3-19-2015

Using Software Defined Networking To Solve Missed Firewall Architecture In Legacy Networks

Jared Dean Vogel
Illinois State University, jdvogel@ilstu.edu

USING SOFTWARE DEFINED NETWORKING TO SOLVE MISSED FIREWALL ARCHITECTURE IN LEGACY NETWORKS

Jared D. Vogel

66 Pages May 2015

This study is concerned with migrating traditional networks and their inherent firewall architecture to Software Defined Networking (SDN) architecture to provide an initial attempt at preventing application downtime due to hidden firewall domain rules. In legacy organization environments the networking engineers, firewall teams, and application analysts are often silo groups, but Software Defined Networking (SDN) can blur the lines between these group silos.

This thesis first outlines the interworking of SDN, traditional firewall architecture and how it interacts with SDN, an experiment of implementation, and the resulting conclusions.

Testing with SDN shows we are approaching new environments where the edges of network are no longer dominated by firmware on switches and routers. The technologies behind SDN allow for the programmability of the entire network, which creates a logical flow of both network traffic and firewall policies that allow us to bypass traditional errors that may arise from physically segmented networks.

The physical and logical level network programming inherent in SDN allows organizations to merge and adapt skill sets of networking engineer and application developers to reduce the risk and reliance on firewall expertise.

Utilizing OpenFlow protocols and flow table concepts presented in SDN we can propagate firewall rules centrally and logically, which provides end-to-end traffic with firewall rules in our network. Using these concepts reduces the traditional firewall complexity for organizations. In this study we present a paper prototype that demonstrates that we may add in firewall rules to a centralized instance allowing our SDN controllers to provide firewall protection throughout the entire network instead of isolated risk domains or tiers. In the prototype application developers are prevented from calling incorrect ports and possibly missing hidden local firewalls not previously known.

The approach described in this paper is based on a case study of several large American firms.

USING SOFTWARE DEFINED NETWORKING TO SOLVE MISSED FIREWALL ARCHITECTURE IN LEGACY NETWORKS

JARED D. VOGEL

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

School of Information Technology

ILLINOIS STATE UNIVERSITY

2015

USING SOFTWARE DEFINED NETWORKING TO SOLVE MISSED FIREWALL ARCHITECTURE IN LEGACY NETWORKS

JARED D. VOGEL

COMMITTEE MEMBERS:

Yongning Tang, Co-Chair

Douglas Twitchell, Co-Chair

ACKNOWLEDGMENTS

I would like to thank and dedicate this to the faculty at Illinois State University. I would specifically like to thank Dr. Yongning Tang and Dr. Douglas Twitchell the participating committee members. Acknowledgements to my family Sherri, Justin, and John for always staying with me and offering there love and time. I would like to also like to thank the academic community as a whole. I have been lucky enough in series of random events to be rewarded with nothing but the most patient, brilliant, and understanding minds in the field of Information Technology. Last but certainly not least, Diana, Harley and Ashley. Terrible days vanish every time I walk through my home door and I am greeted with eight paws and a warm embrace.

"Did I not say before, that I was writing this *Requiem* for myself?" – Mozart, "After saying this, he looked yet again with tears in his eyes through the whole work." – Maynard Solomon in Mozart: A Life p. 494

"Education is the key to success in life, and teachers make a lasting impact in the lives of their students." –Solomon Ortiz

"I have come to believe that a great teacher is a great artist and that there are as few as there are any other great artists. Teaching might even be the greatest of the arts since the medium is the human mind and spirit." – John Steinbeck

J.D.V.

CONTENTS

TABLES

FIGURES

ABBREVIATIONS & ACRONYMS

SDN	Software Defined Networking
ONF	Open Networking Foundation
OVSDB	Open Virtual Switch Database
JVM	Java Virtual Machine
QoS	Quality of Service
SDNi	Software Defined Networking interface
IETF	Internet Engineering Task Force
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OSI	Open Systems Interconnection
NAT	Network Address Translation
WAN	Wide Area Network
LAN	Local Area Network
IIS	Internet Information Services
OF	OpenFlow

CHAPTER I

INTRODUCTION

Can Software Defined Networking (SDN) help organizations manage their security architecture? How can SDN help safely implement new elastic firewall environments within an organization? These are questions facing enterprises today. Enterprises today are increasing in data aggregation, data utilization and multilayered security architecture. According to the study Exploiting In-network Aggregation for Big Data Applications, [t]his generates high network traffic, which is hard to support using traditional, oversubscribed, network infrastructures. Coinciding with oversubscribed network infrastructure are common firewall policies and access control rules.

In a traditional network, if an organization was to alter an application from using Hyptertext Transfer Protocol (HTTP), which is on port 80, to Hypertext Transfer Protocol Secure (HTTPS) on port 443 and firewall rules in place for allowing HTTP, the result would be the failure of the application. At Oracle, inc. Bartley, et al. found that, "[a]s the IT portfolio grows, IT legacy investments and architectures begin to stifle business innovation and increase operational costs [40]." Bigger portfolios mean an increasing the amount of responsibilities and expertise required for application developers. Allowing problems like a simple port switch and firewall rule violation to occur brings applications down.

"80% of unplanned outages are due to ill-planned changes made by administrators ("operations staff") or developers [41]." This statistic taken out of the IT Process Institute

Visible Ops handbook shows the need for solutions in common enterprise environments. Colville and Spafford [42] predicted that, "[t]hrough 2015, 80% of outages impacting mission-critical services will be caused by people and process issues, and more than 50% of those outages will be caused by change/configuration/release integration and hand-off issues." The misconfiguration problem explored in this paper includes the people and process issues in firewall policy management. Depending on the organizations size and business model, firewall policy management errors can be costly to overall operations by taking offline time sensitive revenue generating applications.

This paper will focus on abstracted enterprise examples of how SDN can potentially reduce the complexity of security architectures. The primary focus will be on firewall consolidation to enable the deployment of rapid and secure environments, which should help organizations, avoid downtime resulting from an application configuration that violates firewall rules. Secondarily it will delve into the potential benefits organizations would see after implementation. Using SDN we will outline how the number of firewall layers in an organization can be potentially managed more effectively, paving the way for a new solution of using SDN. Effective management of these policies contributes to eliminating the process portion of our problem.

This study will also outline how SDN is currently being used and customized to meet organizational needs in alignment with best practices of the Open Network Foundation (ONF), the current leaders in the software defined networking field. Google, a large enterprise currently uses OpenFlow technology. Recognizing the previous problems listed, the company has separated out its hardware from software deploying SDN switches. In the words of Amin Vahdat, "It provides logically centralized control

that will be more deterministic, more efficient and more fault-tolerant [43].” At the Open Network Summit in 2014, Vahdat explained Andromeda, their SDN based substrate for network virtualization efforts. “Rather than being forced to create compromised solutions based on available insertion points, we can design end-to-end secure and performant solutions by coordinating across the stack [44].” By using SDN to create logical stacks across the globe, they can provide elastic network connectivity and scalability. This utilizes the networks processing power to provide the high availability and elasticity other traditional organizations don’t have.

The security examples presented in this paper include anonymous data collected from several enterprises currently not using SDN. They are compliant with federal security standards such as National Institute of Standards and Technology SP 800 and Sarbanes-Oxley regulation. We will test a solution that uses this anonymous data to build a multi layered firewall environment. With the environments built our final goals are the reduction of complex policies, having a simpler firewall architecture, and the creation of domain risk classifications, which will enable robust network architecture. In Chapter 5, we propose an SDN-based paper prototype that fulfills these goals. We show that this prototype allows us to move an application from HTTP to HTTPS without downtime due to firewall rule violation.

“Most programmers mainly focus on functionality and make security a secondary priority [44].” The lack of focus on network security and network flexibility creates common risk problems such as misconfigurations and omitted processes. Compounding these problems is the need to maintain end-to-end flows within an organization’s multi firewalled environment. Redefining multiple firewall environments or even changing

firewall policies to meet new application development is often overlooked. When changing the priority of applications organizations need to review the network and security architecture that operates within the organization. Even if they have begun implementation of SDN, Organizations still have the challenge of maintaining separate legacy corporate, and local firewalls. The need for a manageable combined solution is present, but the tools, standards and adoption of SDN are still in its infancy creating a challenging environment for realizing its full benefit.

We also show that future work will be needed as the literature on aggregating firewall and access control language for policies fail to mention the actual limitation of the hardware.

Thesis Organization

The remaining chapters of this thesis contain the following:

- Chapter 2 – Review of related literature covering SDN, software defined networking with firewalls and how current firewall environments operate.
- Chapter 3 – Software defined networking and how it is being currently proposed, implemented, developed, researched and utilized.
- Chapter 4 – Abstract organizational security architecture, firewall problem being proposed with anonymous data collected.
- Chapter 5 – Using SDN to remove multiple risk environments and firewalls decreasing the complexity of security architecture.
- Chapter 6 – Did it work? Proof-of-concept with hypothetical working model. Conclusion on results and proposals.

- Appendix – Scripts, possible code to utilize in implementing the proof-of-concept, and all pseudo code used.

CHAPTER II

REVIEW OF RELATED LITERATURE

The literature analyzed for this paper includes established SDN technologies, established firewall technologies, and emerging concepts combining the two fields together. An abundance of information is present for both topics even though SDN is a technology still in its infancy being rapidly developed. Numerous methodologies are present when applying new concepts to SDN and its open source projects due the modularity and availability of the software. Large amounts of information and literature reside within the ONF, Stanford and Berkley SDN repositories. “Implementation of SDN is being touted by large enterprises such as Google, Amazon, and hardware providers like Cisco [2].”

A large player in the SDN field is The Open Networking Foundation (ONF). “Open Networking Foundation (ONF) is a user-driven organization dedicated to the promotion and adoption of Software-Defined Networking (SDN) through open standards development [1].” ONF was formed due to two studies on SDN gaining traction from the Universities of Stanford and Berkley. “McKeown and colleagues developed a standard called OpenFlow that essentially opens up the Internet to researchers, allowing them to define data flows using software--a sort of "software-defined networking [2].” Carrying this SDN technology forward McKeown and colleagues formed the ONF along with other participating Universities and organizations.

According to the member listing of the ONF [3] over one hundred and seventy organizations and corporations are listed as members including major players such as Citrix, IBM, Intel, Broadcom, Dell, Cisco, and T-Mobile. With a large participating membership and committee the ONF has gained support among hardware vendors adopting the standards and protocols. Because many members are leaders in their respective technology fields, SDN could not be disregarded as a fad or passing phase technology. It has gained a strong foothold in the turning years since its inception around 2009.

Software Defined Networking Infancy

The ONF provides technical documentation introducing the concepts of SDN and the OpenFlow protocol. Methodologies that motivate the ONF seem to have shifted from academic purposes [3] to more commercial. "Today, our Technical Communities continue to analyze SDN requirements, evolve the OpenFlow Standard to address the needs of commercial deployments, and research new standards to expand SDN benefits [1]."

SDN being utilized with numerous approaches and problems they are solving, it is difficult to find research that brings fruition of working models within their conclusions. The ONF does provide a complete setup of architecture explanations and designs to the details of SDN ability and function. Unfortunately, a lot of the literature provided on the front end of their foundation reads like advertising [7]. Open source organizations are easily found to compliment the ONF literature and provide current working projects to engage with [8]. The white papers on the actual hardware specification for their switches [13] down to the actual protocol coding and scripts [29] provided by the ONF are not

only comprehensive, but also allow for abstracting new theories. Entire working models along with associated code repositories are provided by the ONF.

Protocols and Standards

The intense competition among hardware vendors has been beneficial for SDN and the OpenFlow protocol. For example, vendors have begun to advertise how well their hardware is utilizing the SDN and OpenFlow standard as a selling point for cloud services. When researching vendor claims, however, it is considered best practice to check with independent third party laboratories. One such laboratory is the Tolly Group, which claims to be "…positioned to certify vendor solutions and thereby provide evidence that their products meet or exceed marketing claims [44]." The Tolly Group has been conducting tests recently concerning switches that are OpenFlow enabled and operating on an SDN. In an example test they found that, "the IBM BNT RackSwitch G8264 [an SDN-based switch] demonstrated up to 100 times the packet buffering capacity and up to 70 percent less energy consumption than competitive switches, while maintaining full line rate and providing 160 Gbits/second more capacity than any other switch tested [12]." This private competition combined with international organizations such as Internet Engineering Task Force and Institute of Electrical and Electronics Engineers has provided strong backing and standardization of the OpenFlow protocol. Journals and independent consultants refer to the newest version of OpenFlow as the de facto standard [11], however other protocols have been in development. OpenFlow being the standard is not without flaws as evidenced by the ONF adopting other protocols in development into the OpenFlow standard. "The Open Networking Foundation (ONF) recently embraced NETCONF and made it mandatory for the configuration of

OpenFlow-enabled devices [45].” Adoption of these protocols in development broadens the OpenFlow standard providing a wider environment in which to develop. This expanding scope of OpenFlow provides flexibility and depth upon to develop better standards. NETCONF a protocol recently integrated into OpenFlow is very useful to both OpenFlow and to solving this studies problem. “It provides an administrator or network engineer with a secure way to configure a firewall, switch, router, or other network device [45].”

