5-14-2023

Software-Defined Networking Security Techniques and the Digital Forensics of the SDN Control Plane

Abdullah Alshaya
Louisiana State University and Agricultural and Mechanical College

SOFTWARE-DEFINED NETWORKING SECURITY TECHNIQUES AND THE DIGITAL FORENSICS OF THE SDN CONTROL PLANE

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Division of Computer Science and Engineering

by
Abdullah Alshaya
Bachelor of Electrical Engineering (Communication) 2002
Master of Telecommunications Networks 2013
August 2023

This thesis is dedicated to my father, my mother, my wife, my kids, my brothers and sisters, my family, and my friends

Acknowledgments

For everything that Professor. Golden G. Richard III has done for me since the beginning of my Ph.D. program, I am forever grateful. I will never forget the encouragement, support, and guidance that Prof. Richard III gave me throughout my research and education.

Additionally, I wish to convey my deep appreciation to the members of my Dissertation Committee, Professor. Ibrahim (Abe) Baggili and Dr. and Aisha Ali-Gombe, for all of their support and supervision. To my supervisor Mr. Andrew Case (Volexity), I am also thankful for his assistance. I couldn't have gotten where I am today in my pursuit of a Ph.D. without the support of my family. I would like to thank my parents, whose love, advice, and support are with me in whatever I continue pursuing. Above all, I want to express my gratitude to my wonderful wife, Fawzayh, and to our four lovely children. Faisal, Rema, Lana, and Talal are amazing children who never fail to inspire their father. I'd also like to say thanks to my brothers and sisters for all the love and encouragement they've given me. As a last note, I'd want to express my gratitude to my colleagues in the Applied Cyber Security Lab. Special thanks for their contributions and teamwork to my friends Adam Kardorff and Christian Facundus.

Contents

List of Tables

List of Figures

Abstract

Software-Defined Networking (SDN) is an efficient networking design that decouples the network's control plane from the data plane. When compared to the traditional network architecture, the SDN architecture shares many of the same security issues. The centralized SDN controller makes it easier to control, easier to program in real-time, and more flexible, but this comes at the cost of more security risks. An attack on the control plane layer of the SDN controller is a major security concern.

First, centralized design and the existence of a single point of failure in the control plane compromise the accessibility and availability of data or services. A failure of the SDN controller will make the entire network unavailable. In this research, we propose an architecture made up of a set of open-source RYU controllers that work together to achieve an effective level of performance, availability, and scalability against the threat of a single point of failure on the control plane of the SDN and the threat of a DDoS attack on the control plane.

Second, in this project, we discussed Man-In-The-Middle (MITM) attacks and their tremendous impact on the SDN control plane as one of the most serious threats in SDN. With a specific focus on the ARP poisoning attack against the SDN, we developed a security tool that is capable of identifying and preventing MITM attacks in the SDN. Our solution, titled the RYU SDN Controller ARP Poisoning Security Application (Ryu-ARP), is a security application that runs on the RYU controller and provides a way for detecting and preventing the ARP poisoning attack in SDN.

Finally, in the era of SDN, the SDN controller is in charge of the majority of the information passing across the network. As a third thrust of this dissertation, we explored

the OpenDaylight (ODL) SDN controller's memory for forensically useful information. This was accomplished by creating controller memory samples with different networking configurations, analyzing the memory samples, and then constructing an SDN-Controller-Network-Discovery-Tool (SCoNDT). SCoNDT is a memory analysis tool that examines and analyzes the ODL controller's host tracker service from an acquired memory dump.

Chapter 1. Introduction

Technology has become so crucial today that it is used in every aspect of life. If we take a moment to think about what all technology is built on, we can see how networking affects every part of every kind of technology. Computer networking is a branch of engineering that focuses on the interconnection of computing devices such as servers, laptops, desktops, tablets, cellphones, and other devices. Sharing resources, working together effectively, and communicating efficiently are all improved through computer networking.

Networking, like other disciplines of technology, has several types of advancements and significant developments. One of the most recent innovations is software-defined networking (SDN), which takes networking to the next level in terms of performance, functionality, and capability.

1.1. Software Defined Networking

The concept of Software-Defined Networking (SDN) refers to a networking architecture in which software-based controllers are used to handle traffic on the network and maintain connectivity with network devices through application programming interfaces (APIs). The most important thing about SDN is that the control plane and the data forwarding plane are separated from the network's hardware architecture. This decoupling of control and data planes enables the adoption of the OpenFlow protocol and other more open protocols, which can allow network engineers to access network devices such as switches and routers that often employ proprietary and closed firmware. In a Software Defined Network (SDN), the activity of the overall network and its components is completely designed to be managed and controlled using programable software applications and

services and through open Application Programming Interfaces (APIs). SDN has evolved into one of the most popular methods for businesses to deploy network applications and services into their networks over time. SDN technology enables enterprises to implement software applications more quickly and at a lower total cost. Over time, the SDN idea has been seen as the next big thing in the networking industry. Moreover, SDN enables the network to connect and communicate with network applications via APIs; hence, this relationship between SDN and APIs enhances application security, management, and performance and helps create a scalable, dynamic network architecture. SDN creates a network environment that is dynamic and flexible enough to keep up with changing business needs and trends.

1.1.1. SDN vs Traditional Network

In a traditional network, network equipment combines the control and data planes. The control plane is primarily responsible for configuring the data flow paths and defining the network node settings. After determining the network routes by the control plane, they are directed to the data plane. The forwarding of data at the hardware level is determined by these control plane decisions. In this conventional method, after the forwarding rules have been created, the only method for modifying the flow policy is through modifications of the network device configuration. Figure 1.1 illustrates this evolution from traditional to SDN networking.

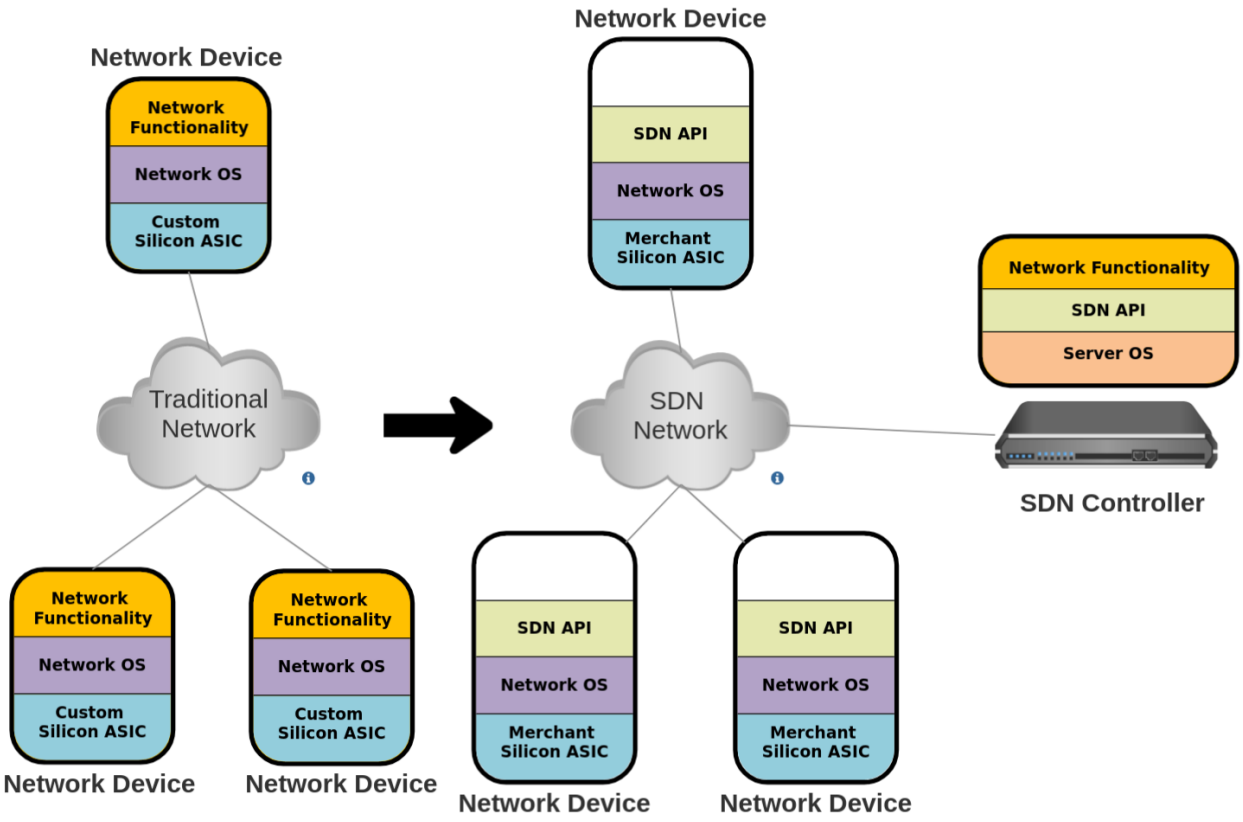

Figure 1.1. SDN Vs Traditional network Architecture

1.2. SDN Architecture

A basic SDN architecture design consists of three layers: the application layer, the control layer, and the infrastructure layer as shown in Figure 1.2. These layers communicate via northbound and southbound application programming interfaces (APIs).

1.2.1. The Application Layer

The application layer consists of SDN applications designed to execute a variety of functions, including enforcing security mechanisms and policies, conducting network management, and executing services on the SDN. This may include firewalls, load balancing, or Intrusion Detection Systems(IDS) [40].

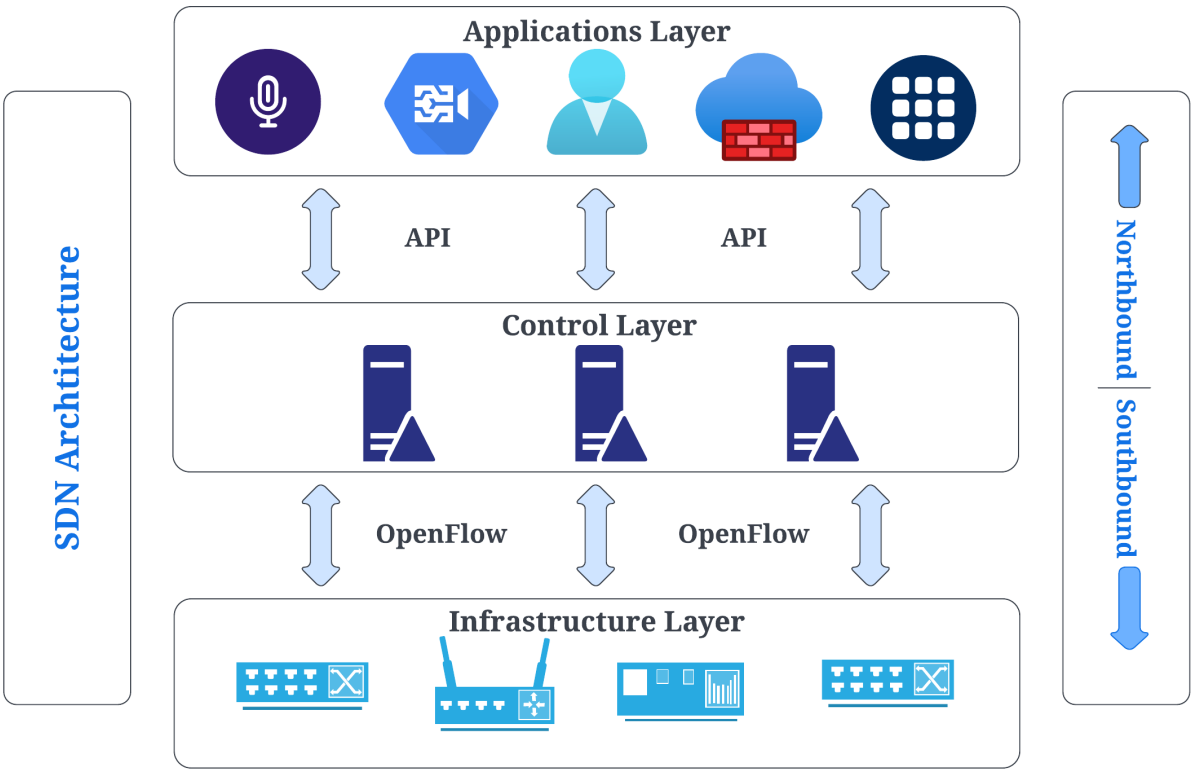

Figure 1.2. SDN Architecture

1.2.2. The Control Layer

The control plane contains the SDN controller. It represents the centralized SDN controller software that serves as the brain of the SDN. The controller manages the OpenFlow network equipment in the infrastructure layer and the SDN applications in the applications layer [97].

1.2.3. The Infrastructure Layer

The infrastructure layer consists of the network's OpenFlow hardware switches. These switches route network traffic to their respective destinations based on specific flow rules [39].

1.2.4. The OpenFlow Protocol

The OpenFlow protocol is a foundational element for building SDN solutions. Any device that intends to connect with an SDN controller in an OpenFlow environment must support the OpenFlow protocol. The communication format between the OpenFlow controller and OpenFlow device uses a secure channel [92]. This communication path is typically protected by TLS-based asymmetric encryption, while unencrypted TCP connections are also possible.

1.3. Research Objective

This research presents a set of techniques beginning with the development of a methodology for implementing a multi-RYU SDN controller architecture, which consists of a collection of open-source RYU controllers collaborating to achieve an effective level of performance, availability, and scalability against the threat of a single point of failure on the SDN's control plane against DDoS attacks. Next, we use the SDN controller and forensics tools to get information and digital evidence from the SDN network. We have also set up the process for getting the memory of the OpenDaylight SDN controller and built forensics tools for it. Lastly, we propose research to develop an SDN security application for the RYU controller to identify and prevent MITM attacks in the SDN. The objective of this work is three-fold:

- Commercial SDN controllers can work with the distributed controller design, but open-source SDN controllers can't. So, in our framework architecture, we showed how multiple open-source RYU SDN controllers could work together to make a highly available and reliable system against DDoS attacks.
- Develop a lightweight solution for the RYU SDN controller to detect and prevent the ARP spoofing MITM attack in the SDN environment.

- Analyze and inspect the SDN controller's acquired memory to help enable SDN controller forensics. In addition, create a Software-Defined Networking Controller Network Discovery Tool (SCoNDT) to analyze and pull data from the OpenDaylight SDN controller's acquired memory.

1.4. Contribution and Outline

This research has the following contributions:

- **Contribution 1:** Elimination of single points of failure in the control plane of the SDN, and enhancement of SDN network availability and scalability against the threat of DDoS attacks can be achieved by constructing the control plane with multiple open-source RYU SDN controllers and using these multiple RYU controllers that work together to increase the network's fault tolerance in case DDoS attacks hit the primary controller and to make the SDN network more reliable.

- **Contribution 2:** Developing a lightweight security application (Ryu-ARP) for the RYU SDN controller to detect and prevent MITM attacks, with a particular focus on the ARP spoofing MITM attack in the SDN environment.

- **Contribution 3:** In-depth research and the development of a forensics framework to provide the tool (SCoNDT) for analyzing and examining the OpenDaylight SDN controller's acquired memory, as well as the generation of an investigation report about the SDN at any required investigation stage.

1.4.1. Outline

This work's outline is as follows:

- Chapter 2 lists the related works that assisted in the organization of this dissertation. This chapter is based on previously published literature, which includes:
  - Security in SDN.
  - SDN Control Plane Security.
  - SDN Multi-controller Based.
  - The Man in the Middle (MITM) attack in SDN.
  - Digital Forensics of SDN.

- Chapter 3 presents the Multi-RYU Software-Defined Network Controller Architecture Against the DDoS Attacks. This work orchestrates the design and the methodology of multi-RYU controller collaboration to address the challenge of the single point of failure in the SDN control plane against DDoS attacks.

- Chapter 4 discusses the security application of detecting and preventing the ARP poisoning attacks in SDN by delivering an SDN security application that improves the RYU SDN controller's security and performance.

- Chapter 5 describes the Memory Forensics of The OpenDaylight SDN Controller by examining its acquired memory and delivering helpful information to investigators.

- Chapter 6 discusses the conclusion and the scope of this dissertation.

Chapter 2. Literature Review

2.1. Security in SDN

There are benefits and drawbacks to SDN, just as there are to any new technology. When it comes to security, for instance, SDN technology may be used to lessen the impact of, or perhaps completely eliminate, certain threats and vulnerabilities that are routinely exploited in traditional networks. However, the innovative design of SDN technology introduces it to new vulnerabilities and attack methods. Actually, a single point of failure that may be exploited to jeopardize an entire SDN network is exposed by the separation of control and data planes and the logical centralization of all network intelligence. in the SDN security survey papers, [26,56,84,88]The authors review the security issues confronted by SDNs across their different layers (application, control, and data). Possible attacks and threats because of these issues and some techniques for addressing, mitigating, and eliminating them are also provided by the authors.

The security of SDN is important to keep networks reliable. Without good security, any data moving through the network is at risk of being stolen or tampered with. One security flaw that was found in SDN is discussed by [108], where the authors exploit race conditions found in SDN in order to close the connection to switches, disrupt the services of the SDN, and block applications from receiving network events. It is worth noting that the exploits found by the authors have been now patched. [35] and [89] provide an overview of potential security risks that can occur in SDN.

2.2. DDoS Attacks against Control Plane

The authors [6] examine how the differences between SDN architecture and conventional networks interpret the security vulnerabilities in the SDN control plane and provide a methodology for comparative research that concentrates on the core network attributes required by traditional networks. This analysis provides a study of how these attributes are conveyed by the control planes of SDN and traditional networks, as well as a comparison of security threats and countermeasures. In this work [33], the authors provide an overview of DoS/DDoS threats and solutions in SDNs. Their study provides a comprehensive background on this issue, with an overview of Dos/DDoS attacks and the existing countermeasures. They classify current solutions in this space as either intrinsic or extrinsic, with intrinsic focusing on SDN components and their elements and extrinsic concentrating on network flows and their typical properties. This research [71] suggests using the centralized control of SDN for DDOS attack detection and provides a strategy based on the entropy fluctuation of the destination IP address and it can identify the attack within the first 500 packets of the attack flow. In [22], the authors provide a lightweight method for detecting DDoS attacks based on traffic flow characteristics, in which information is extracted with a very low overhead compared to conventional approaches. in [19], the study offers LineSwitch as a mitigation approach against control plane saturation attacks. LineSwitch combines SYN proxy methods with probabilistic network traffic blacklisting. AvantGuard [93] tries to overcome the saturation attack issue. AvantGuard proposes a module that implements a SYN proxy and only exposes flows that complete the TCP handshake. DoSGuard [60]proposes a module for SDN controllers that consists pri-

marily of three essential components: a monitor, a detector, and a mitigator. The monitor keeps information between network devices and hosts to identify anomalies. OpenFlow message and flow properties are used by the detector to detect the attack. The mitigator defends networks by screening malicious transmissions. Also, the authors in this work, provide a Moving Target Defense (MTD) [61] system designed to protect against Blind DDoS attacks. The method utilizes a pool of multiple controllers to handle the saturation issue, and it may dynamically switch the active controllers connected to switches based on the flood flow rate.

2.3. SDN Multi-controller Based against DDoS

In this publication [46], the authors provide a concise overview of multi-controller research in SDN. they provide an overview of multi-controller techniques, covering their development and the issues they present. they divide multi-controller research into four categories (scalability, consistency, reliability, and load balancing) based on the multi-controller implementation method. This paper [109]reviews the most recent research on SDN multiple controllers. The advantages and disadvantages of several controllers are examined. The detailed design ideas and topologies of SDN with multiple controllers are discussed in depth. Also, current research on the placement and scheduling of multiple controllers is thoroughly presented and evaluated. In DISCO [21], The authors propose a distributed SDN control plane for mission-critical networks. Each DISCO controller is in charge of an SDN domain and utilizes a controllable pub/sub technique to communicate combined local and network-wide data with neighboring SDN controllers. This work demonstrates a solution to network failures caused by DDoS attacks managed by the avail-

able distributed controllers. [36] provides an additional method of resiliency by providing a backup NOX controller in the case that the active controller fails. The authors provide a method for transferring existing SDN data to the new controller. In this study [42], the authors confront DDoS attacks by utilizing multiple controllers that communicate signals to one another while under attack. As such, some of the controllers can step in for other controllers that are currently under attack. In this paper [63], the authors introduce PATMOS, a protocol for preventing DDoS Attacks in SDN networks with multiple controllers by use of controller clustering. PATMOS recognizes and then removes controllers that are overloaded. Then, it selects the controller with the best performance to maintain the mitigation process and reorganizes the controller clustering to tolerate DDoS attacks. K-Critical [51], an algorithm developed by the authors, strategically positions controllers to ensure stable operation. K-Critical finds the bare minimum of controllers and their placements to build a fault-tolerant control architecture that distributes load fairly among the chosen controllers. In order to maximize the control plane's dependability [87], the authors combine the Greedy technique with simulated annealing to choose the best candidates for controller nodes. The goal of the provided optimization issue is to reduce the connection distance and delay between switches and controllers. In this paper [101], the authors combine multiple SDN controllers with Redis and Zookeeper to provide a scalable and fault-tolerant SDN control layer but they did not use any kind of attack on the control plane. Based on the architectural methods against DDoS attacks presented in the aforementioned papers, and to the best of our knowledge, our design against DDoS attacks using the RYU multi-controller is the first of its kind and provides an architectural solution for the SDN's control plane that uses the RYU controllers against the DDoS attacks by implementing

multiple RYU controllers in combination with Zookeeper and Redis to ensure the reliability of the SDN operations under the DDoS attacks.

