td>
<td>3.34</td>
</tr>
<tr>
<td>3</td>
<td>Device Characteristics</td>
<td>2.22</td>
<td>6.44</td>
<td>3.83</td>
</tr>
</tbody>
</table>

5.5.1.2 Environmental Group Methodology

The BaseScore is weighted against the quality of the actual measurement and their relation to a user’s network. Drawing again on standardized CVSS equations, the Environmental group is reflected as a new value, EnvirScore. Many of the ‘user-defined’ factors of the EnvirScore are not necessarily end-user set, rather a combination of end-user and network administrator. More human testing is necessary to determine the exact amount of end-user customization versus central administration. Calculated by reassessing the weighting of the metrics based upon the user’s actual network (see example in Table 5-4 below); this new value is the Modified.MetricQuality. I again applied the coefficients and values from CVSS and my SMV survey.

\[
Modified\ BaseScore = (0.6 \times Impact + 0.4 \times Modified\ MetricQuality - 1.5) \times f(Impact) \quad Eq. 9
\]

\[
Modified\ MetricQuality = 20 \times Modified\ Spoofability \times Modified\ Measurability \times Modified\ Variability \quad Eq. 10
\]

\[
EnvirScore = (Modified\ BaseScore + (10 - Modified\ BaseScore) \times Collateral\ DamagePotential) \times Network\ Complexity \quad Eq. 11
\]
I modified Equations 9 and 10 of CVSS in a similar way to what was described above for Equations 5 and 6. Unlike CVSS, my framework does not contain a temporal, time base assessment, so I modified the Environment Score formula, Equation 11, to be based off of BaseScore instead. The lower the value of Modified.SMV, the better the metric at predicting a secure data path in that environment.

CollateralDamagePotential retains the same meaning and measurements as described in the CVSS documentation. I replaced the variable TargetDistribution with NetworkComplexity. This metric measures the user’s specific network, its complexity, and the level of control. It is scored as follows:

<table>
<thead>
<tr>
<th>Measure</th>
<th>Score</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>0.25</td>
<td>Small-scale, fully SDN controlled, LAN</td>
</tr>
<tr>
<td>Medium</td>
<td>0.75</td>
<td>Multi-site, single admin</td>
</tr>
<tr>
<td>High</td>
<td>1.00</td>
<td>Large-scale; diverse HW, SW, &amp; Management</td>
</tr>
<tr>
<td>Undefined</td>
<td>1.00</td>
<td>Internet-at-Large</td>
</tr>
</tbody>
</table>

5.5.1.3 Final SDN Data Path Confidence Methodology

The BaseScore and the EnvirScore are measured independently and can provide users and network administrators with an industry established norm and a network specific analysis of the overall security of a given data path. The confidence analysis can then be assessed to determine whether data decryption keys are provided. Each SDN metric has a range of 0-10, with 0 offering the highest level of confidence in data path security.
Table 5-4: Example of Environment Group

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Mod.MQ</th>
<th>Mod.BS</th>
<th>CDP*NC</th>
<th>EnvirScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Duration</td>
<td>2.21</td>
<td>3.82</td>
<td>0.075</td>
<td>4.29</td>
</tr>
<tr>
<td>Packet Size</td>
<td>1.18</td>
<td>3.34</td>
<td>0.075</td>
<td>3.84</td>
</tr>
<tr>
<td>Device Characteristics</td>
<td>2.22</td>
<td>3.83</td>
<td>0.075</td>
<td>4.29</td>
</tr>
</tbody>
</table>

The metrics are not weighed directly (i.e., packet size * weighting), but rather scored based upon their deviation of the median. Metric value expectation is determined by measuring the standard deviation of the pre/post/actual-traffic scoring and calculating degrees of sigma from these values (see Chapter 8, Section 4).

Table 5-5: Example of Final SDN Confidence Assessment

<table>
<thead>
<tr>
<th>#</th>
<th>Metric Name</th>
<th>Raw Score</th>
<th>Base Score</th>
<th>Envir Score</th>
<th>Final Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flow Duration</td>
<td>0-10</td>
<td>3.95</td>
<td>4.95</td>
<td>9.90-18.90</td>
</tr>
<tr>
<td>2</td>
<td>Packet Size</td>
<td>0-10</td>
<td>3.28</td>
<td>4.49</td>
<td>8.77-17.77</td>
</tr>
<tr>
<td>3</td>
<td>Device Characteristics</td>
<td>0-10</td>
<td>3.77</td>
<td>5.41</td>
<td>10.17-19.17</td>
</tr>
</tbody>
</table>

The results of the SDN Data Path Confidence Analysis can then be accessed as a service via a northbound controller API by any network tool or system for route verification and overall security of the data path.

5.5.2 Sample SDN Confidence Analysis

To better illustrate the confidence analysis process, I developed an application to collect three data points (flow duration, packet size, and switch characteristics) for analysis. The purpose of this collection is to provide an example of the confidence analysis concept which utilizes multiple metrics. See Appendix E for the complete data set of metrics collected [59].
Table 5-6: Flow Duration Confidence Matrix

<table>
<thead>
<tr>
<th></th>
<th>Host A-C (ms)</th>
<th>Std. Dev., s:</th>
<th>Total, N</th>
<th>Sum (ms):</th>
<th>Ratio of Std. Dev. ( n ),</th>
<th>Median (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Traffic</td>
<td>4.3980</td>
<td>0.0050332</td>
<td>1</td>
<td>13.192</td>
<td>2620.984</td>
<td>4.3980</td>
</tr>
<tr>
<td>Actual Traffic</td>
<td>4.4020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post-Traffic</td>
<td>4.3920</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After raw collection, the data was analyzed for the standard deviation, \( s \), between each point of collection. The ratio of deviation, \( n \), was then assessed as to whether it meets expected values. If the tool does not have historical data to baseline expected values, then the current average rating is assessed. This is a known vulnerability and thus prior to full scale use on a network it is recommended to conduct a series of initializing data transmissions.

Presently the analysis tool stores the deviation metric, the last three values, and the transmissions, with a confidence rating of ‘High’ or greater. These stored values are factored into future deviation scores. Long-term, I intend to apply machine learning to the deviation metric. Table 5-6 depicts this analysis for a given data transmission; similar matrices are generated for each metric. Table 5-7 shows the analysis of the sampled packet size as it traverses across from Host A to Host C.

Table 5-7: Packet Size Confidence Matrix

<table>
<thead>
<tr>
<th></th>
<th>Host A-C [Switch 1] (bytes)</th>
<th>Host A-C [Switch 2] (bytes)</th>
<th>Host A-C [Switch 4] (bytes)</th>
<th>Std. Dev., ( s )</th>
<th>Total, N</th>
<th>Sum (ms):</th>
<th>Ratio of Std. Dev. ( n ),</th>
<th>Median (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Traffic</td>
<td>1454</td>
<td>1443</td>
<td>1267</td>
<td>36.59</td>
<td>3</td>
<td>4287</td>
<td>117.16</td>
<td>4.3940</td>
</tr>
<tr>
<td>Actual Traffic</td>
<td>1446</td>
<td>1452</td>
<td>1332</td>
<td>35.36</td>
<td>3</td>
<td>4373</td>
<td>123.67</td>
<td></td>
</tr>
<tr>
<td>Post-Traffic</td>
<td>1387</td>
<td>1498</td>
<td>1466</td>
<td>101.4</td>
<td>3</td>
<td>4065</td>
<td>40.06</td>
<td></td>
</tr>
</tbody>
</table>
The raw data assessments are then weighted based upon the criteria listed above in Section 5.5.1. In keeping with the example above, I am applying the weighted values listed in Tables 5-2 and 5-4.

Table 5-8: Combined Confidence Analysis

<table>
<thead>
<tr>
<th>Metric / Run</th>
<th>Base Score</th>
<th>Envir Score</th>
<th>Final Weighted Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Duration</td>
<td>7.07</td>
<td>7.54</td>
<td>7.30</td>
</tr>
<tr>
<td>Packet Size</td>
<td>5.84</td>
<td>6.34</td>
<td>6.09</td>
</tr>
<tr>
<td>Device Characteristics</td>
<td>4.83</td>
<td>5.29</td>
<td>5.06</td>
</tr>
<tr>
<td>Combined Confidence Analysis</td>
<td>6.91</td>
<td>7.39</td>
<td>7.15</td>
</tr>
</tbody>
</table>

To derive a summary SDN data path confidence analysis, see Table 5-8, the average of the final weighted measures for each point of collection are assessed against an acceptable confidence matrix. The confidence analysis ranges correspond with that of CVSS on a scale of 0-10.

5.5.3 SDN Confidence Analysis Experiments

Returning to my original hypothesis that a SDN data path confidence analysis can identify security concerns within a given transmission, the following two experiments illustrate its effectiveness in (1) an in-depth single metric review and (2) a multi-factor analysis.

5.5.3.1 Single Metric Review

To evaluate the performance of the framework, I conducted numerous tests under normal conditions to evaluate flow duration on my testbed. The individual points of data
were compared against prior simulations (a baseline for the data flow) to provide a confidence analysis for each switch in each simulation, see Appendix E.

Following initialization of the tool, I utilized the results of prior university research on the impact of MitM attacks on network timing [60] to emulate an “unhealthy” network state. They observed data transfer timing on 20+ popular public facing websites located across the globe. When using the Cain and Abel man-in-the-middle attack tool there was a median delay of 50ms and a maximum delay of 242ms. I applied these results to a group of varying-sized data transmission (5Mb, 10Mb, 20Mb, 50Mb). I conducted my experiments by introducing increasing levels of switch latency to simulate malicious activity on a switch. Figure 5-6 illustrates the impact of the delay when introduced into the confidence analysis tool. From the graph, you can see the impact malicious activity induced latency (0ms, 50ms, 100ms, 200ms) has on the overall confidence score. The graph not only displays the impact of prolonged switch delay, it also illustrates the linear impact (green trend line below) on both increasing the file size as latency/malicious activity increases. The graph shows as attack induced latency reaches near 50ms, the overall confidence score transition from high to moderate. My experiments validate my hypothesis that as network anomalies, in this case latency, increase there is a linear impact to my confidence assessment regardless of the flow/packet size.
5.5.3.2 Multi-Factor Analysis

It is simple enough to assess the impact of a single metric, but let’s assume the malicious activity is clever and advanced in their implementation. How well can my SDN Data Path Confidence Analysis measure multiple metrics and only a single anomaly?
In this scenario, I assume a similar MitM malicious activity, as well as measuring an additional five metrics (packet size, flow size, device characteristics, hop count, and average packets per flow).

The results in Figure 5-7 shows that initially the single flow duration metric analysis is impacted to a greater degree (by 5%) than the overall analysis; however as malicious activity increases, the impact on overall confidence compensates for the singularly impacted metric. As latency reaches it maximum, the overall confidence score reflects an impact of 26% over the single metric.

5.6 Conclusion

My objective was to help users and administrators achieve the desired security and dependability characteristics of specific network data paths. I address several techniques to verify and validate data transmissions over a specific network path using SDN and traditional network metrics with the desired outcome being that sensitive data is transmitted to the right person, at the right place, at the right time over a known and verifiable network path. My SDN Data Path Confidence Analysis framework provides a framework to gather, analyze and store a set of network behavior patterns. This framework also provides tools that verify and validate network functionality. Using these tools, this service provides an analysis of the security and dependability of a specific network path or segment.
CHAPTER 6

SDN SECURITY RESEARCH MODELS
AND SIMULATION

With a framework in place, the need for a variety of model and simulation platforms is evident. This chapter focuses identifying and developing various testing environments.

6.1 Contribution

In this chapter I provide a series of models and simulations to be utilized for testing any number of SDN research projects. Additionally, I discuss the server testbed that was created at University of Colorado, Colorado Springs (UCCS) and how to build a simple SDN/OpenFlow-enabled router. Finally, I provide information and use-cases for joining the network consortium PlanetLab.

6.2 Large-Scale Modelling and Simulation

One of the early problems identified by SDN/OpenFlow researchers was an attempt to develop and test new Internet protocols on large scale networks. Their initial research work went by the name of Ethane, a way to enable global policy throughout a network. Although Ethane was ultimately absorbed into the OpenFlow protocol, the goal to allow for large-scale testing environments has yet to be realized. As mentioned previously, few tools are available to effectively model, simulate, and potentially permit emulation for SDN/OpenFlow research. The complexity, cost, and speed of the current networking model, see Figure 6-1, needs a 21st century refresh.
6.2.1 Hierarchical

As discussed previously in my exploratory research, there are limitations to the various M&S tools. EstiNet provides the most robust set of M&S and emulation options, but is limited to ~200 node simulations. To get around this limitation and build better large-scale models I employed a hierarchical approach, like the traditional networking approach of Philip Huynh in NS-3 [62]. After discussing it with EstiNet developers, I proposed two approaches for helping solve this problem. However, I did not pursue either of these approaches due to licensing issues/costs with EstiNet. As a free and open source alternative to EstiNet, I developed and tested several custom Mininet topologies to model a data center. Reference Appendix F for tutorials for two different data center hierarchical simulations (one simple and one more realistic).
6.2.2 Network Slicing

The idea for this model is to allow for slicing of the SDN for different traffic and user-types, as well as multiple controllers. FlowVisor creates rich slices of network resources and delegates control of each slice to a different controller [63]. Slices can be defined by any combination of switch ports (Layer 1), src/dst ethernet address or type (Layer 2), src/dst IP address or type (Layer 3), and src/dst TCP/UDP port or ICMP code/type (Layer 4). FlowVisor enforces isolation between each slice, i.e., one slice cannot control another's traffic [63]. Developing a FlowVisor model for my research prepares it for an opportunity of deploying it on a slice of production networks or a large-scale testbed, like PlanetLab.
6.2.3 Emulation with a Testbed

Although I initially intended to capitalize on EstiNet’s ability to allow for real-life networking device exchange of packets (e.g., set up a TCP connection) with nodes (e.g., host, router, or mobile station) in a network simulated by EstiNet, due to reasons mentioned above I developed an approach for a similar capability through open-source products. This feature is very useful as the function and performance of real-world networking devices can be tested under various simulated network conditions. Reference Appendix F for tutorials for SDN emulation [65].

After reading through these use-cases, it is obvious there many more complex simulations involving multiple attributes of the use-cases listed in Chapter 4. Certainly, real-world complex networks would likely involve elements of these scenarios. It is
necessary to isolate the various use cases for testing and validation purposes, but also a requirement to develop the most realistic M&S environments possible.

6.3 University Testbed

Building upon the M&S, a University SDN/OpenFlow testbed provides the physical infrastructure to experiment upon. Large-scale M&S provide repeatability and scale that a testbed cannot offer; however, it is essential to validate research across physical hardware. A UCCS testbed is not a single thing, but rather a series of testbeds each giving researchers a wide range of environments in which to develop, debug, and evaluate their systems.

Like the M&Ss described above, UCCS testbeds were developed to meet different research criteria and challenges. Early testbeds include simple OpenFlow topologies utilizing virtual machine, software-based OpenFlow switches, and a single subnet. Subsequent testbeds incorporated more physical hosts, switches, and subnets. See Appendix G for information of the setup of the UCCS testbed.

This testbed development is based upon virtualization using a VMware vSphere Server and a series of virtual machine hosts, software switches, and controllers. For those who want to experience basic OpenFlow functionality without committing much hardware resources, virtual machines provide a viable alternative. Figure 6-4 is such network that I created for evaluating software OpenFlow switch.
Moving to a more advanced model to have different Hypervisors using a common Virtual Switch using an OpenFlow controller and a tunneling method to virtualize the network, reference Figure 6-5.
Going beyond a virtualized testbed was more difficult, because unfortunately most of the SDN/OpenFlow hardware currently available is designed for industry use (+$10,000 per switch). To avoid the large expense and possible lack of use of these commercial grade switches (>100 ports), I researched a method to utilize existing open-source switch firmware and the SDN open-source switching software to create cost-effective hardware for the university testbed. My research indicates that OpenWRT [67] and DD-WRT [68] allow for some modification to their code-base to allow OpenvSwitch [26] to be ported. This allows for very inexpensive wireless SDN testing. There are some limitations to upgrading firmware and not all consumer grade switch hardware support the conversion. See Appendix H for information on OpenWRT setup.

6.4 PlanetLab Research Network Investigation

6.4.1 Introduction and Motivation

PlanetLab is a global research network that supports the development of new network services. Although joining the PlanetLab is not a direct portion of my research, I investigated the processes and requirements for joining the network.

The need for security research, specifically SDN, is only growing. The volume of DDoS attacks is constantly growing – as of March 2017 volume reached 1.7 Tbps. SDN provides potential for greater network state visibility and control.

As UCCS increases its research in network security the need for large-scale testbed continues to grow. Joining PlanetLab would provide the university with:

- Access to large scale distributed resources
- Ability to run experiments with complete control over each node
- Scale from one to few to many nodes
- Monitor CPU and network traffic
- Deploy long-running experimental services

6.4.2 Overview of PlanetLab

Since the beginning of 2003, more than 1,000 researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing [69]. PlanetLab, as of March 2018, consists of 1353 nodes at 717 sites, with over 600 active research projects running. Corporate sponsorship for PlanetLab includes: Intel, Hewlett Packard, Google, AT&T, France Telecom, AT Corporation, DoCoMo Communications Laboratories USA, Lucent - Bell Labs, NEC Laboratories, and Telecom Italia. Government sponsorship includes NSF and DARPA.

PlanetLab is used for deploying widely distributed services in relative isolation, evaluating competing approaches in a realistic setting and evolving the network architecture to better support such services. However, PlanetLab is not intended to be a platform for a distributed supercomputer, a simulation platform, an Internet emulator, an arena for repeatable experiments, representative of the current Internet, or a power grid.

6.4.3 How It Works and Current SDN Research

Before an understanding of PlanetLab can be achieved, a few terms need to be defined.

- **Node**: A dedicated server that runs components of PlanetLab services
- **Slice**: A set of allocated resources distributed across PlanetLab. To most users, a slice means UNIX shell access to private virtual servers on the same number of PlanetLab nodes. After being assigned to a slice, a user may
assign nodes to it. Slices may be assigned to a user selected set of PlanetLab nodes. After nodes have been assigned to a slice, virtual servers for that slice are created on each of the assigned nodes. Slices have a finite lifetime and must be periodically renewed to remain valid. All data associated with a slice is deleted when the slice expires.