The open source of the controllers APIs and interfaces are concerning. “[O]pen APIs for security functions to SDN have not yet appeared and have not begun to standardize, so API incompatibilities may also cause security holes to appear [14].” Without a clear guidance or grasp on the controller APIs themselves, numerous solutions are still being proposed such as westbound models to have controllers communicate to each other [10] known as SDNi or vertical topologies.

Traditional Firewalls

Literature on firewalls and their detailed workings is in over abundance. Traditional firewalls have been around since the dawn of the Internet and their use is considered by some to be obsolete or failing [15]. Massive amount of firewall architecture and policies that have been created can readily find new literature and dated literature. Dated literature [18] will provide the framework of our firewall architecture due to their current and established dominance in the field [16]. Although firewall literature is often dated, new sources are being generated daily for new concepts and designs. Literature current this year continues to categorize firewalls into the same three

types proposed 27 years ago [17]: packet filtering firewalls, circuit gateways, and application gateways [19].

Research is also being conducted on solving similar problems presented by virtual local area networks (VLANs) and other moving network topologies. “To reduce complexities in identifying various networks using [VLANs][20].”

The limited research into modelling firewall rules and how they will fit within SDN is fragmented. Most research falls into two categories, identifying the potential mistakes or complexity of the proposals [21]. Dissertations such as [22] and even Sigcomm proceedings such as [HOT SDN] often conclude with more work required or a simple model of the structure.

Studies on SDN firewalls are extremely rare. Applicable results on studying traditional complex firewalls and systems developed to quantify them such as [23] are available. Combining traditional firewall studies with SDN studies that begin to step forward in eliminating dedicated hardware firewalls that produce SDN flows [26]. This study will then utilize firewall anomaly discovery algorithm research [27] to produce a viable solution to moving our ports and not violating our firewall rules. Building upon this research we have all the tools to detect our change but then require programming constraints to add our decisions to the firewall policy and make complex queries to it [24].

Conclusion

The literature reviewed in this paper seems to be missing a reoccurring theme. This may be in due to the infant nature of SDN. However, this study’s use of flow tables [31] and concepts behind running our SDN [34] is supported by reoccurring research and

is well documented. The study does, however, require the most updated form of OpenFlow. “Switches using OpenFlow 1.0 forwarding model cannot perform more than one operation during the packet forwarding process [31].” Potentially proving paper prototype viable, our proposed structure tests a real-world environment with firewall rule omissions.

Research and development environments are often scaled back and redundant. Production networks are expensive resulting in efficient network utilization being a repeated discussion. Expanding networks and additional complexities from the increased sized, creates problems for application developers. It has created need for sophisticated algorithmic control across this studies network [31]. New technologies are being proposed just to run parallel with SDN [33] or even using big data applications to run SDN itself [31] to in turn manage and run the network efficiently [33]. In the white paper SDN System Performance, we can see the pitfalls [37] of the hardware itself or optimization of the APIs [36]. Protocol utilization standards such as which fields are optional or required [38], and the overall complexity of the central logic controllers may hinder applicable solutions [35]. Last, this paper uses all combined research and constructs firewall models that simply call and use OpenFlow field standards [25] and proposed OpenFlow protocol configurations [26] In doing so this study creates the paper prototype presented for this thesis. Referring back to the ONF, on combining our hypothesis and pseudo code can easily be done by utilizing the SET-FIELD within OpenFlows field options [29]. Using proven implementations of other research and development projects we were able to produce a viable paper prototype. “We modify the controller to export an install route API to install a shadow-MAC-based label routed path

to a destination [30].” Installing this route API is the final stepping stone from identifying our rule violations, to implementing our detections in the violation, adding our new rule in and finally expanding the flow or path our traffic will take

CHAPTER III

SDN CURRENT IMPLEMENTATIONS

In this chapter we focus on first understanding SDN architecture, its uses, proposed uses, current implementations and many of the various ways it is utilized. We assume that the reader has a background of basic networking experience.

Why Was SDN Proposed?

Computers communicating over a mesh of networks throughout the entire globe are a reality that was thought fantasy when the first network design was conceptualized. Conventional networks are hierarchical in structure, built with tiers of Ethernet switches arranged in a tree formation [4]. This design was best used when client-server computing was dominant initially during the first public computer network adoption [4]. Static architecture is ill suited to the dynamic computing and storage needs of today's enterprise servers, data centers and mirrored backups, campuses, and carrier environments [4].

Key Factors for Proposing SDN

Users are the ultimate goal for any information technology project. Several trends in networking provide the impetus for the development of SDN technologies including (1) dynamic network traffic patterns; (2) mobilization; (3) security and controls; (4)large bandwidth applications;... These also provide a foundation for understanding the need the proposed SDN Firewall solution presented in this paper.

Dynamic Network Traffic Patterns

The model of communications from client to the server and back are long gone with applications communicating with applications and cloud services communicating with other virtual machine environments. The traffic is no longer hierarchical, which slows down current network infrastructures based on switches and routers. In the Andromeda project Google's core SDN were benchmarked on throughput and speed using netperf TCP_STREAM [43]. The results showed an approximate 300% increase in performance that shows the direct benefits from an SDN implementation.

Mobilization

The same study of Google's SDN network also showed the impact of mobilization on traffic patterns. By benchmarking the total number of TCP Streams they showed that traditional networks operate at 2 Gigabits per second throughput versus 3.5 Gigabits per second [43]. Because of the increased throughput, SDN multiplied in a literal sense with the number of devices communicating on the same network. Streaming efficiency also increased with SDN. Two hundred streams were monitored on a traditional network, which performed at 1.5 Gigabits per second versus an increase in the Andromeda SDN to 5.1 Gigabits per second.

Security and Controls

Accessing a Fortune 50 companies documentation or office work files from a smartphone was not conceived when the first network topologies were designed. This equivalent scenario would be Thomas Edison predicting electrical networks to handle wind and solar power plants and the circuitry being able to handle those loads. Yet both of these scenarios are actively being played out today with information being accessed on

the go and new wind turbines being erected in several countries. The need for improved security is one motivation for organizations implementing SDN technologies due its logical and central control.

Using SDN to implement security reform leaves data scientists from Clemson University and Arizona State University asking how to solve the security challenges that will pop up with new software defined networking proposals. "One of the fundamental challenges is to build robust firewalls for protecting OpenFlow-based networks where network states and traffic are frequently changed [5]." If network architects can build robust firewalls with SDN OpenFlow organizations can eliminate many conventional downtime errors.

Voice, Television, and Big Data

Skype, Netflix, and Hadoop are applications, services and companies well known in 2015. Bandwidth and latency concerns continue to climb as services like Netflix begin to aggregate large amounts of traffic. Big Data applications such as Hadoop are also bandwidth intensive and have spawned numerous projects involving the need for SDN. Network engineers at Sigcomm propose not only using SDN, but using it with unique SDN topologies:

> These three trends taken together – software-defined networking, dynamically reconfigurable optical circuits, and structured big data applications – motivate us to explore the design of an SDN controller using a "cross-layer" approach that configures the network based on big data application dynamics at run-time [6].

Policies, Scalability, and Usability

To implement a network-wide policy, IT may have to configure thousands of devices in a large organization, from client based personal computers, to routers, switches and servers. For example, every time a new virtual machine is brought up, it can take hours, in some cases days, for IT to reconfigure ACLs across the entire network [4].” The problem of building a robust firewall also falls into implementing network wide policies. The process of a new virtual machine being brought up is a core component of this papers problem. Dynamic policies create issues when an application is moved and begins to drop legitimate traffic. When moving a virtual server the firewall policy should be elastic enough to either reject the change or allow it to ensure legitimate traffic is not dropped. This is the critical need for SDN in current legacy organizations that explored in this paper.

SDN Architecture

To begin understanding SDN architecture the basic premise of networking first must be understood. The primary network stack includes switches and routers that network administrators utilize. In SDN the *control plane* forwards traffic to the selected destination. The *data plane* (sometimes called the *forwarding plane*), are contained within the hardware of switches and visible in firmware. The data plane, however, is limited in manipulability by administrators. Since the data plane is decoupled from the control plane in SDN, we need a communication medium to coordinate the two, which is OpenFlow [3]. These planes build tables on the switches and routers; then sift through designating traffic from Point A to B or C. Traditional networking has all planes

implemented in the firmware of routers and switches which may be unique but conform to IEEE standards.

Understanding the data plane and control planes we then can progress to SDN architecture, which utilizes the same concepts but through abstraction. “This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services [7].”

In Figure 3, we can see how the SDN architecture uses the “OpenFlow” protocol. This protocol is essential in using SDN and is a standard. The protocol is a standard endorsed by ONF for SDN. Babara Liskov is often quoted when reviewing SDN architecture “Modularity based on abstraction is the way things get done.” Figure 3 shows the switch hardware forwarding traffic being controlled by the control layer. The control layer in turn is being accessed through an application layer, which communicates what traffic it has available and needs moved.

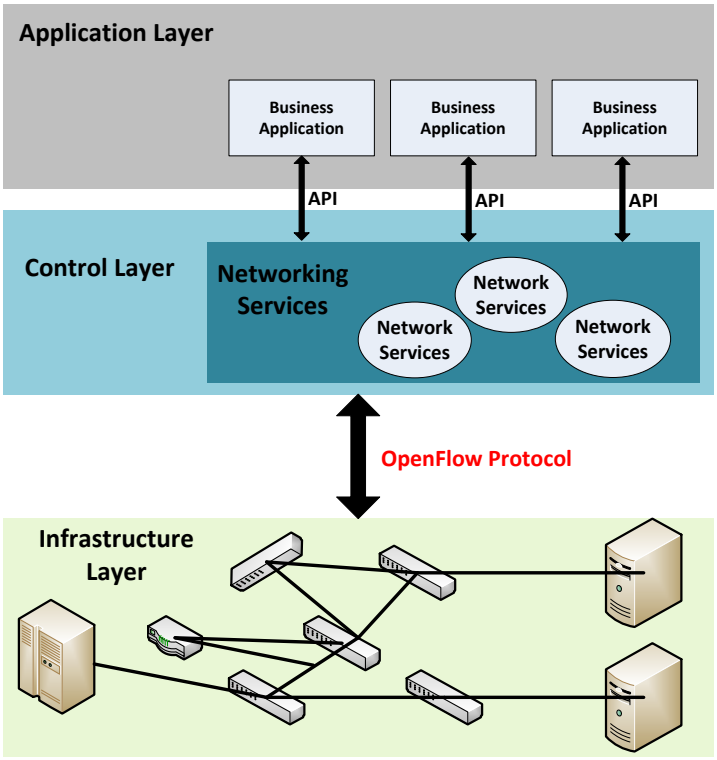

Figure 1. Software Defined Networking Architecture Abstract

By decoupling the network control and forwarding functions, we can then begin to program our network directly. Direct control over the network gives way to two major key points. First, our network is now dynamically moving and adjustable giving the network agility and verbose with network engineers being able to configure, manage and even secure the network manually or even automatically with programs that they can write themselves due to non-proprietary firmware or software. Secondly, we may even move to integrating application developers into the network realm or vice versa. The doors to programming the network are blown wide open by allowing developers to have access to program the network to the specific needs of their application.

Decimating through this information we have gathered that the OpenFlow protocol allows for the standard communications within the SDN architecture, the

decoupling of the planes allowing programmability, direct control over our network, its agility and potential for automation. Lastly, the most critical part of SDN architecture is the centrally managed portion. Similar in networking when speaking physically and logically with Ethernet traffic, we do the same with SDN. "Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch [7]." Managing this "logical" centralized instance leads us to the (SDN) controllers.

SDN Controllers

SDN shifts the architecture as previously mentioned into a logical centralized instance. Coming with this instance leads to the need of a controller of sorts. The controller or main CPU processing unit of the entire network is a control point. "It is the strategic control point in the SDN network, relaying information to the switches/routers 'below' (via southbound APIs) and the applications and business logic 'above' (via northbound APIs) [8]." The southbound APIs are application logic that dictates it's needs top down from the application layer to the control layer. In vis-à-vis the turn of the traffic and decisions are being communicated back to the application layer. This controller would use common interface OpenFlow and presents another common protocol open virtual switch database (OVSDB).