2.4. ARP Poisoning in SDN

ARP Cache Poisoning defenses developed for traditional network are offered for the SDN environment. Dynamic ARP Inspection (DAI), Secure ARP (S-ARP), and Static setup of ARP mappings have given solutions for ARP poisoning attacks in traditional networks, which may also be used to the SDN architecture. All of them have limitations, such as the fact that they are not scalable, increase network overhead, or are sometimes incompatible with the SDN's nature and degrade its performance [90]. The SARP-NAT technique is proposed in this paper [14]. Maintaining a list of hosts that have submitted an ARP request message makes ARP stateful. When a host submits an ARP request, the request is initially forwarded to the POX Controller where the SARP-NAT component is executing. SARP-NAT is able to identify ARP poisoning attacks based on the list matching. Scalable Ethernet traffic Architecture using SDN (SEASDN) is another method offered in this research [48]. This module enables the POX Controller to establish IP and MAC address mappings using DHCP packets exchanged between hosts and the DHCP server. Another IP-MAC address binding-based method is L3-ARPSEC [79]. A solution module is deployed to the POX Controller. This module, like SEASDN, keeps track of the IP and MAC addresses bound to the hosts on the network. In contrast to SEASDN, which generates its table by listening to DHCP messages, this one is constructed on the fly whenever an ARP request message is received. This work [76] provides an approach for the verification and detection of ARP-based attacks, titled Traffic Pattern-Based Solution to ARP-

Related Attacks (FICUR).

In the suggested solution, a customized SDN controller collects the necessary network characteristics and analyzes them to verify and identify attacks. In addition, mitigation is conducted in real-time using SDN capabilities. This study [67]presented a design approach for a new algorithm to combat ARP poisoning attacks by employing certain SDN-based properties. Their algorithm consists of the SDN DYN and SDN STA scenarios. The technique takes into account both dynamic and static IP address assignments and validates IP addresses. The Network Flow Guard for ARP (NFGA) [30] is a security module for software-defined networks (SDNs) that enhances OpenFlow-enabled switches' standard MAC learning protocols. By real-time hashing of a host's physical address with an appropriate IP:port match, NFG eliminates ARP spoofing. According to the paper's authors, this method allows for dynamic and static port allocations, doesn't change the network's architecture or protocols, and doesn't necessitate installing client software. [13]provided a security framework that minimizes host discovery messages to prevent ARP and DHCP attacks.

Using machine learning (ML) techniques, the SDN has different methods for preventing the ARP Poisoning attack. The authors of this paper [62] offer the Bayes approach for estimating the probability of an attack. Several ML techniques were utilized for attack detection according to the probability. Also,in this study [10], the authors presented a machine learning (ML) approach that captures and logs the attack-detection-required information into a file known as a traffic dataset. This data is used to build the machine learning model and to detect ARP poisoning attacks. All of the non-SDN-based solutions to the ARP Cache Poisoning attack have been analyzed in detail by the authors

of these works [28], [44], [82] and [45].

2.5. Digital Forensics of SDN

This paper [105] presented ForenGuard, which monitors and records runtime actions as well as their causal linkages involving both the SDN control plane and the data plan. ForenGuard can identify the underlying cause of a problem by retracing earlier activity in both the control and data planes using causal links. ForenGuard is compatible with Floodlight and provides an intuitive interface that enables users to choose the detection point and analyze complex network issues. Based on [7], we may utilize the scientifically-proven tools, techniques, and procedures of digital forensics to locate and analyze digital artifacts. for the purpose of preserving potential evidence in SDN, [73] presented a Digital Forensic Readiness method. In fact, the aim of their study was on creating a proactive method for the identification, management, collecting, and preservation of digital artifacts in SDN. Also, [59] proposed a proactive DFR framework for SDN with a trigger-based automatic collection system. Their suggested methodology combined an intrusion detection system with an SDN controller.

Some challenges related to conducting network forensics in a virtualized environment were explained by [95], such as the potential for multiple VMs in the same network sharing a single MAC address. SDN security with a comparison to Cloud Security and Network Security are expressed by [106]. Current advantages and challenges relating to SDN forensics are also explained, and suggestions for future research directions are made. [80] provided a description of methods employed to acquire artifacts from SDN. They posited that the most feasible method is to collect data from the southbound interface of

the controller. This is the interface between the controller and the hardware of the network, such as routers. A prototype was successfully implemented for the OpenDaylight controller. However, the prototype must be further developed to work with other controllers such as Floodlight and Ryu. While the information that goes through the southbound interface is collected, the authors did not conduct any memory forensics of the SDN controller. [41] explained the concept of active security, and how it can be used for SDN. The work discussed multiple security techniques, with one component being digital forensics. They developed a prototype system to provide active security that works with the Floodlight SDN controller. In this prototype, host devices automatically load the LME if they are discovered to be infected.

As for other research related to our work, [18] talked about a mechanism to automate the extraction of network artifacts from SDN and [53] described the Mininet we used to simulate a network in our work. Mininet is an open-source network emulator, that allows users to create virtual networks without the need for networking hardware.

2.5.1. Memory Forensics

Memory Forensics entails analyzing the RAM of a computer, also known as the volatile memory. A memory dump is acquired by copying the RAM of a computer. By analyzing the memory dump, we can uncover what a device was doing when the memory was acquired. Memory forensics can also reveal information that was never written to the disk and can be found exclusively in memory. [102] explained the main requirements of a forensically sound memory acquisition involve correctness, atomicity, and integrity. Atomicity refers to the idea that a single operation in memory such as a read or

write is executed in one indivisible action, without interruption. This ensures that the memory state keeps its correctness and integrity without corruption. Post-acquisition, the next step in memory forensics involves analysis. [78] outlined the processes of memory analysis methods. Currently, the primary tool used for analyzing memory is Volatility https://github.com/volatilityfoundation/volatility. To search for malware signatures or patterns in memory, researchers have employed the YARA library and scanner [27]. YARA helps in the identification and classification of malware, malicious files, or patterns using user-defined rules. Automatically search the memory for malware signatures. While it is primarily used to scan for malware, Yara signatures may be used to search for any patterns in memory.

Other work focused on the memory of mobile phones [96]. The authors managed to retrieve outgoing and incoming messages from the memory of the phone a short time after the messages were sent. Other works explored the memory forensics of VR devices [25], cyptocurrency hardware wallets [99], USB attack platforms [98], the V8 JS engine [104] and .NET core applications [66].

[24] described the state of the field of memory forensics and the challenges that lay ahead. While there have been incremental research, many of the challenges related to memory forensics still exist. One of the unexplored areas in memory forensics is SDN memory forensics. While little work exists in SDN forensics, there has been prior work published in the domain of network forensics. When talking about application specific analysis, it is mentioned that memory forensics is gaining popularity in all types of investigations to obtain information not possible through network or disk forensics. It is also mentioned that the memory forensics of IoT devices may also be feasible. [17] compared

the two methods, live response and memory imaging, used to acquire a memory dump.

2.5.2. Network Forensics

Network forensics focuses on extracting and analyzing network interaction data. Prior work constructed a network forensics framework and described basic components of network forensics [81]. The main purpose of network forensics is described as tracking down the attacker of a network, with the goal to prosecute them in a court of law. [38] discussed and compared four main network forensics tools: Xplico, OmniPeek, NetDetector, and NetIetercept. [69] described additional tools and the common techniques used in network forensics such as packet and email sniffing, web forensics, and IP trace-back techniques. Packet sniffing involves packet reconstruction, a network forensic technique that provides insight into a network's activities. Analyzing packets may reveal images, messages, and files that were sent over a network [94].

When collecting network forensic artifacts, a lot of insignificant data will be collected. [72] discussed network forensic analysis using AI techniques to make sorting the data simpler. The author's goal was to use AI to collect key data while keeping non-relevant information confidential. [12] explored a memory forensics approach to network investigations. They showed how to find a large number of connections made by other devices in a device's memory. One hour after the attack occurred, a large number of malicious packets were still present in the memory of Windows and Linux devices. [9] analyzed the instant messaging application WhatsApp with the goal of extracting the IP address of any device that sent a WhatsApp message. They achieved this by analyzing network traffic. Other researchers were able to decrypt the WhatsApp signaling protocol

for its voice calling feature [52]. In [103], the authors analyzed over twenty Android social messaging applications, and they were able to reconstruct partial and full messages from network traffic. Popular video conferencing applications such as WebEx and Zoom were also analyzed using network, disk, and memory forensics [55, 64]. While past work specifically on SDN controller memory forensics did not exist, prior work did explore forensics and security concepts in SDN.

Chapter 3. Multi-RYU SDN Controller Architectural-Based Solution Against the DDoS Attacks

The Internet is becoming a highly intricate backbone for the ever-developing information technologies that have a significant effect on people's daily lives. However, administrators face challenges in operating and managing a dynamic, complex network due to the constraints imposed by traditional network architecture. So, to promote network evolution, programmable SDN was been proposed. It has the potential to greatly improve network administration and pave the way for future developments. The traditional network architecture is split into a centralized control plane and a programmable data plane. This makes it easier to design, monitor, and run networks in the future. SDN promises to make it much easier to manage networks and to encourage new ideas and the creation of network applications. However, Security, reliability, load balancing, and traffic engineering are just some of the issues brought up by SDN's separation of the network's control and data planes. [34] As the SDN controller gives a centralized logical perspective of the entire network and maintains and configures all network devices, it also poses a potential security risk. Because of this, the status of the network as a whole depends on the controller being operational. In a communications network, this is a highly undesirable situation that exists.

DDoS attacks present significant security challenges for SDNs [91]. The SDN controllers can identify an attack due to the network's programmability and centralized awareness of the entire network's topology, but the centralized control architecture is seen as more vulnerable. As a result, the SDN paradigm is likely to be attacked via DDoS. In general, DDoS attacks potentially overwhelm the infrastructure layer, the control layer, or

the communication channel. But an attack on the control layer might result in the complete failure of the SDN services. This study focuses on control plane security challenges, particularly against DDoS attacks targeted at the SDN controller. One of the key goals of SDN architecture was to provide a centralized management and control structure. Thus, a stand-alone single controller was adopted by the vast majority of SDN Controllers. This caused problems with scalability and single points of failure. SDN Controller's distributed capability allows the deployment of multiple controllers operating in a flat or hierarchical structure inside a domain. However, not all open-source SDN controllers, including our main SDN controller in this study (RYU), can support multiple controller designs [110]. So, we proposed a technique utilizing multiple RYU SDN controllers to mitigate and counteract DDoS attacks on the control layer.

The **contribution of this research** includes:

- Elimination of single points of failure in the control plane of the SDN, and enhancement of SDN network availability and scalability against the threat of DDoS attacks by constructing the control plane with multiple open-source RYU SDN controllers.
- Proposing a multi-controller architectural design for an open-source SDN controller RYU that only supports centralized distributions.
- Implement multiple cooperative RYU controllers to increase the network's fault tolerance in the case of DDoS attacks against the primary controller and thereby improve the SDN network's reliability.

The rest of the research is as follows: Section 3.1 presents the background of this research; Section 3.2 provides an overview of our design and implementation; Section 3.3 presents the evaluation of the proposed Multi- RYU controller architecture against DDoS attacks; Section 3.4 summarizes our findings.

3.1. Background

The implementation of SDN should make network administration, innovation, and application development simpler. The SDN is centralized, and there is currently no reliable way to keep open-source SDN controllers working together using their features. As a result, SDN is struggling to guarantee the availability and dependability of its services with the existence of a single point of failure in its control plane. If the SDN controller goes down, the entire network will be down [74]. In this study, we show how to set up multiple open-source RYU-SDN controllers' architectures that work together to provide high performance, availability, and scalability, even though the SDN's control plane could have a single point of failure. Distributed controller capability is only available in commercial SDN controllers, not open-source ones. We came up with a framework architecture that lets many open-source SDN controllers work together so that the network is always available and reliable against DDoS attack. We're attempting to reduce the danger posed by a single SDN controller, which is often the weak spot in SDN networks where DDoS attacks happen.

3.1.1. The SDN Controller

The SDN controller is the foundation of the SDN paradigm. It is the software that coordinates the allocation and utilization of all of the network's fundamental resources. SDN controllers minimize the need for manual settings of each network device by directing traffic based on the most recent forwarding rules established by a network administrator. Programmable network administration is made possible by the centralized controller, which removes the control plane from the networking equipment and runs it as

software [65]. This simplifies the process of integrating and administering technology solutions. To some extent, the SDN controller may be thought of as a network Operating system. The brains of an SDN lie in its controller. So, the controller is the central point of contact for all data exchanges between applications and network nodes. There are two types of SDN controllers: open-source SDN controllers and commercial SDN controllers. Open-source SDN controllers such as RYU, POX, ONOS, and Floodlight. The commercial SDN controller market includes companies like Cisco, HPE, Juniper Networks, and Pica8. We focused our work on the RYU controller [20].

3.2. Methodology and System Design

There is a single point of failure in SDN because of the centralized nature of its control architecture [58]. To address this issue, we use a multi-controller architecture to guarantee a constant delivery of services in the case of DDoS attacks or other disruptive interference against the primary RYU controller. We rely on a single controller to run the entire network and we call it the Primary controller. The remaining controllers are referred to as Standby controllers and are viable substitutes for the Primary controller. The main advantage of the Standby controllers is that they are ready to take over the function of the primary one if the Primary controller fails.

3.2.1. Apparatus

This section outlines the software and hardware used in our system design. Table 5.2 provides information on the computer workstation and software used to create and run this SDN design. VMware was used with Linux Ubuntu 20.04.4 LTS to build and run the complete system. In addition, we used Mininet to simulate the whole network architec-

Table 3.1. Workstation and Software Details

System Details		Software Details	
Device	**Details**	**Software Name**	**Version**
Processor	Intel(R) Core(TM) i7-8550U CPU @1.99 GHz	VMware	Ubuntu 20.04.4 LTS
System Type	64-bit operating system, x64-based processor	Mininet	2.3.0
Graphics Card	NVDIA GeForce MX150	Apache ZooKeeper	3.7.1
Manufacturer	HP Spectre x360	Redis	6.2
Memory (RAM)	16.0 GB	Ryu	4.34
Raspberry Pi3	64-bit quad-core ARM Cortex-A53, 1GB RAM	OpenFlow	1.1, 1.2, 1.3
Raspberry Pi4	Broadcom,quad-core Cortex-A72 64-bit, 2GB RAM	Kali-Linux	2022.3

ture. We selected the Ryu controller as the SDN controller. Also, Redis was employed as a database and message broker, while Apache ZooKeeper was used for distributed synchronization. The Linux 5.18.0-kali5-amd64 machine, together with the hping3 tool, was used to simulate Distributed Denial of Service (DDoS) attacks. In one testing topology, we employ two Raspberry Pi3 as Open vSwitchs and four Raspberry Pi4 as hosts. We used Kali Linux 2022.3 with Metasploit tools for the DDoS simulation.

3.2.2. Benefits and Challenges of the design

Here, we highlighted the advantage of our architecture to circumvent the issue of being vulnerable to DDoS attacks with a single controller in operations. Therefore, with a multiple-controller architecture, we provide high availability of SDN services by achieving fault tolerance and increasing stability, since the network will continue providing its services even if the primary controller or link to the Primary controller fails.

On the other hand, this design presents additional challenges and issues that must be addressed in order to achieve the aforementioned design's advantages. In general, Multi SDN controller design faced issues with consistency, scheduling, and placement approaches for controllers. In our design, we prioritize the consistency and availability of SDN network operations in the event of any sort of DDoS attack.

3.2.3. Design Overview

In this subsection, we discuss the multi-SDN controller architecture against DDoS attacks and explain each solution component in depth.

RYU Multi-Controller Architecture

The Ryu Controller is an open-source SDN Controller that supports traffic management and allows for rapid adaptation of innovative network control and development. Ryu is compatible with a wide variety of network management protocols. Ryu has complete compatibility with OpenFlow versions 1.0, 1.2, 1.3, 1.4, 1.5, and Nicira Extensions. In addition, Ryu's OpenFlow operations may be accessed using a REST interface [86]. We chose the Ryu controller as it is a centralized, open-source controller that does not support multi-controller architecture on its own. Additionally, the RYU controller does not provide clustering capabilities. So, In order to make multiple RYU controllers operate together in the control plane of the SDN, we employ Apache Zookeeper capabilities and Redis as part of our multi-RYU controller design. This enables the RYU controllers to establish a highly available system with multiple RYU controllers servicing the SDN.

As a starting point for our design, we built the SDN environment where switches and routers in the network are all managed by the same single controller. We've designated this controller as the "primary" controller. We included multiple RYU controllers in the design to ensure there is no single point of failure. This implies that the primary RYU controller has full control of the entire network, while the secondary RYU controllers function as Standby controllers that are connected to our design and receive updates of the flow tables from Redis, as well as contribute to the ZooKeeper elections process in the

case that the primary controller fails. Control messages, such as the handshake, switch configuration, modify state messages, and others, are exchanged between the network devices and the Primary controller. On the other hand, the Standby controller only SYN and keepalive messages are sent and received by it. But again, it is not able to transmit or receive control messages to or from the data plane devices. The connectivity of the RYU multi-controller design's control plane components to each other and to the data plane components is illustrated in figure 3.1. There should be only one primary controller for all data plane devices, but multiple standby controllers may be configured as backups.

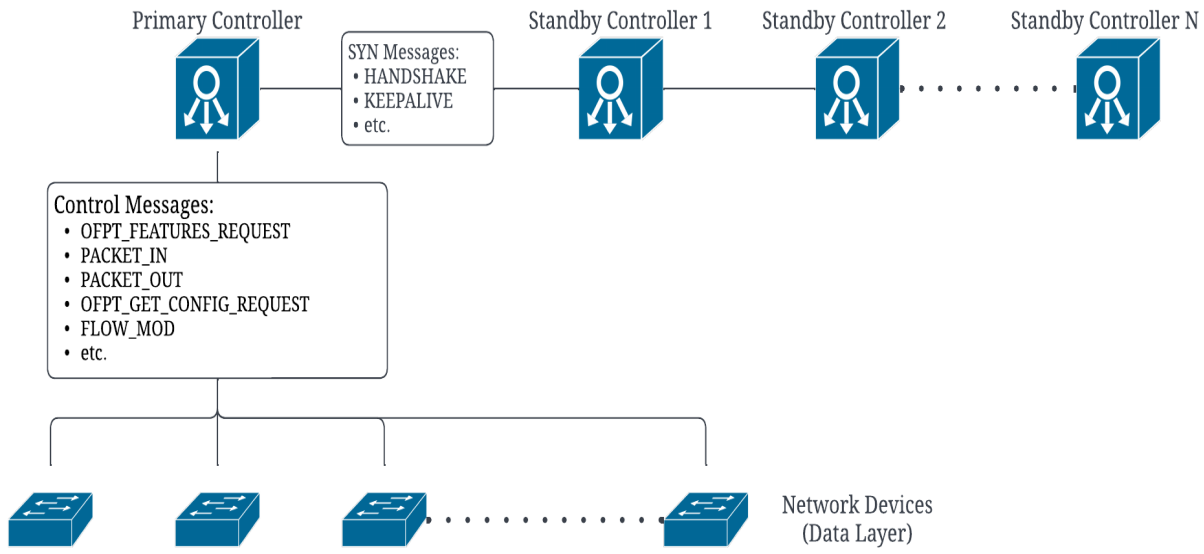

Figure 3.1. RYU Multi-controller roles and messages

In order to control and maintain this RYU multi-controller structure, we rely on Apache Zookeeper to handle the election of a replacement primary controller in the case that the currently active controller goes down. Further, all of the controllers share network data using the Redis data store services.

Apache ZooKeeper

ZooKeeper is an open-source distributed application coordination solution. Distributed applications may expand on their bare-bones set of primitives to provide more sophisticated features like group and name management, configuration management, and synchronization. ZooKeeper offers a simple but effective coordination service through a readily available interface that combines the wait-free characteristics of shared registers with an event-driven approach analogous to distributed file cache invalidation. Through the ZooKeeper interface, high-performance service implementation is feasible. In addition to its wait-free nature, ZooKeeper ensures FIFO request processing for each client and linearizability for all requests that affect ZooKeeper's state [111].

Zookeeper plays a crucial role in our design as a coordination service to coordinate our multiple controllers by maintaining and choosing the primary controller for the SDN architecture. Once connected to the Zookeeper services, the process of managing and selecting the primary controller among the available multiple controllers may begin. Algorithm 1 depicts the primary controller election procedure.

Establishing a persistent Znode along the path "election" is a need. Following that, at the initialization phase, each controller will construct an ephemeral and sequential Znode under the persistent Znode in the election path "/election". ZooKeeper will append a numeric value to the name of each controller Znode formed during the election algorithm execution. The primary controller will be selected by this numbering procedure, with the lowest numeric value among all controllers being elected as the primary controller. ZooKeeper will locate all controller Znodes from the "/election" Znode's path and

Algorithm 1: The Primary Controller Election

```
1  Input: All Controllers
2  Output: The Primary Controller
3  if CurrentRole == NULL then
4  |  while CurrentRole == NULL do
5  |  |  Multicast Current.Priority
6  |  |  check FailedController.priority
7  |  end
8  |  if !Primary||Current.Priority < FailedController.priority then
9  |  |  CurrentRole ←Primary
10 |  else
11 |  |  CurrentRole ←Standby
12 |  end
13 |  if CurrentRole == Primary then
14 |  |  while CurrentRole == Primary do
15 |  |  |  multicast heartbeat packets
16 |  |  |  Listen to newPrimary.priority
17 |  |  end
18 |  |  if Received New.Primary.priority then
19 |  |  |  Send New.Primary.priority
20 |  |  end
21 |  |  if Current.Priority > newPrimary.priority then
22 |  |  |  CurrentRole ← Standby
23 |  |  else
24 |  |  |  if CurrentRole == Primary then
25 |  |  |  |  Listen to heartbeat packets
26 |  |  |  end
27 |  |  end
28 |  |  if heartbeat == NULL then
29 |  |  |  CurrentRole == NULL
30 |  |  end
31 |  end
32 end
```

check for the controller Znode with the lowest numeric value. The election procedure will make the current controller the primary controller if the value of the controller Znode is the same or lower than the value of the controller Znode being used. If not, the procedure for determining the primary controller will proceed by looking for the controller with the lowest numerical value among all available controllers in the SDN environment. The cur-

rent process receives a notification from ZooKeeper when the previously elected ephemeral controller Znode is deleted.