- **Sliver**: A slice running on a specific node. You can use ssh to login to a sliver on a specific node.

In a distributed virtualization environment like PlanetLab, you want to isolate from other activities on those nodes on which you run. PlanetLab provides a level of isolation which gives you your own file system and process control. You share processing cycles and network bandwidth with other active slivers on each node. The concept of slice aggregates the presence of your slivers within the system.

---

**Figure 6-6: PlanetLab – How It Works [69]**
6.4.4 Consortium Membership

6.4.4.1 Consortium Membership Levels

There are several levels of membership within PlanetLab [69]; I am focused primarily on the Academic-level. I believe this is a good fit for UCCS and we are a good candidate for this level of participation.

- Charter ($300k annual dues)
  - Permanent seat on Steering Committee.
  - Unlimited number of slices.
  - Access to PlanetLab events, research papers, and working groups.

- Full ($75k annual dues)
  - Rotating seat on Steering Committee.
  - 10 slices.
  - Access to PlanetLab events, research papers, and working groups.

- Associate ($25k annual dues)
  - 2 slices.
  - Access to PlanetLab events, research papers, and working groups.

- Sponsor ($10k annual dues)
  - Access to PlanetLab events and research papers.

- Academic (no annual dues)
  - Seat on Steering Committee by invitation.
  - 10 slices.
  - Access to PlanetLab events, research papers, and working groups.
6.4.4.2 Consortium Membership Requirements

6.4.4.2.1 Personnel

• Administrative Contact: business contact to handle contracts, invoices, etc.

• Principal Investigator: accepts responsibility for researchers at your site. Often a professor / lead researcher; must be employee of institution.

• Authorized Official: can bind your institution contractually/legally; a provost or contracting officer.

• Technical Contact: contact when a node goes down or when an incident occurs. Commonly a system administrator or graduate student.

6.4.4.2.2 Materials

• Minimum 2 Nodes
  • 6 cores
  • 24GB RAM
  • 2TB Storage
  • PlanetLab Linux image
  • PCU with remote-access capability

• Networking
  • 400kbps minimum of bi-directional bandwidth
  • Recommend outside IDS/Firewall in demilitarized zone (DMZ)

6.4.4.3 Consortium Membership Cost Estimate

The following estimates were developed in June 2014 and assume Dell R320 server hardware and local CenturyLink ISP service pricing.
6.4.4.3.1 One-Time Expenses

Total: $5700

- Node Hardware ($2100/per node)
- Network Hardware ($500)
- System Integration / Local Sys Admin ($1000)
  - 40 hours

6.4.4.3.2 Recurrent

Total: $1160/yr.

- Network Bandwidth ($360/yr.)
  - 12Mpbs
- Sys Admin Maintenance ($200/yr.)
  - 8 hours / month
- Utilities ($600/yr.)

6.4.4.4 Consortium Membership Process

6.4.4.4.1 Step 1: Review Documents and Apply for Membership

- Review Documents:
  - Consortium Government Plan
  - Hosting Requirements
  - Acceptable Use Policy
  - Terms and Conditions of Membership
- Submit Application with Key Personnel Identified
6.4.4.2 Step 2: Connect Machines

- Minimum two Server “Nodes”
- Static IPs – outside firewalls, not network address translation (NAT’ed) or subject to many traffic restrictions
- Allow for remote management

6.4.4.3 Consortium Hosting Site Responsibilities

- Provide IP connectivity for the node, including a single static IP address and a DNS name (including both forward and reverse lookup).
- Place the nodes outside the local firewall, in a network DMZ. This implies not filtering traffic into and out of PlanetLab nodes. In general, sites should take reasonable steps to isolate their PlanetLab nodes from the rest of their institution's computer systems.
- Allow the PlanetLab operations team to administer the node, including have root access, install and maintain the operating system, and set up research accounts. Local administrators do not have root on PlanetLab nodes, but they do have several administrator capabilities, as described below.
- Define a point-of-contact that can be called to re-boot a PlanetLab node.
- Forward complaints from external system administrators to the PlanetLab operations team.
- Enforce the PlanetLab Acceptable Use Policy (AUP) regarding the actions of local users on any PlanetLab machine, whether hosted locally or at another institution. The PlanetLab community relies on each hosting site to stop unacceptable activity originating at that site.
6.4.4.4 PlanetLab Ops Team Responsibilities

The PlanetLab operations team must install software on PlanetLab nodes that enforces constraints on application programs, thereby limiting their effect on other network users. These constraints include:

- Limit outgoing network bandwidth. The local system administrator can set the total outgoing bandwidth that can be consumed by the PlanetLab nodes they host.
- Filter packets addressed to certain destinations. Our policy is to not filter outgoing packets unless explicitly asked to do so by a network administrator that believes his or her network has been "attacked" from a PlanetLab node.
- Not allow applications to spoof IP addresses, or send well-known bad packets (e.g., "ping of death").
- Limit the rate at which probe packets and other potentially disruptive packets leave the site. The PlanetLab operations team establishes limits that are consistent with Internet norms.

6.4.4.5 Consortium Membership Security and Integrity Policies

- No users other than the PlanetLab operations team have root access to PlanetLab nodes.
- To reduce the chance of a remote root exploit, all PlanetLab nodes run only a limited set of remotely accessible system services as root. All other standard system services are disabled. Services that are enabled on the nodes include SSH (RSA authentication only), HTTP, and finger. For SSH, we use OpenSSH v3.4, which is free of any known security vulnerabilities. For HTTP and finger, we use bare-minimum daemons (both a page of Python code) that responds to all incoming connections the same way: by
returning a single file with contact information. (Both HTTP and finger will soon be running as the user nobody to further reduce the chance of a remote exploit.)

- To reduce the chance of a local root exploit, all nodes are kept up-to-date with security patches. PlanetLab nodes currently run a Linux distribution that is largely based on Fedora Core 4. The operations team keep track of the latest security patches and update all the nodes. They also track security advisories, and security vulnerabilities posted to security mailing lists.

- To further reduce the chance of a local root exploit, remote access to PlanetLab nodes is done using sandboxed execution environments. These execution environments are chroot'ed and further constrained by also limiting the set of processes, network interfaces, and so on, that can be accessed with a sandboxed execution environment. In summary, once you're placed in a sandbox, the scope of your activities is limited to that sandbox. Therefore, even if an account is compromised, a hacker still won't have access to root on the machine, and the limitations outlined above are enforced.

- Monitoring software installed on nodes provides an audit trail in the event of a security breach.

- To respond to security issues and potential security breaches in a timely manner, report incidents to the PlanetLab operations team.

- To reduce the opportunity for unknown users to abuse PlanetLab services, the PlanetLab operations team reserves the right to restrict end-users (clients of services running on PlanetLab) to those affiliated with PlanetLab sites.
6.4.5 Current SDN Research Projects using PlanetLab

6.4.5.1 OpenFlow on PlanetLab

PlanetLab nodes can be used as OpenFlow switches with the help of OpenvSwitch. OpenFlow experiments are invoked and configured via Vsys. To create an OpenFlow experiment you first need to obtain a set of privileges, create an OpenvSwitch datapath and provide the identifier of this data path to other users who would like to add ports to your switch. You can also use this identifier to add your own virtual or physical network interfaces as ports. Finally, you can configure the data path to be controlled by an OpenFlow controller running in your slice [70].

6.4.5.2 NMSL - Efficient Media Multicast Using OpenFlow Switches

High-quality video streaming, which once dictated high-end equipment, are gradually available on commodity workstations and networks. For example, multiparty video conferencing calls are enabled by Google+, iChat, and Skype, and large-scale video surveillance, say city-wide, networks have been deployed in many countries. While these applications can largely benefit from Layer 3 multicast, very few Internet service providers enable the multicast feature on their routers, perhaps for the sake of maintenance overhead. In this project, we study how to incorporate OpenFlow in current network infrastructure, either within LANs or WANs, to enable centralized multicast support for bandwidth-hungry multimedia applications. With OpenFlow, the access or aggregation routers may help multiparty video conferencing clients to duplicate or transcode video streams, to relieve the narrow access links from carrying multiple video streams. This in turn leads to multiparty video conferences with higher video quality, lower interactive latency, and better system scalability. Moreover, compared to full-fledged routers, using less expensive
OpenFlow switches significantly reduces the cost of deploying video surveillance networks [71].

6.4.5.3 Indrakshi Ray and Indrajit Ray at Colorado State University (CSU)

An increasing amount of personal data is automatically gathered and stored on servers by administrations, hospitals, insurance companies, etc. Citizens themselves often count on Internet companies to store their data and make them reliable and highly available through the Internet. However, these benefits must be weighed against privacy risks incurred by centralization. The project suggests a radically different way of considering the management of personal data [72]. By embedding a full-fledged Personal Data Server in mobile devices, user control of how his/her sensitive data is shared by others (by whom, for how long, according to which rule, for which purpose) can be fully reestablished and convincingly enforced. To give sense to this vision, Personal Data Servers must be able to interoperate with external servers and must provide traditional database services like durability, availability, query facilities, and transactions.
CHAPTER 7

SDN ON-THE-GO PHYSICAL TESTBED

In addition to modeling and simulation, a physical testbed for repeatable research was determined be of use in SDN testing. This chapter describes a specific portable, low-cost SDN testbed implementation. The initial findings of this research were presented at the International Conference on Dependable Systems and Networks (DSN) [73]. The final results of this research were presented at the IEEE Conference on Dependable and Secure Computing (DSC) in 2017 [74].

7.1 Contributions

My solution is to build a low-cost, portable, standalone SDN testbed. Called SDN On-The-Go (OTG), it is a complete, self-contained testbed that consists of four dedicated Zodiac FX SDN switches, four RaspberryPi3 hosts, a dedicated Kangaroo+ controller with 4GB RAM and two routers to form the network isolation. The testbed supports many configurations for pseudo real-world SDN experiments that produce reliable and repeatable results. It can be used as a standalone research tool or as part of a larger network with production quality traffic. SDN OTG is designed to be used as a portable teaching device, moved from classroom to classroom or taken home for private research. I achieved the repeatability factor of an order of magnitude greater than emulation-based testing. The SDN OTG physical testbed weighs only twenty pounds, costs about a thousand US dollars, provides repeatable, precise, time-sensitive data and can be setup as a fully functional SDN testbed in a matter of minutes.
7.2 Introduction

An extension or amalgamation of earlier proposals, Software Defined Networking is like programmable networks and control-plane separation projects of the past [75]. Gaining momentum, SDN has been adopted by academia and industry as the next programmable network [76]. The rise in popularity of SDN amongst researchers has increased the need for adequate SDN testing facilities [77]. The network infrastructure industry is responding with embedded support for SDN in their network hardware, but their response is slow and includes the inevitable implementation differences and proprietary bias [78]. Commercial SDN hardware costs thousands of dollars per switch, which for many researchers is cost prohibitive to building a physical testbed with embedded SDN support. While this expense may be affordable in larger research centers or in community networks like PlanetLab [69] or GENI [79], for many researchers these facilities are expensive to join, subject to community schedules, and complex to implement jobs and verify results [77].

SDN research requires a detailed and accurate environment for experimentation [77]. Many researchers that lack access to a large research center with expensive SDN enabled hardware or find community SDN networks untenable are limited to software network simulation tools such as Mininet [34], NS3 or EstiNet [37] [80].

While these simulation tools play a crucial role in testing new and innovative research, many SDN experiments and applications depend on accurate and repeatable metrics. It is not a new conclusion to state that simulation can provide an understanding of temporal relationships in networking, but arguably does not adequately emulate real-world behavior. Simulation also tends to become a hidden mess of virtual devices, each effected by the
other in ways that are unpredictable, difficult to troubleshoot and not representative of a real-world network [75] [80].

I propose an alternative, SDN On-The-Go (OTG), a low-cost, portable, standalone, four switch, four host network testbed with a dedicated SDN controller. It is based on the Zodiac FX [11] SDN switches, RaspberryPi3 [81] hosts and Kangaroo+ [82] controller. It is a highly configurable testbed that can support a wide variety of pseudo real-world experiments running any of the SDN controllers including ODL [56]. My implementation fits into a small carrying case, weighs less than twenty pounds and is a complete, standalone SDN testbed.

My SDN OTG testbed is designed with academia in mind. It is a self-contained network testbed that can be used as a portable teaching device, moved from classroom to classroom or taken home for private research. Ease of setup is also a key requirement; therefore, I built it as a standalone network with a monitor, keyboard and mouse to locally control the four RaspberryPi3s and Kangaroo+ computers. The switch board, host board, routers and controller all remain connected and fold together to fit inside the carry case. SDN OTG also has an isolated public network that enables remote control of all four hosts and switches as well as the controller.
7.3 Background and Current SDN Testbed Research

7.3.1 Background

Software Defined Networks create a paradigm shift in traditional network architecture and management [76] [75] [83]. To understand the shortcomings of traditional networks and how SDN can mitigate these shortcomings, we must explore the networking model. A computer network can be logically divided into two basic parts; the switch plane and the
control plane. The switch plane is the set of hardware switches and routers that form the network. The control plane is the decision-making process that resolves what switch to switch path that a data flow will take across the network [48]. A data flow, or a series of packets that make up a single transmission of data, is sent from switch to switch across the network from the source address to the destination address. In traditional networks, data traverses across the network according to the rules and logic defined on each switch in the switch plane. These rules and logic control how data is transferred from switch to switch and is referred to as the “control plane” of a network. So, with the “control” decisions being made on each individual switch, the control plane is said to be distributed on the switch plane [48]. Hence, much of the control (data flow, configuration and management) of the network is relegated to each switch and the switch’s statically configured rules and logic. This contributes to making traditional network configuration and management tedious and error prone [78].

With the switching logic (control plane) on each switch, once a data flow enters the switch plane, there is little ability to control the actual physical path of the data flow or the behavior of that flow across the network [48]. As designed, traditional networks cannot easily specify the exact network path that data will traverse. This can be very important for some types of data. A data center or ISP example is Quality of Service (QoS) agreements on certain types of data streams. Another QoS policy might be that sensitive data such as medical records must traverse across a known and secure network path and must not be stored outside secure in-country storage. The list of QoS qualifications is continuing to grow. The architecture of traditional networks allowed for autonomous processing and control of the network for it to function. To fulfill these QoS qualifications
I need to know far more about a network than the traditional architecture can provide. There are a plethora of protocols and tools built on top of traditional network design that enable some of this insight into the network, but not easily and many times these design extensions depend on other design extensions, further complicating what should be a basic network function. Consumers demand these qualities of service but extending traditional networks in this fashion quickly becomes a configuration and management burden.

Traditional network design can provide some of these QoS policies, but the methods to do so are an amalgamation of protocols that at best are extensions of network designs, not implementations of a baked-in policy enforcement mechanism. By centralizing the control or decision-making process (the how, where and when routing and logical decisions are made) about a data flow I gain a more granular and feature-rich control over how data is transferred across the network. This control is enabled by SDN and as a function of SDN network design, not an amalgamation of traditional extensions and protocols [78].

Software-simulated networks are limited by the constraints of the host and outside demands for host resources. Mininet is an extremely useful software tool for simulating networks. Mininet uses Linux process-based virtualization on a single OS kernel to run many hosts and switches. Communication and functionality are also provided by the same OS. Using network namespaces, Mininet can isolate these processes and assign separate network interfaces, routing and ARP tables to virtual devices such as hosts or switches. These virtual network devices are limited by the CPU and memory resources of the single OS kernel. While scaling the system or adding skew values can lessen the impact of these limitations, there will always be processing overhead with unexpected events and resource limitations that restrict simulation accuracy. Mininet also has a weak concept of virtual
time as a part of real time so event timing can vary depending on system load. Mininet uses OpenvSwitch switches running on the same OS as the virtual hosts, so all the hosts and switches have similar round-trip time (RTT) between each other and the controller. Also, buffering by the OS compounds the timing issues between devices leaving it very difficult to accurately simulate real-world timing metrics. Due to these interdependencies and the unpredictable behavior of the OS, accurate, repeatable, real-world network behavior is very difficult to simulate [80] [84] [85] [86].

7.3.2 Current SDN Testbed Research

In 2014, researchers from Ajou University developed a very small-scale physical SDN testbed [87]; however, their work was limited to utilizing OpenvSwitch and non-dedicated switching hardware. Additionally, their comparison of a physical testbed versus a simulation was limited to throughput (i.e., performance) and not reliability and repeatability of the testing data.

Following up on this work in 2015, researchers from the University of Peradeniya, developed a larger scale SDN testbed [88]. Their goal was like the Ajou University work, although they did not provide an evaluation data with their testbed proposal.

Neither University provided resources for the replication of their testbed or the use of dedicated SDN hardware.

7.4 SDN On-the-Go Testbed Architecture

7.4.1 Switch Plane

The switch plane section of the testbed consists of four Zodiac FX switches based on the Atmel ARM® Cortex®-M4 processor-based microcontroller, programmed for the OpenFlow protocol [89]. Each switch has four 10/100Mbs Ethernet ports; three isolated
SDN virtual local area networks (VLANs) and the fourth is an isolated controller VLAN in the Out-of-Band (OOB) “Ships in the Night” deployment model for an OpenFlow network [90]. The manufacturer notes that an in-band controller configuration might be possible with some modifications to the source code [91].

The switch plane can be configured in any way that is supported by four, four port OOB OpenFlow switches. At a minimum, four switches are needed to support the major network topologies. The switch plane can be configured in Linear, Ring, Star, Bus, Tree and Mesh topologies. Four of these four-port OpenFlow switches are enough to create these topologies. The OTG default topology is a partial mesh, which provides two, three-switch routes between all four host nodes.

7.4.2 Control Plane

The control plane VLAN is formed by the dedicated port 4 on each Zodiac FX switch in conjunction with the SDN concentrator hub. The controller machine is a Kangaroo+ by InFocus, 1.44 GHz 64-bit quad-core Atom x5 Z8300, 4GB RAM; the primary Ethernet port, eth0, is connected to a NetGear WNDR2000v5 LAN2. A secondary Ethernet port via USB conversion is connected to external LAN3 for remote access.