The controller is like a modular plugin platform, which then runs our network performing various tasks such as routing, balances, inventorying, and statistics. Even though the controller is performing basic tasks, it can be extensible and made to perform advanced tasks such as running custom code or algorithms citing new rules throughout the network and conducting on the fly analytics. This is the feature we are interested in

with solving our problem with an SDN perspective because citing new rules or implementing our own custom algorithm is what will make our solution a viable one. "The first SDN Controller was NOX, which was initially developed by Nicira Neworks, alongside OpenFlow. In 2008, Nicira Networks (acquired by VMWare) donated NOX to the SDN community (it was open sourced), where it has become the basis for many subsequent SDN Controller solutions [8]." Basic operations of a NOX SDN controller are displayed in Appendix A.

Concluding on the SDN controller, a major proponent of SDN open source studies is not only the ONF but also the OpenDaylight project, which is part of the Linux Foundation. This SDN controller runs in a Java Virtual Machine (JVM) supporting both OpenFlow and the previously mentioned southbound API's. Different types of SDN controllers exist in the ecosystem. NOX is a C++ multi-threaded controller that is written on top of a POX library, single threaded python controller, Beacon is another Java based controller [8], and many more variations. Each type of controller has its strengths and weaknesses; however, the first open source SDN controller was NOX. Along with various types of controllers, we can identify weak points of the SDN controllers and all their different versions. The larger the network the more taxing it will be on a centralized instance possibly even bringing the entire network down if the controller is corrupted, overloaded or simply under resourced or poorly optimized. A lightweight python based controller couldn't handle the load to a certain critical point. The need for multiple controllers is then present.

SDNi

In 2011, there was a technology news publication from the IEEE society approaching a solution to managing multiple SDN controllers. Providing an interface protocol between the controllers allows engineers to create an "interfacing SDN Domain Controllers [9]" referred to as SDNi, which was progressively being developed by the Internet Engineering Task Force (IETF). This allows for the scalable environment that we can operate in larger organizations by deploying multiple SDN controllers. In Figure 2 SDNi Overview we can see how the switches and OpenFlow protocol fits within this SDNi environment. This environment shown below is a horizontal SDNi structure. Each switch is paired and communicating with each other on decisions being made. A single controller could be added on top of this diagram turning it into a single controller dedicated to controller all sub controllers, which is a vertical SDNi approach.

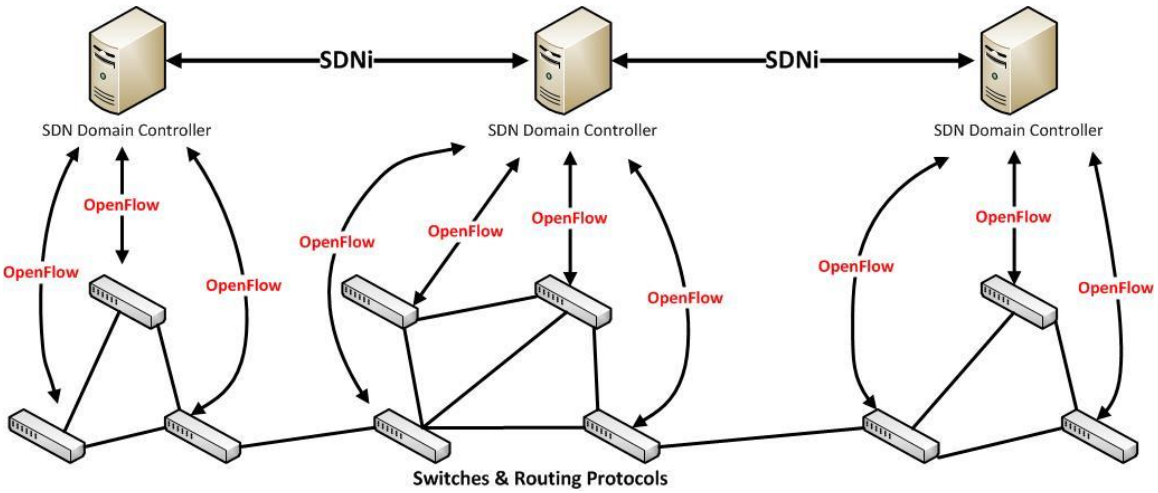

Figure 2. SDNi Overview

SDNi allows the SDN controllers to communicate various information details such as "network topology, network events, user defined request information, Quality of Service (QoS) requirements from user application request, integration infrastructure

status, and more [10].” SDN controller communications as explained previously can have horizontal and vertical designs. These designs arrived due to having one controller may not be suitable to cover the entire network. SDNi, in turn, solves this. Further understanding the in depth nature of SDN we must look at three more elements that compromise it overall. The OpenFlow enabled switches displayed above in Figure 2, OpenFlow Protocol itself, and Flow Tables. “An SDN controller communicates with OpenFlow compatible switches using the OpenFlow protocol running over the Secure Sockets Layer (SSL) [11].” Analyzing these switches and routers we can see how SDN and the protocols work by decoupling the planes within the hardware.

OpenFlow Protocol and Switches

As previously mentioned in SDN Controllers section we discussed that OpenFlow is not the only protocol on the rise in development and use, but it is one of the most widely utilized and researched currently. OpenFlow provides the standards for the interface on controlling the data packets. This is the main goal of the ONF foundation is to set global standards and interoperability in place. “The OpenFlow standard also provides a basic set of global management abstractions, which can be used to control features such as topology changes and packet filtering [12].” Breaking down an OpenFlow enabled switch we can segregate it into three distinct parts, the flow table, secure channel, and the OpenFlow Protocol. In Figure 3 OpenFlow Switch shows how these three distinct parts reside within our OpenFlow enabled switch.

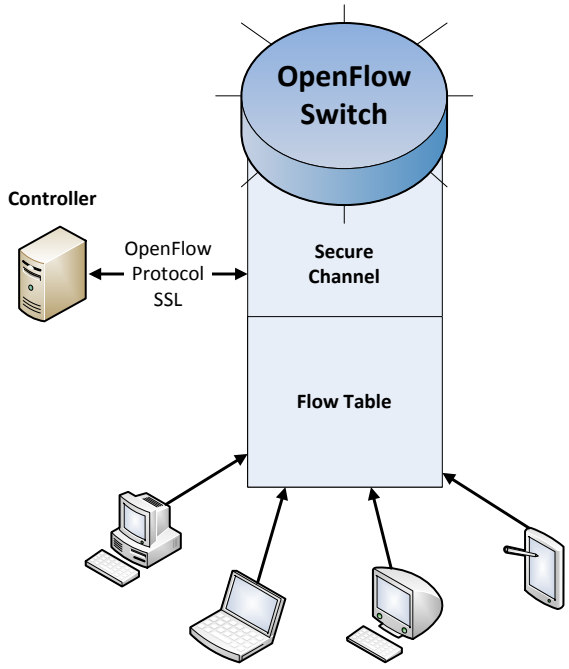

Figure 3. OpenFlow Switch [25]

First we can see the "Flow Table - Tells the switch how to process each data flow by associating an action with each flow table entry [12]." The Flow Table will be populated with entries via allowance the OpenFlow protocol defined by a server external to the switch. By populating this flow table we can have various entries and dictations through policies that are enabled by secure channel. This flow table and secure channel is where the basis of our experiment will begin.

Utilizing certain fields and actions within the flow tables we can dictate our own programmability of the network. "Secure Channel - Connects the switch to a remote control processor (called the Controller) so commands and packets can be sent between the controller and the switch [12]." This is the programmability and modularity that is often discussed when SDN is proposed. Not only can we develop specific firewall rules or policies there are numerous activities that can be done with this programmability such as time sensitive express lanes or even a new type of Quality of Service (QoS). The last

part of the switch is the OpenFlow protocol, which we have covered is a standardized interface for the controller to the switch. Below in Table 1 we can see the reserved ports provided by the ONF and suggested optional ports as well.

Table 1. OpenFlow Ports

Port Name	Description	Required / Optional
ALL	This is for all ports the switch may potentially use for forwarding a packet. Is an egress port only.	Required
CONTROLLER	Representative of the channel that controls with the SDN controller. Can be an ingress and egress port.	Required
TABLE	Beginning of OpenFlow pipeline. Valid as an output action in the action list of a packet-out message.	Required
IN_PORT	Packet ingress port, can only be used as packet output port.	Required
ANY	Specialized value in certain OpenFlow syntax when no port is called. A wild card value which we will be using for our firewall complexities.	Required
LOCAL	Switches local networking stack and the management stack associated with it. Both an incoming and outgoing port it enables remote entities to interact with the switch via the OpenFlow network.	Optional

Flow Tables

Each switch maintains an OpenFlow pipline, a virtual pipeline, which maintains the multiple flow tables which therein contain multiple flow entries. This is how the packets interact with the tables. The switch must have at least one table for it to be active. "Each flow table entry has a specific action associated with a particular flow, such as forwarding the flow to a given switch port (at line rate), encapsulating and forwarding the flow to a controller for processing, or dropping a flow's packets (for example, to help prevent denial of service attacks) [12]."

A very basic way of how the Flow Table or groups of Flow Tables methodology and mechanics work is with how they handle a short process of finding the highest priority matching flow entry. Then it will apply instructions based off the matches. Those instructions could be to modify the packet and update the match fields, update action set. The update action set is what we will be using in our effort to produce a viable algorithm

to reduce our firewall complexity, which will be referenced later as our set-field when discussing the OpenFlow port.

The Update actions are set clear actions or write action instructions. One can begin to formulate our methodology and process for achieving a certain set of algorithmic functions to avoid moving our application into a restrictive environment eventually crashing it by seeing this port and OpenFlow standard. It may also update metadata within step two. Lastly, we will send match data and the actions set to the next table and this denotes the processing pipeline.

Below in Table 2. Flow Table Components we can see the main component entries provided by the ONF in a Flow Table.

Table 2. Flow Table Components [13]

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Further specifications on the Flow Table will be included in the Appendix A.

SDN Vulnerabilities

Like any new technology, vulnerabilities and risks are often associated with new changes. Simply reviewing our architecture explanations we can quickly identify that a central point such as the SDN controller would be an exceptional point of attack. An attacker is able to compromise the controller than he is able to compromise the network and propagate through it rapidly. “According to Ramnath Venugopalan of Intel Security (formerly McAfee), SDN opens potential security holes, especially in connections between controllers and network elements [14].” The controllers and switches are no exception to traditional attacks either. Depending on which language the controller is written in depends on the vulnerabilities associated with it, compounding on top of the

risks are introductions of new risks by the flow tables and OpenFlow protocol misconfigurations.

Physical risks include overloading the controller as previously mentioned, which will bring down the entire network not just parts. If the demands or traffic for the network is too large the controller may fail, dropping legitimate traffic or slowing the service to an unacceptable level.

Conclusion

In conclusion, we have discussed aspects to why SDN was proposed, the architecture behind SDN and our focus within the technologies. Progressing forward we can take away key points such as one of the proposals for SDN was Security and Policy based decision making which aligns to this studies proposed problem of decision making for changing our application from HTTP to HTTPS. Not only is this proposal a widely regarded one, it is the proposal we will be focusing on.

Trying to implement a robust firewall design is complicated; however once done successfully our hypothesis of it reducing the complexity of our firewall environment will prove beneficial in new ways that were unrealized. The SDN architecture summed up is a central logical location versus how switches use to see when the data packet arrived at the switch, and then the firmware would send it off to the destination with its rules built in. Network engineers had no control over these rules and that is what SDN is enabling. “"On a network running OpenFlow, computer scientists can add to, subtract from, and otherwise meddle with these rules [3].”

The rules that we will be meddling with are located within the SDN controller and switches. The OpenFlow protocol and flow tables are what will allow us to change our

landscape and fulfill the solution to our problem. Moving an application server when doing a patch or adding a port can break the entire application. If we were able to somehow have the programmability of a network at our fingertips we could prevent this. This problem is widely recognized within the SDN community; however, it is slightly different then the thesis problem of updating an entire traditional environment over to an SDN environment to solve development problems. This thesis will achieve this by diving into the SDN architecture, down to the controllers, down to the switch, and into the flow tables themselves. Once this project has proposed the solution it will traverse back up to the controller level and have the SDNi level roll out this projects proposal.

CHAPTER IV

FIREWALL ARCHITECTURE

Discussing firewall architecture this thesis will focus on select items including how traditional firewalls work briefly, current real world implementations that are anonymous due to the sensitive nature of firewall architecture, following in parallel of anonymous data current security practices, policies, and standards and closing with firewall implementation options with SDN.

Firewalls essentially have a narrow job list which includes closing off ports, applying certain routing rules to packets and preventing large attacks on the network. They also prevent large illegitimate traffic from getting out if there is a compromise within the network. "Traditional firewalls can also be expensive to operate, especially if you need to supplement them with additional security technologies [15]." Firewalls are not only expensive to operate, but the overall costs are often not efficiently recorded as well.