The current process then retrieves all the possible controller Znodes in the path of the election "/election" and repeats the processes to choose a new primary controller or maintain the current one [83].

Redis Failover

Redis is an open-source software in-memory database, cache, message broker, and streaming engine. Redis supports a wide variety of data structures, including strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams. Redis offers high availability through Redis Sentinel and automated partitioning with Redis Cluster in addition to its built-in replication, Lua scripting, LRU eviction, transactions, and various degrees of on-disk storage [4]. The Redis failover system provides a fully-featured, fully-automatic primary/standby failover configuration.

While ZooKeeper excels at reading data, writing data is an extremely slow process. Additionally, Znode data is limited in size. Redis, on the other hand, has a bigger data size and is faster while reading and writing. Based on the previous characteristics of ZooKeeper and Redis, We utilize both ZooKeeper and Redis in collaboration to provide a high degree of performance, availability, and reliability in our architecture. So, in our design, the Zookeeper service is in charge of keeping track of and coordinating all the controllers in the control plane. It also keeps track of which controllers are working and runs the process of choosing the primary controller from the multiple controllers that are available. On the other hand, Redis is used for database services that need to write in-

formation faster about SDN topology and network forwarding. Figure 3.2 demonstrates the complete architecture of the Multi SDN controller with Zookeeper and Redis against DDoS attacks.

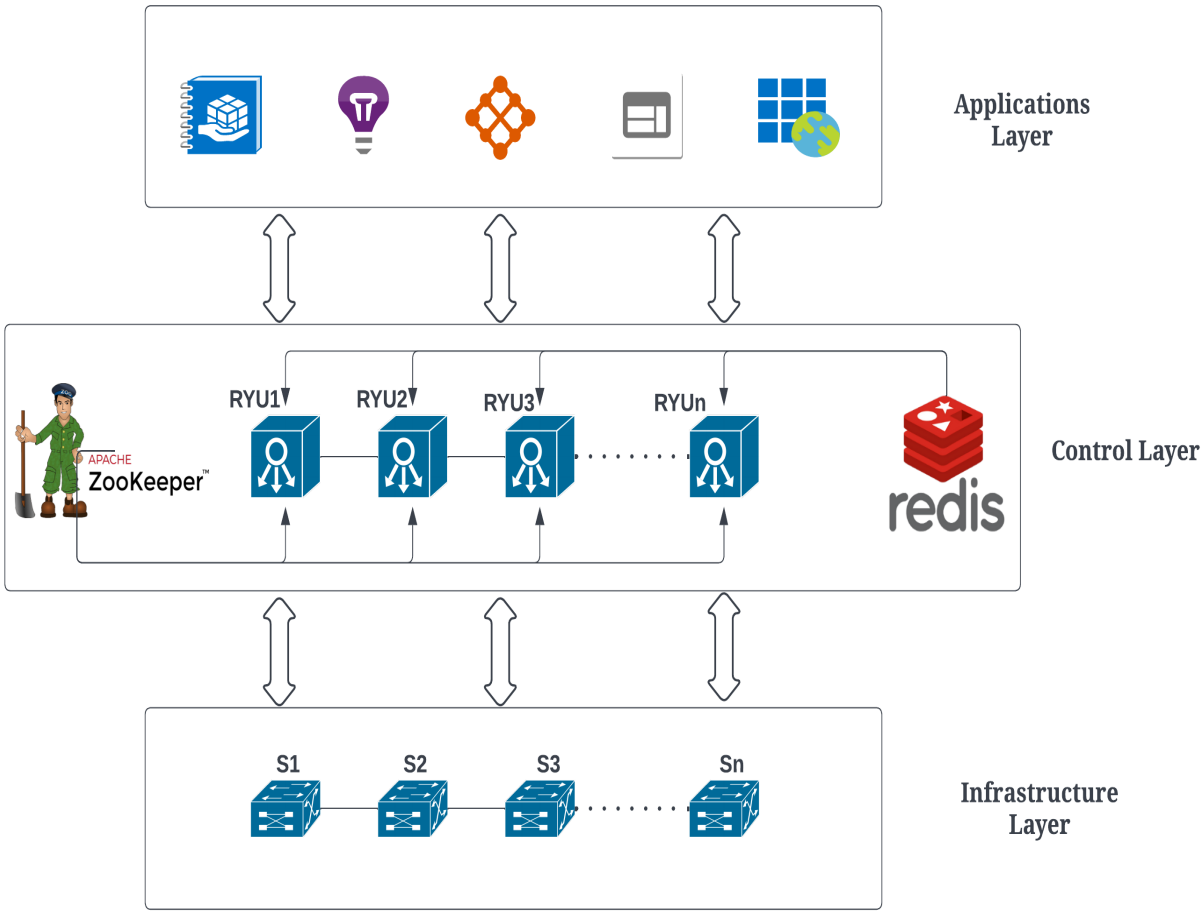

Figure 3.2. Multi-Controller SDN Design with Zookeeper and Redis

3.3. Evaluation

3.3.1. Implementation

In our design, we use two different implementations. The first implementation is based on attempting to utilize a Raspberry Pi3 with OpenvSwitch capabilities, together with four Raspberry Pi4 as network hosts, to establish a hardware environment for the software-defined networking as shown in Figure 3.3. The second implementation uses Mininet's capabilities to create SDN network topologies that are more subject to accommodating complex and challenging SDN layouts for our evaluations of the design.

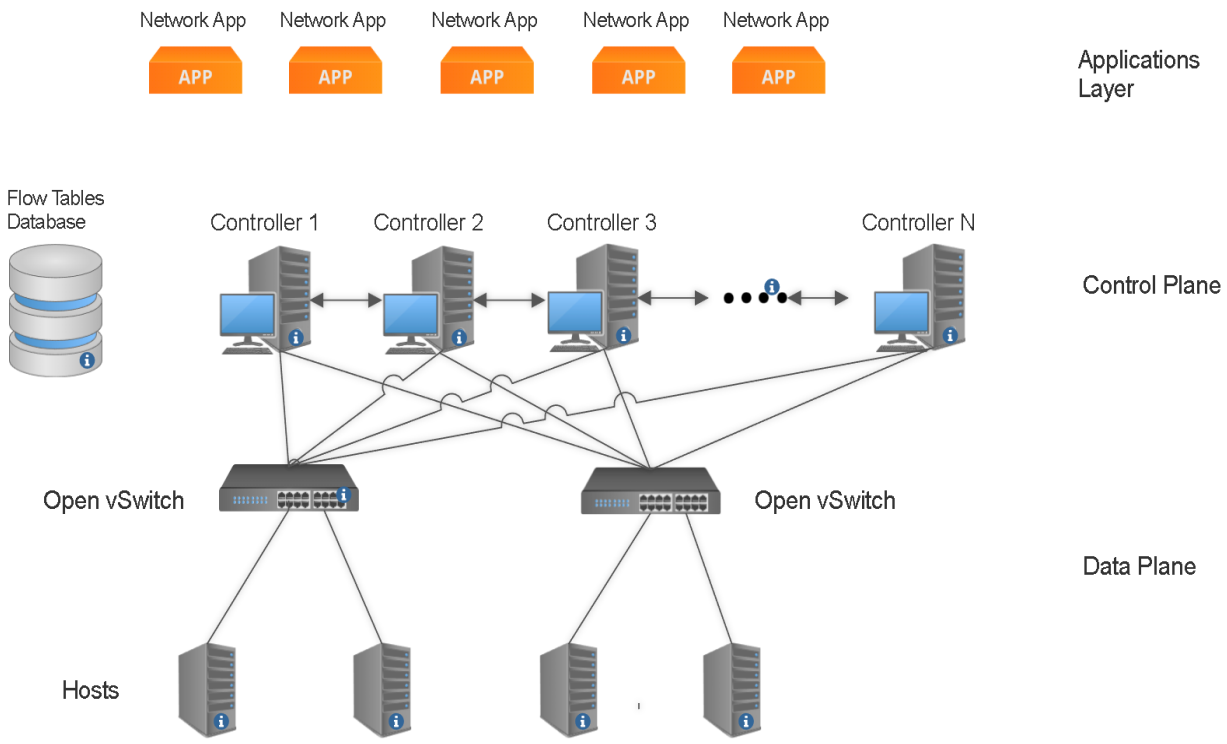

Figure 3.3. Multi-RYU Controllers Design with OpenvSwitch Raspberry Pi

3.3.2. DDoS Attacks Results

In this section, we conducted two different types of tests for our two different implementations. The first test evaluates the performance of our multi-RYU controller's design during normal operation, with no DDoS attacks targeted at the primary controller. The second test is being performed during the DDoS attacks.

Hardware Implementation Results

Using the hardware implementation in Figure 3.3, we use four RYU controllers (one primary controller and three standby controllers) in the control plane. The goal is to put our approach to the test in a setting with functionally equivalent Open Flow network devices with the multi-RYU Controller design.

The D-ITG was used as an analytical and evaluation tool in our research. D-ITG, which stands for "Distributed Internet Traffic Generator," is a system that can create traffic at the packet level by accurately replicating performance characteristics like IDT (Inter Departure Time), PS (Packet Size), and variables like exponential and uniform.

Based on D-ITG results, An example of a normal traffic flow in our SDN environment is illustrated in Figure 3.4. There are a total of 134445 packets transmitted and received throughout the duration of 60 seconds. We found that the longest delay was 0.145938 seconds and the average delay was 0.000745 seconds. Importantly, no data is lost in this normal course of network traffic between devices.

The next type of test involves DDoS (distributed denial of service) attacks. During these DDoS attacks, several machines flood the primary controller with false or fake traffic. And therefore, the controller will get overwhelmed by requests and unable to support

```
|-----------------------------------------------------
Flow number: 1
From 10.1.1.1:54982
To    10.1.1.4:8999
-----------------------------------------------------
Total time                =     59.998956 s
Total packets             =        134445
Minimum delay             =      0.000007 s
Maximum delay             =      0.145938 s
Average delay             =      0.000745 s
Average jitter            =      0.000058 s
Delay standard deviation  =      0.006992 s
Bytes received            =      68835840
Average bitrate           =   9178.271702 Kbit/s
Average packet rate       =   2240.788990 pkt/s
Packets dropped           =              0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
-----------------------------------------------------

*****************  TOTAL RESULTS   ********************

Number of flows           =              1
Total time                =     59.998956 s
Total packets             =        134445
Minimum delay             =      0.000007 s
Maximum delay             =      0.145938 s
Average delay             =      0.000745 s
Average jitter            =      0.000058 s
Delay standard deviation  =      0.006992 s
Bytes received            =      68835840
Average bitrate           =   9178.271702 Kbit/s
Average packet rate       =   2240.788990 pkt/s
Packets dropped           =              0 (0.00 %)
Average loss-burst size   =              0 pkt
Error lines               =              0
-----------------------------------------------------
```

Figure 3.4. Normal Test Traffic (HW Implementation)

network operations.

As shown in Figure 3.5, when our multi-RYU controller design is subjected to DDoS attacks, we have observed the following:

- The magnitude of the impact on the performance of the SDN network during the DDoS attacks is shown by the fact that the maximum transmission delay increased from 0.145938 sec to 0.260879 sec under the same circumstances and with the same type of network traffic.

- With compared to the normal traffic conditions when there is no DDoS attack on the network, the number of packets transmitted while our SDN architecture is under attack drops from 134445 to 125877 total packets.

```
|------------------------------------------------------------
Flow number: 1
From 10.1.1.1:58446
To    10.1.1.4:8999
------------------------------------------------------------
Total time                     =     59.999152 s
Total packets                  =        125877
Minimum delay                  =      0.000011 s
Maximum delay                  =      0.260879 s
Average delay                  =      0.002424 s
Average jitter                 =      0.000059 s
Delay standard deviation       =      0.019925 s
Bytes received                 =      64449024
Average bitrate                =   8593.324652 Kbit/s
Average packet rate            =   2097.979651 pkt/s
Packets dropped                =              0 (0.00 %)
Average loss-burst size        =      0.000000 pkt
------------------------------------------------------------


****************  TOTAL RESULTS    ******************

Number of flows                =              1
Total time                     =     59.999152 s
Total packets                  =        125877
Minimum delay                  =      0.000011 s
Maximum delay                  =      0.260879 s
Average delay                  =      0.002424 s
Average jitter                 =      0.000059 s
Delay standard deviation       =      0.019925 s
Bytes received                 =      64449024
Average bitrate                =   8593.324652 Kbit/s
Average packet rate            =   2097.979651 pkt/s
Packets dropped                =              0 (0.00 %)
Average loss-burst size        =              0 pkt
Error lines                    =              0
------------------------------------------------------------
```

Figure 3.5. Traffic During the DDoS Attack (HW Implementation)

- Normally, the average packet rate is 2240.79 pkt/s, however during the DDoS attack, it plummeted to 2097.98 pkt/s.

- No packets were dropped during the DDoS attacks, according to the results gathered from the SDN network.

Mininet Implementation Results

In this implementation, the Mininet is used to create an SDN topology with four RYU controllers (one primary controller and three standby controllers). The topology of the SDN is a tree with a depth of five. So, we have 31 switches and 32 hosts in the SDN simulation.

The normal traffic in our Mininet setup is depicted in Figure 3.6. D-ITG found that throughout the 60-second time period, a total of 157884 packets were transmitted and received. Delays can range from 0.000008 to 0.942808 seconds, with the average being 0.006689 seconds. It's crucial to note that during this normal operation of the Mininet simulation and transmission packets between hosts, no information is ever lost.

```
/------------------------------------------------------------
Flow number: 1
From 10.1.1.1:40118
To    10.1.1.32:8999
------------------------------------------------------------
Total time                =     59.998987 s
Total packets             =        157884
Minimum delay             =      0.000008 s
Maximum delay             =      0.942808 s
Average delay             =      0.006689 s
Average jitter            =      0.000095 s
Delay standard deviation  =      0.071899 s
Bytes received            =      80836608
Average bitrate           =  10778.396375 Kbit/s
Average packet rate       =   2631.444428 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
------------------------------------------------------------

____________________________________________________________
****************  TOTAL RESULTS    ******************
____________________________________________________________
Number of flows           =             1
Total time                =     59.998987 s
Total packets             =        157884
Minimum delay             =      0.000008 s
Maximum delay             =      0.942808 s
Average delay             =      0.006689 s
Average jitter            =      0.000095 s
Delay standard deviation  =      0.071899 s
Bytes received            =      80836608
Average bitrate           =  10778.396375 Kbit/s
Average packet rate       =   2631.444428 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =             0 pkt
Error lines               =             0
------------------------------------------------------------
```

Figure 3.6. Normal Test Traffic (Mininet Implementation)

```
------------------------------------------------------------
Flow number: 1
From 10.1.1.1:40290
To    10.1.1.32:8999
------------------------------------------------------------
Total time                =     59.999258 s
Total packets             =        143319
Minimum delay             =      0.000013 s
Maximum delay             =      1.966813 s
Average delay             =      0.007989 s
Average jitter            =      0.000118 s
Delay standard deviation  =      0.103261 s
Bytes received            =      73379328
Average bitrate           =   9784.031396 Kbit/s
Average packet rate       =   2388.679540 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =      0.000000 pkt
------------------------------------------------------------

****************  TOTAL RESULTS    ******************
Number of flows           =             1
Total time                =     59.999258 s
Total packets             =        143319
Minimum delay             =      0.000013 s
Maximum delay             =      1.966813 s
Average delay             =      0.007989 s
Average jitter            =      0.000118 s
Delay standard deviation  =      0.103261 s
Bytes received            =      73379328
Average bitrate           =   9784.031396 Kbit/s
Average packet rate       =   2388.679540 pkt/s
Packets dropped           =             0 (0.00 %)
Average loss-burst size   =             0 pkt
Error lines               =             0
```

Figure 3.7. Traffic During the DDoS Attack (Mininet Implementation)

However, throughout the DDoS attacks, a total of 143319 packets were transmitted. The smallest possible delay time is now 0.000013 seconds, while the largest is now 1.966813 seconds. As a whole, the average latency is also 0.007989 seconds. The fact that no data has been lost due to the primary controller's failover problem during the DDoS attacks is further proof that the system is working as intended (see Figure 3.7). The average packet rate decreased during the failover by 9.22%, from 2631.4 packets per second to 2388.7 packets per second. This may indicate that the design's performance decreased by around 9%.

3.3.3. Restore Time

Restore time of the primary RYU controller in the network when it fails due to DDoS attacks, relative to the size of the attack, is one of the most important performance indicators for determining the effectiveness of our design against DDoS attacks. Figure 3.8 depicts the results of our tests comparing the magnitude of DDoS attacks on the primary controller and the latency that the system needs to elect a new primary controller when the affected primary controller fails. After a DDoS attack, it typically takes the RYU controllers between 6.16 and 6.93 seconds to select the new primary controller after the previous one has failed. This time has no effect on the operation of the controller or the network, as each network device has a predetermined amount of time before communicating with the primary controller for the flow table update. This timeframe exceeds the time required by our design to select the new primary controller and therefore will not affect the operation of the network.

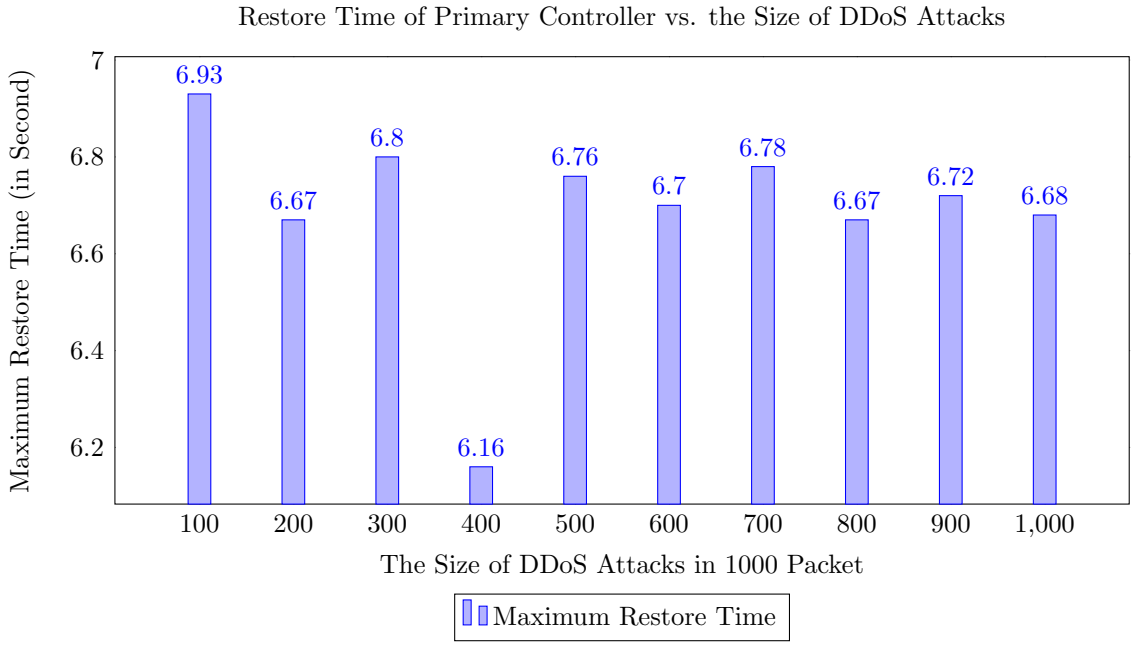

Figure 3.8. Restore Time with different Sizes of the DDoS Attacks

3.3.4. Time Delay and the Size of Network

Figure 3.9 shows the link between the network's traffic and the maximum latency in our hardware-based SDN network design. The direct connection to the RYU controller, the simple topology design, and the Open vSwitch capability are essential for the design's better performance compared to the emulating environment (Mininet) on different VMs machines. Delays might be as high as 0.58 seconds and as low as 0.30 seconds.

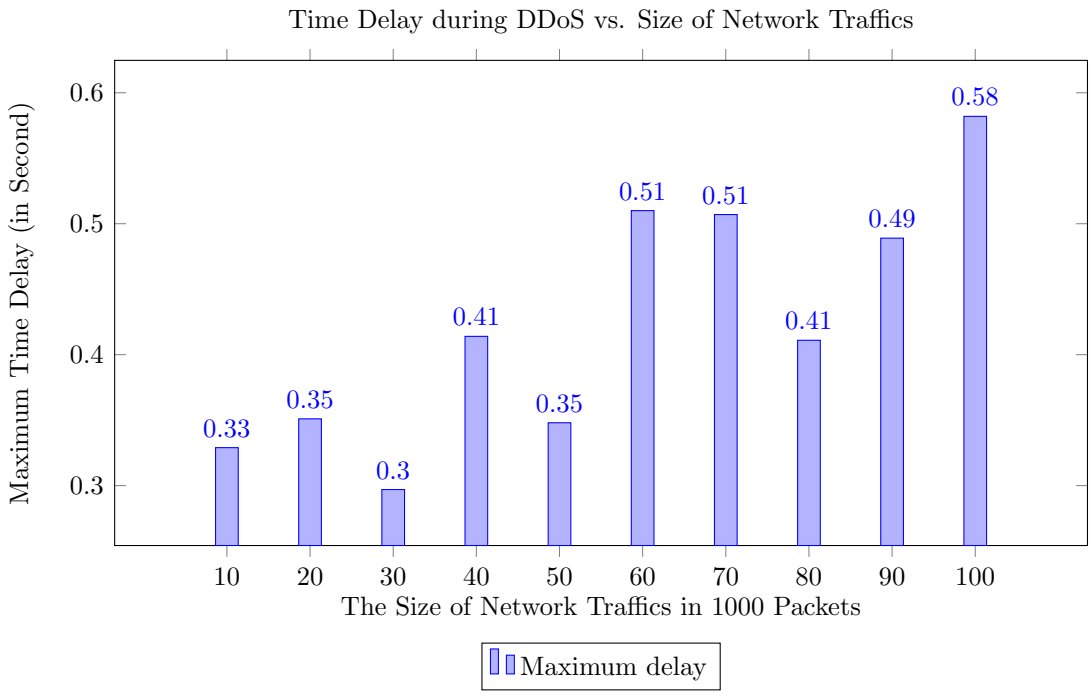

Figure 3.9. Transmission Delay during the DDoS Attacks (HW Implementation

On the other hand, Figure 3.10 illustrates how the amount of network traffic affects the maximum latency in our designed SDN network with Mininet and different VMs. Although packet losses are eliminated even during peak traffic times, the maximum delay grows as the number of packets transmitted in the network increases.