The SDN OTG Controller is highly configurable and customizable thanks in large part to the Kangaroo’s 4GB of RAM. In my default configuration, I utilize ODL version Boron-SR2 and following components:

- Basic ODL functionality:
  - odl-l2switch – the ODL default switch
  - odl-dlux – provides the DLUX web UI
  - odl-restconf – provides a REST interface
- odl-openflowplugin-flow-services – provides flow services

- Time Series Data Repository:
  - odl-tsdr-core – datastore and collectors
  - odl-tsdr-hbase – default Hadoop database
  - odl-tsdr-openflow-statistics-collector – one of many data collector options

![Kangaroo+](image)

**Figure 7-2: SDN OTG ODL Controller Model [16]**

### 7.4.3 External Network

The SDN OTG testbed is a standalone network. In the default configuration, the four RaspberryPi3 hosts’ secondary network are connected to a separate virtual private network (VPN). There are two conventional routers that can be used in the switch and control plane. These routers can be configured to represent most common (and not so common) small network architectures, but the default configuration provides a public facing VPN to allow
remote control of the hosts and controller. To add another dimension to the configuration
the Kangaroo+ controller, RaspberryPi3s and the two routers are wireless and Bluetooth
capable.

7.4.4 Configuration Examples

OTG is a general-purpose implementation that can be extended to include various
topologies, controllers, switches, and other hardware and configurations. For example, all
the topologies described in the Switch Plane section are supported by the SDN testbed.
These configurations, linear, star, tree, etc., can be extended using the three routers on the
main board. Two of the three routers have a wireless radio making hybrid, ad hoc wireless
network configurations possible. There are spare Ethernet ports on the routers for
extending the configuration. The Controller, Kangaroo+ computer and the RaspberryPi3
are wireless and Bluetooth capable. Four hosts, four switches and three routers on three
independent transmission spectrums can create many significant and challenging
topologies. The following are a few example topologies that can be created with the OTG:
Figure 7-3: 4 Switch, 2 Host Partial Mesh

Figure 7-4: 4 Switch, 4 Host with Direct WAN Access
7.4.5 SDN OTG Default Configuration

7.4.5.1 LAN1

SDN Network consists of four Zodiac FX switches connected in a partial mesh configuration. Each switch Port 1 is connected to the primary Ethernet port on the RaspberryPi3.

7.4.5.2 LAN2

Port 4 of each Zodiac FX switch is a dedicated VLAN for the control plane. These ports are connected to a DLink DL1005E hub which is connected to a NetGear WNR2000v5 router, 10.0.1.0/24. The controller computer is connected through its primary eth0 to this NetGear router, completing the control plane.
7.4.5.3 LAN3

External LAN for remote control of each host and the controller using USB-ethernet converters connected to a NetGear WNR5000v5 switch. External devices are connected to this switch.

7.5 Testbed Implementation

7.5.1 Testbed Components

The standalone testbed is designed to be easy for one person to carry and setup in a few minutes. All components and boards can be mounted on the 2’x3’ vertical display board for the purposes of teaching or demonstration. The switch board and the host board attach to the display board via a simple bar/catch. The monitor slips into a bracket and the Kangaroo+ slips into a fan cooled mount.

Almost all the cabling can remain hooked up when the unit is disassembled and stored. The switch and host board fold together and fit in a small hard sided case without unhooking most of the cabling. The controller, monitor, routers, keyboard, mouse, power supplies and spare cabling fit in a soft-sided bag, reference Figure 7-6.
The many power supplies remain connected to three, fused, 120V power strips so the unit is powered by a single 120V cable and the power cabling can be left connected while disassembled.

In addition to portability and functionality, low-cost is also a concern. The OTG components can be purchased for approximately one thousand US dollars. For a complete component list and more installation details, see Appendix I.
7.5.1.1 Zodiac FX Switch Board

The switch board consists of the four Zodiac FX switches, a five-port hub and a five-port router. The four Zodiac switches are arranged in two rows of two pointing in the same direction. This provides a more simplified wiring arrangement for different network topologies. The hub and router are mounted to the right of the switches. The hub acts as a concentrator for the SDN network. The router acts as a concentrator and provides a separate VPN for the four Raspberry Pi3 hosts. I use this VPN as the public network access point. This wiring can be removed without effecting the capabilities of the testbed.

In the default configuration (reference Figure 7-7 and 7-8), each Raspberry Pi3 host is connected to its own switch, providing 5V power via USB to the micro-USB on the switch.
(i.e., h1-s1, h2-s2, etc.). The Zodiac FX provides serial connectivity through this USB port for changing the switch configuration. Each Zodiac FX also has a web interface. This way each host powers a switch and is responsible for configuring the switch via the serial USB connection or web interface. Using the on-board monitor, keyboard and mouse you can change the configuration on each switch and the controller. If the dedicated VPN is hooked up, you can remote control each host and the controller to change configuration.

Figure 7-8: Zodiac FX Switch Board

7.5.1.2 Host Board

With the four RaspberryPi3 hosts, a variety of pseudo-real-world network traffic can be generated at any given time over the configured network, see Figure 7-9. The default configuration connects host Ethernet 0 to port 1 on each switch. Each host connects to a combination USB keyboard/mouse switch, (lower right-hand corner of Figure 7-9) to
provide switching between the Logitech MK240 2.4 GHz Wireless keyboard and mouse. Each host connects HDMI video output to an IOGear IR HDMI switch (upper right) which drives the 10” monitor.

![Figure 7-9: OTG Host Board](image)

**7.5.1.3 Kangaroo+ Controller**

The Kangaroo+ is mounted to the main board on the left of the monitor, inside the well-constructed box. It does generate heat, so I installed a small 5V blower fan in the box with ducting to inject air into the side cooling vents of the Kangaroo+. The Kangaroo+ can be removed from its fan cooled mount, if needed.

Like the four hosts, the Controller can be used with the on-board keyboard/mouse/monitor or you can remote control from an external network. Laptops and other devices can also be plugged directly into the main board public network.
7.5.1.4 Routers

Two routers with Wi-Fi form the subnets for the testbed, see Figure 7-10. These routers can be configured with the wired or wireless subnets or both. The routers, RaspberryPi3 hosts and Kangaroo+ controller are also wireless and Bluetooth capable.

![Figure 7-10: Routers with Wi-Fi](image)

7.6 Testbed Performance Evaluation

SDN simulators, Mininet or otherwise, use native virtualization techniques to create the virtual components which function as the individual hardware components (i.e., network switches or host computers) of the network. Typically, these simulations utilize the virtualization infrastructure provided by the Linux kernel-based virtual machine (KVM). KVM uses hardware virtualization extensions to simulate the complete virtualization of the hardware environment where an unmodified guest operating system
can run in isolation using the same instruction set as the host machine. It provides an interface (\(/dev/kvm\)) to these hardware extensions so each component can create network namespaces and assign separate network interfaces and routing tables that use the host’s network stack. While each virtual component exists in process isolation, they lack a unique clock to precisely control the execution of events. Therefore, each component relies on the host’s kernel CPU scheduler to order those events. As the virtual components compete for the same host resources such as CPU cycles, memory and input/output, timing-related experiments pose a serious reliability problem for researchers when trying to represent real-world timing metrics [75] [80] [84] [85] [86].

To better illustrate the need for repeatability, I ran a large number of experiments using Mininet on a variety of hardware and software configurations to include (1) Linux OS running on a 1.44GHz 64-bit quad-core Atom x5 Z8300 with 4GB RAM, (2) Linux running on a 2.90GHz i7 2630 with 16GB RAM, and (3) in a VirtualBox on a 3.4GHz i7 6800k 32GB RAM PC.

All Mininet tests demonstrated some unpredictable behavior as would be expected from virtual devices running on a single compute node. For the three metrics measured (throughput, switch latency and jitter), I saw a larger standard deviation on the Mininet test verses those on the SDN OTG. The actual bandwidth observed by the various tests is less important than the degree of variation between the measurements of each test iteration.

These tests produced an order of magnitude greater repeatability than the tests run on Mininet shown in the 4-hop Bandwidth Comparison in Figure 7-11. The SDN OTG tests in purple show a much smaller standard deviation than all three platforms running Mininet.
My simple tests for throughput, latency and jitter were enough to show some benefit from a dedicated hardware SDN testbed.

![4-Hop Bandwidth Comparison](image)

**Figure 7-11: Mininet vs. Physical Testbed**

### 7.7 Conclusion

My research has produced reasonable network metrics from a software emulated SDN that are very useful for identifying and demonstrating behavior and quantifying performance [45] [59]. However, with more accurate, repeatable measurements on a physical SDN, I can further enhance my ability to draw conclusions about network health and discover subtle indicators that could point to abnormal or malicious behavior. As an alternative to using simulation or buying and scheduling time on an existing SDN network like GENI or PlanetLab, I had a genuine need for a low cost, portable SDN testbed to further expand my testing and teaching capabilities. Using the new low-cost Zodiac FX
SDN switches, I built a four switch, four host (RaspberryPi3) and a controller machine network that is portable and can also be hooked up to a larger SDN or hybrid network.

I identified and reviewed current problems in SDN testing and the solution to some of these problems using my SDN OTG testbed. This testbed can run any SDN controller including ODL, Floodlight and Ryu SDN controllers [56]. I completed my tests using primarily ODL, but noticed similar results when using other controllers. I developed a northbound application to conduct a network analysis [59]. Using the OTG testbed and ODL TSDR [57] to collect data from the testbed, the network analysis tool performs the tests to generate precise network metrics and produces a confidence score for that specific network path. SDN On-The-Go is a great tool for generating precise network metrics and demonstrating SDN and traditional network behavior and misbehavior in the classroom.
CHAPTER 8

SDN NETWORK ANALYSIS TOOL
IMPLEMENTATION

8.1 Contributions

Applying the results of the previous seven chapters, I implemented a functional SDN Network Analysis Tool. It provides a real-time assessment of the data flow and data path for a given transmission. The purpose of this implementation is to utilize the features of SDN to enhance network security for sensitive data transmission. First, I provide an overview of the implementation, key tasks it must perform, and the fundamentals of creating and utilizing a SDN Northbound application. Second, I describe my methodology for implementing the framework of Chapter 5. Then, I cover how the SDN Floodlight controller allows of the implementation of such an application. Next, I describe the functional details of the tool. Finally, I tested this tool under numerous conditions and in various models, simulations, and testbeds. The results and evaluation verify the utilization of this application in detecting data anomalies.

8.2 Implementation Overview

This application’s confidence assessment uses a large amount of information about the network (see metrics listed below in Section 8.4.1) to calculate a confidence level from three transmissions of data (a pre-transmission, actual-transmission, and post-transmission). The results of these calculations are returned to the data sender (transmitter) so they can determine whether or not to release the encryption key.
Key Tasks:

- Capture metrics for a given SDN data flow
- Analyze metrics using methodology listed below in Section 8.4.1
- Computer Confidence weightings that are user configurable, see Section 8.4.8
- Provide “confidence assessment” output to user
- Store prior “confidence assessments” for review

8.2.1 Floodlight Programming Model

The term ‘modular architecture’ is used to describe the architecture of the Floodlight Controller, which is shown in Figure 8-1. The core architecture includes various modules, such as topology management, device/end-station management, path/route computation, infrastructure for web access (management), counter store (OpenFlow counters), and a

Figure 8-1: Floodlight Programming Model [92]
state storage system, that are well stitched by a module-management system. Below I describe some of the important components of the controller architecture.

![SDN Controller Architecture](image)

**Figure 8-2: SDN Controller Architecture [92]**

### 8.2.2 Rest-API Based Applications

Floodlight comes with a few applications that use the exposed REST-APIs. Circuit Pusher utilizes Floodlight REST-APIs to establish a path between any two IP-addressable devices by adding flow-entries in all the switches that together constitute the path. In addition to circuit-pusher, Floodlight can be run as the network backend for OpenStack using a Neutron plugin. There are two main components to this solution: RestProxy and VirtualNetworkFiler (see Figure 8-3).
8.3 Description of Methodology

8.3.1 Metrics Measured

- Packet Size – packet size measured as data flows from switch to switch per flow.
• Flow Size – total flow size as the data flows from switch to switch, measured per flow.
• Packet Lapse Time – total amount of time from when packet arrives at a switch until it arrives at the next switch, measured at each SDN switch.
• Flow Duration – the total time it takes a flow to travel from host A to host B.
• Hop Count – simple hop count for a flow from host A to host B.
• Average Packets per Flow – the average number of packets in each flow.
• Median Bytes per Flow – determine a median value of flow size, see equation below where X represents the number of bytes and n represents the number of flows. Equation 12 is borrowed from Guo’s research in attack detection:

\[
\text{md}(X) = \begin{cases} 
\frac{X(n+1)}{2}, & \text{if } n \text{ is odd} \\
\frac{X(\frac{n+2}{2})+X(\frac{n+1}{2})}{2}, & \text{otherwise}
\end{cases}
\]

Eq. 12

8.3.2 Metric Analysis

8.3.2.1 Calculate and Assess Deviation

• Calculate the standard deviation between all measured values. If the standard deviation is 0, then set a value of 1 for standard deviation. Equation 13 is used in my implementation for calculating standard deviation.

\[
\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2} \quad \text{Eq. 13}
\]

• Calculate the median of the measured values.

• Measure degrees of separation from the median value + or – a degree of deviation. These values were selected utilizing a six sigma/normal distribution approach. The values below range from a sigma level of 4.3 through level 1.
o median +/- (0.25*σ) = DeviationScoreMultiplier of 1
o median +/- (0.62*σ) = DeviationScoreMultiplier of 3.25
o median +/- (1.0*σ) = DeviationScoreMultiplier of 5.5
o median +/- (1.5*σ) = DeviationScoreMultiplier of 7.75
o median +/- (2.0*σ) = DeviationScoreMultiplier of 10
o median +/- (3.0*σ) = DeviationScoreMultiplier of 20
o Anything outside of +/- (3*σ) = DeviationScoreMultiplier of 30

8.3.3 Calculate the BaseScore

- Referring back to Chapter 5.5.1 for more information on the following equations and terminology.

- BaseScore = (0.6 × (10.41 × (1 − (1 − CI) × (1 − II) × (1 − AI)))) + 
  0.4 × (20 × S × M × V) − 1.5) × f(Impact)

  o S = Spoofability
  o M = Measurability
  o V = Variability
  o CI = ConfImpact
  o II = IntegImpact
  o AI = AvailImpact

  o f(Impact) = 0 if Impact = 0, else 1.176

<table>
<thead>
<tr>
<th>Configurable Impact Value</th>
<th>none</th>
<th>partial</th>
<th>complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfImpact</td>
<td>0</td>
<td>0.275</td>
<td>0.66</td>
</tr>
<tr>
<td>IntegImpact</td>
<td>0</td>
<td>0.275</td>
<td>0.66</td>
</tr>
<tr>
<td>AvailImpact</td>
<td>0</td>
<td>0.275</td>
<td>0.66</td>
</tr>
</tbody>
</table>
8.3.4 Calculate the EnvirScore

- Modified.BaseScore = \((0.6 \times (10.41 \times (1 - (1 - M.CI) \times (1 - M.II) \times (1 - M.AI)))) + 0.4 \times (20 \times M.S \times M.M \times M.V) - 1.5) \times f(Impact)

- EnvirScore = \((M.BS + (10 - M.BS) \times CDP) \times NC\)
  
  o NetworkComplexity = NC (Reference Table 5-3 for Network Complexity values)
  
  o CollateralDamagePotential = CDP

Table 8-2: Configurable Collateral Damage Potential

<table>
<thead>
<tr>
<th>Configurable Collateral Damage Potential</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>0.0</td>
</tr>
<tr>
<td>low</td>
<td>0.1</td>
</tr>
<tr>
<td>low-med</td>
<td>0.3</td>
</tr>
<tr>
<td>med-high</td>
<td>0.4</td>
</tr>
<tr>
<td>high</td>
<td>0.5</td>
</tr>
<tr>
<td>undefined</td>
<td>0.0</td>
</tr>
</tbody>
</table>

8.3.5 Calculate the BaseScore Confidence Rating

I created Equation 14 specifically for use in this framework/implementation.

\[
BSCR = \frac{(Pre.BS+Pre.DSM)+Actual.BS+Actual.DSM+(Post.BS+Post.DSM)}{3.5} \quad \text{Eq. 14}
\]

- BaseScoreConfidenceRating = BSCR
- BaseScore = BS
- DeviationScoreMultiplier = DSM
8.3.6 Calculate the EnvirScore Confidence Rating

I created Equation 15 specifically for use in this framework/implementation.

\[
ESCR = \frac{(\text{Pre.ES} + \text{Pre.DSM}) + (\text{Actual.ES} + \text{Actual.DSM}) - (\text{Post.ES} + \text{Post.DSM})}{3.5} \quad \text{Eq. 15}
\]

- EnvirScoreConfidenceRating = ESCR
- EnvirScore = ES
- DeviationScoreMultiplier = DSM

8.3.7 Calculate and Assess the Confidence Rating

I created Equation 16 specifically for use in this framework/implementation. The addition of variables b and e to Confidence Rating are present to increase the severity of individual metric measurement on the overall scoring.

\[
CR = \begin{cases} 
\frac{\text{BSCR} + \text{ESCR}}{2}, & \text{if all DSM} \leq 1 \\
\frac{\text{BSCR} + \text{ESCR}}{2} + b + e, & \text{otherwise}
\end{cases} \quad \text{Eq. 16}
\]

- ConfidenceRating = CR
- b = the total number of BS.DSM > 1 when the Median of BSCR is greater than one \( \sigma \)
- e = the total number of ES.DSM > 1 when the Median of ESCR is greater than one \( \sigma \)
- To assess ConfidenceRating reference the table below:

<table>
<thead>
<tr>
<th>Confidence Rating</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest Confidence</td>
<td>0.1 – 3.9</td>
</tr>
<tr>
<td>High Confidence</td>
<td>4.0 – 6.9</td>
</tr>
<tr>
<td>Moderate Confidence</td>
<td>7.0 – 8.9</td>
</tr>
<tr>
<td>Low Confidence</td>
<td>9.0 – 10.0</td>
</tr>
</tbody>
</table>

8.3.8 List of All User Configurable Values

- Spoofability (S) – reference Table 5-1 for values
- Measurability (M) – reference Table 5-1 for values
8.4 Floodlight Controller

8.4.1 Floodlight Controller REST-API

Floodlight includes a REST-API server, which uses the Restlets library. With the Restlets, any module developed can expose additional REST-APIs through an REST-API service – typically, the modules that depend on REST server, expose APIs by implementing RestletRoutable in a class. The controller itself presents a set of extensible REST-APIs, to get and set various types of information. The REST-API is the recommended interface to develop applications utilizing Floodlight supported features.