Overall costs are referencing bureaucracy in developing alongside firewall teams such as the time it would take to request a firewall change. In smaller environments one or two individuals usually do the changes; however, working with large organizations and application developer may have to wait an unacceptable amount of time until he or she gets the desired firewall change. Not only does this study utilize technology to reduce risk and complexity, it also should eliminate unnecessary red tape bureaucracy and allowing organizations to move with agility and ease.

Progressing forward we will discuss how current firewalls are being used, phased out and how we can use SDN to our advantage solving the problem listed above. Firewalls can be grouped into traditional, distributed, embedded and others. Our hypothesis we will focus on distributed firewalls and how they create certain risk domains within an organization.

Abstract Firewall Architecture

Starting off with firewall architecture we can examine Figure 4. Single Firewall Abstract.

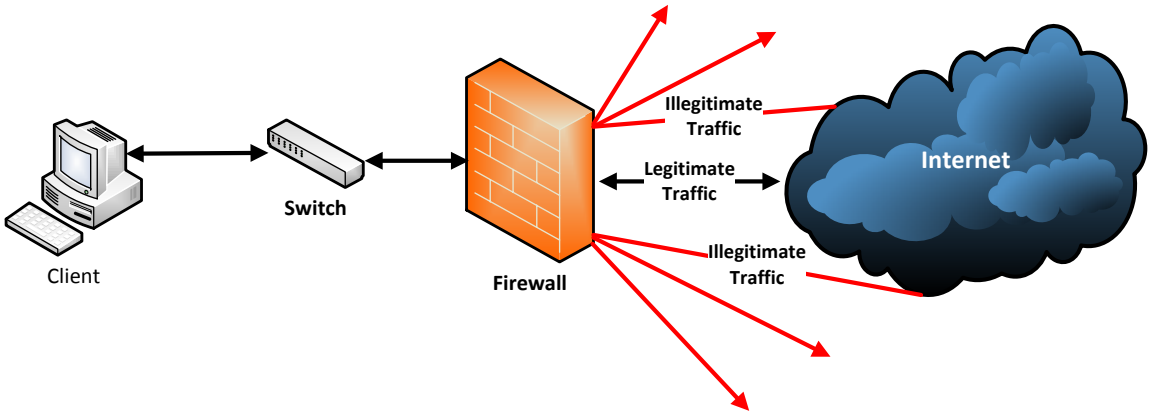

Figure 4. Single Firewall Abstract

Above we can see how the client has legitimate traffic that is traversing usually through a switch or a router then hits our single firewall then communicates to the Internet. This border firewall will be combined into our Flow Table in SDN. Therefore the complex decision making logic will reside within the field settings of our SDN and not a traditional tree based access control language border firewall. Even at the level of users with home networks they still have software firewalls behind the border hardware firewall at the router or switch level.

This divides the firewalls into two types of categories, hardware firewalls and software firewalls. In Figure 4, we are displaying a hardware firewall. “Hardware firewalls are normally situated between the network and the connecting cable/modem. These are external hardware devices usually called Network firewalls [16].” Finding the solution to our problem, we see in SDN how traditional hardware or network firewalls are converted over to a software firewall while still maintaining its status as a hardware firewall.

The programmability of an SDN allows for us to make an elastic robust firewall for communicating traffic across our network. “Software firewalls are basically software components that are internal to the computer system. They work hand-in-hand with the computer’s operating system [16].” Working with the computers operating system software firewalls are designed to protect the client they reside on. Using the single focus and expanding it throughout our SDN by programming our flow tables will provide a unique firewall solution.

Types of Firewalls

Firewalls were one of the first inventions of security when the Internet was brought into existence. The Internet grew and along with it so did the types of firewalls. Cautiously proceeding, material referenced in 1999 is still relevant today, hinting the need for new technologies on protecting our networks.

Packet Filtering

“Filtering firewalls screen packets based on addresses and packet options. They operate at the IP packet level and make security decisions (really, "to forward, or not to forward this packet, that is the question") based on the headers of the packets [17].”

Packet filtering takes place at the third layer of the OSI model of networking. At the IP Internet Protocol (IP) Layer we can afford to have a robust implementation because this is present in nearly every device on the network such as routers, switches, wireless points and much more.

Expanding upon Figure 4 we can see in Figure 5 Filtering, how the firewall examines characteristics of the packet and then matches them to an accept policy rule or reject. If a match is not found typically the firewall will refer to its own Quality of Service policy. In exploring our options for certain firewalls we have three distinct groups. In Figure 5 the packet characteristics are source Internet protocol (IP) address, source port, destination IP address, and destination port. Filling the last spot is the IP Protocol, which could be Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). The figures begin to dive deeper into the SDN framework for the study's proposal. Figure 5 shows the standard filtering logic that will have to be dynamically applied inside the SDN proposal. The same permit and deny logic will not be simple and concise within the SDN switch. Taking into account numerous types of filtering the proposal begins to become complex.

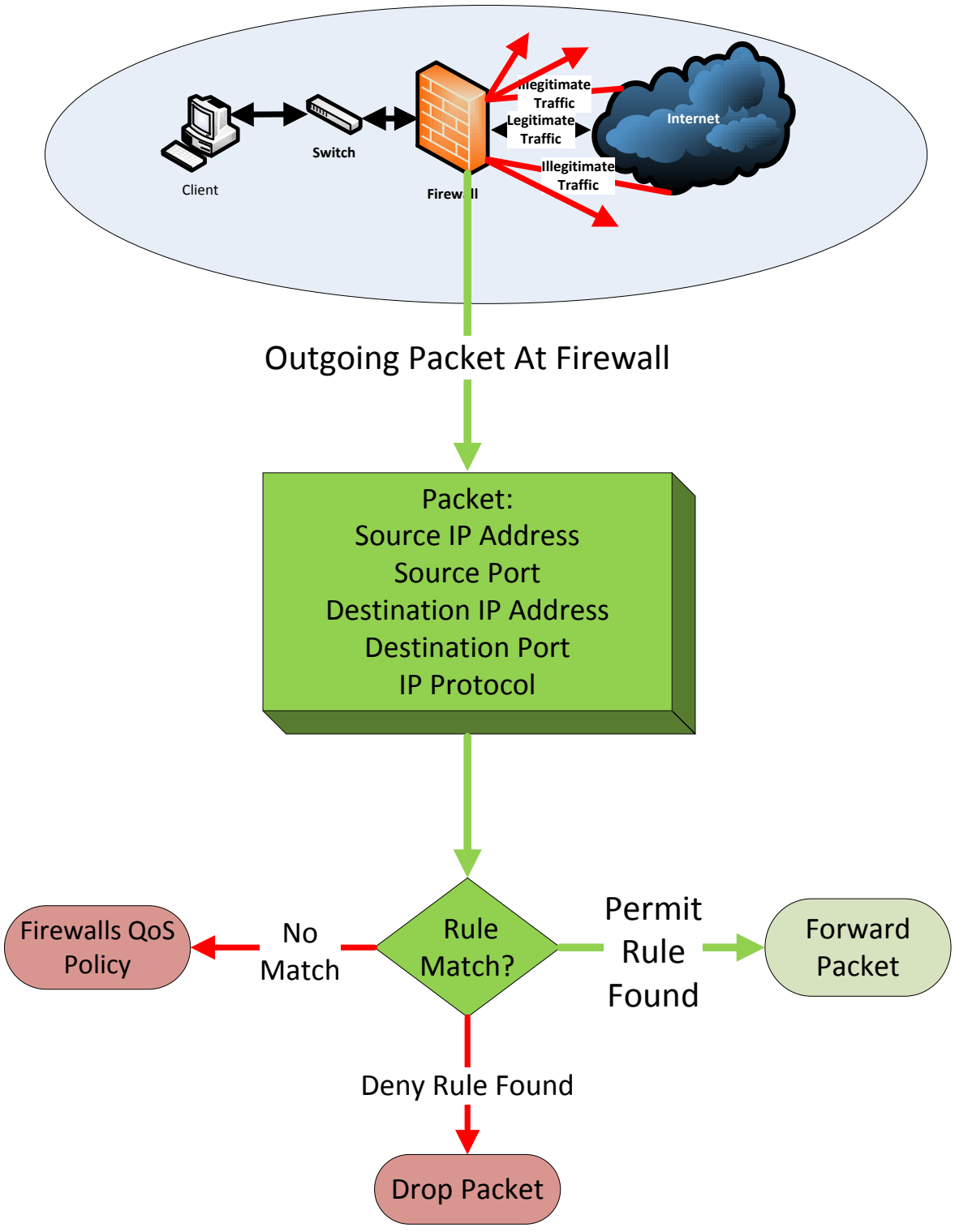

Figure 5. Filtering

Packet filtering can then be divided into three more subsections listed below.

Table 3. Packet Filtering Subtypes

Packet Filtering Subtypes	
Stateful Inspection	Similar to dynamic packet filtering adding on the granular inspection of data contained within the IP Packet. This gives the Firewall the ability to see what is in the packet, which may prove useful when implementing our SDN.
Dynamic Filtering	"Dynamic packet filtering tracks the outgoing packets it has allowed to pass and allows only the corresponding response packets to return. When the first packet is transmitted to the public network (Internet), a reverse filter is dynamically created to allow the response packet to return. To be counted as a response, the incoming packet must be from the host and port to which the outbound packet was sent." [18]
Static Filtering	Most common type of filtering, displayed in Figure 5. This filtering must be manually changed.

Application – Gateway Firewalls

Gateway firewalls, like packet filtering, determine whether or not a connection will be made through it also determines how each connection should be made. This information is crucial for building the study proposal. The gateway firewalls logic is the closest firewall rule logic that will apply to the thesis proposal. "This type of firewall stops each incoming (or outgoing) connection at the firewall, and, if the connection is permitted, initiates its connection to the destination host on behalf of whoever created the initial connection. This type of connection is called a proxy connection [18]." In short

this process should be transparent to the user and is acting merely like a simple proxy server that provides to specific applications. This type of firewall is what closely aligns to our SDN hypothesis. “By using its database, which defines the types of connections allowed, the firewall either establishes another connection (i.e., permitting the originating and destination host to communicate) or drops the original connection [18].” This type of firewall ensures protocol conformance and can even inspect individual sessions and decide to drop packets based on information in the headers or payloads.

Circuit Level – Gateway Firewalls

Combining gateway firewalls to the study proposal circuit level gateway firewalls will contribute to the proposal. These firewalls operate at the session layer of the OSI Model. “They monitor TCP handshaking between the packets to determine if a requested session is legitimate [19].” Network Address Translation or NAT is a large part of circuit level gateway firewalls. This part of the firewall will allow for a public IP address at the firewall level and internal private IP address therefore traffic being routed and possibly remote controlled into the environment is not exposed to potential intruders.

With the three types of firewall architectures explained there are stateful multilayer inspection firewalls which combine the aspects of all the other types of firewalls and filter packets amongst the transport layers, network layers and application layers. The firewalls could allow packets to pass through individuals, direction connections or the algorithms they choose to recognize at the application layer instead of specific rules.

Multiple Firewall Environments

After discussing the different types of firewalls and how their overall architecture and how our SDN is implemented, we begin to turn towards the crux of our problem. Multiple firewall environments are present in nearly every large organization and corporate network. Corporate firewalls are usually separated out into multiple risk domains. For example the company's intranet would have a firewall separating it between a low risk domains then another firewall separating the low risk domain with a high risk domain which is usually outward facing towards the internet.

On top of these risk domains we would have separate instances of local firewalls for different uses and applications. "The consistency between those firewall policies is crucial to corporate network security. However, the managing of these has become a complex and error-prone task. Bad configurations may cause serious security breaches and network vulnerabilities. In particular, conflicting filtering rules lead to block legitimate traffic or to accept unwanted packets [20]."

Understanding the complexity of these problems we need to thoroughly review a proposed abstract environment. The environment in Figure 6, which we have named Firewall Risk Domain Problem, is based off a real world model provided by a large American Fortune 500 firm.