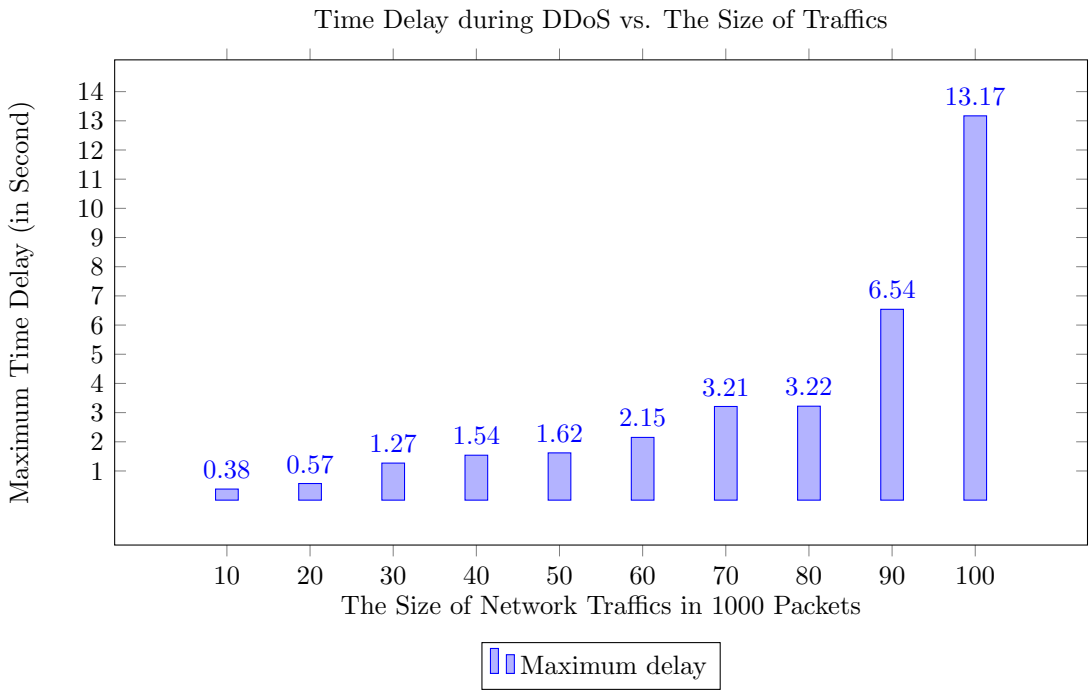

Figure 3.10. Transmission Delay during the DDoS Attacks with Mininet

3.4. Summary and Limitations

Our SDN control plane architecture is based on the open-source RYU SDN controller, which is free to use. In the future, this may be expanded to include POX, ONOS, NOX, and Floodlight, which are all open-source controllers in the SDN space. This means that each open source has its own set of shortcomings in this area, making it more challenging to get several controllers to work together to provide things like synchronization and information sharing across networks and countermeasure the threat of DDoS attacks against the control plane of the SDN. Also, the new challenge is in developing a variety of open-source controllers for the SDN control plane that all rely on the same flow tables and network information to provide services to all of the devices in the same network. We may also put our architecture to the test by putting the features of multiple open-source controllers on an already-existing network with a wide range of nodes, traffic volumes,

and decision-making needs. In this study, we want to make sure that DDoS attacks don't cause an SDN control plane with a single point of failure to fail completely. We do this by suggesting a distributed open-source RYU controller architecture. Our proposed architecture for open-source SDN controllers involves multiple open-source RYU controllers, the Zookeeper service, which is in charge of multi-controller and primary controller election, and the Redis database service, which provides high availability and reliability. In our tests, we use Mininet, Open vSwitch, Raspberry Pi, and Ryu controllers to see how well the design works against DDoS attacks. The design shows a very effective way to deal with the problem of a single SDN controller failing during a DDoS attack, with smooth operation and performance and no packet loss when the primary controller fails over. In future work, we plan to use a real SDN environment to put our design to the test and show how well it works.

Chapter 4. Detecting and Preventing ARP Poisoning Attacks on the RYU SDN Controller

The man-in-the-middle attack is a significant cause of concern in the security field and has received much attention in recent years. Aside from attempting to obtain sensitive information, the attackers' goal, also, is to modify or destroy the data itself as it travels between the nodes in the network. It's important to protect the network from these attacks since they significantly affect the security of all the information on it [29]. According to [28], the Man-In-The-Middle (MITM) attack is the most common and successful method used by cybercriminals to get access to private information while it is being transmitted between two ends, in this case, the target organization and its customers. Man-in-the-middle attacks are particularly dangerous because an attacker can effectively shut down the controller by taking over its functions. More than that, the Attacker may secretly listen in on any discussion taking place between the Controller and the Hosts or between the hosts. The Attacker can eliminate any incoming or outgoing packet and can reroute any packet for their own malicious intent [8]. All routing decisions in an SDN are handled centrally by the controller. This means that the entire network is vulnerable to attack once the Man-In-The-Middle (MITM) compromises the communications between the hosts. It's a failure for the network providers and their clients. Man-in-the-middle attacks include ARP poisoning, IP spoofing, DNS spoofing, SSL hijacking, Wi-Fi eavesdropping, email hijacking, session hijacking, and HTTP spoofing [75]. This paper focuses on ARP poisoning in RYU-controlled SDNs and attempts to develop an SDN solution to detect and protect against this type of MITM attack.

This work addresses the following research objectives:

- Provide the RYU SDN Controller ARP Poisoning Security Application (Ryu-ARP) as a lightweight and efficient solution for the RYU controller to detect and prevent Man-In-The-Middle (MITM) Attacks, particularly poisoning ARP attacks in the SDN.

- Identify the Internet Protocol (IP) address, Media Access Control (MAC) address, and port number of the MITM attacker, as well as the IP, MAC Address, and port numbers of the compromised SDN hosts.

- As an advantage of SDN technology, enhance the security and reliability of the RYU controller with a lightweight and efficient software solution.

In order to secure the communication between the RYU SDN controller and the switches as well as the communication between the hosts in the network from ARP poisoning attacks, it is crucial to detect and prevent MiTM attacks.

Therefore, This research proposes a solution as an application on the RYU SDN controller for identifying and preventing ARP poisoning attacks. This part of the dissertation is structured as follows: Section 4.1 outlines the problem's background. The solution is described in Section 4.2. In Section 4.3, the conducted experiment and evaluation are illustrated. Finally, the chapter concludes with Section 4.4.

4.1. Background

The Address Resolution Protocol (ARP) is a protocol by which a dynamic IP address is linked to a static MAC address or the address of a physical computer in the network. A computer is assigned an IP address when it connects to a network so that it can be recognized and communicate with other computers on the network. The primary function of ARP is to map 32-bit IPv4 addresses to 48-bit MAC addresses [23]. In order to determine if the IP-to-MAC-address mapping has already been performed, a device checks its ARP cache each time it requests a MAC address to send a packet to another device on the network. If it already exists, there's no need to make another request. If the translation between the IP and the MAC address has not been done yet, the request for network addresses is submitted, and Address Resolution Protocol (ARP) is used for this purpose [100].

4.1.1. MITM Attacks in SDN

A Man-In-The-Middle (MITM) attack is a form of eavesdropping attack in which the attacker disrupts an ongoing data transmission by placing themselves in the "middle" of the data transfer. Figure 4.1 demonstrates some MITM threats in SDN.

A MITM attack includes three components: two victims on opposite ends of a communication channel, and an attacker posing as a middleman. With access to the channel, the attacker may watch and change messages between the two ends of the communication channel. Now, the attack will mislead both ends of the communication channel into thinking the attacker's MAC address is the legitimate one [32].

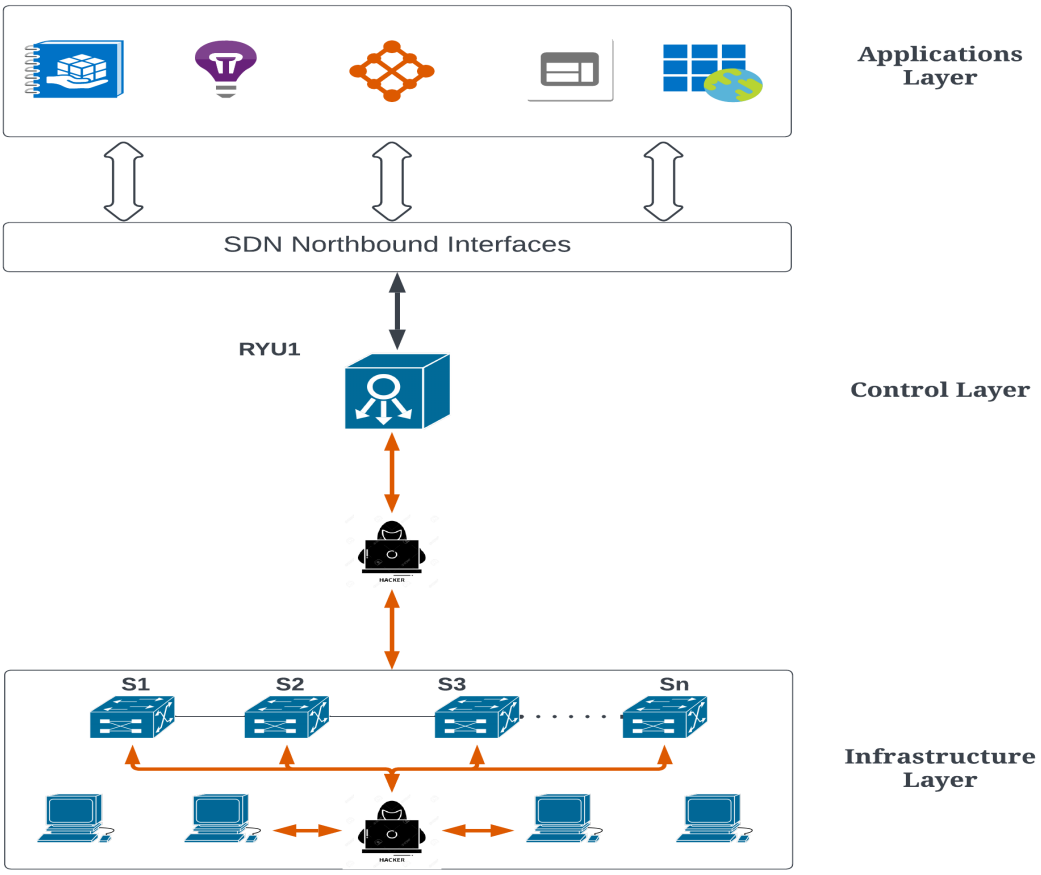

Figure 4.1. Example of MITM Attacks in the SDN

As shown in Figure 4.2, for a typical MITM attack, two victim Endpoints exchange public keys (Endpoint 1 PK and Endpoint 2 PK) to establish encrypted communication. The attacker places themselves in the middle of this communication channel between the two victims (Endpoint 1 and Endpoint 2) and provides them with its public key (MITM PK). Every Endpoint now believes the MITM PK to be legitimate. The first victim (Endpoint 1) then encrypts and sends a message to the second victim using the attacker's public key (MITM PK) (Data 1). The attacker is in possession of the private key and uses it to decrypt Data 1. The attacker will then transmit plaintext that has been encrypted with Endpoint 2's public key to Endpoint 2 (Data 2) [28].

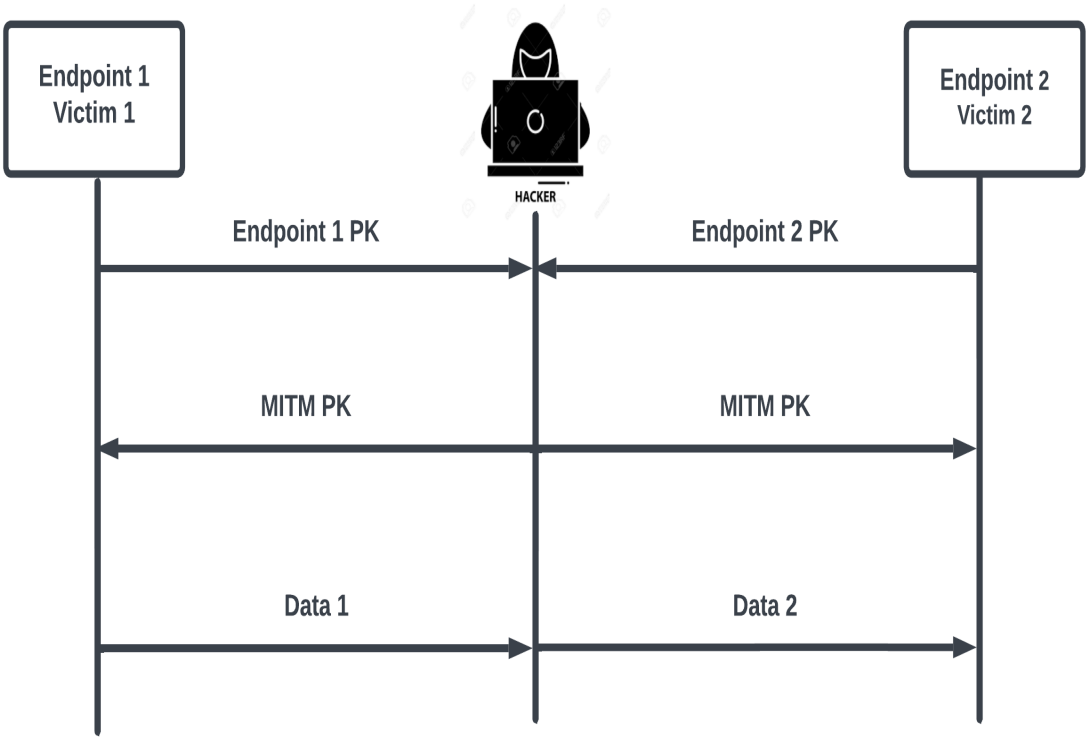

Figure 4.2. MITM Attacks Messages

The SDN's southbound interfaces are vulnerable to Man in the Middle (MITM) attacks. Host-to-host communication must be protected from man-in-the-middle attacks, and the controller-to-data-plane-devices connection must also be secured. That should make a MITM attack less likely to happen. Without proper protection, data integrity issues might arise when moving information between control and data plane nodes [70]. This form of attack will have consequences on both the control and data planes. One of the primary challenges of software-defined networks is the data modification between the control and data planes [11].

4.1.2. ARP Poisoning in SDN

The Address Resolution Protocol (ARP) is commonly employed to convert IP addresses to MAC addresses. Every host on a subnet has its own ARP cache for mapping IP addresses to MAC addresses. Modern network protocols maintain this table by monitoring ARP packets unless manually specified. Despite its widespread use, ARP has significant flaws. For example, there is no built-in way for a receiver host to verify the authenticity of the sending host in an ARP transmission [43]. In addition, ARP is a stateless protocol, meaning that nodes may send ARP answers even if they haven't received an ARP request. Since an attacker may employ ARP Cache Poisoning to launch further attacks, including as Denial of Service (DoS), Man-in-the-middle (MITM), and Host Impersonation attacks, its prevention is crucial in both traditional and SDN networks [85]. This research [68] is the first to quickly describe the working principles of six methods to mitigate this threat in SDN. In this study [90], they conduct a comprehensive literature review of the methods available for mitigating ARP Cache Poisoning attacks in SDN. With SDN, an ARP cache poisoning attack may be launched against both Proxy ARP and Regular ARP.

Taking advantage of the centralized nature of SDN, Proxy ARP allows the Controller to respond to all ARP inquiries on the network. When an ARP packet is received, the switch sends it on to the Controller. On the other hand, Regular ARP in SDN functions in the same way as it does in traditional networks. The host broadcasts an ARP request across the network and receives a MAC address from the replying host in a unicast ARP reply [31] [62] and [14].

Proxy ARP Poisoning

The target of a proxy ARP poisoning attack is the SDN controller's ARP cache, which the intruder will attempt to corrupt with malicious entries. Fake packets might be sent to the controller in the form of ARP requests or replies to achieving this goal [15]. Those kinds of fake packets could potentially corrupt the controller's ARP cache by posing as either a request or a reply. Poisoning the SDN controller's ARP cache could have many other effects on the network as a whole. For example, a man-in-the-middle attack could be used to get important information from the network flow [47], [49].

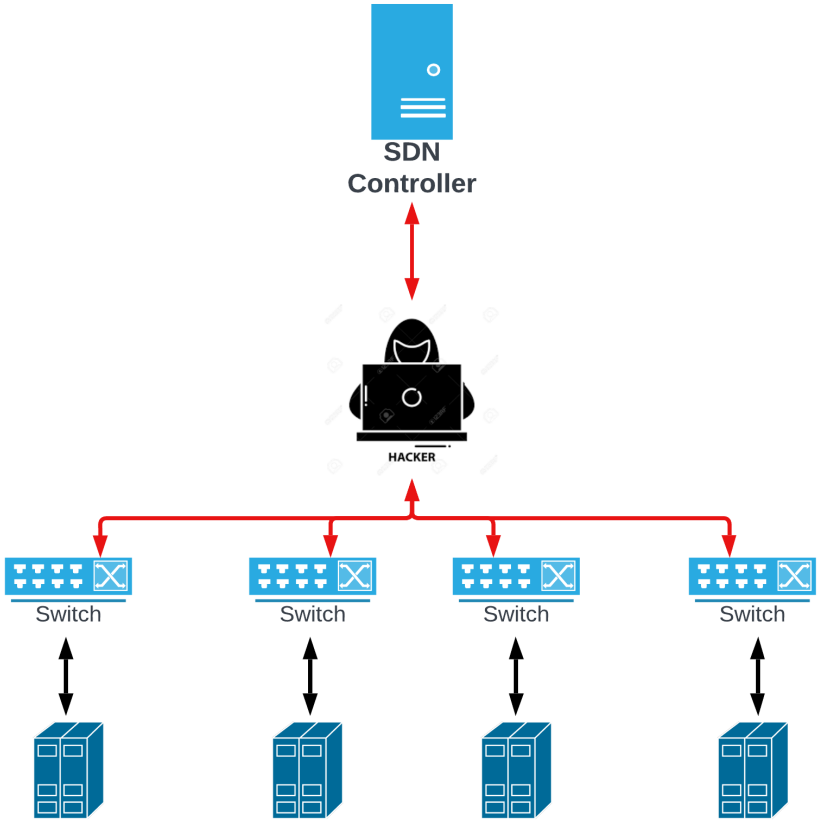

Figure 4.3. Proxy ARP Poisoning in the SDN

Regular ARP Poisoning

The attacker's goal in this regular ARP poisoning attack is to corrupt victims' caches on the SDN's hosts. Similar to attacks on traditional networks, this attack involves sending fake ARP request packets in an effort to poison hosts' ARP caches. This attack may also be carried out by sending fake ARP replies or sending unnecessary ARP packets [16].

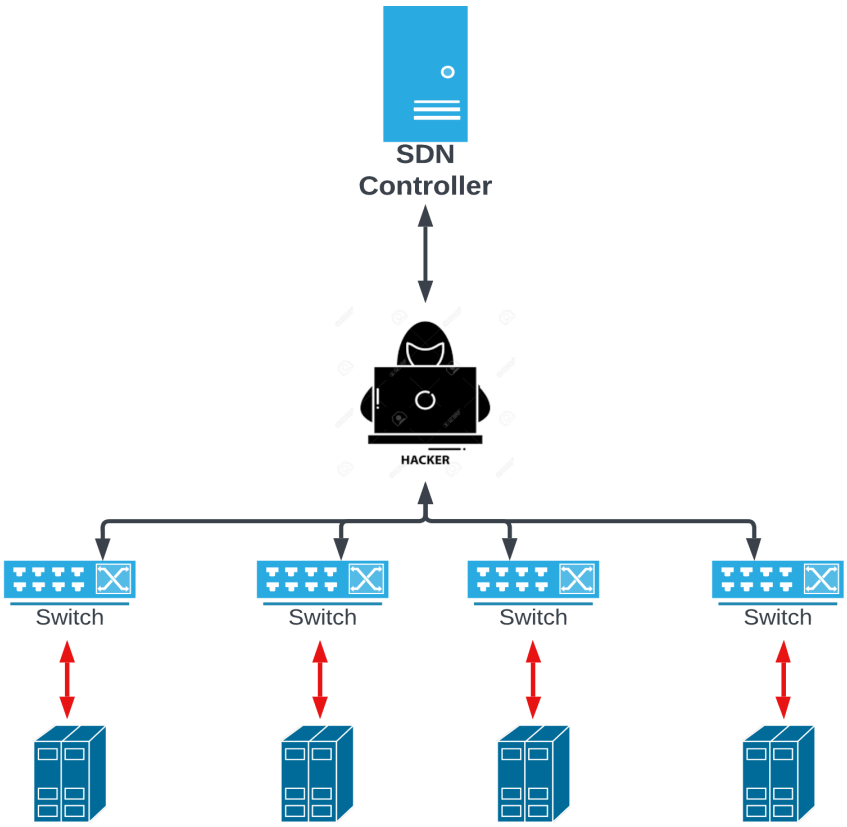

Figure 4.4. Regular ARP Poisoning in the SDN

4.2. Methodology and Implementation

Section 4.2.1 covers the apparatus, Section 4.2.2 describes the research environment, and Section 4.2.3 explains the experiment.