Once the Floodlight is up and running, one can use the APIs that are already supported in the controller. Figure 8-4 summarizes various functions that are supported by the Floodlight. For example, the below curl command retrieves the switches that are connected to the controller with IP as 10.0.0.1.

```
curl http://10.0.0.1:8080/wm/core/controller/switches/json
```
8.4.2 Writing a Floodlight Module

As summarized in Figure 8-5, the process involves scoping the module functionalities, followed by translating those requirements into specific events, messages and utilities. Once the requirements are known, a class matching those functionalities are declared and implemented – including handling messages and events, as well as implementing the exposed APIs.
In summary, the OpenFlow-based Floodlight controller consists of a rich set of modules, where each module provides a service that can be accessed through either a simple Java API or a REST-API.

8.4.3 Applications Backend Development

The Backend server development of the application involves all the API services which were developed to maintain activities mentioned below. The entire database is maintained online.

**Server:** Providers of the resource or service. On the server there is an application that is operated through a remote database and web client application.
Client: Service requesters. These are the client applications that remain active on the web browsers.

Process: Client initiates the interaction and server receives request and then server with client cooperation performs the requested processes and provides the requested service.

8.5 Application Implementation

An overview of this framework implementation along with design documents, use-cases/evaluations and work-in-progress source code are available in Appendix J.

The workflow for the tool is illustrated in Figure 8-6. When Host A initiates a new data transfer to Host B, a new flow is created. Assuming that there is no existing entry in the OpenFlow-enabled switch connected to Host A, the switch will contact the Floodlight controller IFlowManagerService module for routing information. The flow tables on all switches involved in the routing will receive flow rules from this module. Then utilizing the IRestAPIService, the controller will communicate with the IStatisticsService. This service module conducts bi-directional communication with the switches to get metric data. Again utilizing the IRestAPIService, the metrics are shared with the newly created ConfidenceService Module, which analyzes and calculates an assessment. This information is then transmitted back to Host A, the message originating point.
Figure 8-7 illustrates the functional process steps in the network analysis framework. This implementation focuses on the first four steps of this process. The traffic/key release determination and process is unique to the entity applying this tool; therefore, outside the scope of this research.
The first step in the process/implementation is identifying the traffic and initiating the tool. In Figure 8-8, you see user interface (UI) which lets the user select source and destination of the data.

Step Two is to transmit and monitor the data path as traffic is transmitted across the network. In the administrator workflow, you may choose to modify various configurable
values prior to transmitting data. Figure 8-9 shows the UI to adjust the values listed in Section 8.4.8.

![Configuration Menu](image)

**Figure 8-9: Configuration Menu**

Next, Step Three, the analysis tool captures and assesses the base and environment scores for the data transmission. In general, this level of data is not visible to the user (viewable by selecting “View Details”) as it is deemed unnecessary in assessing the security of the data transmission, see Figure 8-10 below.
Lastly, the overall data path confidence rating is assessed. This is a summary view of the data shown in Figure 8-11 and provide the important color-coded assessment described in Section 8.4.7.
During the tool initialization process, an administrator can configure the preset values then conduct a series of tests to ensure confidence scoring meets their needs. The UI allows for easy modification and fine tuning of the preset values to minimize effort and complexity, see Figure 8-12.

An added functionality, a history of assessments is included within the UI. This allows administrators the ability to review previous transmissions and provides auditing if needed for the administrator, see Figure 8-13.
8.6 Application Experiments and Evaluation

8.6.1 General Application Testing and Experimentation

Now that the tool is implemented, it is necessary to conduct a series of experiments to baseline the tool and provide a general assessment of its impact on network traffic. I tested this against a basic SDN topology simulation, a data center simulation and on the OTG testbed. The goals of this testing were (1) to verify analysis functionality, (2) to compare and access impact on throughput, and (3) to provide a reference point for future security testing.

8.6.1.1 Baseline Simulation

The baseline simulation is a simple Mininet command line argument:

```
sudo mn --controller=remote,ip=172.26.2.101,port=6653 --mac
--topo=linear,3
```

The command produces a topology like the one illustrated in Figure 8-14.

![Figure 8-14: Baseline Topology](image)

The purpose of using this simple model for initial testing was to minimize the impact of outside factors and conduct testing under ideal conditions. Sending traffic from one host to another on this simple network, I verified the functionality of the tools and
algorithms. I compared the transmission size for traffic generation tools with the measurements of the assessment tool.

As you can see from the graph in Figure 8-15, there were limited differences (avg. 20.26 vs. 19.97) between measurements of the tool versus the actual traffic generation. The realized variance of 1.44% is well within the margin of error for Mininet M&S. The graph below depicts the data collected and a polynomial trendline is included to illustrate the similarity between assessment tool and \texttt{iperf} measurements. A sampling of the raw data of this testing is provide in Appendix J.

![Baseline Bandwidth Comparison](image)

**Figure 8-15: Bandwidth Comparison of Baseline**

### 8.6.1.2 Data Center Simulation

For the data center simulation, I utilized the custom topology developed in Chapter 6 and listed in Appendix F. For traffic generation I applied techniques researched and detailed in Section 4.4; these include \texttt{ping}, \texttt{iperf}, \texttt{hping}, and others.

In Figure 8-16, you can see the impact of bandwidth testing on a semi-realistic data center simulation. From the graph you can see that I continue to observe a limited
nonconformity between my assessment tool and the traffic generation methods. The variance for the Mininet Data Center M&S is 1.58% or an increase from the baseline topology of 10%. After repeated experimentation, I conclude this increased variance is due to the complexity (i.e., additional routing time) of a data center versus the small linear topology.

![Data Center Bandwidth Comparison](image)

**Figure 8-16: Bandwidth Comparison for Data Center**

### 8.6.1.3 OTG Physical Testbed

For the physical testbed, I utilized the partial mesh topology illustrated in Figure 7-3 in Chapter 7, Section 4. Again, for traffic generation I continued to apply the techniques researched and detailed in Chapter 4, Section 4; these include ping, iperf, hping, and others.

In Figure 8-17 you can see the impact of bandwidth testing on the SDN OTG. From the graph you can see the tool experienced limited deviation between the assessment tool measurements and the traffic generation methodologies own measurement. It was surprising to note that the variance did increase to 1.76%. Although I expected the results
of the OTG to be more accurate and repeatable, I attribute this variation to several factors: (1) a slightly more complex topology that the OTG represents versus the base line linear topology, (2) networking oddities of physical media versus a model, and lastly (3) processing capability of the M&S system versus the limited CPU of the OTG.

![OTG Bandwidth Comparison](image)

Figure 8-17: Bandwidth Comparison of SDN OTG

### 8.6.2 Evaluations Conditions

With a series of baseline and reference testing complete, I experimented with two forms of malicious activity. The purpose of this testing was to (1) verify if the tool can effectively identify a threat in real-time and to (2) assess the impact of this assessment.

#### 8.6.2.1 Man in the Middle Attack

After building my analysis framework and completing my initial experiments, I identified that data path analysis is well suited for the detection of network anomalies, specifically detection of MitM attacks. Recall from Chapter 5, Section 5, that my framework can detect MitM showing 26% change in the total assessment score compared to a single metric.
The framework testing is based upon three metrics, only two of which were dynamically measured. In this current implementation, I am computing seven real-time network measures for my confidence assessment.

To evaluate the effectiveness of this implementation, I tested against the data center topology which I used in the Section 8.6.1.2. In addition to this topology, this test environment was operating on an Amazon Elastic Cloud-2 instance and unconstrained bandwidth. For attack generation, I utilized a combination of the methods described in Chapter 5, Section 5 and research from the Naval Postgraduate School on SDN MitM attacks [94]. I employed a MitM reflection attack with varying degrees of delay. I used the same traffic generation techniques as in the Section 5.5. Measurements of network data were recorded using Wireshark, traffic generation tools, and my assessment tool.

![Data Center MitM Assessment](image-url)

**Figure 8-18: Data Center MitM Attack Assessment**
Figure 8-18 shows the impact of a MitM attack and how this attack impacts an individual and combined confidence assessment. Attacks were conducted with increasing maliciously induced latency. See Appendix J for additional graphs of the data collected.

I then conducted a second set of experiments using my SDN OTG. All other conditions from the above experiments remained the same. In Figure 8-19, I noted a similar linear relationship (1) between a single metric analysis and the combined analysis and (2) the corresponding increase between increasing latency and assessment scoring. I observed the same performance as my framework testing in which initially single metric analysis more quickly identifies an anomaly; however, the combined assessment shows 22% greater impact from anomalies.
In further analyzing this data, I reviewed the difference between the Packet Lapse Time metric and the overall Combined Assessment, see Figures 8-20 and 8-21. Again, the linear relationship is present in my data.

**Figure 8-20: Δ in Data Center Measurements**

**Figure 8-21: Δ in SDN OTG Measurements**
The following conclusions were drawn from these sets of experiments:

- There is a linear relationship between individual assessment metrics and the overall combined assessment exists. Therefore, increasing MitM induced latency, increases the likelihood of detection via the confidence assessment.

- One key factor impacting the increasing MitM latency is the traffic throughput. Increasing from 50Mbps to 350Mbps, doubled the measured latency for detection. Recall prior research on large-scale MitM identified a minimum delay of 50ms and average maximum of 242ms.

- From my experiments and data, it is evident that increasing the number of measurement opportunities (i.e., nodes, switches, routes, etc.) positively impacts the ability of the assessment tool to identify anomalies. On average the tool had a 19% greater degree of anomaly detection as network complexity increased from nine nodes to twenty-one (a factor of 2.3), see Figure 8-22.

![Impact of Network Complexity](image-url)  

**Figure 8-22: Impact of Network Complexity on Detection**
8.6.2.2 Denial of Service Attack

During my exploratory research and my research proposal, the topics of DoS/DDoS detection and SDN were discussed. During the development of the assessment framework and the implementation from this chapter, I paid specific attention to metrics and measurements which may help identify an attack of this type. Specifically, these were the average number of packets per flow, the median Bytes per flow, and flow duration. In the cases of these measures, average traffic exhibits a moderate to large number of packets or Bytes per flow (or duration). A low number of packets or Bytes (or duration) in a flow is a possible indicator of malicious activity. By assessing multiple metrics, I hoped to decrease the rate of false positives.

Unfortunately, my current implementation proved to be not well suited for real-time DoS/DDoS detection, not because of an inability to identify the anomaly, but rather because of the processing time required for the application. Through testing on both Data Center M&S and the SDN OTG, an average assessment measures nine seconds of traffic with an additional overhead between 1.1-1.7 seconds. In real terms this means a 20-30 second between metric sampling. A delay of this magnitude does not greatly impact user transmission of individual data, but it does limit the real-time IDS capabilities.

Despite these limitations, I tested the tool against this attack scenario. Using the BoNeSi DoS/DDoS attack tool, described in Appendix A, I tested my assessment tool. The attack conducted lasted 30 seconds and increased traffic volume on the network nearly 3000 times. Whenever the assessment tool sampled network traffic, all scoring increased by 24%; however, it took 20 seconds before the assessment scoring could alert the user or administrator. By the time my assessment tool takes another measurement, the attack is
completed, see Figure 8-23. A knowledgeable attacker could simply employ a burst on/off attack to defeat this tool. Additionally, numerous other DoS/DDoS detection tools can identify this type of malicious activity more efficiently. I added the SDN DDoS tool developed by Braga [15] and approaches based upon the KDD-99 data set for reference. To replicate these approaches, I built an experiment in which a DDoS brute force flooding attack is utilized. Neither Braga nor the KDD-99 have a scoring system, rather a simple detection Boolean. For the simplest of reading the figure, I substituted 1.5 and 8.5 for Braga’s detection Booleans and 1 and 8 for the KDD-99. Detection occurs ~10 seconds faster utilizing the Braga approach and ~6 seconds faster with the KDD-99 approach.

![Figure 8-23: Data Center DoS/DDoS Attack](image)

8.6.3 Results Analysis

Knowing that my network analysis tool is measuring metrics accurately, I moved to testing data flows with the tool. Key areas of emphasis for this phase of experimentation
is to show the accuracy/error performances of my confidence-score system with legitimate traffic.

To review and evaluate the data captured during baseline testing, I applied a Receiver Operating Characteristics (ROC) curve analysis [95]. The baseline test was executed ten times, generating 1,200 TCP transmissions per iteration for a total of 12,000 data transmission. A total of 660 confidence scores were generated during this testing of those, 607 confidence scores rate legitimate traffic with a score less than 7.0 which indicates negative malicious activity (e.g., a secure transmission). The remaining 53 confidence scores for legitimate traffic were at a score of 7.0 or higher which indicates positive malicious activity (e.g., an insecure transmission). This is an overall false positive rate of 8.03%. The True Positive Rate of my tool from the baseline testing described above is calculated using Equation 17, a standard formula.

\[
True\ Positive\ Rate = 1 - \frac{True\ Negatives}{Total\ True\ Negatives}\ Eq.\ 17
\]

The False Positive Rate of my tool from the baseline testing above is calculated using Equation 18, a standard formula. The resulting ROC Curve Analysis graph is shown in Figure 8-24.

\[
False\ Positive\ Rate = \frac{True\ Positives}{Total\ True\ Positives}\ Eq.\ 18
\]
I attribute the false positive rate primarily to dynamic routing changes that occurred during the transmissions. Above in my attack analysis (Section 8.6.2), I described the impact of latency induced by malicious activity. I found that modest network latency, less than 40ms, does not appear to impact the number of false positives as much as additional hops (e.g., routing changes).

The proposed use-case for this application is when transmitting sensitive data; it is not intended for use as a general network traffic security monitor. An 8% false positive rate would be unacceptable for an IDS; however, when utilized as an extra layer of security for sensitive data transmission it is acceptable. Assuming a few hundred transmissions per day, network administrators could reasonably review anomalies of a few dozen confidence alerts. Increasing the number of pre and post traffic traceroute measurements would likely decrease amount of false positives by creating more data points for the anomaly analysis.
Additionally, fine tuning of the specific deployments EnvirScore configurable values could potentially reduce the rate of false positives even further.

The baseline collection focused on the combined score, so it is difficult to identify the specific impact of each measurement. Additional research can be conducted to assess the specific impact each recorded metric has on the rate of false positives and false negatives through principle component analysis.

In future work, I intend to continue testing the tool utilizing two additional forms of traffic generation. First, I will employ a capture and replay process for legitimate network traffic. To conduct such an approach, I will use tcpdump and tcpreplay. The purpose of tcpdump is to capture and write traffic to a file. tcpreplay allows for reply of the tcpdump file and offers the ability to manipulate flow size and rate. Second, I plan to use standardized test data sets, like West Point Cyber Defense Exercise Data Set, to test the tool against larger-scale malicious activity.
CHAPTER 9

FUTURE RESEARCH AND CONCLUSION

In this research, several topics associated with the advancement in securing SDN platforms were explored. The concept and protocol of SDN/OpenFlow were assessed for their ability to enhance network security through better knowledge and control of the data path. I conducted exploratory research of the tools necessary to conduct larger more in-depth SDN experimentation. From this work, I identified an opportunity to improve data path security through data analysis and acknowledged a need for a robust set of testing tools and environment. This included the creation of SDN OTG for research, as well as a teaching tool for networks and security. The results are a network analysis tool implementation and numerous SDN modeling, simulation, and testbeds.

9.1 Concerns and Lessons Learned

In this section, some of the concerns and issues that were encountered before and during the research process are detailed, as well as the lessons I learned while dealing with them.

1. The foundation of the new research is often a survey of literature to understand the domain, opportunities, shortcomings, and risks. Since SDN was very much in its infancy when I began, the amount of existing security-related literature was limited, although the possibilities that SDN might provide to network security were clearly great. To address the new and changing landscape of SDN
required repetitive surveying of new existing work and when necessary incorporating those ideas. I learned that you are never done learning.

2. Exploratory research with EstiNet proved to be a challenge early on. Not only did they have a commercial focused business model (which did not accommodate academia), their software functioned under very tight tolerances. EstiNet was and is an excellent and complete SDN M&S. It offers test repeatability and many additional features that are not part of other M&S options. Unfortunately, I could not capitalize on their software. Pivoting to Mininet and eventually to SDN OTG lead my research in a new and novel direction. If my relationship with EstiNet persisted, then I would not have built my own testbed. I learned to embrace change.

3. When I finally began developing a framework to enhance network security, I was unsure if my ideas showed promise. Fortunately, I had an opportunity to present an early version of the network analysis framework during an industry exposition. The response from industry personnel provided needed confidence and motivation. I learned the benefit of getting “expert” feedback early; you might be smarter than you thought.

4. I am certain that most researchers experience the need to test, modify, and re-test their hypotheses and implementations. My research is certainly no different. The final network analysis tool is truly the result of an iterative process. I learned to not get discouraged, to stick with it. And probably more importantly, I learned to not settle for good enough, but push beyond.
9.2 Evaluation of Success Criteria

The following is a self-evaluation of the success criteria of this research work:

1. A review of the capabilities of SDN to enhance network security was conducted, allowing researchers a view of possible opportunities for future security research. Current state of the art SDN security research was also surveyed to help identify gaps and points of synergy.

2. Exploratory research into SDN security resulted in an assessment of current M&S applications, identification of several SDN specific traffic/attack tools, and a review of SDN controllers. Although the M&S tools selected from this section of research was ultimately not used, it identified limitations with the selected application and spurred the development of SDN OTG.

3. Building from the SDN capabilities assessment, an opportunity to capitalize on the better network knowledge transformed into a framework for data path analysis. The framework developed a service that inspects a network path before and after sending sensitive information and compares this inspection to our known behavior and security patterns to provide a user with a dependability assessment of that specific network path. This assessment allows users to decide whether the network path is secure enough for releasing sensitive information.

4. The work presented at ICCCN highlighted the need for developing a series of modeling, simulation, and testbeds for SDN research. Throughout my studies, I developed several M&S environments, built a virtualized tested, and identified and tested many traffic/attack tools. I used much of this research when testing
my own SDN security applications, as well as made them available to the University at large (via GitHub, the A210 Lab, or on my personal research page).