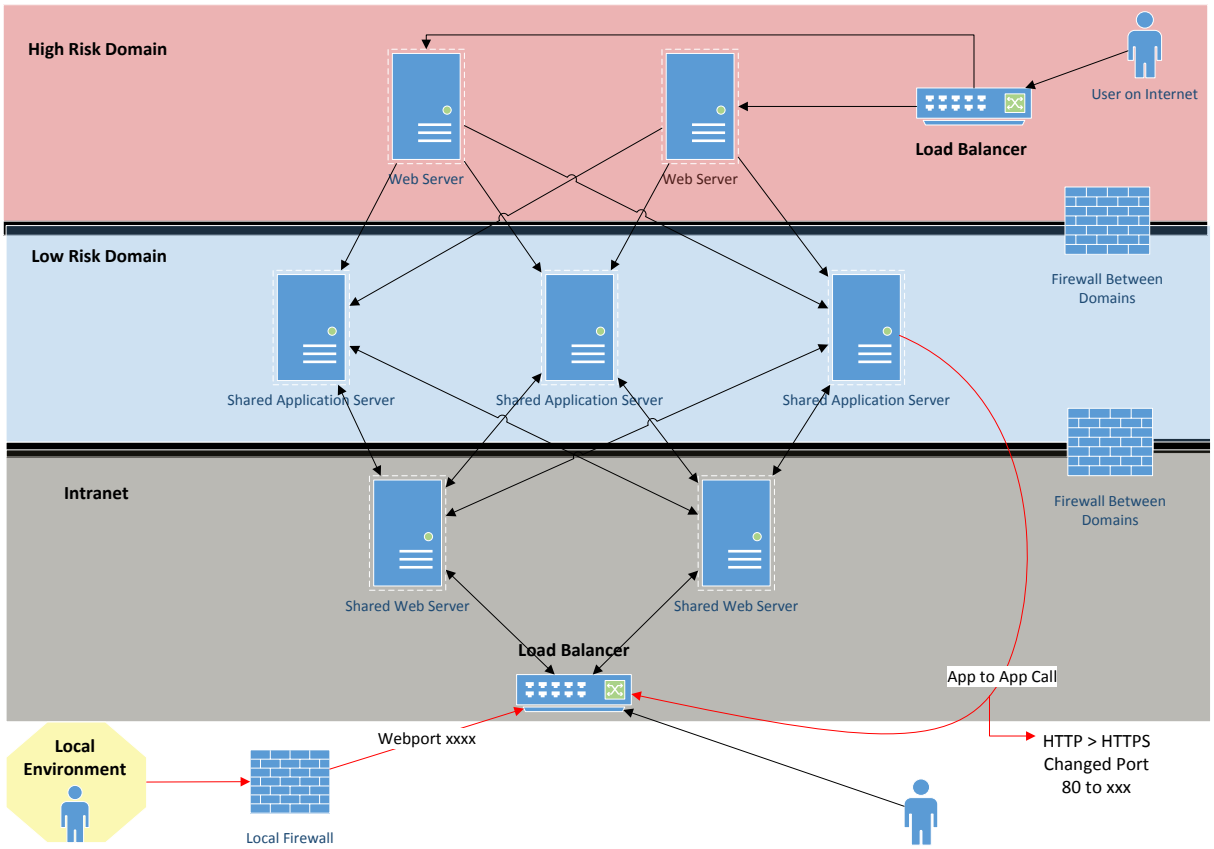

Figure 6. Firewall Risk Domain Problem

Abstracting this real world setup we will focus on using this firewall architecture as the basis for our research problem. First, we must understand the flow of data within our network and firewalls before addressing the problems it presents.

In Figure 6, we have five environments starting at the lowest level in which we first see a local environment. A good example for a local environment would be a facility or offices that are offsite from a headquarters of an enterprise or organization. This local facility has chosen to be behind a local firewall for purposes that align with their policies and usually contains computers, servers of their own, printers and other devices operating on the network.

The traffic moves from the local environment to the load balancer via a web port and also a rogue user outside the firewall. This user is shown because even though the

firewall is in place a user may find a way to be accessing the intranet load balancer not through their firewall. Firewall configuration errors could lead to major attacks; misconfiguration could lead to applications being reduced all traffic being dropped; or entire domains could be restricted or error prone. “One challenge is that large networks usually have several firewalls scattered across the network each with their own firewall policy. This makes designing and deploying an effective firewall policy difficult [20].”

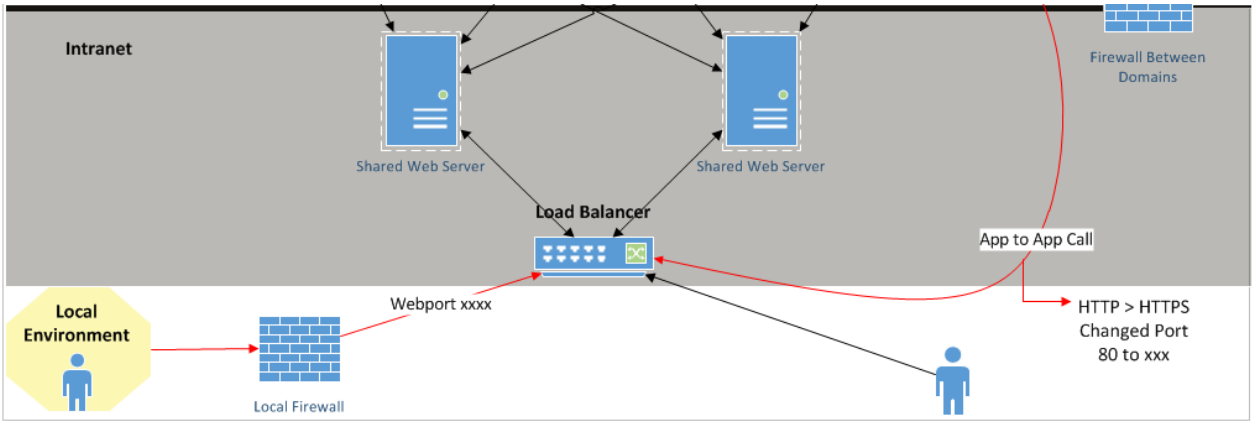

Figure 7. Close Up Intranet

In a more detailed view in Figure 7 we can see we have the corporate intranet shared with a load balancer and two shared webservers. The load balancer could be a cisco blade or any other commercial load balancer along with the web servers being java virtual machines or even windows web servers running Internet Information Services (IIS). A special note to take in is the service from one shared application server is showcasing traffic with an application-to-application call. In a real world example when changing over the HTTP to HTTPS we change the ports from 80 to 443. “Therefore, unawareness of policy conflicts and errors can significantly increase the risk of policy inconsistency thus increasing network vulnerability [21].”

Not knowing the app-to-app call which is very easy to do could cause a network vulnerability or error if the application developer did not know this was implemented. This would be an easy miss for consultant developers who inherit completed systems and begin working on them for the first time with organizations who have had the applications in maturity or retirement age. “An error in a firewall policy can be a wrong definition of being legitimate or illegitimate for some packets. This can lead to a firewall either accepting some malicious packets, which consequently creating security holes in the firewall, or discard some legitimate packets, which consequently disrupt normal business [22][23].”

This problems complexity would be further compounded if we considered both Wide Area Networks (WAN) and Local Area Networks (LAN). For simplicity we will be assuming everything is residing on a Local Area Network. However, we will take into account how complex the actual rules for the firewall are. These can be so complex that Avishai Wool dedicated his entire paper for developing a quantitative analysis for firewall rule complexity: “RC = Rules + Objects + Interfaces(Interfaces -1)/2, where RC denotes rule complexity, Rules denotes the raw number of rules in the rule set, Objects denotes the number of network objects, and Interfaces denotes the number of interfaces on the Firewall [23].” Not even getting out of the intranet environment we can begin to see the complexity of the environment.

Progressing upward we will pass through another firewall separating out intranet and our low risk domain. The traffic going from our web servers in the intranet and the application servers in the low risk domain are bidirectional web service calls. This is an important distinction due to our policies on the firewall separating out these two

environments. Within the low risk domain we have three shared application servers running various applications within their environments.

Last we have the high-risk domain with one-way traffic coming into the low risk domain separated by our last firewall. Within the high-risk domain we have two web servers and a load balancer. A great example for this environment would be of a user trying to use the same application within the intranet or low risk domain but is currently off campus and travelling mobile. Therefore the user would be high risk because they are coming from our last environment the Internet.

Analyzing and trying to resolve this complex environment is a fairly traditional problem since the inception of the Internet. Expert systems have been developed and in use for a long time and they generally work by using a database with an engine to make sure there is uniform policy and decisions being made. "This goal is usually achieved by combining a logical inference engine with a knowledge base. The information in the knowledge base contains a set of known facts and a set of production rules that allow if-then inferences on the facts and other acquired information [24]." The critical point of expert systems is there is not a defined way in our previous example to prevent an application developer changing the HTTP to HTTPS and breaking the entire application because of an unknown policy. The expert system is not interested what the application is doing only if it meets the firewalls requirements.

Table 4. Example of Firewall Rules

Protocol	Source IP	Port	Destination IP	Port	Action
UDP	192.168.10.1	80,001	10.1.1.12	80,100	Deny
TCP	DMZ	443	Any	Any	Allow

The policy matrix could be a table format or simply algorithm saying the example of a destination for this packet is the Internet and the source is DMZ which is allowed on certain ports like 80 and 443 but on intranet is allowed for all ports. Instead of merely throwing away legitimate traffic which an expert system would do if we changed our HTTP to HTTPS SDN will allow us to be robust enough to save the application developer from crippling the system with an outage.

Conclusion

The complexity of multiple firewall organizations like the one described above proves to be a problem that is pressing for traditional networks. Reviewing the original architecture of one firewall and one entry and exit point we noted that the basic premise of firewall is to allow or deny traffic. We defined the difference between a hardware firewall which is usually a network firewall sitting on a router or between the hardware servers.

Software firewalls we analyzed are specifically designed for the client they are residing on such as a windows computer. We analyzed how the firewalls operated with the three different types including filtering with subsets of static filtering, dynamic filtering and stateful inspection. Adopting the stateful inspection methodology we will push forward towards our SDN solution combining it with our second firewall type application gateway firewalls.

These modular and programmable options push us in the direction of controlling the network to a level where we can fix our exuberant problem of firewall miss configuration along the lines with application developers. Last type of firewall we discussed was circuit level gateways, which utilize the network address translation area. Using this critical information we then thoroughly analyzed our real world problem being presented in Figure 6. Examining the risk domains and how the organization is setup we must provide a solution to where if we move an application server, web server, or simply change or HTTP protocol to HTTPS it does not crash our system due to unknown firewall policy and rule. "In this case, the filtering rules and VLANs need to be well defined such that no desired traffic is blocked before reaching its destination and no undesired traffic is allowed to flow through the various firewalls in the distributive environment [20]." Publications only a few years old are proposing future work for this very problem however they are using expert systems with traditional network topologies to try and solve it. Instead we are proposing to upgrade the traditional network to a software-defined network which potentially solves our problem and provides more benefits than originally intended that is discussed in our analysis chapter.

CHAPTER V

SDN REDUCING COMPLEX ARCHITECTURES

In Chapter 4 we have reviewed basic firewall architectures and a solid foundation of our real world problem moving protocols or traffic amongst different risk domains, firewalls, and policies. Chapter 5 will focus solely on defining how we will achieve upgrading our traditional or legacy network example into an SDN network. Continuing forward we will propose how the SDN will handle HTTP being switched to HTTPS and prevent application developers from bringing an application down due to unknown rules and firewalls. "Researchers can control their own flows - by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP [25]." With Figure 6 being our proposed problem topology, Figure 4 will be our proposed solution topology.

SDN Topology Upgrade

We begin with an analysis to understand how the traditional environment has been converted into a hypothetical SDN environment in Figure 7.