4.2.1. Apparatus

The hardware and software required to create and perform this ARP poisoning for the SDN study are detailed in Table 4.1. To simulate our network, we used Mininet `http://mininet.org/`. We used the Ettercap tool `https://www.ettercap-project.org/` to generate the ARP poisoning attacks. For our study, we used the Ryu controller as the main controller in the control plane `https://ryu-sdn.org/`. This controller is open-source and can operate on MacOS, Linux, or Windows.

Table 4.1. HW and SW Details for the MITM Experiment

System Details		**Software Details**	
Device	**Details**	**Software Name**	**Version**
Processor	Intel(R) Core(TM) i7-8550U CPU @1.99 GHz	VMWare Workstation Pro	16.2.4 build-20089737
System Type	64-bit Windows 11 OS, x64-based processor	Linux	Ubuntu 20.04.3 LTS
Graphics Card	NVDIA GeForce MX150	Mininet	2.3.0
Manufacturer	HP Spectre x360	RYU	4.34
Memory (RAM)	16.0 GB	Ettercap	0.8.3.1-Bertillon

4.2.2. Experimental Environment

For the research environment, we configured an RYU SDN controller and a network for the experimental study. We used a Mininet VM to create a virtual network, and a Linux Ubuntu version 5.4.0-105-based VM to host the RYU controller. Using a Windows 11-based laptop, VMware was used to manage both VMs. To simulate a network managed by the RYU controller, we connected the RYU controller to the Mininet virtual network. This enabled us to have configuration control over our experimental network without the need for additional hardware. Tools to conduct ARP spoofing attacks have proliferated in response to the widespread use, significance, and actual risks of such exploits. Most operating systems have support for these ARP simulation applications. The most common ARP spoofing attack software are Ettercap [1], Dsniff [2], Yersinia [5], and Cain and

able [3]. To test our solution against the MITM attack, we have performed the MITM attack using Ettercap tools.

4.2.3. Experiment

Our experiment is done in three phases - First, the RYU SDN controller was the central element of our network design with RYU version operating on a standalone VM. The Mininet emulator, which generates a realistic virtual network by executing a genuine kernel, switch, and application code on a separate VM, was also used. We then connected our SDN network architecture on Mininet to the RYU controller as a remote controller in order to govern and run the entire simulation network. In our SDN network, we created several network topologies, tested different forms of network traffic, and validated that the network was operational. Figure 4.5 is an example of a network architecture. In the second phase, we implement our detection and prevention tool, Ryu-ARP, as a security application for the RYU controller. The third phase involved running the MITM ARP poisoning tool Ettercap against our SDN system to create the MITM attack. We compared the performance of our SDN network before, after, and during the MITM attack after we conducted the experiment on our setup with different topologies.

4.3. Ryu-ARP

Ryu-ARP (RYU SDN Controller ARP Poisoning Security Application) is an SDN security application that is added to the RYU SDN controller to make it better at detecting and stopping the threat of a man-in-the-middle attack, specifically ARP poisoning. It is written in Python and can be executed directly with the ryu-manager command used to control the RYU controller.

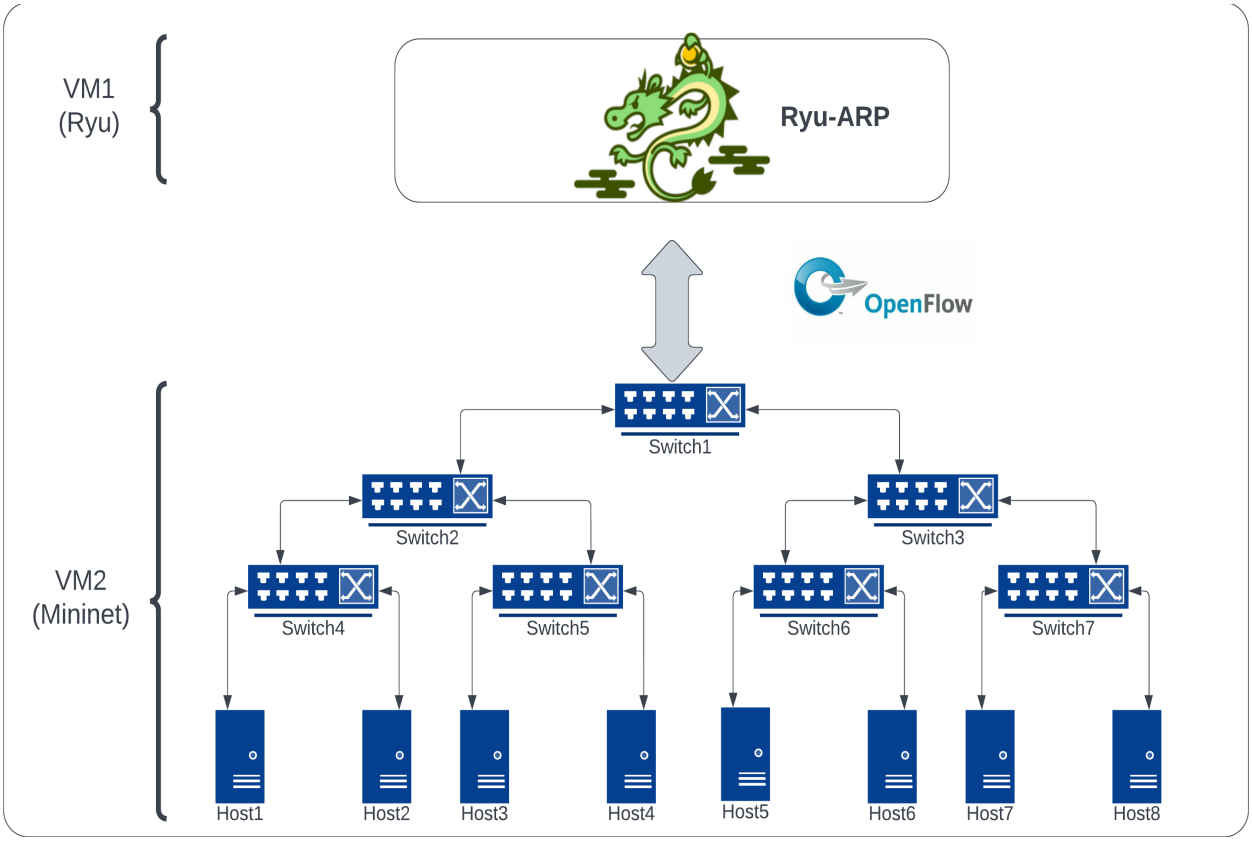

Figure 4.5. Example of One Experimental Network Topology

The Ryu-ARP concurrently works with the RYU controller. The Ryu-ARP is designed with three main components. The first component begins by constructing network-specific ARP cache entries. The ARP cache table consists of IP- MAC matching entries. This component collects IP addresses, Mac addresses, and port numbers for every device connected to the SDN network. All of this data is maintained on the RYU controller for use as a reference and as the foundation for detecting and preventing ARP poisoning attempts. Changes in the network flow table trigger updates to the ARP cache entries.

The second component deal with detecting an ARP poisoning attack. In this function, the Ryu-ARP compares the source Mac addresses of the ARP request to the entries in the ARP cache table that we created in the previous function. If the source Mac ad-

dress of the ARP request does not already exist in the ARP cache table, then the table must be updated with the new value of that Mac address, along with its IP addresses and port number. On The other hand, if the source Mac address does exist in the ARP cache table so Ryu-ARP will check the source IP and port number with the stored values. If it is the same, then this is a valid ARP request and no further action will be taken, but if it is different, then this is an ARP poisoning attack and Ryu-ARP will alert the RYU controller for further action.

The third component is to prevent ARP poisoning attacks. This function is responsible for the actual prevention of the attacks. After detecting a poisoning attempt on the previous function, the Ryu-ARP will block the attacker's port, thereby removing them from the network.

4.3.1. Ryu-ARP Algorithm

The following algorithm 2 explains how the Ryu-ARP performs.

4.3.2. Ryu-ARP Usage

The RYU security application (Ryu-ARP) provides two main functions:

- Detecting the presence of a man in the middle attack (ARP Poisoning).
- Preventing the threat of MITM attack, in particular the (ARP poisoning).

Ryu-ARP can be used with the following commands:

```
ryu-manager Ryu-ARP.py
```

Algorithm 2: RYU SDN Controller ARP Poisoning Security Application (Ryu-ARP)

Input: ARP Cache Table
Output: Detection and Prevention of ARP Poisoning
function BUILDING THE ARP CACHE ENTRIES(
)**if** $MAC == New$ **then**
| $ARPCacheTable$ ←Add the New MAC, IP Port No.
else
| $ARPCacheTable$ ←No Updates
end
end function

function DETECTION OF ARP POISONING(
)**if** $SrcMAC == Exists$ **then**
| $CheckSrcIP == ARPInfo.Stored$ ←No Action
else
| $SrcIP == NoMatchingARPInfo.$ ←ARP Attack
end
end function

function PREVENTION OF ARP POISONING(
)**if** $ARPPoisoningAttack == True$ **then**
| $AttackerPortNo.$ ←Blocked
else
| $ARPPoisoningAttack == False$ ←No Action
end
end function

The Ryu-ARP will be working alongside the RYU controller and will be responsible for monitoring each and every ARP request that is sent throughout the network in order to reach a conclusion regarding the validity of ARP reply with the entries in the ARP cache. We also have the option of capturing the traffic flow by using the tcpdump command, after which we can generate a .pcap file that has further information regarding the ARP protocol messages.

4.4. Evaluations

In this section, we test and apply Ryu-ARP to a number of different SDN topologies and configurations using Mininet simulation to get a better idea of how well our tools work with different sizes and layouts of SDN networks.

4.4.1. ARP Cache Dataset

This learning process begins with the controller constructing an ARP cache database including information about every SDN-connected device. IP addresses, Mac addresses, and port numbers are the three essential components of the Arp cache table. This procedure begins when our tool is launched alongside the RYU controller. The information about each device (IP address, Mac address, and port number) is periodically updated in case a communication connection between hosts is requested.

Figure 4.6 demonstrates the construction of an ARP table with seven switches attached to an RYU controller.

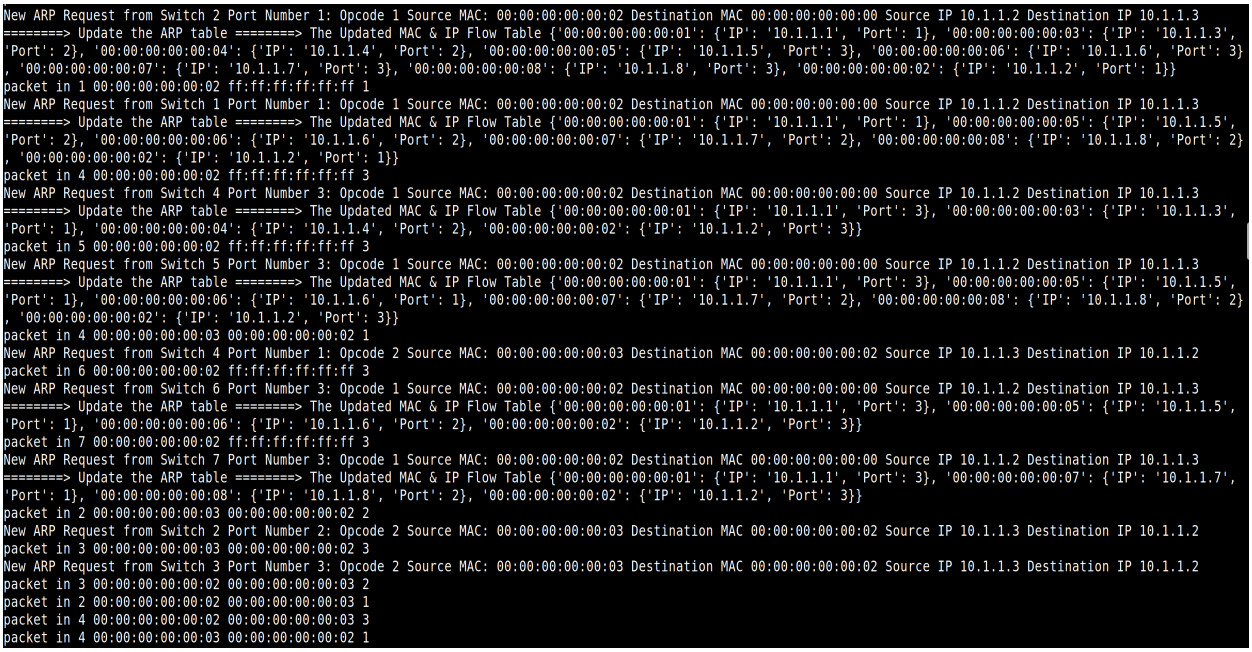

Figure 4.6. Building The ARP Table

The outcome of this procedure is illustrated in Figure 4.7, this table contains the IP Addresses and Mac Addresses of all devices connected to each switch and host in this SDN network.

```
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> h1 arp -a
? (10.1.1.6) at 00:00:00:00:00:06 [ether] on h1-eth0
? (10.1.1.5) at 00:00:00:00:00:05 [ether] on h1-eth0
? (10.1.1.4) at 00:00:00:00:00:04 [ether] on h1-eth0
? (10.1.1.3) at 00:00:00:00:00:03 [ether] on h1-eth0
? (10.1.1.8) at 00:00:00:00:00:08 [ether] on h1-eth0
? (10.1.1.2) at 00:00:00:00:00:02 [ether] on h1-eth0
? (10.1.1.7) at 00:00:00:00:00:07 [ether] on h1-eth0
mininet> h8 arp -a
? (10.1.1.1) at 00:00:00:00:00:01 [ether] on h8-eth0
? (10.1.1.7) at 00:00:00:00:00:07 [ether] on h8-eth0
? (10.1.1.2) at 00:00:00:00:00:02 [ether] on h8-eth0
? (10.1.1.3) at 00:00:00:00:00:03 [ether] on h8-eth0
? (10.1.1.4) at 00:00:00:00:00:04 [ether] on h8-eth0
? (10.1.1.5) at 00:00:00:00:00:05 [ether] on h8-eth0
? (10.1.1.6) at 00:00:00:00:00:06 [ether] on h8-eth0
mininet>
```

Figure 4.7. The ARP Table Information

4.4.2. Detection Process

In one of our performance evaluation experiments for Ryu-ARP, we executed an ARP attack on a topology consisting of seven switches and eight hosts, as shown in Figure 4.5. The IP address of the attacker is 10.1.1.3, while the victim devices are 10.1.1.1 and 10.1.1.8.

The procedure of detection begins by focusing on the source Mac address, which is 10.1.1.1. The Ryu-ARP verifies the source Mac addresses of the ARP request with the entries in the ARP cache table that were constructed when the tool was launched. If the source Mac address of the ARP request does not already exist in the ARP cache table, the

table must be updated with the new value of that Mac address, as well as its IP addresses and port number, which is not the case with 10.1.1.1. The source Mac address does exist in the ARP cache table, thus Ryu-ARP will compare the source IP and port with the cached values, which are 00:00:00:00:00:01, which is different from what we have while the SDN is under attack from 10.1.1.3. Here, 10.1.1.1's Mac address is 00:00:00:00:00:03 as shown in Figure 4.8. Hence, this is an ARP poisoning attack, and Ryu-ARP will notify the RYU controller to block the attacker's port on 10.1.1.3. In addition, another endpoint of the attack, 10.1.1.8, is also affected by this attack and has an incorrect mac address of the attacker: 00:00:00:00:00:03.

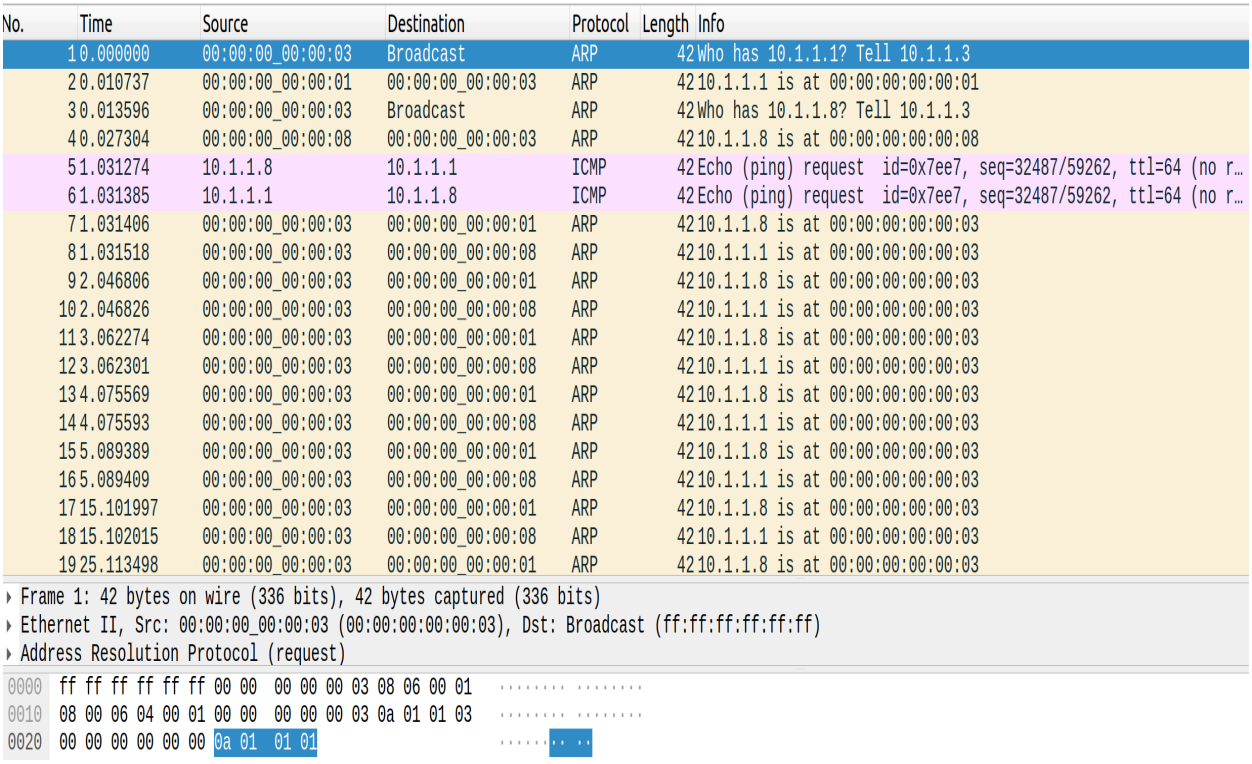

Figure 4.8. The ARP Poisoning attack

4.4.3. Prevention Process

During the preventive phase of our approach, the Ryu-ARP will eliminate the attacker host from the network and block the port that the attacker is using. The warning message will appear on the controller terminal, as shown in figure 4.9, together with all of the information that the administrators require in order to identify the IP address of the attacker (10.1.1.3), as well as his or her port number and Mac address (00:00:00:00:00:03). In addition, Ryu-ARP profiles the targeted devices, which are identified by the IP addresses 10.1.1.1 and 10.1.1.8, as well as the devices' Mac addresses.

```
New ARP Request from Switch 4 Port Number 1: Opcode 2 Source MAC: 00:00:00:00:00:03 Destination MAC 00:00:00:00:00:01 Source IP 10.1.1.8 Destination IP 10.1.1.1
======> Warning: There is an ARP Poisoning Attack=========> The Attacker is: MAC 00:00:00:00:00:03 Targted IP 10.1.1.8 ========
The Correct IP Address for The Attacker MAC  00:00:00:00:00:03 is {'IP': '10.1.1.3', 'Port': 1}
Action: Blocking the Attacker Port
packet in 4 00:00:00:00:00:03 00:00:00:00:00:08 1
New ARP Request from Switch 4 Port Number 1: Opcode 2 Source MAC: 00:00:00:00:00:03 Destination MAC 00:00:00:00:00:08 Source IP 10.1.1.1 Destination IP 10.1.1.8
======> Warning: There is an ARP Poisoning Attack=========> The Attacker is: MAC 00:00:00:00:00:03 Targted IP 10.1.1.1 ========
The Correct IP Address for The Attacker MAC  00:00:00:00:00:03 is {'IP': '10.1.1.3', 'Port': 1}
Action: Blocking the Attacker Port
```

Figure 4.9. The Detection and Prevention of The ARP Poisoning attack

Following the poisoning attack, the status of the ARP Poisoning attacker is shown in Figure 4.10. Because host 3 has been removed from the network, it is no longer able to communicate with any of the other hosts that are part of the network. also, the AARP cash table had the entry for host 3 deleted as well.