5. As stated above, the research, development, and construction of the SDN OTG was the direct result of my exploratory work and the limitation of existing M&S applications. SDN OTG is a physical testbed that can produce repeatable experimentation results under semi-realistic conditions. It is a self-contained testbed that can also be used as a portable teaching device, moved from classroom to classroom or taken home for private research.

6. I completed my research with a functional network analysis tool implementation. Collection and analysis of seven metrics are included in the tool. It was tested and evaluated under a series of modeling, simulation, and testbeds previously developed. Lastly, I assessed the tool under malicious conditions. The results of which show that it is a valuable assessment tool for detecting MitM or other network anomalies. The effectiveness of the tool is positively impacted as network complexity increases.

9.3 Contributions

This research contributes to the advancement of network security by capitalizing on the features and capabilities of SDN.

1. A review of the potential network security enhancements that SDN provides researchers.

2. A survey of simulation and traffic generation tools, as well as exploratory testing and evaluation.
3. A framework for securing sensitive data transmission across a network by analyzing data path metrics.

4. A series of modeling, simulation, and testbeds which were used in experiments and evaluation of my research.

5. A portable SDN testbed which can be reused for research or as a teaching tool.

6. An implementation of a Network Analysis Tool that enhances data path security.

9.4 Future Research

There are a few areas of research that warrant further investigation. The first area is in regard to the confidence analysis framework, specifically taking a finer grained look at the metric weighting structure to consider a larger survey audience as well as including temporal-based weighting like CVSS.

Secondly, in relation to the framework and OTG, I am also looking to extend this application in an area of machine learning to help in the detection of anomalies in IDS/IPS systems. Furthermore, I hope to explore the metric deviation determination and address human cognitive ergonomics via enhanced visualization tools for the confidence analysis.

Additionally, I would like to continue the testing of the implemented framework in large-scale testbeds, like PlanetLab, or as part of a larger campus testbed. The deployment and use of the tool in a real-world environment will help refine the metrics and algorithms.

Finally, this framework may provide a key component to an authentication/key exchange service, Proximity Encryption, that does the security analysis to establish that the recipient of sensitive data is who they say they are and that the data has been sent to the right location before the encryption key is sent.
9.5 Conclusion

Protecting privacy and security of our data is critical to safeguarding users and maintaining the public’s confidence. My research shows that malicious activity can be identified quickly as a single metric indicator and consistently within a multi-factor indicator analysis. My tool gives administrators better understanding of the internal operations and helps in detecting latent, systemic network problems. The SDN OTG physical testbed is a cost-effective solution, providing reliable and repeatable results, providing precise time sensitive data and can be setup as a fully functional SDN testbed in a matter of minutes. The combination of the Network Analysis Tool and series of testing systems provides researchers and security engineers the means to enhance security and improve user confidence in our networks.
REFERENCES


wan/google-s-software-defined-openflow-backbone-drives-wan-links-to-100--utilization.html.


Appendix A – BoNeSi Guide

The guide below walkthrough the process of downloading and installing the DDoS simulation tool BoNeSi in the EstiNet SDN environment [21] [37].

1. Visit BoNeSi website

Visit the following link: https://code.google.com/p/bonesi/

2. Download "bonesi-0.2.0.tar.gz"

3. Unzip BoNeSi

    tar -jxf bonesi-0.2.0.tar.gz

4. Install prerequisite

    yum -y install libpcap-devel
    yum -y install libnet-devel

5. Move to bonesi-0.2.0 folder

    cd ../bonesi-0.2.0

6. Compile BoNeSi

    ./configure
    make
    make install

7. See the usage of BoNeSi
vim README

```bash
# bonesi [OPTION...] <dst_ip:port>

Options:
-i, --ips=FILENAME   filename with ip list
-p, --protocol=PROT0  udp (default), tcp or tcp
-r, --send_rate=NUM   packets per second, 0 = infinite (default)
-s, --payload_size=SIZE size of the payload, (default: 32)
-o, --stats_file=FILENAME filename for the statistics, (default: 'stats')
-c, --max_packets=NUM  maximum number of packets (requests at tcp/http), 0 = infinite (default)
-t, --integer         IPs are integers in host byte order instead of in dotted notation
-u, --url=URL         the url (default: '/') (only for tcp/http)
-l, --url_list=FILENAME filename with url list (only for tcp/http)
-b, --useragent_list=FILENAME filename with useragent list (only for tcp/http)
-d, --device=DEVICE   network listening device (only for tcp/http, e.g. eth1)
-m, --mtu=NUM         set MTU (default 1500). Currently only when using TCP.
-f, --frag=NUM        set fragmentation node (0=IP, 1=TCP, default: 0). Currently only when using TCP.
-v, --verbose         print additional debug messages
-h, --help            print help message and exit
```

Additionally Included Example Files:

```
50k-bots
50,000 ip addresses generated randomly to use with --ips option
browserlist.txt
several browser identifications to use with --useragentlist option
urlist.txt
several urls to use with --urllist option
```

8. Find the binary of BoNeSi

```bash
whereis bonesi
```

[root@localhost ~]# whereis bonesi
bonesi: /usr/local/bin/bonesi

9. Add BoNeSi binary to EstiNet

```bash
cp /usr/local/bin/bonesi /usr/local/estinet/tools
```

10. Move to /usr/local/estinet/tools

```bash
cd /usr/local/estinet/tools
```

11. Try to run BoNeSi on terminal

```bash
ls
./bonesi -c 100 127.0.0.1:8000
```
Send 100 UDP packets to lo device with port 8000

11. Use Wireshark to see lo device

```plaintext
<table>
<thead>
<tr>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:000</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>TCP</td>
<td>1460</td>
<td>“Device protocol: Destination port: 8000”</td>
</tr>
<tr>
<td>10:00:001</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>UDP</td>
<td>1460</td>
<td>“Device protocol: Destination port: 8000”</td>
</tr>
<tr>
<td>10:00:002</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>TCP</td>
<td>1460</td>
<td>“Device protocol: Destination port: 8000”</td>
</tr>
<tr>
<td>10:00:003</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>UDP</td>
<td>1460</td>
<td>“Device protocol: Destination port: 8000”</td>
</tr>
<tr>
<td>10:00:004</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>TCP</td>
<td>1460</td>
<td>“Device protocol: Destination port: 8000”</td>
</tr>
<tr>
<td>10:00:005</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
<td>UDP</td>
<td>1460</td>
<td>“Device protocol: Destination port: 8000”</td>
</tr>
</tbody>
</table>
```

12. Draw the topology in EstiNet
13. Run BoNeSi in "Attacker" (host 1)

Double-click host1

Left-click "Add" button

Enter bonesi 1.0.1.2:8000 (where 1.0.1.2 is the victim server's IP in the simulator)

set Bonesi's running time at 10~20 seconds

Left-click "OK" button to save and leave
13. **Run stcp in "Legitimate User" (host 2) to simulate a legal TCP sender.**

   Double-click host2
   
   Left-click "Add" button
   
   Enter stcp 1.0.1.2 (where 1.0.1.2 is the victim server's IP in the simulator)
   
   set stcp's running time at 1~30 seconds
   
   Left-click "OK" button to save and leave

14. **Run rtcp in "Victim Server" (host 3) to simulate a legal TCP receiver.**

   Double-click host3
   
   Left-click "Add" button
   
   Enter rtcp
   
   set rtcp's running time at 1~30 seconds
   
   Left-click "OK" button to save and leave
15. Run the simulation and see the results.

Due to BoNeSi's UDP DDoS, TCP connections between Victim Server and Legitimate User were interrupted during 10~20 seconds in the total simulation running.
Appendix B – Framework Confidence Analysis
Application

```java
import java.awt.Component;
import java.awt.EventQueue;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.math.RoundingMode;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JLabel;
import org.apache.poi.ss.usermodel.FormulaEvaluator;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.omg.CORBA.NameValuePair;
import org.apache.commons.codec.binary.Base64;
import jxl.Cell;
import jxl.CellType;
import jxl.Sheet;
import jxl.Workbook;
import java.awt.BorderLayout;
import javax.swing.JTextField;

public class test {
    private JFrame frame;
    final JFileChooser fc = new JFileChooser();
    Component c;
    float bs, es;
    private JTextField textField;
    private JTextField textField_1;
    private JTextField textField_2;
    JLabel lblNewLabel_4, lblNewLabel_6, lblNewLabel_8, lblNewLabel_10,
    lblNewLabel_12, lblNewLabel_14;
    float Spoofability, Measurability, Variablitiy, base_score;
    //In response to a button click: //int returnVal = fc.showOpenDialog(aComponent);
    /*** Launch the application.***/
    public static void main(String[] args) {
```
EventQueue.invokeLater(new Runnable() {
    public void run() {
        try {
            test window = new test();
            window.frame.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

/*** Create the application.*/
public test() {
    initialize();
}

/*** Initialize the contents of the frame.*/
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 700, 650);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(null);

    JLabel lblNewLabel = new JLabel("Pre");
    lblNewLabel.setBounds(10, 76, 139, 14);
    frame.getContentPane().add(lblNewLabel);

    JLabel lblNewLabel_1 = new JLabel("Actual");
    lblNewLabel_1.setBounds(10, 116, 139, 14);
    frame.getContentPane().add(lblNewLabel_1);

    JLabel lblNewLabel_2 = new JLabel("Post");
    lblNewLabel_2.setBounds(10, 156, 139, 14);
    frame.getContentPane().add(lblNewLabel_2);

    JTextField textField = new JTextField();
    textField.setBounds(418, 74, 270, 20);
    frame.getContentPane().add(textField);
    textField.setColumns(10);

    JTextField textField_1 = new JTextField();
    textField_1.setBounds(418, 114, 270, 20);
    frame.getContentPane().add(textField_1);
    textField_1.setColumns(10);

    JTextField textField_2 = new JTextField();
    textField_2.setBounds(418, 154, 270, 20);
    frame.getContentPane().add(textField_2);
    textField_2.setColumns(10);

    JButton btnNewButton = new JButton("Calculate");
    btnNewButton.setBounds(264, 201, 210, 23);
    frame.getContentPane().add(btnNewButton);

    JLabel lblNewLabel_3 = new JLabel("Base_Score_Confidence_Rating");
    lblNewLabel_3.setBounds(10, 258, 241, 14);
    frame.getContentPane().add(lblNewLabel_3);

    JLabel lblNewLabel_4 = new JLabel("New label");
    lblNewLabel_4.setBounds(418, 258, 270, 14);
114.  frame.getContentPane().add(lblNewLabel_4);
115.  JLabel lblNewLabel_5 = new JLabel("Base_Score_Confidence_Rating 2");
116.  lblNewLabel_5.setBounds(10, 310, 241, 14);
117.  frame.getContentPane().add(lblNewLabel_5);
118.  JLabel lblNewLabel_6 = new JLabel("New label");
119.  lblNewLabel_6.setBounds(418, 310, 270, 14);
120.  frame.getContentPane().add(lblNewLabel_6);
121.  JLabel lblNewLabel_7 = new JLabel("Envir_Score_Confidence_Rating");
122.  lblNewLabel_7.setBounds(10, 359, 241, 14);
123.  frame.getContentPane().add(lblNewLabel_7);
124.  JLabel lblNewLabel_8 = new JLabel("New label");
125.  lblNewLabel_8.setBounds(418, 359, 270, 14);
126.  frame.getContentPane().add(lblNewLabel_8);
127.  JLabel lblNewLabel_9 = new JLabel("Envir_Score_Confidence_Rating 2");
128.  lblNewLabel_9.setBounds(10, 408, 241, 20);
129.  frame.getContentPane().add(lblNewLabel_9);
130.  JLabel lblNewLabel_10 = new JLabel("New label");
131.  lblNewLabel_10.setBounds(418, 408, 270, 14);
132.  frame.getContentPane().add(lblNewLabel_10);
133.  JLabel lblNewLabel_11 = new JLabel("Confidence_Rating");
134.  lblNewLabel_11.setBounds(10, 459, 241, 14);
135.  frame.getContentPane().add(lblNewLabel_11);
136.  JLabel lblNewLabel_12 = new JLabel("New label");
137.  lblNewLabel_12.setBounds(418, 459, 270, 14);
138.  frame.getContentPane().add(lblNewLabel_12);
139.  JButton btnNewButton_1 = new JButton("Connect");
140.  btnNewButton_1.addActionListener(new ActionListener() {
141.    public void actionPerformed(ActionEvent arg0) {
142.      testopendaylight();
143.    }
144.  });
145.  btnNewButton_1.setBounds(571, 12, 117, 25);
146.  frame.getContentPane().add(btnNewButton_1);
147.  JButton btnNewButton_2 = new JButton("Nodes in Network");
148.  btnNewButton_2.addActionListener(new ActionListener() {
149.    public void actionPerformed(ActionEvent arg0) {
150.      testopendaylight.getNodes("admin", "admin", "http://127.0.0.1:8181")
151.    }
152.  });
153.  btnNewButton_2.setBounds(495, 47, 193, 25);
154.  frame.getContentPane().add(btnNewButton_2);
btnNewButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println(textField.getText().toString());
        System.out.println(textField_1.getText().toString());
        System.out.println(textField_2.getText().toString());
        testdo(Float.parseFloat(textField.getText().toString()), Float.parseFloat(textField_1.getText().toString()), Float.parseFloat(textField_2.getText().toString()), 1);
    }
});

protected void testopendaylight() {
    // TODO Auto-generated method stub
    String user = "admin";
    String password = "admin";
    String baseURL = "http://127.0.0.1:8181/controller/nb/v2/flowprogrammer";
    String containerName = "default";
    try {
        URL url = new URL(baseURL + "/");
        String authStr = user + ":" + password;
        String encodedAuthStr = Base64.encodeBase64String(authStr.getBytes());
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Authorization", "Basic " + encodedAuthStr);
        connection.setRequestProperty("Accept", "application/json");
        // Get the response from connection's inputStream
        InputStream input = connection.getInputStream();
        BufferedReader in = new BufferedReader(new InputStreamReader(input));
        String line = "";
        while ((line = in.readLine()) != null) {
            System.out.println(line);
            System.out.println("---------------------------------------------------");
        }
        System.out.println("connection Successfull");
        System.out.println("---------------------------------------------------");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void read(String inputfile) throws IOException {
    File inputworkbook = new File(inputfile);
    Workbook w;
    try {
        w = Workbook.getWorkbook(inputworkbook);
        Sheet s = w.getSheet(0);
for (int j = 0; j < s.getColumns(); j++) {
    for (int i = 0; i < s.getRows(); i++) {
        Cell c = s.getCell(j, i);
        CellType ct = c.getType();
        if (ct == CellType.LABEL) {
            System.out.println("Labrl" + c.getColumn());
        } else if (ct == CellType.NUMBER) {
            System.out.println("Number " + c.getColumn());
        }
    }
}

void testf(File file) throws IOException {
    float a1 = 0, a2 = 0, a3;
    FileInputStream fis = new FileInputStream(file);
    XSSSSFWorkbook wb = new XSSSSFWorkbook(fis);
    XSSFSheet sheet = wb.getSheetAt(0);
    XSSFSheet sheet1 = wb.getSheetAt(2);
    XSSFSheet sheet2 = wb.getSheetAt(3);
    FormulaEvaluator fe = wb.getCreationHelper().createFormulaEvaluator();
    int a = 0, check = 0;
    for (Row row: sheet1) {
        for (org.apache.poi.ss.usermodel.Cell c: row) {
            //System.out.println("Cell is at "+c.getColumnIndex()+" "+c.getRowIndex()+" "+c.toString());
            if (c.getColumnIndex() == 6 && c.getRowIndex() == 13) {
                // System.out.println("Base Score is "+c.getNumericCellValue());
                bs = (float) c.getNumericCellValue();
            }
        }
    }
    for (Row row: sheet2) {
        for (org.apache.poi.ss.usermodel.Cell c: row) {
            if (c.getColumnIndex() == 7 && c.getRowIndex() == 13) {
                // System.out.println("Environmental Score is "+c.getNumericCellValue());
                es = (float) c.getNumericCellValue();
            }
        }
    }
    for (Row row: sheet) {
        for (org.apache.poi.ss.usermodel.Cell c: row) {
            switch (fe.evaluateInCell(c).getCellType()) {
                case org.apache.poi.ss.usermodel.Cell.CELL_TYPE_STRING:
                    //System.out.println("String is"+c.getStringCellValue()+"\t\t");
                    if (c.getStringCellValue().matches("test 1 "))
                        {
                    }
System.out.println("Matched");
a = 1;
}  else if (c.getStringCellValue().matches("test 2")) {
a = 2;
}  else if (c.getStringCellValue().matches("test 3")) {
a = 3;
}  else if (c.getStringCellValue().matches("test 4")) {
a = 4;
}  else if (c.getStringCellValue().matches("test 5")) {
a = 5;
}
break;

case org.apache.poi.ss.usermodel.Cell.CELL_TYPE_NUMERIC:
if (a == 1) {
    //System.out.println("Values is "+c.getNumericCellValue()+"\t\t");
    check++;
    if (check == 1) {
        a1 = (float) c.getNumericCellValue();
        // System.out.println("Pre is "+a1);
    }
    if (check == 2) {
        a2 = (float) c.getNumericCellValue();
        // System.out.println("Actual is "+a2);
    } else
        if (check == 3) {
            a3 = (float) c.getNumericCellValue();
            // System.out.println("Post is "+a3);
            check = 0;
            testdo(a1, a2, a3, 1);
        }
    } else if (a == 2) {
        //System.out.println("Values is "+c.getNumericCellValue()+"\t\t");
        check++;
        if (check == 1) {
            a1 = (float) c.getNumericCellValue();
            // System.out.println("Pre is "+a1);
        }
        if (check == 2) {
            a2 = (float) c.getNumericCellValue();
            // System.out.println("Actual is "+a2);
        } else if (check == 3) {
            a3 = (float) c.getNumericCellValue();
            //System.out.println("Post is "+a3);
            check = 0;
            testdo(a1, a2, a3, 2);
        }
    } else if (a == 3) {
        //System.out.println("Values is "+c.getNumericCellValue()+"\t\t");
        check++;
        if (check == 1) {
            a1 = (float) c.getNumericCellValue();
            // System.out.println("Pre is "+a1);
        }
        }
if(check == 2) {
    a2 = (float) c.getNumericCellValue();
    // System.out.println("Actual is "+a2);
} else if (check == 3) {
    a3 = (float) c.getNumericCellValue();
    //System.out.println("Post is "+a3);
    check = 0;
    testdo(a1, a2, a3, 3);
}
else if (a == 4) {
    //System.out.println("Values is "+c.getNumericCellValue()+"\t\t");
    check++;
    if (check == 1) {
        a1 = (float) c.getNumericCellValue();
        // System.out.println("Pre is "+a1);
    }
    if(check == 2) {
        a2 = (float) c.getNumericCellValue();
        // System.out.println("Actual is "+a2);
    } else if (check == 3) {
        a3 = (float) c.getNumericCellValue();
        //System.out.println("Post is "+a3);
        check = 0;
        testdo(a1, a2, a3, 4);
    }
} else if (a == 5) {
    //System.out.println("Values is "+c.getNumericCellValue()+"\t\t");
    check++;
    if (check == 1) {
        a1 = (float) c.getNumericCellValue();
        // System.out.println("Pre is "+a1);
    }
    if(check == 2) {
        a2 = (float) c.getNumericCellValue();
        // System.out.println("Actual is "+a2);
    } else if (check == 3) {
        a3 = (float) c.getNumericCellValue();
        //System.out.println("Post is "+a3); //check=0;
        testdo(a1, a2, a3, 5);
    }
} else {
    break;
    default:
    break;
}