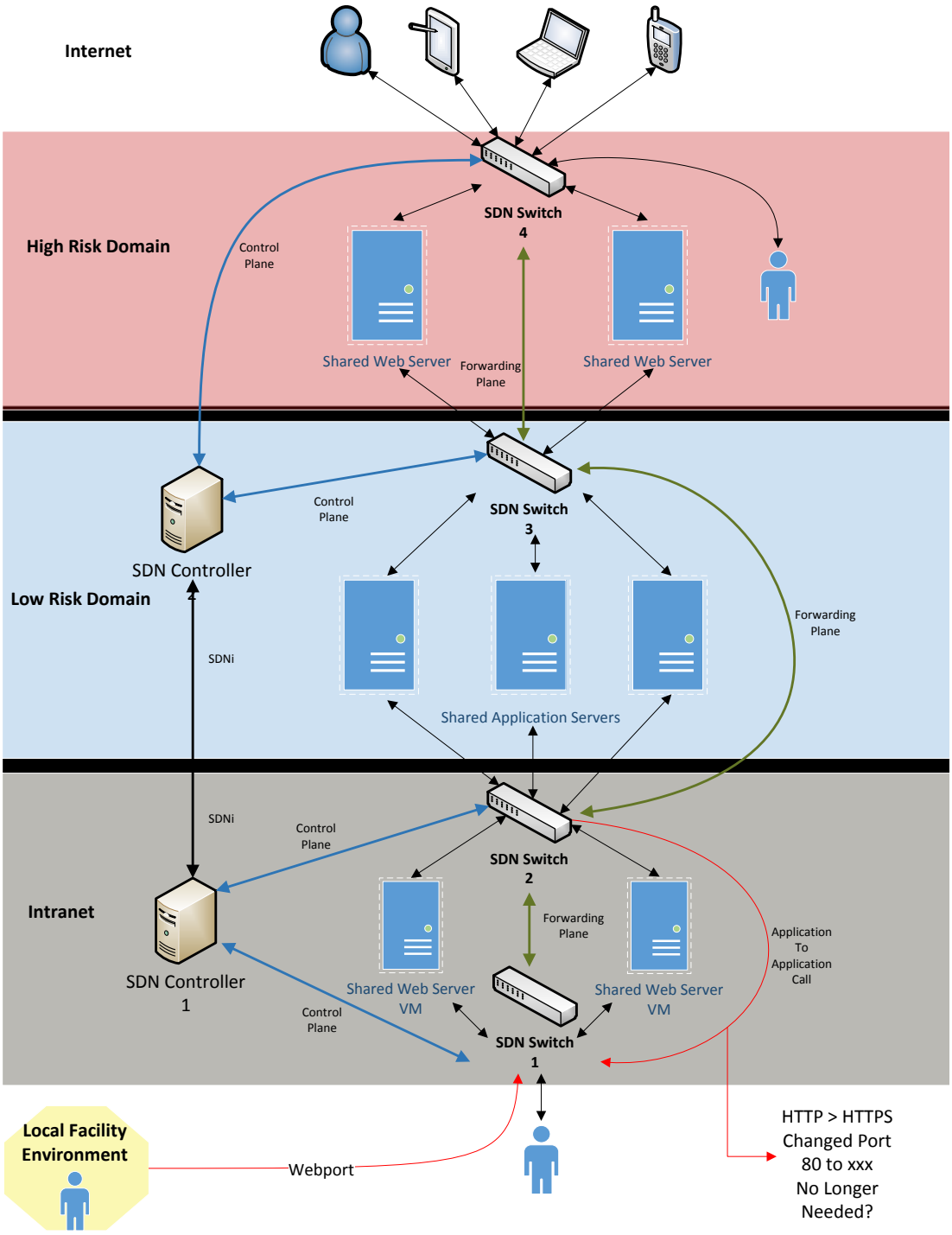

Figure 8. SDN Topology Upgrade

Using Figure 7 and others we will show how to solve the problem of migration and benefitting the developers when we preserve the nodes reachability. All firewalls in this Figure 7 have been absorbed by the OpenFlow switches. “SDN switching equipment supports flow routing tables (Flow Tables) in which processing rules for packet flows are installed. The final step of migration from a traditional topology to an SDN paradigm is installing flow rules into OF switches flow tables [26].”

Comparing to Figure 6 we can see all the firewalls have been aggregated into Flow Rules, which will be explained further on. This topology shows that we have added two SDN controllers to the original environment, removed two of the load balancers because the SDN switches act as load balancers with the flow of traffic controlled, and added the four SDN switches themselves. Noted on the graph we can see the forwarding planes denoted between each of the switches.

Coming from the SDN controller we can see the control plane being used within the secure channel that was shown in Figure 3. How the provisioning of firewall policies and sub policies within we can think of each risk domain (intranet, low risk, high risk) as a subnet or vice versa. The overall topology of the network such as lattice, star, ring, graph, dimensional cube, etc. is not taken into account for this working example. Our only focus is this small slice of the overall real world example and does not include the rest of our network that would be interlaced with extra switches, servers, controllers and clients. We are merely showing that a problem such as HTTP converting over to HTTPS would not bring the entire network down with our solution. All the servers listed on the figure are Virtual Machines (VM), which means they can be easily moved from one domain to another, can have policies changed within the server or simply moved to a

different IP address. In traditional environments the movement of servers is not chaotic; most changes will be done via the application development side of our problem. We now have a clear understanding of how our environment would look with our topology displayed and explained.

Flow Tables and Rules

Focusing on the firewall rule policies themselves we will not list out all rules and matches. Great example of rule policy logic that we can convert into the flow table below is from Modeling and Management of Firewall Policies [26].

In their paper, they provide five definitions for policy rule logic. Definition One aligns to the proposal of disjoint fields which are in turn if the server was looking for IP A to IP B it does not equal the corresponding flow within the SDN table. Definition Two provides the complex logic of meeting the flow table match. Definition Three would be in where the decision and advanced querying within SDN would be vital. The rule dictates if IP A is not matching our flow table but has a partial match communicating to another SDN controller for inclusive matching then IP A has changed while IP B or the superset IP has remained the same. This is applicable to our proposed problem specifically of moving IP A or Port 80 to HTTPS on Port 443. Definition Four would be rule logic for checking our SDN proposal all the way end to end to see if the firewall flow has been broken in another risk domain. Definition Five builds the firewall flow outward towards the ending destination. If rules one through four have been a match, the SDN must begin to build the flow creating an end-to-end flow by correlating with previous flows in place. All detailed formulas can be found in Table 5.1.

Table 5. Policy Rule Logic [26] [27]

Definition Number	Description
Definition 1	***Definition 1*** — Rules R_x and R_y are *completely disjoint* if every field in R_x is not a subset nor a superset nor equal to the corresponding field in R_y. Formally, $R_x \Re_{CD} R_y$ if $\forall i: R_x[i] \nprec R_y[i]$ where $\prec \in \{\subset, \supset, =\}$, $i \in$ {protocol, s_ip, s_port, d_ip, d_port}
Definition 2	***Definition 2*** — Rules R_x and R_y are *exactly matching* if every field in R_x is equal to the corresponding field in R_y. Formally, $R_x \Re_{EM} R_y$ if $\forall i: R_x[i] = R_y[i]$ where $i \in$ {protocol, s_ip, s_port, d_ip, d_port}
Definition 3	***Definition 3*** — Rules R_x and R_y are inclusively matching if they do not exactly match and if every field in R_x is a subset or equal to the corresponding field in R_y. R_x is called the subset match while R_y is called the superset match. Formally, $R_x \Re_{IM} R_x$ if $\forall i: R_x[i] \subseteq R_y[i]$ and $\exists j$ such that $: R_x[j] \neq R_y[j]$ where $i, j \in$ {protocol, s_ip, s_port, d_ip, d_port} For example, in Fig. 1, Rule 1 inclusively matches Rule 2. Rule 1 is the subset match while Rule 2 is the superset match.
Definition 4	***Definition 4*** — Rules R_x and R_y are *partially disjoint* (or partially matching) if there is at least one field in R_x that is a subset or a superset or equal to the corresponding field in R_y, and there is at least one field in R_x that is not a subset and not a superset and not equal to the corresponding field in R_y. Formally, $R_x \Re_{PD} R_y$ if $\exists i, j$ such that $R_x[i] \prec R_y[i]$ and $R_x[j] \nprec R_y[j]$ where $\prec \in \{\subset, \supset, =\}$, $i, j \in$ {protocol, s_ip, s_port, d_ip, d_port}, $i \neq j$ For example, Rule 2 and Rule 6 in Fig. 1 are partially disjoint (or partially matching).

Definition 5

Definition 5 — Rules R_x and R_y are *correlated* if some fields in R_x are subsets or equal to the corresponding fields in R_y, and the rest of the fields in R_x are supersets of the corresponding fields in R_y. Formally, $R_x \Re_C R_y$ if

$$\forall i: R_x[i] \asymp R_y[i] \text{ and}$$

$$\exists j, k \text{ such that } :R_x[j] \subset R_y[j] \text{ and } R_x[k] \supset R_y[k]$$

where $\asymp \in \{\subset, \supset, =\}$,

$i, j, k \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}, j \neq k$

For example, Rule 1 and Rule 3 in Fig. 1 are correlated.

With the programmability offered by SDN an organization can slowly create small trial environments to see how they might affects their overall network and production environments. Creating more and more of these pocket environments will eventually lead to the overall network being converted into a software defined network instead of a traditional.

Many different versions of pseudo code have been presented traditional firewall policy research. For example, “If filtering rules on two different routes between subnets are different, i.e. if an end point on one route is reachable and on the other is not, the warning message “Conflict Found” is displayed and computation stops” is example pseudocode from Bob, et al. [26], which has a similar goal to this study. However, what most papers fail to identify is the pseudo code and process by which we should be using our rules.

Pseudo Code Logic Proposal

```
BOOLEAN Change Application Network (Application){
     IF (App.HasChanged) AND {FlowPolicy.Firewall =
FlowPolicyAcceptance
          Then
               Controller.UpdateFlowPolicy
               Controller.SwitchFlowTableUpdate
               RETURN TRUE;
     ELSE IF FlowPolicy.Firewall != FlowPolicyAcceptance
          THEN
               App.FirewallHybrid
               Controller.UpdateFlowPolicy
               Controller.SwitchFlowTableUpdate
               RETURN TRUE;
     ELSE
          Controller.DoNotUpdateFlowPolicy
          Controller.SwitchFlowTableUnchanged
          RETURN FALSE;
```

Mathematical Proof Pseudo Code

$$f(App1): SC1\Delta(N1 \vee F1) \rightarrow f(PCY): SC1 \cap F1 = PA \rightarrow SC1\Delta(SW1 \wedge F1) \sim PCY$$

$$= PR \rightarrow SC1(SW1 \wedge Fx^{N}) = (PA \wedge \Delta SC1) \sim SC1 \cap QoS$$

$$\rightarrow SC1(SW1 \wedge F2)$$

Above in our pseudo code proposal, this study proposes an application created and running Java or C++ on the controller. This application checks for the change in state of the applications currently listed in the controller and communicating on its network. The network is defined as the domains previously discussed with intranet, low risk domain and high risk domain. If an application listed on the controller presents a change as proposed in Figure 7, the proposed application of this study will follow this logic.

First, the program has a Boolean if an application has changed in its listing and the change meets the flow rules and firewall rules in the form of flow policy acceptance on the network, it will update the controllers table and update the switches table which contain both the flow rules and combined firewall rules returning a true value. Else if, the

application plans to create a move that does not present a policy acceptance it will call the function App.FirewallHybrid that is outlined abstractly in Table 5 to create the proper flow rules updating the policy on the controller and the flow table on the switch.

Last, if none of the criterion meet the designed specifications and terms of quality service it should return a false, not changing or updating the controllers flow and switch table, preventing the changed application flow to be interrupted and the application crashing. The Mathematical proof is an abstract of the pseudo code outlining similarly.

Figure 9. Flow Controller Logic

In Greg Ferro's essay *SDN Use Case: Firewall Migration in the Enterprise* [28] he also proposes flow migrations with traditional networks. However, comparing to our research with Greg's is proposing a logical step-by-step migration of each firewall. "Now you have forced the flow over to the alternate firewall while other flows continue to traverse the old firewall." While this API has not been developed, it shows this study is not the only approach combining SDN and firewalls. Our method also proposes that application developers not migrate over single firewall rule policies, but upgrade pieces of the network with SDN all at once.

How do Flows, Flow?

With our top-level logic explained we will now briefly review how the OpenFlow Switch provides MAC forwarding and IP Forwarding. The example provided by the ONF is in Figure 7.