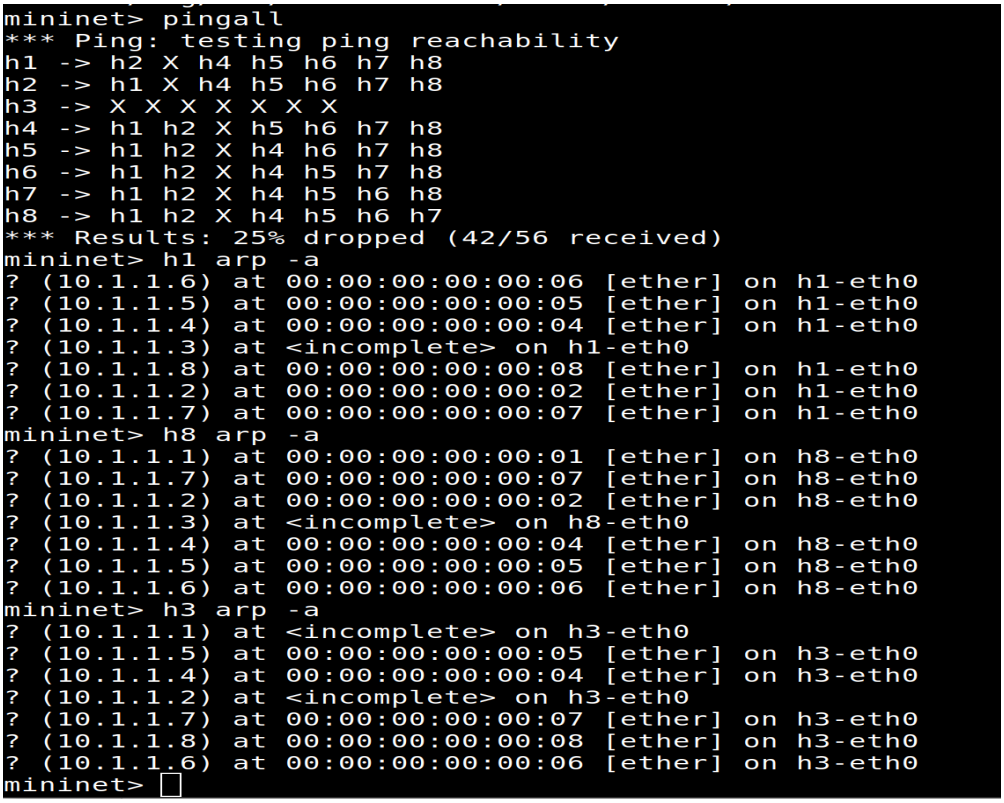

Figure 4.10. The ARP poisoning Attacker Status

4.5. Summary and Future Work

Ryu-ARP is a very effective solution that operates on the RYU controller and helps to protect against the threat posed by an ARP poisoning attack. It enables the RYU controller to detect and prevent any MITM attempt in the SDN network, and it provides the network administrators with the ability to defend against such an attack without causing any overhead or a lower level of performance.

Ryu-ARP operates with three functions. constructing the ARP cache database with matching IP and Mac addresses, and then utilizing this information as a reference for the second function of the detecting process. The third function is to prevent and eliminate an ARP attacker. Hence, we could use the same three functions in the development of a security application that could be an add-on to other SDN controllers and upgrade

their security application library with an SDN application that could defend against the ARP poisoning attack.

In further work on our ARP poisoning tool (Ryu-ARP), we will look into the possibility of using the exact same technique and algorithm for several different types of SDN controllers. Because the ARP poisoning attack poses a significant risk to the operation of the SDN network, it is essential and a good idea to employ the same strategy when defending against this type of threat. In addition, as future work, we are going to attempt to deploy our security application to hardware devices that are already part of an SDN network. This will allow us to put our tool through its tests and evaluate its performance in addition to the overhead it may impose in the real world.

Chapter 5. Memory Forensics of The OpenDaylight SDN Controller

Software-defined networking (SDN) introduces new issues for cybersecurity approaches and network forensic investigation methodologies. In traditional networking, analytics and network security were based on the control and data information for the network infrastructure being in one place. However, since SDN is a highly centralized scheme, the separation of the data plane and the control plane into two separate layers would definitely require new forensics analysis and security methodologies. SDN is a widely popular approach for virtualizing networking functions. While there has been extensive research on SDN security, little emphasis has been placed on acquiring and analyzing data in the SDN to collect digital evidence for cyber incidents [80]. In the case of a traditional network attack, it is necessary to collect disk and memory data from a wide range of devices on the network in order to figure out what kind of cyber operations have happened and when. However, with the introduction of SDN, the SDN controller now manages the vast majority of network data, providing the investigator with the unique perspective of finding all the network data in one location. The controller is employed to centrally administer the whole network. This technology is beneficial for network users as well as network administrators, and we intend to determine how SDN affects digital forensics investigations.

The main focus of this research is on how SDN, a new technology in the networking field, affects memory and network forensics analytics and security techniques. For the OpenDaylight SDN controller, our main goal is to build a framework for memory forensics and network forensics. By pulling information about network devices and flow tables from the controller's acquiring memory and analyzing it as part of our work in digital forensics

for the OpenDaylight SDN controller, we can look into and evaluate the security of the SDN environment. As part of this study, we introduce a mechanism for retrieving and analyzing memory data from the OpenDaylight SDN controller. On that basis, we are developing a forensics tool to help detectives work in SDN environments. This research would open up a new dimension for the investigator by identifying new methodologies for acquiring, analyzing, and evaluating digital evidence in SDN settings. The acquisition of memory dumps from the SDN controller is crucial for obtaining information about the network and each device that is connected to it.

Given the importance of a controller to SDN, we focus our research on it. The contributions of this part of the dissertation are given below:

- Enhancing digital forensics research on the SDN controller's memory and Orchestrating the process of obtaining the SDN controller's memory.
- We conducted the primary research on SDN network controller memory. Also, we devised and tested the process of acquiring the SDN controller memory of an ODL controller.
- We developed a tool, (SCoNDT) that extracts host IPs and other forensically relevant data from a controller's memory image.
- We share our collected dataset of memory images publicly for future researchers and reproducibility.

The remainder of the chapter consists of the following sections: Section 4.1 describes the background of our research; Section 4.2 offers an overview of our methodology and implementation; Section 4.3 describes the Exploratory Analysis of the proposed approach; Section 4.4 contains the SCoNDT's explanation; Section 4.5 provides the evaluation of SCoNDT; Section 4.6 provides some discussion about SCoNDT; and Section 4.7 contains the summary and future work.

5.1. Background

SDN was conceived to virtualize traditional networks. Through network virtualization, the control layer and infrastructure layer are separated. The infrastructure layer refers to the hardware found in a network, such as switches and routers. The control layer refers to the software traditionally integrated into the hardware, such as the software used to route packets in routers. In SDN, there is a software controller which manages this separation and performs these tasks for the network. Some advantages of SDN are:

1. A centralized point for network administrators to access the network and check the status of all devices.
2. A centralized point for applications to interact with the network.
3. Greater scalability of networks, new devices will be managed by the controller.
4. Potential cost savings, especially on hardware.

While the centralization of SDN simplifies management and application development, it also introduces certain security risks. SDN is vulnerable to Denial of Service attacks, if the controller is overloaded then it affects the entire network. [107] and [77] describe SDN in further depth and examine some of the obstacles this technology faces. The packet routing decisions and control tasks in SDN are made by the SDN controllers. In SDN, the southbound interface of the SDN controller is connected to the rest of the network; hence, the majority of information entering the network will have to traverse the memory of the SDN controller. As information that flows through the memory of the SDN controller would leave traces, we limit the scope of our work to examining the memory of the controller. To further explain our work, we provide more background information about SDN controllers in Section 5.1.1.

5.1.1. SDN Controller

The controller is the software component responsible for coordinating the allocation of network resources. Figure 5.1 depicts the core elements of the SDN controller, providing a more in-depth look at the controller's architecture. The components of the controller serve as the structural basis for the controller's operation [57].

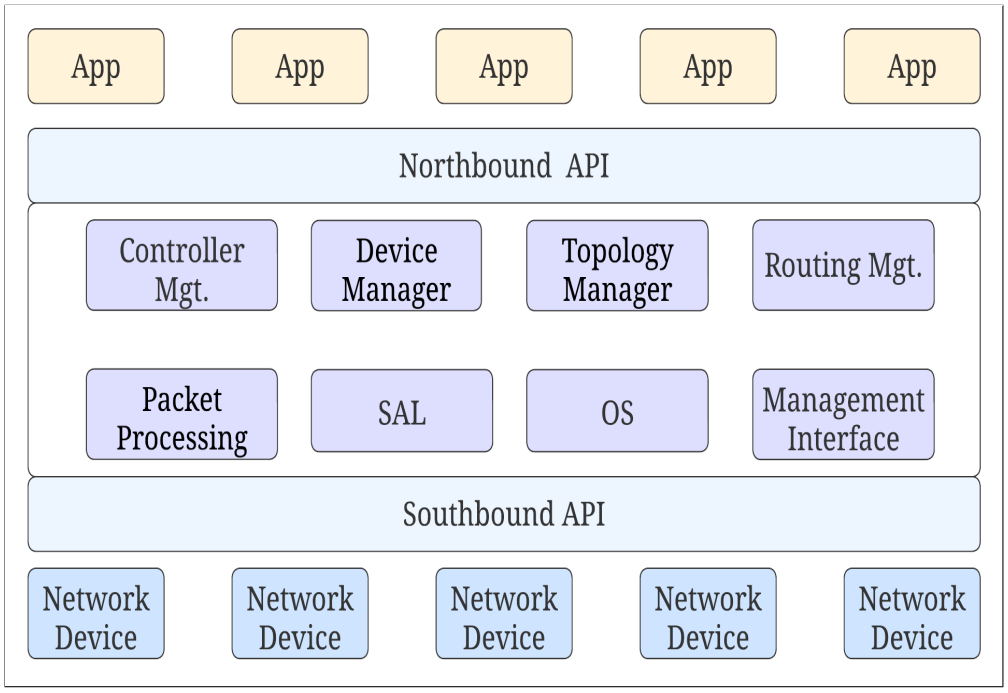

Figure 5.1. SDN Controller Components

The Controller Management unit maintains all network devices linked to the controller. The Device Manager handles the connections between devices on the network. The Topology Manager describes what connections are made between different devices in the network. The Routing Manager routes packets between source and destination addresses. However, this depends on the protocol used. The Packet Processing component processes packet headers and payloads depending on the network protocol. In order to support other device protocols, developers may implement a SAL. The OS component is the operating

system that the controller runs on top of, typically it is a Linux system. Web-based administration panels are the standard for providing access to the controller through the Management Interface component. These panels provide quick and clear navigation of the controller's available features. Every controller has an OF module installed, which is responsible for handling messages, actions, table entries, flow rule matching, message queues, and statistics.

There are several types of SDN controllers being used. Some of them are described with their specifications in table 5.1. For our experimental environment, we choose the OpenDayligh `https://www.opendaylight.org/` controller. This Java-based, open-source controller is compatible with macOS, Linux, and Windows.

Table 5.1. SDN Controllers Features

Controller	**Programming Language**	**Architecture**	**Platform**	**License**
OpenDaylight	Java	Distributed Flat	Linux, MacOS, Windows	EPL 1.0
Ryu	Python	Centralized	Linux, MacOS	Apache 2.0
Floodlight	Java	Centralized	Linux, MacOS, Windows	Apache 2.0
ONOS	Java	Distributed Flat	Linux, MacOS, Windows	Apache 2.0
NOX	C++	Centralized	Linux	GPL 3.0

5.1.2. Memory Forensics

The field of Memory Forensics focuses on analyzing the RAM of a computer, also known as the volatile memory. A memory dump is acquired by copying the RAM of a computer. By analyzing the memory dump, you can potentially see information about what a device was doing when the memory was acquired. Memory forensics can also reveal information that was never written to the disk and can be found exclusively in memory. [78] outlines the processes of memory acquisition and memory analysis methods. As

of now, the primary tool used for analyzing memory is Volatility `https://github.com/volatilityfoundation/volatility`. [24] describes the state of the field of memory forensics as of the year 2017. When talking about application-specific analysis, it is mentioned that memory forensics is gaining popularity in all types of investigations to obtain information not possible through network or disk forensics. It is also mentioned that the memory of certain IoT devices is also able to be analyzed, showing that memory forensics may be performed on devices other than traditional computers. [17] compares the two methods, live response, and memory imaging, used to acquire a memory dump. There are three requirements for a forensically sound memory acquisition: correctness, atomicity, and integrity [102]. [27] discusses using the Yara library and scanner to automatically search the memory for malware signatures. While it is primarily used to scan for malware, Yara signatures may be used to search for any patterns in memory.

The memory of mobile phones was analyzed by [96]. The authors managed to retrieve outgoing and incoming messages from the memory of the phone a short time after the messages were sent. [25] explores the memory forensics of VR devices. In this research, the physical location of the VR user was found in memory.

5.1.3. Network Forensics

The field of network forensics focuses on extracting and analyzing network interaction data. Prior work constructed a network forensics framework and described basic components of network forensics [81]. The main purpose of network forensics is described as tracking down the attacker of a network, with the goal to prosecute them in a court of law. [38] discusses and compares four main network forensics tools: Xplico, OmniPeek,

NetDetector, and NetIetercept. [69] describe additional tools, as well as some of the common techniques used in network forensics such as packet and email sniffing, web forensics, and IP trace-back techniques.

Packet analysis is another network forensic technique, and it can provide insight into the activities of a network. Analyzing packets can potentially reveal images, messages, and files that were sent over a network [94]. When collecting network forensic artifacts, a lot of insignificant data will be collected. [72] discusses network forensic analysis using AI techniques to make sorting the data simpler. The author's goal is to use AI to collect key data while keeping non-relevant information confidential. [12] talks about a memory forensics approach to network investigations. In their research, they showed how to find a large number of connections made by other devices in a device's memory. One hour after the attack has occurred, a large number of malicious packets were still able to be found in the memory of Windows and Linux devices. [9] analyzes the instant messaging app WhatsApp with the goal of extracting the IP address of any device that sent a Whatsapp message. They achieved this by analyzing network traffic.

In [103], the authors analyzed over twenty Android applications and were able to reconstruct partial and full messages from network traffic. Popular video conferencing apps, WebEx and Zoom, were analyzed using the network, disk, and memory forensics [55] [64].

This study is concerned with the digital forensics of the OpenDaylight SDN controller. We were attempting to acquire the memory of the controller and then obtain as much information as possible about the SDN network.

5.2. Methodology and Implementation

The apparatus is described in Section 5.2.1, the research environment in Section 5.2.2, and the experiment in Section 5.2.3.

5.2.1. Apparatus

In this section, we describe the devices and software used in our methodology. Volatile was selected because it is widely used, open source, extendable, utilizes Python, and is compatible with Windows, Mac, and Linux. Table 5.2 details the hardware and software used to produce and perform this SDN-based memory forensics research. Mininet (`http://mininet.org/`) was used to simulate our network. To collect the memory dump we used Surge Collect (`https://www.volexity.com/products-overview/surge/`), and Volatility was used to analyze the collected memory dumps. We used the ODL (`https://www.opendaylight.org/`) controller for our tests. This controller is open source, programmed in Java, and can run on MacOS, Linux, or Windows.

Table 5.2. Hardware and Software Details

System Details		**Software Details**	
Device	**Details**	**Software Name**	**Version**
Processor	Intel(R) Core(TM) i7-8550U CPU @1.99 GHz	VMWare Workstation Pro	16.2.4 build-20089737
System Type	64-bit operating system, x64-based processor	Linux	Ubuntu 20.04.3 LTS
Graphics Card	NVDIA GeForce MX150	Mininet	2.3.0
Manufacturer	HP Spectre x360	The Volatility Framework	2.6.1
Memory (RAM)	16.0 GB	OpenDaylight	karaf-0.6.4-Carbon

5.2.2. Experimental Environment

For the research environment, we used Mininet in our experiment because it is one of the most effective tools for simulating SDN and virtual networks. It offers a convenient and realistic environment for a low cost [53], [50] [54]. In addition, Mininet-WiFi serves as a tool for simulating wireless OpenFlow/SDN scenarios, enabling experiments with high fidelity that replicate real networking environments [37]. We configured an SDN controller and a testing network. We created a virtual network using a Mininet VM and a second VM running Linux Ubuntu version 5.4.0-105 to host the controller. Using a Windows 10 laptop, both VMs were operated by VMware (`https://www.vmware.com/`). In order to simulate a network controlled by the ODL controller, we linked the ODL controller to the Mininet virtual network. This allowed us to control the setup of our experimental network without requiring additional hardware. Volexity's Surge Collect, a tool for memory capture, was installed on the ODL VM. Throughout our experiments, Surge enabled us to acquire memory dumps from the controller. To run Volatility on the memory dumps, a profile of the operating system on which the ODL controller was running was generated.

We began by establishing our research environment and concentrated on the ODL SDN controller running on a Linux operating system. After establishing our environment, we examined the memory of the controller after the regular operation. We concluded by evaluating the memory of the controller following a network attack. Our objective during the process of implementation was to retrieve forensic evidence from memory. We were particularly interested in the network's structure, current network activity, and recent network connections.

5.2.3. Experiment

The experiment had two different stages. In the beginning, we utilized the ODL SDN controller as the core element of our network design. On a standalone VM, version karaf-0.6.4-Carbon of the ODL ran. The Mininet emulator, which generates a realistic virtual network by executing a real kernel, switch, and application code on a separate VM, was also used. Then, we connected the Mininet network topology to the ODL controller as a remote controller to manage and run the full simulation network.

In our SDN network, we created several network topologies, executed different types of network traffic, and confirmed that the network was operational. Figure 5.2 illustrates an example of a network's topology.

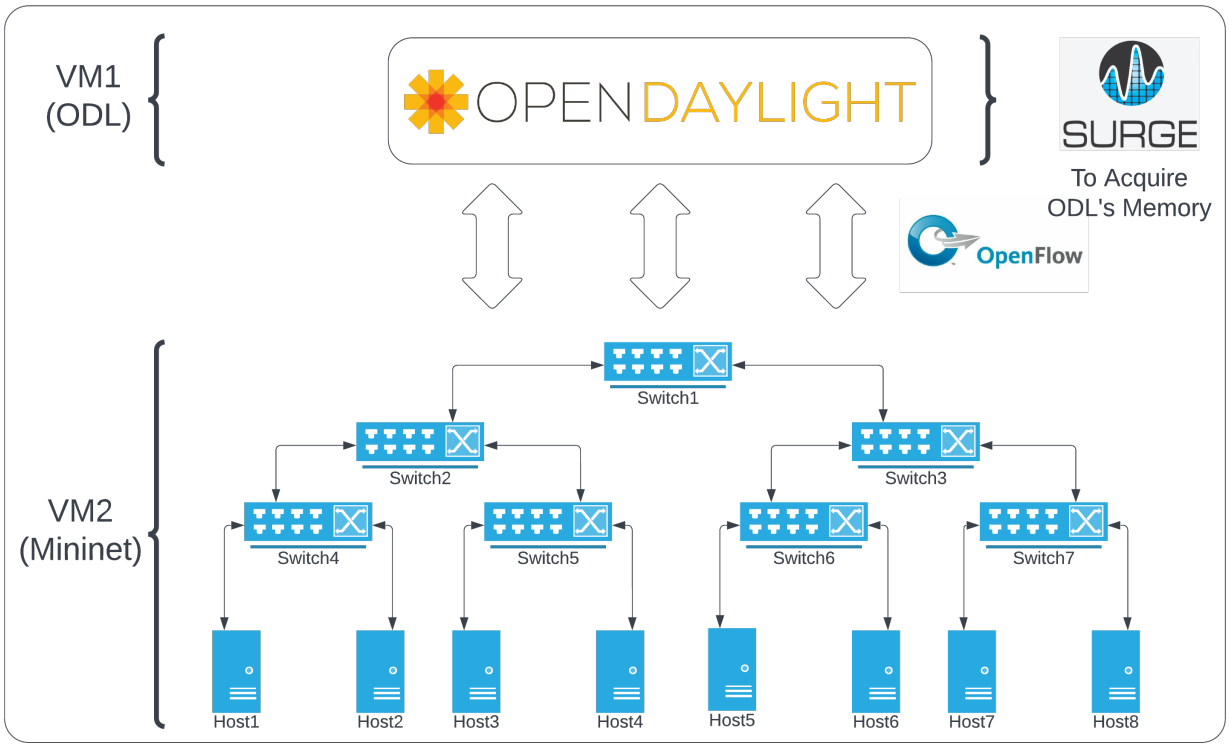

Figure 5.2. Example of One Experimental Network Topology

In the second step, multiple memory samples of the ODL memory were collected. Surge Collect was used for acquiring different samples of ODL memory images under various setups and topologies (See Table 5.3). After collecting all of the memory samples, we evaluated them with Volatility and conducted the manual analysis. Then, SCoNDT was developed (See Section 5.4).

Table 5.3. Memory Samples of ODL Controller

Memory Sample	Topology	Number of Switches	Number of Hosts
1 [2GB]	Linear	20	20
2 [2GB]	Tree	1	2
3 [2GB]	Tree	3	4
4 [2GB]	Tree	7	8
5 [2GB]	Tree	15	16
6 [2GB]	Tree	31	32
7 [2GB]	Single	1	15
8 [2GB]	Single	1	25
9 [2GB]	Single	1	50
10 [2GB]	Single	1	100

5.3. Exploratory Analysis

5.3.1. Volatility Analysis

By investigating memory samples with Volatility, we were able to determine the controller's execution time. Using Volatility's *linux bash* plugin, we restored a log containing command-line entries from the controller. These logs can identify if unauthorized access has occurred and if an adversary has been misusing or misconfiguring a controller in secret. Figure 5.3 displays a sample of the results from our memory analysis, which depicts the various commands submitted into the controller's terminal. In this figure, you can see the commands run to collect the device's memory dump using Surge Collect.

```
1957 bash               2022-11-01 20:27:21 UTC+0000   cd distribution-karaf-0.6.4-Carbon
1957 bash               2022-11-01 20:27:21 UTC+0000   cd bin/
1957 bash               2022-11-01 20:27:21 UTC+0000   export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
1957 bash               2022-11-01 20:27:21 UTC+0000   ./karaf
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd mnt/
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd /mnt
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd hgfs/
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd ODL volatility
1957 bash               2022-11-01 20:27:21 UTC+0000   cd 'ODL volatility'
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cp surge-collect-linux-ab.zip /root
1957 bash               2022-11-01 20:27:21 UTC+0000   cd
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   mkdir
1957 bash               2022-11-01 20:27:21 UTC+0000   mkdir surge
1957 bash               2022-11-01 20:27:21 UTC+0000   mv surge-collect-linux-ab.zip /surge
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd surge/
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   ls -a
1957 bash               2022-11-01 20:27:21 UTC+0000   cd
1957 bash               2022-11-01 20:27:21 UTC+0000   ls
1957 bash               2022-11-01 20:27:21 UTC+0000   cd /surge
1957 bash               2022-11-01 20:27:21 UTC+0000   cd ~/odl
1957 bash               2022-11-01 20:27:21 UTC+0000   cd
1957 bash               2022-11-01 20:27:21 UTC+0000   cd ..
1957 bash               2022-11-01 20:27:21 UTC+0000   cd surge
1957 bash               2022-11-01 20:27:21 UTC+0000   exit
1957 bash               2022-11-01 20:27:24 UTC+0000   ls
1957 bash               2022-11-01 20:27:29 UTC+0000   cd
1957 bash               2022-11-01 20:27:31 UTC+0000   sudo surge
1957 bash               2022-11-01 20:27:31 UTC+0000   ls
1957 bash               2022-11-01 20:27:35 UTC+0000   sudo surge
1957 bash               2022-11-01 20:27:36 UTC+0000   cd surge
1957 bash               2022-11-01 20:27:38 UTC+0000   ls
1957 bash               2022-11-01 20:28:06 UTC+0000   sudo chmod +x surge-collect-linux-ab
1957 bash               2022-11-01 20:28:16 UTC+0000   ./surge-collect-linux-ab primitivejoysalt
1957 bash               2022-11-01 20:28:40 UTC+0000   sudo ./surge-collect-linux-ab primitivejoysalt
```

Figure 5.3. Commands run on the SDN controller

Using the "linux netstat" plugin from Volatility, we were able to discover active connections to the SDN controller. The ability to view connections reveals whether or not any unauthorized devices are connected to a network. After detecting malicious connections, additional steps might be taken to block an attacker, such as banning specific IP addresses from the network. Figure 5.4 is an example of output from the controller. This output displays a variety of established connections as well as the listening ports of the controller.