///direct working
void basescore(float s, float m, float v) {
    float smv, impact;

    smv = s * m * v * 20;
    System.out.println("SMV :" + smv);

    impact = (float)(10.41 * (1 - (1 - 0.66) * (1 - 0.275) * (1 - 0.275)));
    System.out.println("Impact is :" + impact);

    base_score = (float)((0.6 * impact + 0.4 * smv - 1.5) * 1.176);
    System.out.println("Base score is "+ base_score);
private void testdo(float a1, float a2, float a3, int v) {
    float m;
    float add;
    float sa1, sa2, sa3;
    float sra1, sra2, sra3, tr;
    if (v == 1) {
        System.out.println("<-------------------------------");
        System.out.println("Test 1");
        System.out.println("Pre is "+a1);
        System.out.println("Actual is "+a2);
        System.out.println("Post is "+a3);
        System.out.println("<-------------------------------");
    }
    else if (v == 2) {
        System.out.println("<-------------------------------");
        System.out.println("Test 2");
        System.out.println("Pre is "+a1);
        System.out.println("Actual is "+a2);
        System.out.println("Post is "+a3);
        System.out.println("<-------------------------------");
    }
    else if (v == 3) {
        System.out.println("<-------------------------------");
        System.out.println("Test 3");
        System.out.println("Pre is "+a1);
        System.out.println("Actual is "+a2);
        System.out.println("Post is "+a3);
        System.out.println("<-------------------------------");
    }
    else if (v == 4) {
        System.out.println("<-------------------------------");
        System.out.println("Test 4");
        System.out.println("Pre is "+a1);
        System.out.println("Actual is "+a2);
        System.out.println("Post is "+a3);
        System.out.println("<-------------------------------");
    }
}
if (v == 5) {
    System.out.println("<---------------------------------------->");
    System.out.println("Test 5");
    System.out.println("<---------------------------------------->");
    System.out.println("Pre is " + a1);
    System.out.println("Actual is " + a2);
    System.out.println("Post is " + a3);
    System.out.println("<---------------------------------------->");
}
add = a1 + a2 + a3;
m = a1 + a2 + a3;
m = Float.parseFloat(format(m, 2));
System.out.println("Add is " + m);
m = m / 3;
m = Float.parseFloat(format(m, 2));
System.out.println("Mean is " + m);

sa1 = a1 - m;
sa2 = a2 - m;
sa3 = a3 - m;
sa1 = Float.parseFloat(format(sa1, 2));
sa2 = Float.parseFloat(format(sa2, 2));
sa3 = Float.parseFloat(format(sa3, 2));

System.out.println("First Value is " + sa1);
System.out.println("Second Value is " + sa2);
System.out.println("Third Value is " + sa3);
sra1 = (float) Math.pow(sa1, 2);
sra2 = (float) Math.pow(sa2, 2);
sra3 = (float) Math.pow(sa3, 2);

System.out.println("Square of first " + sra1);
System.out.println("Square of Second " + sra2);
System.out.println("Square of Third " + sra3);
tr = sra1 + sra2 + sra3;
tr = (float)(tr * 0.4);
tr = (float) Math.sqrt(tr);
tr = (float)(tr + 0.0001);
System.out.println("Original Standard Devation is:" + tr);
tr = Float.parseFloat(format(tr, 5));

//System.out.println("Affter conversion SD "+tr);
if (tr == 0) {
    System.out.println("Standard Devation(If Zero):" + 1);
} else {
    System.out.println("Affter conversion SD "+ tr);
}
System.out.println("No. of Measures:" + 1);
System.out.println("Original Sum is:" + add);
add = Float.parseFloat(format(add, 5));
System.out.println("5 DP sum is " + add);
System.out.println("5 DP Ratio of Deviation:" + add / tr);
float rsf = Float.parseFloat(format(tr, 5));
System.out.println("5 DP Raw Score Factor :" + rsf);
float median = a1 + a3;
median = median / 2;
System.out.println("Raw Score Factor second column " + median);

median = Float.parseFloat(format(median, 5));
System.out.println("5 DP Raw Score Factor second column " + median);

range_lowest_dev = (float)(rsf * 0.25);
range_lowest_dev = (float)(median - range_lowest_dev);
System.out.println("Range Lowest Deviation " + range_lowest_dev);
range_lowest_dev = Float.parseFloat(format(range_lowest_dev, 5));
System.out.println("5 DP Range Lowest Deviation " + range_lowest_dev);

range_lowest_dev2 = (float)(rsf * .25);
range_lowest_dev2 = (float)(median + range_lowest_dev2);
System.out.println("Range Lowest Deviation second column " + range_lowest_dev2);
range_lowest_dev2 = Float.parseFloat(format(range_lowest_dev2, 5));
System.out.println("5 DP Range Lowest Deviation second column " + range_lowest_dev2);

range_low_dev = (float)(rsf * 0.66);
range_low_dev = range_lowest_dev - range_low_dev;
System.out.println("Range Low Deviation " + range_low_dev);
range_low_dev = Float.parseFloat(format(range_low_dev, 5));
System.out.println("5 DP Range Low Deviation " + range_low_dev);

range_low_dev2 = range_low_dev2 + range_lowest_dev2;
System.out.println("Range Low Deviation second column " + range_low_dev2);
range_low_dev2 = Float.parseFloat(format(range_low_dev2, 5));
System.out.println("5 DP Range Low Deviation second column :" + range_low_dev2);

range_partial_dev = rsf * 1;
range_partial_dev = range_low_dev - range_partial_dev;
System.out.println("Range Partial Deviation " + range_partial_dev);
range_partial_dev = Float.parseFloat(format(range_partial_dev, 5));
System.out.println("5 DP Range Partial Deviation " + range_partial_dev);

range_partial_dev2 = rsf * 1;
range_partial_dev2 = range_partial_dev2 + range_low_dev2;
System.out.println("Range Partial Deviation second column " + range_partial_dev2);
range_partial_dev2 = Float.parseFloat(format(range_partial_dev2, 5));
System.out.println("5 DP Range_Partial_Deviation second column "+ range_partial_dev2);

Float range_high_dev = (float)(rsf * 1.5);
range_high_dev = range_partial_dev - range_high_dev;
System.out.println("Range_High_Deviation "+ range_high_dev);
range_high_dev = Float.parseFloat(format(range_high_dev, 5));
System.out.println("5 DP Range_High_Deviation "+ range_high_dev);

Float range_high_dev2 = (float)(rsf * 1.5);
range_high_dev2 = range_partial_dev2 + range_high_dev2;
System.out.println("Range_High_Deviation second column "+ range_high_dev2);
range_high_dev2 = Float.parseFloat(format(range_high_dev2, 5));
System.out.println("5 DP Range_High_Deviation second column "+ range_high_dev2);

float rawscorepree = raw_score(a1, range_high_dev, range_partial_dev, range_low_dev, range_lowest_dev, range_high_dev2, range_partial_dev2, range_low_dev2, range_lowest_dev2);
System.out.println("Raw Score of Pre "+ rawscorepree);
rawscorepree = Float.parseFloat(format(rawscorepree, 5));
System.out.println("5 DP Raw Score of Pre "+ rawscorepree);

float rawscoreactual = raw_score(a2, range_high_dev, range_partial_dev, range_low_dev, range_lowest_dev, range_high_dev2, range_partial_dev2, range_low_dev2, range_lowest_dev2);
System.out.println("Raw Score of Actual "+ rawscoreactual);
rawscoreactual = Float.parseFloat(format(rawscoreactual, 5));
System.out.println("5 DP Raw Score of Actual "+ rawscoreactual);

float rawscorepost = raw_score(a3, range_high_dev, range_partial_dev, range_low_dev, range_lowest_dev, range_high_dev2, range_partial_dev2, range_low_dev2, range_lowest_dev2);
System.out.println("Raw Score of Post "+ rawscorepost);
rawscorepost = Float.parseFloat(format(rawscorepost, 5));
System.out.println("5 DP Raw Score of Post "+ rawscorepost);

//clear up
bs = (float) 5.31;
System.out.println("Base Score of Pre "+ bs);
System.out.println("Base Score of Actual "+ bs);
System.out.println("Base Score of Post "+ bs);
bs = Float.parseFloat(format(bs, 5));
System.out.println("5 DP Base Score of Pre "+ bs);
System.out.println("5 DP Base Score of Actual "+ bs);
System.out.println("5 DP Base Score of Post "+ bs);

es = (float) 3.53;
System.out.println("Environmental Score of Pre "+ es);
System.out.println("Environmental Score of Actual "+ es);
System.out.println("Environmental Score of Post "+ es);
es = Float.parseFloat(format(es, 5));
System.out.println("5 DP Environmental Score of Pre "+ es);
System.out.println("5 DP Environmental Score of Actual " + es);
System.out.println("5 DP Environmental Score of Post " + es);
Float final_weighted_measure1 = rawscorepree + bs + es;
System.out.println("Final Weighted Measure of Pre " + final_weighted_measure1);
final_weighted_measure1 = Float.parseFloat(format(final_weighted_measure1, 5));
System.out.println("5 DP Final Weighted Measure of Pre " + final_weighted_measure1);

Float final_weighted_measure2 = rawscoreactual + bs + es;
System.out.println("Final Weighted Measure of Actual " + final_weighted_measure2);
final_weighted_measure2 = Float.parseFloat(format(final_weighted_measure2, 5));
System.out.println("5 DP Final Weighted Measure of Actual " + final_weighted_measure2);

Float final_weighted_measure3 = rawscorepost + bs + es;
System.out.println("Final Weighted Measure of Post " + final_weighted_measure3);
final_weighted_measure3 = Float.parseFloat(format(final_weighted_measure3, 5));
System.out.println("5 DP Final Weighted Measure of Post " + final_weighted_measure3);

float basescore_confidence_rating2 = ((rawscorepree + bs) + (rawscoreactual + bs) + (rawscorepost + bs)) / 3;
float basescore_confidence_rating1 = basescoreconfidencerating1(basescore_confidence_rating2);
System.out.println("Base Score Confidence Rating is " + basescore_confidence_rating1);
lblNewLabel_4.setText("" + basescore_confidence_rating1);
System.out.println("BaseScore Confidence Rating second column is "+basescore_confidence_rating2);
lblNewLabel_6.setText("" + basescore_confidence_rating2);

basescore_confidence_rating1 = Float.parseFloat(format(basescore_confidence_rating1, 5));
System.out.println("5 DP Base Score Confidence Rating is "+basescore_confidence_rating1);
basescore_confidence_rating2 = Float.parseFloat(format(basescore_confidence_rating2, 5));
System.out.println("5 DP BaseScore Confidence Rating second column is "+basescore_confidence_rating2);

float envir_score_confidence_rating2 = ((rawscorepree + es) + (rawscoreactual + es) + (rawscorepost + es)) / 3;
float envir_score_confidence_rating1 = envirscoreconfidencerating1(envir_score_confidence_rating2);
System.out.println("EnvirScore Confidence Rating is "+envir_score_confidence_rating1);
lblNewLabel_8.setText("" + envir_score_confidence_rating1);
System.out.println("EnvirScore Confidence Rating second column is "+envir_score_confidence_rating2);
lblNewLabel_10.setText("" + envir_score_confidence_rating2);

envir_score_confidence_rating1 = Float.parseFloat(format(envir_score_confidence_rating1, 5));
System.out.println("5 DP EnvirScore Confidence Rating is "+envirscore_confidence_rating1);

envir_score_confidence_rating2 = Float.parseFloat(format(envir_score_confidence_rating2, 5));
System.out.println("5 DP EnvirScore Confidence Rating second column is "+envir_score_confidence_rating2);

float confidencerating2 = (basescore_confidence_rating2 + envir_score_confidence_rating2) / 2;
System.out.println("Confidence Rating is " + confidencerating1);
lblNewLabel_12.setText(""+ confidencerating1);
System.out.println("Confidence Rating second column is " + confidencerating2);
confidencerating2 = Float.parseFloat(format(confidencerating2, 5));
System.out.println("5 DP Confidence Rating is " + confidencerating1);
confidencerating2 = Float.parseFloat(format(confidencerating2, 5));
System.out.println("5 DP Confidence Rating second column is " + confidencerating2);
}

public String format(Number n, int a) {
NumberFormat format = DecimalFormat.getInstance();
format.setRoundingMode(RoundingMode.FLOOR);
format.setMinimumFractionDigits(0);
format.setMaximumFractionDigits(a);
return format.format(n);
}

public float raw_score(float c4, float m4,
float l4, float k4, float j4, float m5,
float l5, float k5, float j5) {
float C4, M4, L4, K4, J4, M5, L5, K5,
J5;
C4 = c4;
M4 = m4;
L4 = l4;
K4 = k4;
J4 = j4;
M5 = m5;
L5 = l5;
K5 = k5;
J5 = j5;
if (C4 < M4) {
return 1;
} else {
if (C4 >= M4 && C4 < L4) {
return (float) 7.75;
} else {
if (C4 >= L4 && C4 < K4) {
return (float) 5.5;
} else {
if (C4 >= K4 && C4 < J4) {
return (float) 3.25;
} else {
if (C4 >= J4 && C4 <= J5) {
return 1;
} else {
if (C4 > J5 && C4 <= K5) {
    return (float) 3.25;
} else {
    if (C4 > K5 && C4 <= L5) {
        return (float) 5.5;
    } else {
        if (C4 > L5 && C4 <= M5) {
            return (float) 7.75;
        } else {
            return 10;
        }
    }
}

public Float basescoreconfidencerating1(Float r5) {
    Float R5;
    R5 = r5;
    if (R5 > 9) {
        return (float) 1;
    } else {
        if (R5 > 7 && R5 <= 8.9999999) {
            return (float) 2;
        } else {
            if (R5 > 4 && R5 <= 6.9999999) {
                return (float) 3;
            } else {
                if (R5 > 0 && R5 <= 3.99999999) {
                    return (float) 4;
                } else {
                    return (float) 5;
                }
            }
        }
    }
}

public float envirscoreconfidencerating1(float s5) {
    float S5 = s5;
    if (S5 > 9) {
        return 1;
    } else {
        if (S5 > 7 && S5 <= 8.9999999) {
            return 2;
        } else {
            if (S5 > 4 && S5 <= 6.99999999) {
                return 3;
            } else {
                if (S5 > 0 && S5 <= 3.99999999) {
                    return 4;
                } else {
                    return 0;
                }
            }
        }
    }
}

public float confidencerating1(float t5) {
    float T5;
    T5 = t5;
    if (T5 > 9) {
        return 1;
else {
    if (T5 > 7 && T5 <= 8.9999999999999) {
        return 2;
    }
    if (T5 > 4 && T5 <= 6.9999999999999) {
        return 3;
    }
    if (T5 > 0 && T5 <= 3.9999999999999999) {
        return 4;
    }
    else {
        return 0;
    }
}
Appendix C – Framework OpenDaylight Controller Application

```java
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import org.apache.commons.codec.binary.Base64;
import org.codehaus.jettison.json.JSONObject;
import org.opendaylight.openflowplugin.applications.lldpspeaker.LLDPSpeaker;
import org.opendaylight.openflowplugin.applications.lldpspeaker.LLDPUtil;
import org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.yang.types.rev130715.MacAddress;
import org.opendaylight.yang.gen.v1.urn.opendaylight.flow.inventory.rev130819.FlowCapableNodeConnector;
import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeConnector;
import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeConnectorId;
import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeConnectorRef;
import org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.NodeId;
import org.opendaylight.yangtools.yang.binding.InstanceIdentifier;
import org.opendaylight.yangtools.yang.common.RpcResult;
public class testopendayy extends LLDPSpeaker implements PacketProcessingService {
    private PacketProcessingService packetProcessingService;
    private ScheduledExecutorService scheduledExecutorService;
    private ScheduledFuture scheduledSpeakerTask;
    private final MacAddress destinationMACAddress = null;
    private LLDPSpeaker lldpSpeaker;
    static {
        MacAddress mac = new MacAddress("01:23:45:67:89:AB");
    }
}
```
```java
public testopenDayy(PacketProcessingService packetProcessingService, MacAddress addressDestination)
{
    super(packetProcessingService, addressDestination);
}

public static JSONObject getNodes(String user, String password, String baseURL)
{
    StringBuffer result = new StringBuffer();
    try {
        if (!baseURL.contains("http")) {
            System.out.println("Not containing");
            baseURL = "http://" + baseURL;
        }
        baseURL = baseURL + "/controller/nb/v2/switchmanager/default/nodes";
        System.out.println(baseURL);
        // Create URL = base URL + container
        URL url = new URL(baseURL);
        // Create authentication string and encode it to Base64
        String authStr = user + ":" + password;
        String encodedAuthStr = Base64.encodeBase64String(authStr.getBytes());
        // Create Http connection
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        // Set connection properties
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Authorization", "Basic " + encodedAuthStr);
        connection.setRequestProperty("Accept", "application/json");
        System.out.println(connection + "");
        // Get the response from connection's inputStream
        InputStream content = (InputStream) connection.getInputStream();
        BufferedReader in = new BufferedReader(new InputStreamReader(content));
        // result.append("{");
        String line = "";
        while ((line = in.readLine()) != null) {
            result.append(line);
            System.out.println(result);
        }
        // System.out.println("org.codehaus.jettison.json.JSONException: A
        //JSONObject text must begin with '{
        //at character 1 of "+result.toString());
        //JSONArray nodes = new JSONArray(result.toString());
        System.out.println(nodes + "");
        //Future<RpcResult<java.lang.Void>> transmitPacket(TransmitPacketInput input);
        return nodes;
    } catch (Exception e) {
        e.printStackTrace();
        // e.getMessage();
    }
    return null;
```
void test()
{
    lldpSpeaker = new LLDPSpeaker(packetProcessingService, scheduledExecutorService, destinationMACAddress);
    lldpSpeaker.setOperationalStatus(OperStatus.RUN);
}

public void testStandBy()
{
    lldpSpeaker.setOperationalStatus(OperStatus.STANDBY); // Add node connector - LLDP packet should be transmitted through packetProcessingService // lldpSpeaker.nodeConnectorAdded(id, fcnc); // Execute one iteration of periodic task - LLDP packet should be transmitted second time
    lldpSpeaker.run();
    // Check packet transmission // verify(packetProcessingService, times(1)).transmitPacket(packet); // verifyNoMoreInteractions(packetProcessingService);
}

public void testNodeConnectorAdd()
{
    lldpSpeaker.run();
    // Check packet transmission // verify(packetProcessingService, times(2)).transmitPacket(packet); // verifyNoMoreInteractions(packetProcessingService);
}

@Override
public Future< RpcResult< Void >> transmitPacket(TransmitPacketInput arg0) { // TODO Auto-generated method stub
    return null;
}
Appendix D – SMV Survey & Results

Survey of Network Security Metrics Utility

Research:
We intend to design and implement a framework for SDN network security confidence analysis. In order to enhance traditional analysis, we utilize SDN in two key areas; route and destination verification and switch metrics analysis. This framework will allow SDN authentication applications to validate and verify the routing and destination of data as well as assess the network devices for unexpected behavior (i.e., data compromise, man-in-the-middle attacks, etc.)