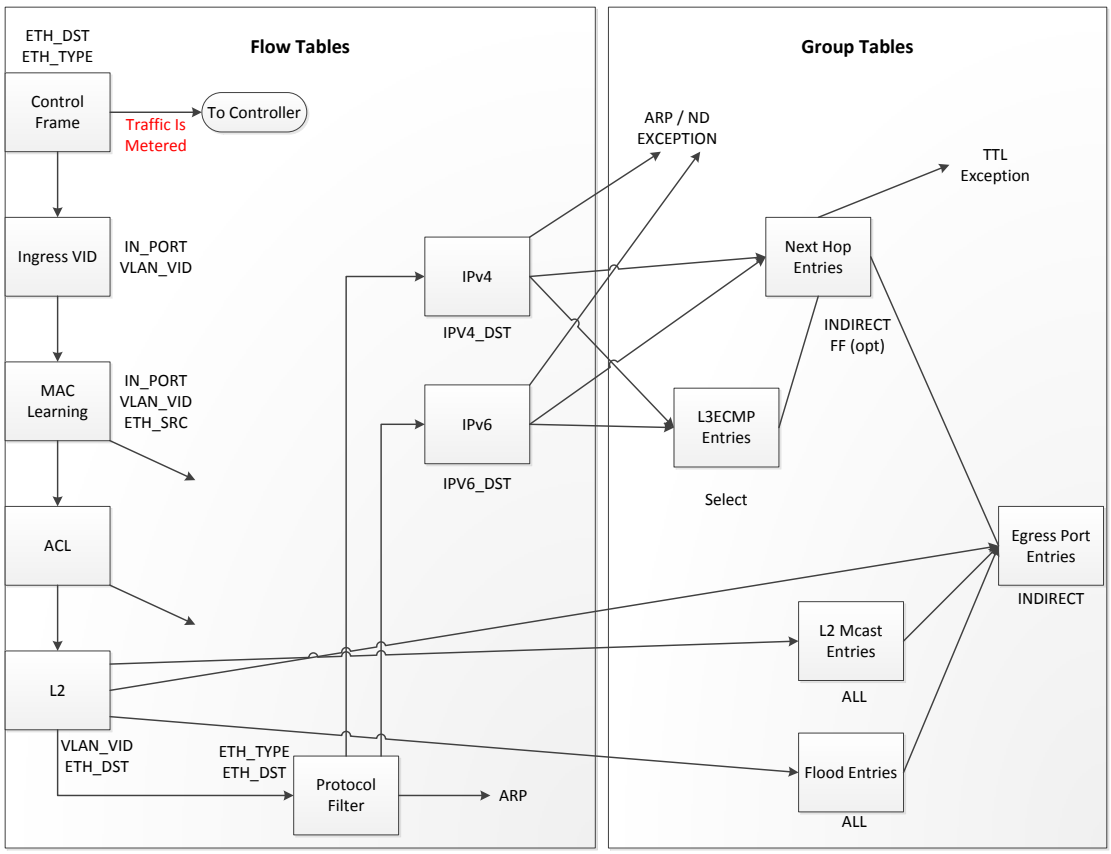

Figure 10. Flow Table Logic [29]

This shows how the controller has the controlling characteristics based on the older version of the OpenFlow protocol. This is a common standard for bridging and forwarding so the descriptions and characteristics are well known. However, as noted in the figure the control frame has not been merged with access control lists table, the key feature we should take away from Figure 10. We can now add in matching rules to the flow table to automate our policies. Our architecture in Table 5 creates the topology shown below in Figure 11.

Simplified Flow

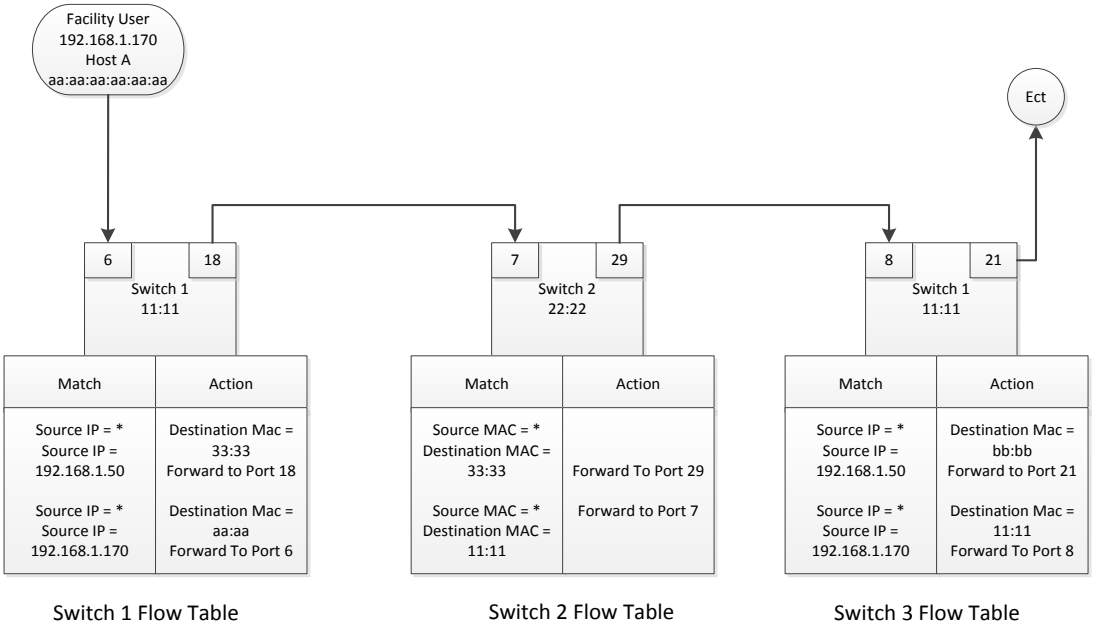

Figure 11. Flow Topology

Using Figure 11, we can see an example of how the MAC address is being issued by the SDN controller with our Firewall policies to complete an end-to-end flow. This end-to-end flow design is an example how we can achieve our goal with SDN and firewall rules being less complex. As Eric Rozner said, "The MAC address rewriting scheme leverages the fact that OpenFlow-compatible switches can rewrite addresses in the data plane at line rate [30]." This means we are able to drive that packet at line rate into a particular path with the rules we subject from the ingress port and the egress port. We can even begin to write rules corresponding to the hypervisor on the switches with the virtual hosts like our shared webservers and shared application servers as seen in Appendix A examples.

CHAPTER VI

DID IT WORK?

Hearing the system is down after a patch is every developer's worse nightmare in a production environment. The purpose of this thesis proposes a solution to this problem of complex and vague firewall policy environments by implementing SDN. Did we succeed at preventing a critical incident with the study prototype? The answer is, yes, theoretically, and inconclusive in a large organization.

This thesis has proposed the pseudo code with the firewall programmable logic in Table 5, which should, in theory, produce solution similar to Figure 11 flow table. This flow table would then follow the general applicable steps provided in Figure 8 to properly route our HTTP to HTTPS while checking for rule violation and integrity along the way with Table 5 logic. Also for local firewall searches this study will use the local firewall search algorithm in Appendix A. We also show that we can program the flow tables to prevent the traffic from dropping with SDN the end-to-end logic flow. However, the number of flows that a switch can store is limited by the hardware itself. Another limiting factor is getting switches with OpenFlow 1.3 or later versions enabled on them. Vendor sentiment has been against creating an open environment because they would like to convert OpenFlow to their own proprietary platforms. "Standards have been a lively part of the SDN debate, but that discussion has been focused more on how forwarding is programmed into individual network devices." [32]

SDN Extended

We have started off explaining the top levels of SDN. We examined why SDN was originally proposed ranging from traffic patterns to our problem of security and controls. SDN will continue to evolve as the technology is still in its infancy and the needs are growing for it every day. It has spawned numerous other technologies and projects running in parallel infancy and growing such as Network Functions Virtualization (NFV). "Whereas NFV focuses on network platform virtualization, SDN is focused on network virtualization [33]."

Our proposed solution today might be outdated tomorrow by functions served up in NFV. Gathering the updated and most recent data, the best conclusion is to participate with leading groups such as the ONF. "Leading standards groups like the European Telecommunications Standards Institute (ETSI) is already devising ways to unify SDN and NFV monitoring data [34]." Expanding even further on the rapid pace of SDN concepts including NFV, this study started off in April, 2013. One year into writing this thesis the subject of running an SDN network efficiently came about.

Understanding SDN and the proposal's limitation understands that every connection in the network is a flow. On top of the flows we are adding firewall rule logic creating a firewall flow. This would mean thousands upon thousands of flow entries in a moderate to large network. We mentioned previously that the limitations of the hardware are one obstacle; however, even running the network that large itself would take a program equally if not greater in mass and complexity. "But exactly how will network management accomplish this feat? Curiously enough, the answer is Big Data [34]." As we mentioned in Chapter 3 a driving force for SDN was Big Data applications, however,

ironically it seems we need Big Data applications to run the network of Big Data applications.

Explaining through the concepts of the SDN controller, we have noted several different types of controllers. While we didn't select a specific controller for future research it should be noted NOX was the original OpenFlow controller. "NOX is the original OpenFlow controller. It serves as a network control platform that provides a high-level programmatic interface for management and the development of network control applications [35]." We covered how the application layer, the control layer and infrastructure layers role are fulfilled with the SDN architecture and eventually explored SDNi.

In our SDNi model, it should be noted that we took an East to West approach with our protocols determining the SDNi. This means the controllers communicate with each other and base their decisions off the connections and peers. However, there is a vertical approach where a row of SDN controllers is in turn controlled by one SDN controller that sits on top of the stack. This one controller will then issue commands in a decomposition method controlling the network as a whole. "The master controller has a global view of the network across all connected SDN domains and can orchestrate the configuration in each domain [36]." We chose the horizontal approach because…

We integrated our designs into the OpenFlow protocol that is a decoupled design with the control plane and data (or forwarding) plane separated. The controllers in this design utilize this control plane in the secure channel to issue commands and our networking flow rules. While the clients and hosts received and issues these commands from the flow table on the forwarding plane. This study shows the ports with the protocol

provided by the ONF manage the switches and communicate with the controllers with the fields All, Controller, Table, In_Port, Any, and Local which have subfields. Briefly describing the SDN vulnerabilities should be expanded upon in future research. Very little is discussed about the current limitations of the flow table design and the hardware limitations behind it as well.

Future Work

The main contribution of this study is using the firewall logic in Table 5 and combining it with psuedocode to provide a flexible, misconfiguration-tolerant SDN firewall. A real-world implementation of the paper prototype could use a NOX C++ controller; create flow tables with firewall rule logic added into the set-field to ensure the application does not drop due to not adding in a rule allowing for traffic on port 443. We can implement firewall policies and ACL's by programming the SDN controller and switch. The policies and logic tree matrixes would work at a top level logic inside the SDN controller communicating to the switches flow table. This means the decision making process would start from the horizontal SDNi chain of controllers and work downward towards the switches. Future work should consist of research into manipulating the set-field option within SDN. "While not strictly required, the support of rewriting various header fields using Set-Field actions greatly increases the usefulness of an OpenFlow implementation [13]."

This Set-Field option is what allows us to apply our modifications to the outermost header of the packet and VLAN header as well. Creating a simple forward flow firewall policy function we are able to forward packets based on simple logic trees. However, despite the scalability is built into the SDN architecture, in practice hardware

limitations may be reached around four thousand flow tables [37]. Depending on the usage of CPU and memory with the controllers and switches depends on the maximum number of flows a developer could achieve.

A large research project should be undertaken with the SDN firewall proposals. There are many excellent ideas and proposals but nothing substantial to initiate an aggregation of project papers and developers to pursue this solution. This thesis should spawn a research initiative to begin solving the complex logic trees and limitation of the SDN switches and their number of flow tables and policies. Hongxin Hu from Arizona State University said, “The goal of this work is to design and develop a systematic solution for building reliable firewalls that enable effective network-wide access control in SDNs [38].” The research they are presenting is how to configure firewalls within SDN environments first and then optimize the firewall after the implementation.

Conclusion

This study proposes future work of developing a SDN solution for traditional networks that incorporate firewall rules into the flow tables. One application dedicated to the monitoring of this would be created with the outline of the pseudocode and placed on the SDN controller. Using the OpenFlow protocol and SDNi, the controller would be able to make firewall access decisions automated for basic or advanced scenarios.

REFERENCES

[1] ONF Overview., “Open Networking Foundation," [online] 2014, https://www.opennetworking.org/about/onf-overview (Accessed: 03 September 2014).

[2] Member Listing., “Open Networking Foundation," [online] 2015, https://www.opennetworking.org/our-members (Accessed: 03 September 2014).

[3] K. Greene., MIT Technology Review," [online] 2014, http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/ (Accessed: 03 September 2014).

[4] "Software-Defined Networking: The New Norm Of Networking” Open Network Foundation. April 13, 2012. October, 2014 <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>

[5] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. “FLOWGUARD: building robust firewalls for software-defined networks. in Proc. Hot SDN ‘14 (SIGCOMM '14), 2014, pp. 97-98.

[6] G. Wang, T.S. Eugene Ng, A. Shaikh. “Programming your network at run-time for big data applications. in Proc. Hot SDN ‘12 (SIGCOMM '12), 2014, pp. 103-104.

[7] Software-Defined Networking (SDN) Definition., Open Networking Foundation," [online] 2015, https://www.opennetworking.org/sdn-resources/sdn-definition (Accessed: 09 September 2014).

[8] What are SDN Controllers?., SDX Central," [online] 2014, https://www.sdxcentral.com/resources/sdn/sdn-controllers/ (Accessed: 10 September 2014).

[9] Steven Vaughan-Nichols, "OpenFlow: The Next Generation of the Network?" Computer, August 2011.

[10] H. Yin, H. Xie, T. Tsou, D. R. Lopez, P. A. Aranda, R. Sidi. “The case for SDNi SDN controller interconnection. in Proc. IETF ‘12 (The IETFJ Journal '12), 2014, pp. 84-89.

[11] W. Stallings. “Software-Defined Networks and OpenFlow.” The Internet Protocol Journal, vol. 16, pp.1-8, May. 2014.[12] IBM. “OpenFlow: The next generation in networking interoperability. Internet: http://public.dhe.ibm.com/common/ssi/ecm/qc/en/qcw03010usen/QCW03010USEN.PDF, May. 01, 2011 [Oct. 10, 2014].