Also, by examining the controller's process list, it is possible to identify any extra processes. This analysis is helpful when figuring out if malware is running on the con-

```
TCP      192.168.153.130 :   22 192.168.153.1   :63383 ESTABLISHED                 sshd/1335
UNIX 39929                   sshd/1335
UNIX 39794                   sshd/1335
TCP      ::1             : 6010 ::              :    0 LISTEN                      sshd/1335
TCP      127.0.0.1       : 6010 0.0.0.0         :    0 LISTEN                      sshd/1335
UNIX 41124                   sshd/1357
TCP      192.168.153.130 :   22 192.168.153.1   :63384 ESTABLISHED                 sshd/1357
UNIX 41099                   sshd/1357
UNIX 40690                   sshd/1357
UNIX 41124                   sshd/1432
TCP      192.168.153.130 :   22 192.168.153.1   :63384 ESTABLISHED                 sshd/1432
UNIX 40689                   sshd/1432
UNIX 41099                   sshd/1432
TCP      ::1             : 6011 ::              :    0 LISTEN                      sshd/1432
TCP      127.0.0.1       : 6011 0.0.0.0         :    0 LISTEN                      sshd/1432
UNIX 41283                   sudo/1465
UNIX 41301                     su/1471
UNIX 41365                   java/1491
UNIX 40803                   java/1491
TCP      ::              :34711 ::              :    0 LISTEN                      java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35276 ESTABLISHED                java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35292 ESTABLISHED                java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35304 ESTABLISHED                java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35320 ESTABLISHED                java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35336 ESTABLISHED                java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35348 ESTABLISHED                java/1491
TCP      ::ffff192.168.153.130: 6653 ::ffff192.168.153.128:35360 ESTABLISHED                java/1491
UNIX 41369                   java/1491
UNIX 41370                   java/1491
TCP      ::ffff127.0.0.1 :34125 ::              :    0 LISTEN                      java/1491
UNIX 41371                   java/1491
UNIX 41372                   java/1491
TCP      ::              : 8185 ::              :    0 LISTEN                      java/1491
TCP      ::              : 6653 ::              :    0 LISTEN                      java/1491
TCP      ::              : 6633 ::              :    0 LISTEN                      java/1491
UNIX 44358                   java/1491
UNIX 44359                   java/1491
TCP      ::              : 8101 ::              :    0 LISTEN                      java/1491
TCP      ::              : 1099 ::              :    0 LISTEN                      java/1491
TCP      ::              :44444 ::              :    0 LISTEN                      java/1491
TCP      ::              : 8181 ::              :    0 LISTEN                      java/1491
TCP      ::              : 8080 ::              :    0 LISTEN                      java/1491
TCP      ::ffff127.0.0.1 : 2550 ::              :    0 LISTEN                      java/1491
```

Figure 5.4. Connections to the SDN controller

troller during an investigation. Because If the controller is hacked, the entire network becomes vulnerable. Here, By applying memory forensics and network forensics techniques to the SDN controller's acquired memory, we were able to extract a significant amount of important information about the operational network and the network devices that were connected to it.

The user name and hashed password of the running controller are retrieved, which is valuable information for gaining access to the controller or other controllers on the same network if they share the same or similar credentials as shown in Figure 5.5.

{"userName":"odl","uid":1000,"gid":1000,"homeDirectory":"/home/odl","shell":"/bin/bash","privileged":{"hash
edPassword":["$6$6zchz/0vggHZKDDM$ypFgPH22KG9BlZ7f96YY/8WidoilPZyGogPnlsFLvOBGD3h.XagdLxRZtzbI3RBHeLwP0tLTp
zqeJOmiZGu0g."]},"passwordChangeNow":false,"lastPasswordChangeUSec":1645142400000000,"passwordChangeMaxUSec
":8639913600000000,"passwordChangeWarnUSec":604800000000}

Figure 5.5. Username and Hashed Password of the SDN controller

Figure 5.6 illustrates how the controller's IP was retrieved from the acquired memory of the ODL controller.

```
Volatility Foundation Volatility Framework 2.6.1
Interface        IP Address           MAC Address        Promiscous Mode
---------------- -------------------- ------------------ ---------------
lo               127.0.0.1            00:00:00:00:00:00  False
ens33            192.168.153.130      00:0c:29:42:ea:96  False
```

Figure 5.6. The IP Address of the SDN controller

In addition, we were able to extract the MAC and IP addresses of the hosts that were connected to the SDN controller, as shown in Figures 5.7 and 5.8, respectively.

stination":{"dest-node":"openflow:4","dest-tp":"openflow:4:2"}},{"link-id":"host:00:00:00:00:00:01/openflow:3:1","source":{"so
"destination":{"dest-node":"openflow:3","dest-tp":"openflow:3:1"}},{"link-id":"openflow:6:2/host:00:00:00:00:00:06","source":{
t-node":"host:00:00:00:00:00:06","dest-tp":"host:00:00:00:00:00:06"}},{"link-id":"host:00:00:00:00:00:02/openflow:3:2","source
:02"},"destination":{"dest-node":"openflow:3","dest-tp":"openflow:3:2"}},{"link-id":"openflow:5:3","source":{"source-node":"op
:1","dest-tp":"openflow:1:2"}},{"link-id":"openflow:4:3","source":{"source-node":"openflow:4","source-tp":"openflow:4:3"},"des
"openflow:5:2","source":{"source-node":"openflow:5","source-tp":"openflow:5:2"},"destination":{"dest-node":"openflow:7","dest-
enflow:7","source-tp":"openflow:7:3"},"destination":{"dest-node":"openflow:5","dest-tp":"openflow:5:2"}},{"link-id":"openflow:
tination":{"dest-node":"openflow:5","dest-tp":"openflow:5:1"}},{"link-id":"openflow:3:2/host:00:00:00:00:00:02","source":{"sou
de":"host:00:00:00:00:00:02","dest-tp":"host:00:00:00:00:00:02"}},{"link-id":"openflow:4:2/host:00:00:00:00:00:04","source":{"
-node":"host:00:00:00:00:00:04","dest-tp":"host:00:00:00:00:00:04"}},{"link-id":"openflow:7:1/host:00:00:00:00:00:07","source"
est-node":"host:00:00:00:00:00:07","dest-tp":"host:00:00:00:00:00:07"}},{"link-id":"openflow:6:1/host:00:00:00:00:00:05","sour
{"dest-node":"host:00:00:00:00:00:05","dest-tp":"host:00:00:00:00:00:05"}},{"link-id":"openflow:4:1/host:00:00:00:00:00:03","s
n":{"dest-node":"host:00:00:00:00:00:03","dest-tp":"host:00:00:00:00:00:03"}},{"link-id":"openflow:3:1/host:00:00:00:00:00:01"
tion":{"dest-node":"host:00:00:00:00:00:01","dest-tp":"host:00:00:00:00:00:01"}},{"link-id":"openflow:1:1","source":{"source-n
openflow:2","dest-tp":"openflow:2:3"}},{"link-id":"openflow:2:2","source":{"source-node":"openflow:2","source-tp":"openflow:2:
ink-id":"openflow:2:1","source":{"source-node":"openflow:2","source-tp":"openflow:2:1"},"destination":{"dest-node":"openflow:3

Figure 5.7. MAC Addressees connected to the SDN controller

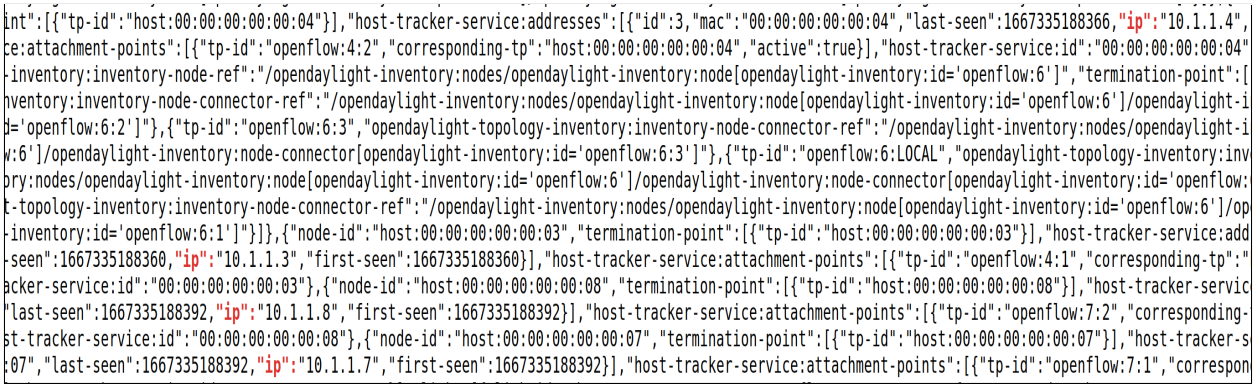

Figure 5.8. IP's connected to the SDN controller

Using Volatility's pslist plugin, we recovered a list of running processes. By analyzing the process list, unusual processes can be uncovered. This is helpful during an investigation for exploring if malware may be actively running.

5.3.2. Strings Analysis

By using the Linux *strings* and *grep* programs to scan the memory dump, we were able to locate the Host Tracker service of the ODL controller. Host Tracker is a service that gathers information about nodes in the network, such as their data layer address, switch type, port type, and network address, and provides APIs for accessing this information. Host Tracker's capabilities can be either static or dynamic. For dynamic activities, the Host Tracker uses ARP to monitor the status of the service. Using northbound APIs, the Host Tracker information is manually created in static mode. This operation stores in memory information about hosts connected to the network, including their IP addresses, MAC addresses, and the timestamps at which they were detected. We developed a Python tool capable of extracting this information.

5.4. SCoNDT

SCoNDT searches a memory dump using the Unix *strings* and *grep* tools for the host tracker service of the ODL controller. This service provides information about each host connected to the network, including its internal IP address, mac address, and the times of its first and last network connections. According to Figure 5.9, it generates this information as an HTML report file that may be viewed using a web browser.

ID	IP	Mac Address	First Seen	Last Seen
0	10.1.1.3	00:00:00:00:00:03	2023-01-16 22:47:15.531	2023-01-16 22:47:15.531
1	10.1.1.1	00:00:00:00:00:01	2023-01-16 22:47:15.626	2023-01-16 22:47:15.626
2	10.1.1.4	00:00:00:00:00:04	2023-01-16 22:47:15.644	2023-01-16 22:47:15.644
3	10.1.1.6	00:00:00:00:00:06	2023-01-16 22:47:15.677	2023-01-16 22:47:15.677
4	10.1.1.5	00:00:00:00:00:05	2023-01-16 22:47:15.694	2023-01-16 22:47:15.694
5	10.1.1.7	00:00:00:00:00:07	2023-01-16 22:47:15.696	2023-01-16 22:47:15.696
6	10.1.1.8	00:00:00:00:00:08	2023-01-16 22:47:15.701	2023-01-16 22:47:15.701
6		00:00\nw:1	1969-12-31 18:00:00.000	1969-12-31 18:00:00.000
7	10.1.1.10	00:00:00:00:00:0a	2023-01-16 22:47:15.702	2023-01-16 22:47:15.702
8	10.1.1.11	00:00:00:00:00:0b	2023-01-16 22:47:15.708	2023-01-16 22:47:15.708
9	10.1.1.12	"00:00:00:00:00:0c	2023-01-16 22:47:15.709	2023-01-16 22:47:15.709
9	10.1.1.12	00:00:00:00:00:0c	2023-01-16 22:47:15.709	2023-01-16 22:47:15.709
10	10.1.1.13	00:00:00:00:00:0d	2023-01-16 22:47:15.710	2023-01-16 22:47:15.710
11	10.1.1.14	00:00:00:00:00:0e	2023-01-16 22:47:15.716	2023-01-16 22:47:15.716
12	10.1.1.15	00:00:00:00:00:0f	2023-01-16 22:47:15.717	2023-01-16 22:47:15.717
13	10.1.1.16	00:00:00:00:00:10	2023-01-16 22:47:15.726	2023-01-16 22:47:15.726
14	10.1.1.19	00:00:00:00:00:13	2023-01-16 22:47:15.735	2023-01-16 22:47:15.735
15	10.1.1.9	00:00:00:00:00:09	2023-01-16 22:47:15.741	2023-01-16 22:47:15.741
16	10.1.1.2	00:00:00:00:00:02	2023-01-16 22:47:15.749	2023-01-16 22:47:15.749
17	10.1.1.20	00:00:00:00:00:14	2023-01-16 22:47:15.750	2023-01-16 22:47:15.750
18	10.1.1.17	00:00:00:00:00:11	2023-01-16 22:47:15.785	2023-01-16 22:47:15.785
19	10.1.1.18	00:00:00:00:00:12	2023-01-16 22:47:15.792	2023-01-16 22:47:15.792

Username	HashedPassword
odl	$6zchz/0vggHZKDDM$ypFgPH22KG9BlZ7f96YY/8Widoil...
root	$3M0TY4hV0GQT..k4$j3YoGWswueW4FAW6LB6ZydlAlnOd...

Figure 5.9. SCoNDT Output From Memory Sample 1

Due to the fact that SCoNDT reads volatile memory, there may be instances where data is incomplete. Host activity triggers the Host Tracker service. Hence, if a host has an extended period of inactivity during a memory capture, it is likely that some data is missing, as shown in ID 4 of Figure 5.10.

ID	IP	Mac Address	First Seen	Last Seen
7	10.1.1.6	00:00:00:00:00:06	2022-11-01 13:39:48.415	2022-11-01 13:39:48.415
4		00	1969-12-31 16:00:00.000	1969-12-31 16:00:00.000
0	10.1.1.2	00:00:00:00:00:02	2022-11-01 13:39:48.270	2022-11-01 13:39:48.270
2	10.1.1.1	00:00:00:00:00:01	2022-11-01 13:39:48.365	2022-11-01 13:39:48.365
3	10.1.1.4	00:00:00:00:00:04	2022-11-01 13:39:48.366	2022-11-01 13:39:48.366
1	10.1.1.3	00:00:00:00:00:03	2022-11-01 13:39:48.360	2022-11-01 13:39:48.360
5	10.1.1.8	00:00:00:00:00:08	2022-11-01 13:39:48.392	2022-11-01 13:39:48.392
6	10.1.1.7	00:00:00:00:00:07	2022-11-01 13:39:48.392	2022-11-01 13:39:48.392

Figure 5.10. SCoNDT Output From Memory Sample 4

The Host Tracker service is part of the L2 switch project `https://test-odldocs.readthedocs.io/en/latest/user-guide/l2switch-user-guide.html#overview`. It is necessary for this to be installed on the ODL in order to use the SCoNDT tool. This is done by installing the ODL-L2 switch-switch-ui feature. After installing, a configuration file needs to be created and placed in the config directory of the ODL controller as shown in Figure 5.11. This modification includes switching the L2 switch from proactive flood mode to reactive flood mode by creating a new XML file with the following content to the directory: opendaylight-15.1.0 /etc/opendaylight/datastore/initial/config.

```
<?xml version="1.0" encoding="UTF-8"?>
 <arp-handler-config
    xmlns="urn:opendaylight:packet:arp-handler-config">
  <is-proactive-flood-mode>false</is-proactive-
  flood-mode>
</arp-handler-config>
```

Figure 5.11. XML File to Change Configuration of L2 Switch in ODL

5.4.1. SCoNDT Algorithm

Algorithm 3 provides insight into how the SCoNDT tool functions. To start, the input file is searched for the string signature related to the host tracker service, usernames, and hashed passwords. After all relevant strings are found, they are parsed and the relevant information is saved. Once all information is parsed, an HTML table is generated as a report.

Algorithm 3: The Pseudocode of SCoNDT

```
1  Input: Memory Dump of The SDN Controller (ODL)
2  Output: ODL Controller's information
   function MAIN
   strings = get_strings()
   hostData = parse_host_string(strings)
   userData = parse_user_string(strings)
   generate_report(hostData, userData)
   end function
   return HTMLfile(ODLInformationReport)
```

We detail the description of SCoNDT:

- Function `get_strings`

  Input: Memory sample file path

  Unix strings and grep operations find signatures for the host tracker service and username and hashed password combinations

  Returns: A list of strings

- Function `parse_host_string`

  Input: List of strings

  Parses strings for host id, IP address, mac address, first seen date, and last seen date, and stores that information in a host object

  Returns: A host object

- Function `parse_user_string`

  Input: List of strings

  Parses strings for username and hashed password, and stores that information in a user object

  Returns: A user object

- Function `generate_report`

  Input: Two data frames and a file name

  Creates an HTML file with a user-provided name or the default name Drop duplicates from the host information data frame and username and hashed password data frame

  Returns: Write data frames to the file

- Function `Main`

  Input: Optional flags, Memory sample file path, Report file name

  Check for -h option, if true show the help page

  Call get_strings function, the store returned list of strings in the search variable

  Call parse_host_string function and store each host object in the data frame

  Call parse_user_string function, store each username and hashed password combination in the data frame

  Call generate_report function

5.4.2. SCoNDT Usage

The tool provides two main functions:

- Parsing network data from ODL computer memory images.
- Parsing usernames and hashed passwords from ODL computer memory images.

SCoNDT can be used with the following commands:

```
python2 scondt.py -h
python2 scondt.py inputFileName outputFileName
```

In order to learn more about SCoNDT, you may use the -h option to bring up a menu with explanations. Memory dump files are only accepted as input, and the report file's name is determined by the name of the file that is written out. In the absence of an explicit file name, the output will be saved under the name "Result". An HTML report will be generated by the SCoNDT and saved in the same folder as the scondt.py file after running this command.

5.5. Evaluation of SCoNDT

Here, we put SCoNDT through its tests by putting it to the test on several memory samples we pulled from the ODL controller.

5.5.1. Memory Samples

The different memory samples used to evaluate SCoNDT are listed in Table 5.4. To determine how many hosts SCoNDT could recover, we created simulated networks with varying numbers of hosts and IP addresses.

5.5.2. Results

Data from Table 5.4 shows that, with the exception of Memory Sample 4, SCoNDT was robust enough to recover both the total number of hosts present in each memory sample that was tested as well as the individual IP and MAC addresses of each host connected to the ODL controller. SCoNDT was also able to retrieve the ODL controller's user name, the root user, and hashed passwords. SCoNDT additionally provides information about what times a device was first and last seen connected to the network, which is critical for investigations. SCoNDT reconstructed data about the SDN network in times ranging from 14.96 seconds to 23.65 seconds. Even with memory smearing, SCoNDT was capable to detect all host IPs in nine of ten memory samples. For the one test where all host IPs were not recovered, SCoNDT still recovered all information but one of the host IP addresses. We also note that for the third memory sample, the full host IP was not recoverable, and only part of the host IP was reconstructed. Overall, we deem this as an acceptable result since memory is volatile, and memory smearing is expected.

5.5.3. Performance

SCoNDT was evaluated on an Intel Core i7-4870HQ processor. The average time for SCoNDT to complete its operation was just 17.9 seconds. Unix strings operation has spent an average of 17.7 seconds during the runtime. The remaining 0.2 milliseconds were used for doing things like parsing the host data, identifying the username, hashed passwords, and generating the report. File sizes for the memory samples were 2GB. When the size of the string is increased, the runtime also increases since the string's operation consumes the majority of the operation time.

Table 5.4. Results From Running SCoNDT on Memory Samples in Table 5.3

Memory Sample	# of Hosts	# of host IPs Detected	User Name Detected	Hashed Password Detected	Processing Time
1 [2GB]	20	20	Yes	Yes	14.9689 s
2 [2GB]	2	2	Yes	Yes	23.6577 s
3 [2GB]	4	4	Yes	Yes	16.2847 s
4 [2GB]	8	7	Yes	Yes	15.5904 s
5 [2GB]	16	16	Yes	Yes	16.5658 s
6 [2GB]	32	32	Yes	Yes	18.9154 s
7 [2GB]	15	15	Yes	Yes	18.2491 s
8 [2GB]	25	25	Yes	Yes	15.2733 s
9 [2GB]	50	50	Yes	Yes	17.5244 s
10 [2GB]	100	100	Yes	Yes	22.2256 s

A network forensics investigation involves collecting as much evidence from the network as feasible. Our objective was to determine which digital evidence could be recovered from the SDN controller's memory. We have been able to obtain forensically relevant data based on our results. Volatility allowed us to recover ODL controller command-line inputs, processes, and open connections and ports. Our SCoNDT tool was also highly effective at reconstructing the number of hosts set by the SDN controller. In every memory dump tested, it was also able to recover the username and hashed password.