Purpose of Survey:
In order to better assess the value and qualities of these various networking metrics, we would like your evaluation.

Sample Scenario:
Use the Scenario and Figure below to answer the questions on this survey:

To better highlight the metrics collected and how they may be analyzed, reference the example network topology in the figure below and assume the host A is sending a sensitive message to host C.
General Criteria:
For assessing the meters/metrics in general, we propose the following criteria:

S: Spoofability--this measures the ability of the metric in general to be falsified in some manner.

M: Measurability – the degree of exactness that SDN allows for measure (whether subjective or objective metric)

V: Variability – measures the range of acceptable values that would be considered within bounds for a given metric.
General Criteria Grid:

<table>
<thead>
<tr>
<th>Scoring/Criteria</th>
<th>Spoofability</th>
<th>Measurability</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Very difficult to spoof and/or easily recognized by most</td>
<td>Easily assessed metric with limited network overhead</td>
<td>Very narrow band of acceptable values</td>
</tr>
<tr>
<td>4</td>
<td>Hard to spoof and/or easily recognized by some</td>
<td>Easily assessed with moderate network overhead</td>
<td>Small band of acceptable values</td>
</tr>
<tr>
<td>3</td>
<td>Alterable and/or recognized with some training/effect</td>
<td>Assessable metric</td>
<td>Moderate band of acceptable values</td>
</tr>
<tr>
<td>2</td>
<td>Alterable by many and/or hard to recognize</td>
<td>Hard to assess and may reduce performance due to overhead</td>
<td>Large array of acceptable values within a single band</td>
</tr>
<tr>
<td>1</td>
<td>Easily altered and/or very difficult to recognize</td>
<td>Very difficult to assess metric with considerable overhead</td>
<td>Wide array of acceptable values, potentially in different bands</td>
</tr>
</tbody>
</table>

Survey Questions

Example SDN Topology:

Variables:

\[ P = \text{Packet} \]

\[ Cr = \text{SDN Controller} \]

\[ F = \text{Flow} \]

\[ T = \text{Time} \]

\[ x, y, z = \text{Transmission Number} \]

\[ n = \text{Size/Bytes} \]

1. Route Verification
Utilizing SDN to provide and compare the expected route (from the controller’s flow table) to the actual route that data flows from the switch forwarding rules. For example, a packet (Px) traveling from Host A to Host C (Reference the diagram above as needed).

Path: PRx, -> A, 1, Cr, 1, 2, 4, C

Verification Track: Px -> A,1,Cr,1,2,Cr,2,4,Cr,4,C

<table>
<thead>
<tr>
<th>Spoofability:</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

2. Packet/Flow Size

Measure the size of the first and last packets. Packet size can then be compared as the data flows from switch to switch and from each of the three elements of message traffic (pre/post and actual)

\[
\begin{align*}
(P_x1 + P_x2 + P_x4).
(P_y1 + P_y2 + P_y4).
(P_z1 + P_z2 + P_z4).
\end{align*}
\]

<table>
<thead>
<tr>
<th>Spoofability:</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

3. Packet Arrival Time to Controller

Assessing the median arrival time from pre/post and actual traffic will provide a metric for any future transmission along the same route. If traffic were intended for multiple recipients, then the arrival time of similar segments of the routing could be compared as well.
4. Packet/Flow Lapse Time

Measuring the packets arrival at two switches provides metrics that should be validated with performance traceroute to assess speed and detect man in the middle attacks.

<table>
<thead>
<tr>
<th>Spoofability</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

5. Packet/Flow Duration

The arrival time of the first packet until the flow exits the switch. Knowing flow sizes and types/protocol, we could compare the traffic against performance tests and standardized metrics to assess and create expectations for transport time and routing. Comparing the time a flow spends in the Flow Table may help determine if a high volume of malicious data is traveling utilizing the same flow table entry (i.e., many flows, but few packets).

<table>
<thead>
<tr>
<th>Spoofability</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

6. Hop Count

This is a simple metric and is available conventionally; however, the SDN controller could have a much more accurate estimation of the hop count per recipient. This estimation helps validate the path, eliminate routing to devices well outside of the network/system control,
and would have limited overhead. It is important to note that hop count can be changed by
the controller/installed Flow Entry Action.

<table>
<thead>
<tr>
<th>Spoofability</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

7. Switch/Device Location

This refers to the geospatial or at least time zone location of a device. This could be used
with varying degrees of trust to assess the strength of a partial route. Some larger level
(ISP) or company internal switches would have a higher level of trust versus the open
internet. Location data such as an authorized IP range could identify a switch/network
owner, combined with an external entity like IANA or a business IT department.

<table>
<thead>
<tr>
<th>Spoofability</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

8. Switch/Device Characteristics

Knowing the type of switch, level within the LAN/WAN hierarchy, and switch owner
could all be utilized to develop an algorithm for trust with the network. For example
physical vs. virtual switches – typically a physical switch would have a higher degree of
trust as it is harder to hack/spoof.

<table>
<thead>
<tr>
<th>Spoofability</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
9. Sender/Receiver Role Based-Access

Linking Active Directory, RADIUS Server, etc., traffic could be validated by message type, access of the users and/or user group, pull in location data of the group and compare routing data. This data could also be held or queued if a user was identified as logged off, so it would not flow to the device until the user was logged in and available to receive it.

<table>
<thead>
<tr>
<th>Spoofability:</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

10. Average Number of Packets per Flow

Data transmission is a two-way street, so it is equally as important to ensure the safety of the receiving node from malicious attack. A side effect of malicious spoofing is the generation of flows with a small number of packets, i.e., about 3 packets per flow. Given that normal traffic usually involves a higher number of packets. If we can determine a median value for this, then we can assess confidence.

<table>
<thead>
<tr>
<th>Spoofability:</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

11. Median Bytes per Flow

Continuing on receiving node security, the attack payload size is often very small (for example TCP flooding attacks typically contain packets of ~100 bytes). If we can determine a median value for this, then we can assess confidence.

Verify how many pair-flows occur in the flow stream during a certain interval. Malicious activity often increases the number of single-flows into the network because they send packets with a fake IP.

13. Packet Timestamp Comparison

By employing timestamping on the first packet in and last packet out of a given flow, we can assess exactly when traffic enters and exits SDN hybrid-networks. Based upon experience, we can identify if traffic is flowing at a different rate. Taking into account, any deviation would at a minimum degrade our confidence in the data path. Additionally, this could potentially hash the packet header and timestamp. Passing this hash to an authentication server (which knows the header) would validate the packet based upon the returned timestamp.

14. Percentage of Correlative Flows
The Destination has the capability/requirement to reply to packet request (whether legitimate or not). Under normal condition, the rate of traffic from the Destination back to the source is typically half, whereas illegitimate requests would see a near zero result.

<table>
<thead>
<tr>
<th>Spoofability:</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Variability:</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
### Sample of the Survey Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**Average:**

<table>
<thead>
<tr>
<th>Route Verification</th>
<th>0.48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Size</td>
<td>0.48</td>
</tr>
<tr>
<td>Flow Size</td>
<td>0.48</td>
</tr>
<tr>
<td>Packet Arrival Time to Device</td>
<td>0.61</td>
</tr>
<tr>
<td>Packet Lapse Time</td>
<td>0.48</td>
</tr>
<tr>
<td>Flow Lapse Time</td>
<td>0.48</td>
</tr>
<tr>
<td>Packet Duration</td>
<td>0.48</td>
</tr>
<tr>
<td>Flow Duration</td>
<td>0.48</td>
</tr>
<tr>
<td>Hop Count</td>
<td>0.66</td>
</tr>
<tr>
<td>Device Location</td>
<td>0.61</td>
</tr>
<tr>
<td>Device Characteristics</td>
<td>0.48</td>
</tr>
<tr>
<td>Average of Packets per Flow</td>
<td>0.61</td>
</tr>
<tr>
<td>Median Bytes per Flow</td>
<td>0.48</td>
</tr>
<tr>
<td>Growth of Single Flows</td>
<td>0.61</td>
</tr>
<tr>
<td>Packet Timestamp Comparison</td>
<td>0.61</td>
</tr>
</tbody>
</table>

**Average of SMV Scores:**

| 4.705882353 | 2.117647059 | 4.529411765 | 4.058823529 | 3.823529412 | 3.647058824 |

**CVSS Scoring:**

| 0.48 | 0.48 |
Appendix E – Framework Data & Results

A small sample of the data captured and analyzed by the SDN Confidence Analysis framework.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Event Type</th>
<th>Time event started (ticks)</th>
<th>Duration of Event (Ticks)</th>
<th>Packet ID</th>
<th>Packet Type</th>
<th>Source node ID per IP Address</th>
<th>Transmitted bytes per MAC</th>
<th>Received bytes per MAC</th>
<th>Count of transmission</th>
<th>Drop Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>26117322</td>
<td>68330</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;1 6&gt;</td>
<td>61 78</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11088000</td>
<td>53985</td>
<td>52200</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>521852</td>
<td>62564</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;3 6&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52200</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52200</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>BTX</td>
<td>11054489</td>
<td>53880</td>
<td>52185</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
<tr>
<td>802.3 TX</td>
<td>TX</td>
<td>24105929</td>
<td>63090</td>
<td>31155929</td>
<td>DATA</td>
<td>&lt;0 0&gt;</td>
<td>62 64</td>
<td>0 NONE</td>
<td>0 NONE</td>
<td>NONE</td>
</tr>
</tbody>
</table>
Using the data above, I calculated the flow duration for a series of varying sized data transfer. Throughout the transfers the latency was manually increased to simulate a man-in-the-middle attack.
Appendix F – Large-Scale SDN Modeling and Simulation Setup

Hierarchical SDN Model

Simple Data Center:

Below is an example simple datacenter some number of racks, each with hosts and a single top-of-rack (ToR) switch. The ToR switches are connected to a central root switch. This represents a simple datacenter. This example includes the ability, from the command line, to specify how many hosts are in each rack and how many racks are in network [96].

```

A simple datacenter topology script for Mininet.

[ s1 ]================================.
 ,--'       |                       |
[ s1r1 ]-.[ s1r2 ]-. ... [ s1r# ]-.
[ h1r1 ]-| [ h1r2 ]-| ... [ h1r# ]-|
[ h2r1 ]-| [ h2r2 ]-| ... [ h2r# ]-|
 ... | ... | ... ... |
[ h#r1 ]-' [ h#r2 ]-' ... [ h#r# ]-'
```

```
from mininet.topo import Topo
from mininet.util import irange

class DatacenterConfigurableTopo( Topo):
    "Configurable Datacenter Topology"

    def build( self, numRacks=4, numHostsPerRack=4):
        self.racks = []
        rootSwitch = self.addSwitch( 's1' )
        for i in irange( 1, numRacks ):
            rack = self.buildRack( i, numHostsPerRack=numHostsPerRack )
            self.racks.append( rack )
        for switch in rack:
            self.addLink( rootSwitch, switch )
```
def buildRack( self, loc, numHostsPerRack ):
    "Build a rack of hosts with a top-of-rack switch"

dpid = ( loc * 16 ) + 1
switch = self.addSwitch( 's1r%s' % loc, dpid='%x' % dpid )

for n in irange( 1, numHostsPerRack ):
    host = self.addHost( 'h%sr%s' % ( n, loc ) )
    self.addLink( switch, host )

    # Return list of top-of-rack switches for this rack
return [switch]

# Allows the file to be imported using `mn --custom <filename> --topo
dcconfig`
topos = {
    'dcconfig': DatacenterConfigurableTopo
}

More Realistic Data Center:

In a real datacenter, there is often more than one root switch linked in a ring pattern
with at least two links going to each ToR switch. This allows for a failure of one of the root
switches without bringing down the entire network. To ensure full redundancy it is
necessary to configure it so that every rack has two ToR switches, each connected with a
single link to the root switches and provide every host with a connection to both ToR
switches in that rack [96].

""
A simple datacenter topology script for Mininet.

,--------------------------------------. Each root switch connected in
ring.
  [ s1 ]------[ s2 ]--- ... ---[ s# ]
     |        |         |   Each ToR switch connects to its
,='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='
    s3

""
from mininet.topo import Topo
from mininet.util import irange

class DatacenterHAFullTopo( Topo ):
    "Configurable Datacenter Topology"

    def build( self, numRacks=4, numHostsPerRack=4, numHASwitches=2 ):
        # This configuration only supports 15 or less root switches
        if numHASwitches >= 16:
            raise Exception( "Please use less than 16 HA switches" )

        self.racks = []
        rootSwitches = []
        lastRootSwitch = None

        # Create and link all the root switches
        for i in irange( 1, numHASwitches ):
            rootSwitch = self.addSwitch( 's%s' % i )
            rootSwitches.append( rootSwitch )

            # If we have initialized at least two switches, make sure to
            # connect them. This handles s1 -> s2 -> ... -> sN
            if lastRootSwitch:
                self.addLink( lastRootSwitch, rootSwitch )

            lastRootSwitch = rootSwitch

        # Make the final link from the last switch to the first switch
        if numHASwitches > 1:
            self.addLink( lastRootSwitch, rootSwitches[0] )

        for i in irange( 1, numRacks ):
            rack = self.buildRack( i, numHostsPerRack=numHostsPerRack,
                                   numHASwitches=numHASwitches )
self.racks.append( rack )

# For every HA switch, add a link between the rack switch and root
# switch of the same ID
for j in range( numHASwitches ):
    self.addLink( rootSwitches[j], rack[j] )

def buildRack( self, loc, numHostsPerRack, numHASwitches ):
    "Build a rack of hosts with a top-of-rack switch"

    switches = []
    for n in irange( 1, numHASwitches ):
        # Make sure each switch gets a unique DPID based on the location
        # in the rack for easy decoding when looking at logs.
        dpid = ( loc * 16 ) + n
        switch = self.addSwitch( 's%sr%s' % (n, loc), dpid='%x' % dpid )
        switches.append( switch )

    for n in irange( 1, numHasPerRack ):
        host = self.addHost( 'h%sr%s' % (n, loc) )

        # Add a link from every top-of-rack switch to the host
        for switch in switches:
            self.addLink( switch, host )

    # Return list of top-of-rack switches for this rack
    return switches

# Allows the file to be imported using `mn --custom <filename> --topo dchafull`
topos = {
    'dchafull': DatacenterHAFullTopo
}

**Multi-Tenancy SDN Model**

To gain an understanding of the basics of SDN network slicing and how to control different slices with different controller, I created the sample topology below. I will create
a network application that create multiple Layer-2 network slices for different portions of the flowspace [63].

The figure below illustrates a single SDN network with two different bandwidth paths (through switches s2 and s3) [97]. The script below will generate this topology:

```python
#!/usr/bin/python

from mininet.topo import Topo

class FVTopo(Topo):
    def __init__(self):
        # Initialize topology
        Topo.__init__(self)

        # Create template host, switch, and link
        hconfig= {'inNamespace':True}
        http_link_config= {'bw':1}
        video_link_config= {'bw':10}
        host_link_config= {}

        # Create switch nodes
```
for i in range(4):
    sconfig = {'dpid': '%016x' % (i+1)}
    self.addSwitch('s%d' % (i+1), **sconfig)

# Create host nodes
for i in range(4):
    self.addHost('h%d' % (i+1), **hconfig)

# Add switch links
self.addLink('s1', 's2', **http_link_config)
self.addLink('s2', 's4', **http_link_config)
self.addLink('s1', 's3', **video_link_config)
self.addLink('s3', 's4', **video_link_config)

# Add host links
self.addLink('h1', 's1', **host_link_config)
self.addLink('h2', 's1', **host_link_config)
self.addLink('h3', 's4', **host_link_config)
self.addLink('h4', 's4', **host_link_config)

topos = { 'fvtopo': ( lambda: FVTopo() ) }

To understand how an SDN “slice” and multi-tenancy intersect, I will explain how to divide the network into two separate slices, upper and lower. To implement this isolation, there is a need to block communication between hosts in different slices. This done by inserting drop rules on certain switches.
Creating Network Slices [98]

Each slice will be managed by a separate controller which will control all the traffic in its slice. To create a slice named upper connecting to a controller listening on tcp:localhost:10001 by running the following command:

```
$ fvctl -f /dev/null add-slice upper tcp:localhost:10001
admin@upperslice
```

Similarly, create a slice named lower connecting to a controller listening on tcp:localhost:10002.

```
$ fvctl -f /dev/null add-slice lower tcp:localhost:10002
admin@lowerslice
```

Ensure that the slices were added by running the following command:
$ fvctl -f /dev/null list-slices

Creating the Flowspaces [98]

Flowspaces associate packets of a particular type in the network to specific slices. When a packet matches multiple flowspaces, FlowVisor assigns it to the flowspace with the highest priority number.