[13] "OpenFlow Switch Specification” Open Network Foundation. October 14, 2013. October, 2014 <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>

[14] InformationWeek. “Beware SDN Security Risks, Experts Warn” http://www.networkcomputing.com/networking/beware-sdn-security-risks-experts-warn/d/d-id/1234667?, Feb. 12,2014 [Oct. 16, 2014].

[15] Dell. “How Traditional Firewalls Fail Today’s Networks – And Why Next Generation Firewalls Will Prevail. Internet: http://software.dell.com/documents/how-traditional-firewalls-fail-todays-networks-ebook-24532.pdf, 2012 [Oct. 21, 2014].

[16] Tech-Faq. “Firewalls. Internet: http://www.tech-faq.com/firewall.html, Sep. 26, 2012 [Oct. 23, 2014].

[17] F. Avolio. “Firewalls and Internet Security, the Second Hundred (Internet) Years.” The Internet Protocol Journal, vol. 2, pp.1-5. Available: http://www.cisco.com/web/about/ac123/ac147/ac174/ac200/about_cisco_ipj_archive_article09186a00800c85ae.html [Nov. 5, 2014].

[18] E. Eugene Schultz. “83-10-41 Types of Firewalls.” Internet: http://www.ittoday.info/AIMS/DSM/83-10-41.pdf, [Nov. 5, 2014].

[19] K. Rajesh. “What are: Packet filtering, Circuit level, Application Level and Stateful Multilayer inspection Firewalls,” http://www.excitingip.com/205/what-are-packet-filtering-circuit-level-application-level-and-stateful-multilayer-inspection-firewalls/, Jun. 13,2009 [Nov. 5, 2014].

[20] Loye L. Ray. “A Matrix Model for Designing and Implementing Multi-firewall Environments.” The International Journal of Information Security Science, vol. 2, pp.119-128. Available: *www.ijiss.org/ijiss/index.php/ijiss/article/download/30/pdf_13* [Nov. 18, 2014].

[21] H. Hamed, E. Al-Shaer, “Taxonomy of Conflicts in Network Security Policies”, IEEE Communications Magazine, Vol. 44, Issue 3, pp. 134-141, March 2006a.

[22] M. Marmorstein, “Formal Analysis of Firewall Policies”, College of William and Mary, doctoral dissertation, 2008.

[23] A. Wool, “A Quantitative Study of Firewall Configuration Errors”, Computer, Vol. 37, No. 6, pp. 62-67, June 2004.

[24] P.Eronen, J. Zitting. “An Expert System For Analyzing Firewall Rules.” Internet: http://astarloa.hit.bg/firewalls/An%20Expert%20System%20for%20Analyzing%20Firewall%20Rules%20%282001%29.pdf, [Nov. 22, 2014].

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008.

[26] D. Gamayunov, I. Platonovm, R. Semlieansky. “Toward Network Access Control With Software-Defined Networking.” Internet: http://www.researchgate.net/profile/Ruslan_Smeliansky/publication/236148336_Toward_Network_Access_Control_With_Software-Defined_Networking/links/00b7d51caf777e947a000000.pdf, Jun. 06, 2013 [Nov. 29, 2014].

[27] E. Al-Shaer and H. Hamed, “Modeling and Management of Firewall Policies,” in IEEE eTransactions on Network and Service Management, vol. 1-1, April 2004.

[28] G. Ferro., “Open Networking Foundation," [online] Mar, 18 2013, http://etherealmind.com/sdn-use-case-firewall-migration-in-the-enterprise/ (Accessed: 03 December 2014).

[29] "OpenFlow Table Type Patterns” Open Network Foundation. August 15, 2014. December 2014 <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/OpenFlow%20Table%20Type%20Patterns%20v1.0.pdf>

[30] K. Agarwal, C. Dixon, E. Rozner, J. Carter. “Shadow MACs: scalable label-switching for commodity ethernet.” in Proc. Hot SDN ‘14 (SIGCOMM '14), 2014, pp. 157-162.

[31] I. Pepelnjak. “Flow Table Explosion with OpenFlow 1.0 (And Why We Need OpenFlow 1.3).” http://blog.ipspace.net/2013/10/flow-table-explosion-with-openflow-10.html, [Dec. 10, 2014].

[32] M. Bushong., “Using big data for SDN: How analytics will enable programmability," [online] Aug, 2013, http://searchsdn.techtarget.com/opinion/Using-big-data-for-SDN-How-analytics-will-enable-programmability (Accessed: 09 January 2015).

[33] Netronome., “Software-Defined Network Functions Virtualization (SDN & NFV)," [online] 2013, https://netronome.com/wp-content/uploads/2013/12/Netronome-SDN-NFV-whitepaper_11-13.pdf (Accessed: 10 January 2015).

[34] A. Cole., “SDN and Big Data: Enterprise, Analyze Thyself," [online] Nov, 7 2014, http://www.enterprisenetworkingplanet.com/datacenter/datacenter-blog/sdn-and-big-data-enterprise-analyze-thyself.html (Accessed: 21 January 2015).

[35] S. Rao., “SDN Series Part Three: NOX, the Original OpenFlow Controller," [online] Jan, 5 2013, http://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/ (Accessed: 09 January 2015).

[36] D. Gupta, R. Jahan., “Inter-SDN Controller Communication: Using Border Gateway Protocol," [online] 2013, http://www.tcs.com/resources/white_papers/Pages/InterSDN-Controller-Communication.aspx (Accessed: 25 January 2015).

[37] J. Liao., “SDN System Performance," [online] June 13, 2012, http://www.tcs.com/resources/white_papers/Pages/InterSDN-Controller-Communication.aspx (Accessed: 02 February 2015).

[38] H. Hu, G.J, Ahn, W. Han, Z. Zaho., “SDN System Performance," [online] June 13, 2012, http://www.tcs.com/resources/white_papers/Pages/InterSDN-Controller-Communication.aspx (Accessed: 02 February 2015).

[39] P. Costa, A. Donnelly, AIT. Rowstron, G. O’Shea. “Camdoop: Exploiting In-network Aggregation for Big Data Applications. in Proc. NSDI ‘12 (USENIX '12), 2014, pp. 3-3.

[40] D. Bartley, P. Andres, R. Hunter, B. Covington, P. Heller., “Architectural Strategies for IT Optimizations: From Silos to Clouds," [online] 2010, http://www.oracle.com/technetwork/topics/entarch/whatsnew/oea-wp-optimization-129780.pdf (Accessed: 29 March 2015).

[41] The Visible OPS Handbook Implementing ITIL In 4 Practical And Auditable Steps, Information Technology Process Institute., Phoenix, AZ, 2014.

[42] R. Colville, G. Spafford., “Configuration Management for Virtual and Cloud Infrastructures," [online] May 17, 2013, http://datasecuritycompliance.blogspot.com/2013/05/configuration-management-for-virtual.html (Accessed: 30 March 2015).

[43] B. Salisbury., “Inside Google's Software-Defined Network," [online] May 14, 2014, http://www.networkcomputing.com/networking/inside-googles-software-defined-network/a/d-id/1234201? (Accessed: 30 March 2015).

[43] A. Vahdat., “Enter the Andromeda zone - Google Cloud Platform’s latest networking stack," [online] April 2, 2014, http://googlecloudplatform.blogspot.com/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html (Accessed: 30 March 2015).

[44] EC-Council. Albuquerque, NM: EC-Council Press, 2010.

[43] A. Vahdat., “Enter the Andromeda zone - Google Cloud Platform’s latest networking stack," [online] April 2, 2014, http://googlecloudplatform.blogspot.com/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html (Accessed: 30 March 2015).

[44] The Tolly Group - Company Background., “Tolly Enterprises, LLC," [online] 2014, http://www.tolly.com/AboutUs.aspx (Accessed: 31 March 2014).

[45] M. McNickle., “Five SDN protocols other than OpenFlow," [online] August 24, 2014, http://searchsdn.techtarget.com/news/2240227714/Five-SDN-protocols-other-than-OpenFlow (Accessed: 31 March 2014).

APPENDIX A

USING SOFTWARE DEFINED NETWORKING TO SOLVE

MISSED FIREWALL ARCHITECTURE

IN LEGACY NETWORKS

OpenFlow Pipeline Example

Denotes how the set-field action takes place.

cookie=0x0, duration=642.651s, table=0, n_packets=30, n_bytes=2586, send_flow_rem tun_id=0x1,in_port=2 actions=goto_table:20

cookie=0x0, duration=563.287s, table=0, n_packets=30, n_bytes=2586, send_flow_rem in_port=3,dl_src=fa:16:3e:1c:fc:3b actions=set_field:0x1->tun_id,goto_table:10

cookie=0x0, duration=644.372s, table=0, n_packets=37, n_bytes=4198, send_flow_rem in_port=1,dl_src=fa:16:3e:e6:a8:9f actions=set_field:0x1->tun_id,goto_table:10

cookie=0x0, duration=562.906s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=8192,in_port=3 actions=drop

cookie=0x0, duration=644.197s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=8192,in_port=1 actions=drop

cookie=0x0, duration=4641.604s, table=0, n_packets=125, n_bytes=11125, send_flow_rem dl_type=0x88cc actions=CONTROLLER:56

cookie=0x0, duration=643.569s, table=10, n_packets=33, n_bytes=3356, send_flow_rem priority=8192,tun_id=0x1 actions=goto_table:20

cookie=0x0, duration=642.293s, table=10, n_packets=19, n_bytes=1614, send_flow_rem priority=16384,tun_id=0x1,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:2,goto_table:20

cookie=0x0, duration=490.806s, table=10, n_packets=15, n_bytes=1814, send_flow_rem tun_id=0x1,dl_dst=fa:16:3e:c8:c8:26 actions=output:2,goto_table:20

cookie=0x0, duration=643.162s, table=20, n_packets=15, n_bytes=1814, send_flow_rem priority=8192,tun_id=0x1 actions=drop

cookie=0x0, duration=643.71s, table=20, n_packets=43, n_bytes=3658, send_flow_rem priority=16384,tun_id=0x1,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=output:1,output:3

cookie=0x0, duration=643.931s, table=20, n_packets=24, n_bytes=2084, send_flow_rem tun_id=0x1,dl_dst=fa:16:3e:e6:a8:9f actions=output:1

cookie=0x0, duration=562.286s, table=20, n_packets=15, n_bytes=1814, tun_id=0x1,dl_dst=fa:16:3e:1c:fc:3b actions=output:3

OpenFlow Pseudo Code Pipeline

```
BOOLEAN Change Application Network (Application){
     IF (App.HasChanged) AND {FlowPolicy.Firewall =
FlowPolicyAcceptance
          Then
               Controller.UpdateFlowPolicy
               Controller.SwitchFlowTableUpdate
               RETURN TRUE;
     ELSE IF FlowPolicy.Firewall != FlowPolicyAcceptance
          THEN
               App.FirewallHybrid
               Controller.UpdateFlowPolicy
               Controller.SwitchFlowTableUpdate
               RETURN TRUE;
     ELSE
          Controller.DoNotUpdateFlowPolicy
          Controller.SwitchFlowTableUnchanged
          RETURN FALSE;
```

$$f(App1): SC1\Delta(N1 \vee F1) \rightarrow f(PCY): SC1 \cap F1 = PA \rightarrow SC1\Delta(SW1 \wedge F1) \sim PCY$$

$$= PR \rightarrow SC1(SW1 \wedge Fx^{N}) = (PA \wedge \Delta SC1) \sim SC1 \cap QoS$$

$$\rightarrow SC1(SW1 \wedge F2)$$

OpenFlow Repository

Stanford University Repository

http://yuba.stanford.edu/git/gitweb.cgi?p=openflow.git;a=summary

sudo apt-get -y install ssh
<ssh into your VM>

sudo apt-get install git-core automake m4 pkg-config libtool
git clone git://openflow.org/openflow.git
cd openflow
./boot.sh
wget http://openflow.org/downloads/openflow-1.0.0.tar.gz
tar xzf openflow-1.0.0.tar.gz
cd openflow-1.0.0

sudo apt-get install gcc

./configure
make
sudo make install

Firewall Local Search

```
repeat
   Δmax = 0
   for i := 1 to n-2 do                    \ For all node pairs
      for j := i+1 to n-1 do
         if dij = 0 then                   \ If not dependent
            begin
               Δ := E(L) - E(L<i↔j>);
               if Δ > Δmax then            \ Consider swapping
                  begin
                     Δmax := Δ;
                     i* := i;              \ Record largest gain
                     j* := j
                  end
            end;
   if Δmax > 0 then
      swap(ri*, rj*)                       \ Implement best swap
until
   Δmax = 0                                \ Until no improvement
```

[27]