- **match** describes a flow or flows. Such flow descriptions comprise a series of field=value assignments, separated by commas.
- **slice-perm** is a comma-separated list of slices that have control over a specific FlowSpace. slice-perm is of the form "slicename1=perm[slicename2=perm[...]]". Each slice can have three types of permissions over a flowspace: DELEGATE, READ, and WRITE. Permissions are a bitmask specified as an integer, with DELEGATE=1, READ=2, WRITE=4.

To create a flowspace named dpid1-port1 (with priority value 1) that maps all the traffic on port 1 of switch s1 to the upper slice by running the following command:

   $ fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=1 upper=7

Similarly, create a flowspace named dpid1-port3 that maps all the traffic on port 3 of switch s1 to the upper slice:

   $ fvctl -f /dev/null add-flowspace dpid1-port3 1 1 in_port=3 upper=7
Now create a flowspace for all the traffic at a switch by using the match value of any.

Use that technique to add switch s2 to the upper slice:

```
$ fvctl -f /dev/null add-flowspace dpid2 2 1 any upper=7
```

Next, create flowspaces to add ports 1 and 3 of switch s4 to the upper slice:

```
$ fvctl -f /dev/null add-flowspace dpid4-port1 4 1 in_port=1 upper=7
$ fvctl -f /dev/null add-flowspace dpid4-port3 4 1 in_port=3 upper=7
```

Ensure that the flowspaces are correctly added:

```
$ fvctl -f /dev/null list-flowspace
```

Then repeat to create flowspaces for the lower slice:

```
$ fvctl -f /dev/null add-flowspace dpid1-port2 1 1 in_port=2 lower=7
$ fvctl -f /dev/null add-flowspace dpid1-port4 1 1 in_port=4 lower=7
$ fvctl -f /dev/null add-flowspace dpid3 3 1 any lower=7
$ fvctl -f /dev/null add-flowspace dpid4-port2 4 1 in_port=2 lower=7
$ fvctl -f /dev/null add-flowspace dpid4-port4 4 1 in_port=4 lower=7
```

Ensure that the flowspaces are correctly added:

```
$ fvctl -f /dev/null list-flowspace
```

Start a controller instance for each slice. The controllers will reactively install routes based on the destination MAC address, and it is provided as an executable. Open two fresh terminal tabs and run the following:

- Terminal 1
$ ./pox.py openflow.of_01 -port=10001 flowvisor_lab1_upper

- Terminal 2

$ ./pox.py openflow.of_01 -port=10002 flowvisor_lab1_upper

This will launch two instances of the controller, listening on port 10001 and 10002 respectively.

Verify that h1 can ping h3 but not h2 and h4 (and vice versa). In the Mininet console run the following commands:

```
mininet> h1 ping -c1 h3
mininet> h1 ping -c1 -W1 h2
mininet> h1 ping -c1 -W1 h4
```

Verify that h2 can ping h4 but not h1 and h3 (and vice versa). In the mininet console run the following commands:

```
mininet> h2 ping -c1 h4
mininet> h2 ping -c1 -W1 h1
mininet> h2 ping -c1 -W1 h3
```

**SDN Emulation**

Step 1: Connect guest OS (i.e., Mininet) to the internet.

In the VirtualBox network setting, make sure the NAT interface is enabled to allow connection to the internet. The ip address should look like this: 10.0.3.15 (a class A
address). Test connectivity by pinging an internet (www.google.com) to make sure the connection to internet from the guest OS (i.e., Mininet) is functioning [65].

---

Step 2: Start the network

Start a Mininet. At a minimum create a network with a switch and two hosts:

```bash
sudo mn --switch ovsk --mac --topo single,2
```

Step 3: Connect the guest interface to the OpenvSwitch bridge

Use the `ovs-vsctl` command to configure openvswitchd(this is a process of OpenvSwitch).

Do this by opening a terminal window for switch 1, s1, as this command does not run directly on mininet. Check the OpenvSwitch configuration using the command: `ovs-vsctl show`

Run the following command to connect eth1 to s1: `ovs-vsctl add-port s1 eth1`

Check the configuration again using `ovs-vsctl show`. The new interface that is added has been highlighted in red.

```
root@mininet-vm:~# ovs-vsctl show
d27a9060-3edf-4ee7-a4cf-09e705c93f56
Bridge "s1"
```
Controller "ptcp:6634"
Controller "tcp:127.0.0.1:6633"
    is_connected: true
fail_mode: secure
Port "eth1"
    Interface "eth1"
Port "s1-eth1"
    Interface "s1-eth1"
Port "s1-eth2"
    Interface "s1-eth2"
Port "s1"
    Interface "s1"
    type: internal
ovs_version: "2.0.1"

Step 4: Run `dhclient` on hosts.

Open terminal windows for host 1, h1, and host 2, h2, and run the following commands to remove the ip from h1-eth0 and the second command gets the ip address for h1-eth0 from dhcp server.

```
root@mininet-vm:~# ifconfig h1-eth0 0
root@mininet-vm:~# dhclient h1-eth0
root@mininet-vm:~# ifconfig
h1-eth0   Link encap:Ethernet   HWaddr 00:00:00:00:00:01
          inet addr:10.0.3.16  Bcast:10.0.3.255  Mask:255.255.255.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:24 errors:0 dropped:0 overruns:0 frame:0
```
Next check the internet connectivity using ping.

root@mininet-vm:~# ping www.google.com
PING www.google.com (216.58.216.164) 56(84) bytes of data.
64 bytes from seal5s02-in-f4.4e100.net (216.58.216.164): icmp_seq=14
ttl=54 time=61.9 ms
64 bytes from seal5s02-in-f4.4e100.net (216.58.216.164): icmp_seq=15
ttl=54 time=60.7 ms
^C
--- www.google.com ping statistics ---
15 packets transmitted, 2 received, 86% packet loss, time 14065ms
rtt min/avg/max/mdev = 60.707/61.336/61.965/0.629 ms
Appendix G – University Small SDN Testbed

Tutorial on VMware Esxi SDN Testbed Setup

1. Within esxi, set IP info (static IP, mask, gateway, DNS, and host name).
2. Install vSphere Client on local PC
3. In local PC, modify “Drivers/etc/hosts” to include IP and name for esxi and vcenter
4. Launch vSphere Client
5. In vSphere Client, Deploy VSphere Appliance
   a. File – Deploy OVF Template, Thin Provision
6. In vSphere Client, do basic configurations
   a. Set Time: Configuration – Time Configuration – Properties – Add NTP Server
      “Start and Stop with Host”
   b. Enable SSH: Security Profile – Properties – SSH “Start and Stop with Host”
7. In vSphere Client, setup vCenter
   a. Edit Settings: Set RAM to 4GB
   b. Console View and Start VM, let is boot to bluescreen
8. On local PC (connected to esxi server router), Log on to provide IP and configure:
   modify time, “admin” for password, and network – address for static ip
9. On local PC, surf to sdnvcenter:xxxx
   a. Verify Time still set
10. WinSCP into vCenter to modify host file to remove host name from first line and add
    esxi and vcenter to bottom of file.
11. On local PC on sdnvcenter site, launch “setup wizard” with default settings.

12. On local PC on sdnvcenter site, reboot server (under system tab) & open in new console window

13. On local PC in vSphere Client, select esxi and VM Startup & Shutdown, modify to:
   a. Allow vm start & stop automatically
   b. Startup in 15 seconds and continue with VM tools
   c. Shutdown in 15 seconds and “guest shutdown”
   d. Move vCenter App into “Auto Startup” group

14. On local PC open new vSphere Client, log in to vCenter to verify connectivity.

15. On local PC goto https://vcenter and select “log in to vSphere Web Client”

16. On local PC, surf to sdnvcenter:xxxx
   a. Verify Time still set

17. On vSphere Web Client, select vCenter Servers – vcenter and then “create datacenter”
   a. Select “add a host” and type esxi, then next to finish

18. Optional – adding more storage (i.e., a data store) in Datacenters – Datacenter, select “add a datastore”
   a. Create “vmfs datastore”, select host “esxi” and select storage and give it a name
   b. Select vmfs 5 (allows for >2TB drives), select next to finish

19. Setup ISO Storage: Click add folder for ISO storage, then download and install “VMware Client Integration Plug-In” Note: You might need to enable plug-ins.

20. Uploading ISOs and existing VMs
Appendix H – Low-Cost SDN Router Setup

A Low-Cost SDN Router can be created using various older Linksys or D-Link WiFi routers and OpenWRT. Below I will describe the steps to download, format, and deploy such a router. The following assumes the user is working in a Linux environment [99][100].

1. Install the dependencies.

   • `sudo apt-get install build-essential binutils flex bison autoconf
gettext texinfo sharutils \ subversion libncurses5-dev ncurses-term
zlib1g-dev gawk`

2. Download OpenWRT source and feeds

   • `mkdir ~/openwrt`
   • `cd ~/openwrt`
   • `sudo apt-get update`
   • `svn co svn://svn.openwrt.org/openwrt/branches/backfire`
   • `cd backfire`
   • `./scripts/feeds update -a`
   • `./scripts/feeds install -a`

3. Get OpenWRT config file

   • `make menuconfig`

4. Configure OpenWRT with dependencies and build
5. Set up OpenFlow and select package.

- `sudo apt-get install git`
- `git clone https://github.com/CPqD/openflow-openwrt.git`
- `cd ~/openwrt/backfire/packages/`
- `ln -s ~/openwrt/openflow-openwrt/openflow-1.3/`
- `cd ~/ofwrt/backfire/`
- `ln -s ~/openwrt/openflow-openwrt/openflow-1.3/files`
- `cd ~/ofwrt/`
- `make menuconfig`
- `make kernel_menuconfig`
- `make`
## Appendix I – SDN OTG Parts List

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Cost/Per</th>
<th>Sub-Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zodiac FX 100Mb 4-Port SDN Switch</td>
<td>4</td>
<td>63.75</td>
<td>255.00</td>
</tr>
<tr>
<td>Raspberry Pi 3 Complete Starter Kit by Vilros</td>
<td>4</td>
<td>69.99</td>
<td>279.96</td>
</tr>
<tr>
<td>Ethernet Cables (multi-color), 3ft length, 8-pack</td>
<td>1</td>
<td>14.98</td>
<td>14.98</td>
</tr>
<tr>
<td>Ethernet Cables (orange), 3ft length, 4-pack</td>
<td>1</td>
<td>15.95</td>
<td>15.95</td>
</tr>
<tr>
<td>Belkin 12-port Power Strip (Model: BP112230-08)</td>
<td>1</td>
<td>27.49</td>
<td>27.49</td>
</tr>
<tr>
<td>Tek Republic USB Sharing Switch - 4 Port Manual Switch (Model: TUS-400)</td>
<td>1</td>
<td>16.99</td>
<td>16.99</td>
</tr>
<tr>
<td>USB Hub, 4-port by Dlink (Model: DUB-H4)</td>
<td>1</td>
<td>19.99</td>
<td>19.99</td>
</tr>
<tr>
<td>USB LAN Ethernet Adapter, 10/100Mb by Manhattan (Model: 506731)</td>
<td>4</td>
<td>16.99</td>
<td>67.96</td>
</tr>
<tr>
<td>A IO Gear HDMI Video KVM (Model: GHDSW4K4)</td>
<td>1</td>
<td>21.89</td>
<td>21.89</td>
</tr>
<tr>
<td>3-foot HMDI Cables</td>
<td>4</td>
<td>5.09</td>
<td>20.36</td>
</tr>
<tr>
<td>TP-LINK WiFi Router (Model: TL-WR1043ND)</td>
<td>1</td>
<td>49.99</td>
<td>49.99</td>
</tr>
<tr>
<td>InFocus Kangaroo Plus</td>
<td>1</td>
<td>169.99</td>
<td>169.99</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>$960.55</strong></td>
<td></td>
</tr>
</tbody>
</table>

Prices and Model Nomenclature as of August 2017
Appendix J – Application Data & Results

General Application Testing and Experimentation

Below is a sampling of the data collected during the testing and experimentation phases of Chapter 8.

- Baseline Simulation

<table>
<thead>
<tr>
<th>Time</th>
<th>Traffic Generation</th>
<th>Assessment Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 sec</td>
<td>134.8 GBytes</td>
<td>136 GBytes</td>
</tr>
</tbody>
</table>

- Data Center Simulation: In this sample you can see a comparison of the traffic generation and the assessment tool at 40.0 seconds into the run. Total throughput at this point according to the traffic generator is 134.8 GBytes and according the tool is 136 GBytes.
- OTG Physical Testbed: In this sample you can see a comparison of the traffic generation and the assessment tool at 70.0 seconds into the run. Total throughput at this point according to the traffic generator is 233.5GBytes and according the tool is 227.1GBytes.
Man in the Middle Testing and Experimentation

Below is a sampling of the data collected during the testing and experimentation phases of Chapter 8.

- **Data Center Simulation:** In the samples below, I broke out each of the scoring groups (base, environmental, and overall confidence). It is easier to see the linear relationship and the cross over points of the data.

```
<table>
<thead>
<tr>
<th>ID</th>
<th>Interval</th>
<th>Transfer Rate</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0-10.0 sec</td>
<td>33.0 GBytes</td>
<td>29.3 Gbits/sec</td>
</tr>
<tr>
<td>2</td>
<td>10.0-20.0 sec</td>
<td>33.2 GBytes</td>
<td>29.5 Gbits/sec</td>
</tr>
<tr>
<td>3</td>
<td>20.0-30.0 sec</td>
<td>34.2 GBytes</td>
<td>29.4 Gbits/sec</td>
</tr>
<tr>
<td>4</td>
<td>30.0-40.0 sec</td>
<td>33.2 GBytes</td>
<td>28.5 Gbits/sec</td>
</tr>
<tr>
<td>5</td>
<td>40.0-50.0 sec</td>
<td>31.5 GBytes</td>
<td>27.0 Gbits/sec</td>
</tr>
<tr>
<td>6</td>
<td>50.0-60.0 sec</td>
<td>34.2 GBytes</td>
<td>29.4 Gbits/sec</td>
</tr>
<tr>
<td>7</td>
<td>60.0-70.0 sec</td>
<td>34.0 GBytes</td>
<td>29.2 Gbits/sec</td>
</tr>
<tr>
<td>8</td>
<td>70.0-80.0 sec</td>
<td>35.2 GBytes</td>
<td>30.3 Gbits/sec</td>
</tr>
<tr>
<td>9</td>
<td>80.0-90.0 sec</td>
<td>35.0 GBytes</td>
<td>30.0 Gbits/sec</td>
</tr>
<tr>
<td>10</td>
<td>90.0-100.0 sec</td>
<td>31.3 GBytes</td>
<td>26.9 Gbits/sec</td>
</tr>
<tr>
<td>11</td>
<td>0.0-100.0 sec</td>
<td>335 GBytes</td>
<td>98 Gbits/sec</td>
</tr>
</tbody>
</table>
```
Normal Traffic Analysis

Below is a sampling of the data collected during the testing and experimentation phases of Chapter 8 used to generate a ROC curve analysis.

<table>
<thead>
<tr>
<th>Sampling (10 per)</th>
<th>Observed Positive</th>
<th>Negative</th>
<th>Cumulative Positive</th>
<th>Negative</th>
<th>FPR</th>
<th>TPR</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0.0332</td>
<td>0.0742</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>16</td>
<td>0.3245</td>
<td>0.3742</td>
<td>0.0547</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>0.24</td>
<td>0.2952</td>
<td>0.0543</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>24</td>
<td>0.4288</td>
<td>0.4752</td>
<td>0.0576</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>13</td>
<td>0.2647</td>
<td>0.3192</td>
<td>0.0631</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>0.1321</td>
<td>0.1852</td>
<td>0.0658</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>9</td>
<td>27</td>
<td>0.6064</td>
<td>0.7692</td>
<td>0.0926</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
<td>27</td>
<td>0.6064</td>
<td>0.7692</td>
<td>0.0926</td>
<td></td>
</tr>
</tbody>
</table>

214
### Appendix K – Acronym List

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AUP</td>
<td>Acceptable Use Policy</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CSU</td>
<td>Colorado State University</td>
</tr>
<tr>
<td>CVSS</td>
<td>Common Vulnerability Scoring System</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>DMZ</td>
<td>Demilitarized Zone</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DSC</td>
<td>IEEE Conference on Dependable and Secure Computing</td>
</tr>
<tr>
<td>DSN</td>
<td>International Conference on Dependable Systems and Networks</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP(S)</td>
<td>Hypertext Transfer Protocol (Secure)</td>
</tr>
<tr>
<td>IANA</td>
<td>Internet Assigned Numbers Authority</td>
</tr>
<tr>
<td>ICCCN</td>
<td>International Conference on Computer Communications and Networks</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>KVM</td>
<td>Kernel-based Virtual Machine</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MitM</td>
<td>Man in the Middle</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>ODL</td>
<td>OpenDaylight</td>
</tr>
<tr>
<td>ONF</td>
<td>Open Networking Foundation</td>
</tr>
<tr>
<td>OOB</td>
<td>Out-of-Band</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OTG</td>
<td>On-The-Go</td>
</tr>
<tr>
<td>PVM</td>
<td>Path Verification Mechanism</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>ROC</td>
<td>Receiver Operating Characteristics</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Term</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-Defined Network</td>
</tr>
<tr>
<td>SMV</td>
<td>Spoofability, Measurability, Variability</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>ToS</td>
<td>Type of Service</td>
</tr>
<tr>
<td>TSDR</td>
<td>Time Series Data Repository</td>
</tr>
<tr>
<td>UCCS</td>
<td>University of Colorado, Colorado Springs</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VLAN</td>
<td>Virtual Local Area Network</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
</tbody>
</table>