5.6. Discussion

A network forensics investigation consists of gathering as much evidence as possible from the network. Our goal was to explore what artifacts could be extracted from the ODL SDN controller's memory to inform digital investigations. Based on our findings, we have been able to retrieve forensically relevant data. The host IP addresses and MAC addresses that are recovered can be used to identify which devices are connected to the network and can assist in identifying unknown devices. The recovered timestamps showing when a device was first and last seen can also be used to determine how long a foreign device has been on the network. Volatility allowed us to recover command-line events, processes, and open connections and ports on the ODL controller. Our SCoNDT tool also

showed strong efficacy in being able to reconstruct the number of hosts configured through the SDN controller. It was also able to recover the username and hashed password in each tested case.

5.7. Summary and Future Work

Our study demonstrates that the controller's memory is of forensic relevance and that existing approaches can be used to retrieve digital evidence in SDN investigations. Our research aimed to discover if useful information could be extracted from the SDN controller's memory using existing memory forensics methods. We were able to successfully get the process list and network information by probing the memory with volatility. We then looked at the storage to see if there was any further data retrievable that is unavailable to the current tools. Using the L2 switch project of ODL, we discovered that the host tracker service stores host-related data in the controller's memory. Using the host tracker service allowed us to get details about the hosts that connected to the network. We developed SCoNDT after examining the memory layout of the host and the user. SCoNDT performs a search of the acquired memory, gets the corresponding user and host data, and then provides a report in HTML format containing the retrieved data. Evaluation of SCoNDT showed that most hosts on the network might be located in memory. This holds for a wide range of network sizes and configurations. We also found that SCoNDT can be executed in a reasonable time frame.

In order to build upon our successes, we should consider extending SCoNDT's capabilities to work with additional SDN controllers using our established methodology. Further research needs to investigate the testing of bigger networks. 100 hosts were con-

nected to the greatest network we tested. By examining data from bigger networks, it may be possible to gain a deeper understanding of how much data is stored in the controller's memory and the limits of retrieving that data. As hosts are added and removed from the network, it is also crucial to understand what changes occur in the controller's memory. Lastly, in our experiments, we simulated a network by connecting our controller to mininet, a simulation program. Important future steps include applying our technique to a network operating on actual hardware to guarantee that simulated and physical networks are the same.

Chapter 6. Conclusion

Using a virtualized network, the SDN architecture converts traditional networks into scalable, customizable infrastructures. Because of this, software-defined networking (SDN) is set to become the de facto standard for future networks. This research intends to explore some aspects of the security of the SDN, in particular the security of the SDN control plane, beginning with the SDN controller's single point of failure against the DDoS attacks that are targeted at the RYU controller. Following this, the next research proposes a framework for memory and network forensics for the ODL SDN controller. This framework is called the SDN Memory and Network Forensics Framework (SCoNDT). In the final work, the research results in the development of a lightweight solution for the detection and prevention of MITM attacks (ARP poisoning attacks) in SDN networks.

In this study, we aim to mitigate the risk of a total breakdown in a single-point-of-failure SDN control plane by recommending a distributed open-source controller architecture. This design was used to enhance the security of the RYU SDN controller against the DDoS attacks that aim to take down the primary controller in the SDN control plane, which leads to the failure of the entire SDN network. Our proposed architecture for open-source RYU SDN controllers involves three different components operating together to achieve a reliable system against DDoS attacks: first, we have multiple open-source RYU SDN controllers as the brain of the SDN; second, the Zookeeper service, which is in charge of multi-controller and primary controller election; and third, the Redis database service, which provides high availability and reliability. Mininet, Open vSwitch Raspberry Pi, and Ryu controllers are used in our experiments to further test and evaluate the design architecture. The design shows a very effective solution to the problem of a single SDN

controller failing during a DDoS attack, with smooth operation and performance and no packet loss during the DDoS attack on the primary controller. To further prove the effectiveness of our concept, we intend to implement it in a real SDN environment in future work.

An ARP poisoning is a type of attack that can compromise the RYU controller if an attacker is able to successfully intercept communications and data transmissions while using it. If an attacker is able to get access to the controller or the communication channels, then they are in a position to manipulate the network in any way they choose. This project proposes Ryu-ARP, which stands for RYU SDN Controller ARP Poisoning Security Application. Ryu-ARP is a lightweight security application that can detect and prevent ARP poisoning threats. It is designed to be used with the RYU SDN controller. The Ryu-ARP demonstrates a technique that is not only efficient but also reliable for detecting and preventing ARP poisoning attacks. After putting the tools through their tests on a number of different network architectures and topologies, we found that they consistently deliver a solid performance when it comes to detecting and preventing ARP poisoning.

Finally, SDN simplifies network administration. It is essential that the security and forensics of this technology be properly investigated due to its widespread application. Our research demonstrates that existing digital forensics approaches may be used to retrieve digital evidence in SDN investigations and that the controller's memory is forensically significant. In this research, we analyzed existing memory forensics methods to evaluate if pertinent data might be extracted from the memory of an SDN controller. We were able to get the process list and network information after successfully utilizing volatility to investigate the memory. In addition, we investigated the memory to discover what more

data may be obtained that existing techniques cannot locate. We've discovered that the ODL L2 switch project's host tracker service stores information about hosts in the controller's memory. By doing a search for the host tracker service, we were able to get information on the network-connected hosts. We also discovered that specific information about the controller's users, such as usernames and hashed passwords, is saved in memory. After understanding how host and user data are saved in memory, we developed the SCoNDT tool. SCoNDT searches the acquired memory for user and host data, retrieves this information, and generates an HTML report including all the data discovered. Evaluation of SCoNDT demonstrated that the majority of network-connected hosts may be identified in memory. This is true for several network sizes and topologies. Our assessment also shows that SCoNDT can be executed within a realistic timeframe.

Bibliography

[1] https://www.ettercap-project.org/downloads.html.

[2] http://naughty.monkey.org/dugsong/dsniff.

[3] Cain and able. http://www.oxid.it/cain.html.

[4] Redis. https://redis.com/?utm_source=google&utm_medium=cpc&utm_campaign=redis360-brand-us-15152278745&utm_term=redis&utm_content=.

[5] Yersinia. http://yersinia.sourceforge.net.

[6] AbdelRahman Abdou, Paul C Van Oorschot, and Tao Wan. Comparative analysis of control plane security of sdn and conventional networks. *IEEE Communications Surveys & Tutorials*, 20(4):3542–3559, 2018.

[7] Inikpi O Ademu, Chris O Imafidon, and David S Preston. A new approach of digital forensic model for digital forensic investigation. *IJACSA) International Journal of Advanced Computer Science and Applications*, 2(12), 2011.

[8] Tinku Adhikari, Malay Kule, and Ajoy Kumar Khan. An ecdh and aes based encryption approach for prevention of mitm in sdn southbound communication interface. 2022.

[9] Waqas Ahmed, Faisal Shahzad, Abdul Rehman Javed, Farkhund Iqbal, and Liaqat Ali. Whatsapp network forensics: Discovering the ip addresses of suspects. In *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–7. IEEE, 2021.

[10] Nisha Ahuja, Gaurav Singal, Debajyoti Mukhopadhyay, and Ajay Nehra. Ascertain the efficient machine learning approach to detect different arp attacks. *Computers and Electrical Engineering*, 99:107757, 2022.

[11] Muhammet Fatih Akbaş, Enis Karaarslan, and Cengiz Güngör. A preliminary survey on the security of software-defined networks. *International Journal of Applied Mathematics Electronics and Computers*, (Special Issue-1):184–189, 2016.

[12] Mohammed I Al-Saleh, Ziad A Al-Sharif, and Luay Alawneh. Network reconnaissance investigation: A memory forensics approach. In *2019 10th International Conference on Information and Communication Systems (ICICS)*, pages 36–40. IEEE, 2019.

[13] Emad Alasadi and Hamed S Al-Raweshidy. Ssed: Servers under software-defined network architectures to eliminate discovery messages. *IEEE/ACM Transactions on*

Networking, 26(1):104–117, 2017.

[14] Talal Alharbi, Dario Durando, Farzaneh Pakzad, and Marius Portmann. Securing arp in software defined networks. In *2016 IEEE 41st conference on local computer networks (LCN)*, pages 523–526. IEEE, 2016.

[15] Talal Alharbi and Marius Portmann. Sproxy arp-efficient arp handling in sdn. In *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 179–184. IEEE, 2016.

[16] Ali Faiq Ali and Wesam S Bhaya. Software defined network (sdn) security against address resolution protocol poisoning attack. *Journal of Computational and Theoretical Nanoscience*, 16(3):956–963, 2019.

[17] Amer Aljaedi, Dale Lindskog, Pavol Zavarsky, Ron Ruhl, and Fares Almari. Comparative analysis of volatile memory forensics: live response vs. memory imaging. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 1253–1258. IEEE, 2011.

[18] Izzat Alsmadi and Mamoun Alazab. A model based approach for the extraction of network forensic artifacts. In *2017 Cybersecurity and Cyberforensics Conference (CCC)*, pages 16–18. IEEE, 2017.

[19] Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Radha Poovendran. Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks. In *Proceedings of the 10th ACM symposium on information, computer and communications security*, pages 639–644, 2015.

[20] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354, 2017.

[21] Mathieu Bouet, Kévin Phemius, and Jérémie Leguay. Distributed sdn for mission-critical networks. In *2014 IEEE Military Communications Conference*, pages 942–948. IEEE, 2014.

[22] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. In *IEEE Local Computer Network Conference*, pages 408–415. IEEE, 2010.

[23] Danilo Bruschi, Alberto Ornaghi, and Emilia Rosti. S-arp: a secure address resolution protocol. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 66–74. IEEE, 2003.

[24] Andrew Case and Golden G Richard III. Memory forensics: The path forward. *Digital Investigation*, 20:23–33, 2017.

[25] Peter Casey, Rebecca Lindsay-Decusati, Ibrahim Baggili, and Frank Breitinger. Inception: virtual space in memory space in real space–memory forensics of immersive virtual reality with the htc vive. *Digital Investigation*, 29:S13–S21, 2019.

[26] Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, and Juan Felipe Botero Vega. Security in sdn: A comprehensive survey. *Journal of Network and Computer Applications*, 159:102595, 2020.

[27] Michael Cohen. Scanning memory with yara. *Digital Investigation*, 20:34–43, 2017.

[28] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A survey of man in the middle attacks. *IEEE communications surveys & tutorials*, 18(3):2027–2051, 2016.

[29] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys Tutorials*, 18(3):2027–2051, 2016.

[30] Jacob H Cox, Russell J Clark, and Henry L Owen. Leveraging sdn for arp security. In *SoutheastCon 2016*, pages 1–8. IEEE, 2016.

[31] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: detecting security attacks in software-defined networks. In *Ndss*, volume 15, pages 8–11, 2015.

[32] Zakaria El Mrabet, Naima Kaabouch, Hassan El Ghazi, and Hamid El Ghazi. Cyber-security in smart grid: Survey and challenges. *Computers & Electrical Engineering*, 67:469–482, 2018.

[33] Lubna Fayez Eliyan and Roberto Di Pietro. Dos and ddos attacks in software defined networks: A survey of existing solutions and research challenges. *Future Generation Computer Systems*, 122:149–171, 2021.

[34] Lubna Fayez Eliyan and Roberto Di Pietro. Dos and ddos attacks in software defined networks: A survey of existing solutions and research challenges. *Future Gener. Comput. Syst.*, 122:149–171, 2021.

[35] Antoine Feghali, Rima Kilany, and Maroun Chamoun. Sdn security problems and solutions analysis. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, pages 1–5. IEEE, 2015.

[36] Paulo Fonseca, Ricardo Bennesby, Edjard Mota, and Alexandre Passito. A replication component for resilient openflow-based networking. In *2012 IEEE Network*

operations and management symposium, pages 933–939. IEEE, 2012.

[37] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. Mininet-wifi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 384–389. IEEE, 2015.

[38] Fahad M Ghabban, Ibrahim M Alfadli, Omair Ameerbakhsh, Amer Nizar AbuAli, Arafat Al-Dhaqm, and Mahmoud Ahmad Al-Khasawneh. Comparative analysis of network forensic tools and network forensics processes. In *2021 2nd International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, pages 78–83. IEEE, 2021.

[39] Evangelos Haleplidis, Spyros Denazis, Kostas Pentikousis, Jamal Hadi Salim, and Odysseas Koufopavlou. Sdn layers and architecture terminology. *Internet Engineering Task Force, Internet Draft, Aug*, 2014.

[40] Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, J Hadi Salim, David Meyer, and Odysseas Koufopavlou. Software-defined networking (sdn): Layers and architecture terminology. Technical report, 2015.

[41] Ryan Hand, Michael Ton, and Eric Keller. Active security. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pages 1–7, 2013.

[42] Muhammad Reazul Haque, Sameer Ali, Saw Chin Tan, Zulfadzli Yusoff, Lee Ching Kwang, Ir Rizaludin Kaspin, and Salvatore Renato Ziri. Motivation of ddos attack-aware in software defined networking controller placement. In *2017 International Conference on Computer and Applications (ICCA)*, pages 36–42. IEEE, 2017.

[43] Sherin Hijazi and Mohammad S Obaidat. A new detection and prevention system for arp attacks using static entry. *IEEE Systems Journal*, 13(3):2732–2738, 2018.

[44] Sherin Hijazi and Mohammad S Obaidat. Address resolution protocol spoofing attacks and security approaches: A survey. *Security and Privacy*, 2(1):e49, 2019.

[45] A Hingne and S Jain. A survey on various detection and prevention mechanism for mitm and arp attacks. *Int J Innov Res Comput Commun Eng*, 4(11):19918–19924, 2016.

[46] Tao Hu, Zehua Guo, Peng Yi, Thar Baker, and Julong Lan. Multi-controller based software-defined networking: A survey. *IEEE access*, 6:15980–15996, 2018.

[47] Harman Y Ibrahim, Parishan M Ismael, Ali A Albabawat, and Ahmad B Al-Khalil. A secure mechanism to prevent arp spoofing and arp broadcasting in sdn. In *2020 International Conference on Computer Science and Software Engineering (CSASE)*,

pages 13–19. IEEE, 2020.

[48] Naseela Jehan and Aneesh M Haneef. Scalable ethernet architecture using sdn by suppressing broadcast traffic. In *2015 Fifth international conference on advances in computing and communications (ICACC)*, pages 24–27. IEEE, 2015.

[49] Samuel Jero, Xiangyu Bu, Cristina Nita-Rotaru, Hamed Okhravi, Richard Skowyra, and Sonia Fahmy. Beads: Automated attack discovery in openflow-based sdn systems. In *Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings*, pages 311–333. Springer, 2017.

[50] Jose M Jimenez, José Oscar Romero Martínez, Albert Rego Máñez, and Jaime Lloret. Analyzing the performance of software defined networks vs real networks. *International Journal on Advances in Networks and Services*, 9(3-4):107–116, 2016.

[51] Yury Jimenez, Cristina Cervelló-Pastor, and Aurelio J Garcia. On the controller placement for designing a distributed sdn control layer. In *2014 IFIP Networking Conference*, pages 1–9. IEEE, 2014.

[52] Filip Karpisek, Ibrahim Baggili, and Frank Breitinger. Whatsapp network forensics: Decrypting and understanding the whatsapp call signaling messages. *Digital Investigation*, 15:110–118, 2015.

[53] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. Mininet as software defined networking testing platform. In *International conference on communication, computing & systems (ICCCS)*, pages 139–42, 2014.

[54] Faris Keti and Shavan Askar. Emulation of software defined networks using mininet in different simulation environments. In *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, pages 205–210. IEEE, 2015.

[55] Zainab Khalid, Farkhund Iqbal, Faouzi Kamoun, Mohammed Hussain, and Liaqat Ali Khan. Forensic analysis of the cisco webex application. In *2021 5th Cyber Security in Networking Conference (CSNet)*, pages 90–97. IEEE, 2021.

[56] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Mohsen Guizani, and Muhammad Khurram Khan. Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art. *IEEE Communications Surveys & Tutorials*, 19(1):303–324, 2016.

[57] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[58] Anand Krishnamurthy, Shoban P Chandrabose, and Aaron Gember-Jacobson. Pratyaastha: an efficient elastic distributed sdn control plane. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 133–138, 2014.

[59] Maxime Lagrasse, Avinash Singh, Howard Munkhondya, Adeyemi Ikuesan, and Hein Venter. Digital forensic readiness framework for software-defined networks using a trigger-based collection mechanism. In *Proceedings of the 15th International Conference on Cyber Warfare and Security, ICCWS*, pages 296–305. ACPi Norfolk, Virginia, 2020.

[60] Jishuai Li, Tengfei Tu, Yongsheng Li, Sujuan Qin, Yijie Shi, and Qiaoyan Wen. Dosguard: Mitigating denial-of-service attacks in software-defined networks. *Sensors*, 22(3):1061, 2022.

[61] Duohe Ma, Zhen Xu, and Dongdai Lin. Defending blind ddos attack on sdn based on moving target defense. In *International Conference on Security and Privacy in Communication Networks*, pages 463–480. Springer, 2014.

[62] Huan Ma, Hao Ding, Yang Yang, Zhenqiang Mi, James Yifei Yang, and Zenggang Xiong. Bayes-based arp attack detection algorithm for cloud centers. *Tsinghua science and technology*, 21(1):17–28, 2016.

[63] Ricardo Macedo, Rafael de Castro, Aldri Santos, Yacine Ghamri-Doudane, and Michele Nogueira. Self-organized sdn controller cluster conformations against ddos attacks effects. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[64] Andrew Mahr, Meghan Cichon, Sophia Mateo, Cinthya Grajeda, and Ibrahim Baggili. Zooming into the pandemic! a forensic analysis of the zoom application. *Forensic Science International: Digital Investigation*, 36:301107, 2021.

[65] Lusani Mamushiane, Albert Lysko, and Sabelo Dlamini. A comparative evaluation of the performance of popular sdn controllers. In *2018 Wireless Days (WD)*, pages 54–59. IEEE, 2018.

[66] Modhuparna Manna, Andrew Case, Aisha Ali-Gombe, and Golden G Richard III. Memory analysis of. net and. net core applications. *Forensic Science International: Digital Investigation*, 42:301404, 2022.

[67] Mohammad Z Masoud, Yousf Jaradat, and Ismael Jannoud. On preventing arp poisoning attack utilizing software defined network (sdn) paradigm. In *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pages 1–5. IEEE, 2015.

[68] Jitta Sai Meghana, T Subashri, and KR Vimal. A survey on arp cache poisoning

and techniques for detection and mitigation. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–6. IEEE, 2017.

[69] Natarajan Meghanathan, Sumanth Reddy Allam, and Loretta A Moore. Tools and techniques for network forensics. *arXiv preprint arXiv:1004.0570*, 2010.

[70] Dmitrij Melkov and Sarunas Paulikas. Security benefits and drawbacks of software-defined networking. In *2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–4. IEEE, 2021.

[71] Seyed Mohammad Mousavi and Marc St-Hilaire. Early detection of ddos attacks against sdn controllers. In *2015 international conference on computing, networking and communications (ICNC)*, pages 77–81. IEEE, 2015.

[72] Srinivas Mukkamala and Andrew H Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence*, 1(4):1–17, 2003.

[73] Howard Munkhondya, Adeyemi Ikuesan, and Hein Venter. Digital forensic readiness approach for potential evidence preservation in software-defined networks. In *ICCWS 2019 14th International Conference on Cyber Warfare and Security: ICCWS*, volume 268, 2019.

[74] Jargalsaikhan Narantuya, Seunghyun Yoon, Hyuk Lim, Jin-Hee Cho, Dong Seong Kim, Terrence Moore, and Frederica Nelson. Sdn-based ip shuffling moving target defense with multiple sdn controllers. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Supplemental Volume (DSN-S)*, pages 15–16. IEEE, 2019.

[75] Gopi Nath Nayak and Shefalika Ghosh Samaddar. Different flavours of man-in-the-middle attack, consequences and feasible solutions. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 5, pages 491–495. IEEE, 2010.

[76] Ajay Nehra, Meenakshi Tripathi, and Manoj Singh Gaur. Ficur: Employing sdn programmability to secure arp. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–8. IEEE, 2017.

[77] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications surveys & tutorials*, 16(3):1617–1634, 2014.

[78] Hannah Nyholm, Kristine Monteith, Seth Lyles, Micaela Gallegos, Mark DeSantis,

John Donaldson, and Claire Taylor. The evolution of volatile memory forensics. *Journal of Cybersecurity and Privacy*, 2(3):556–572, 2022.

[79] R Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, Rogério Luiz Iope, and Ligia Rodrigues Prete. L3-arpsec–a secure openflow network controller module to control and protect the address resolution protocol. *XXXIII Simpósio Brasileiro De Telecomunicações–(SBrT2015)*, pages 158–162, 2015.

[80] Mudit Kalpesh Pandya, Sajad Homayoun, and Ali Dehghantanha. Forensics investigation of openflow-based sdn platforms. In *Cyber Threat Intelligence*, pages 281–296. Springer, 2018.

[81] Emmanuel S Pilli, RC Joshi, and Rajdeep Niyogi. A generic framework for network forensics. *International Journal of Computer Applications*, 1(11):1–6, 2010.

[82] S Prajapati and Zishan Noorani. A survey on arp poisoning and techniques for detection and prevention. *Int. J. Adv. Res. Innov. Ideas Educ*, 3:594–601, 2017.

[83] Ankit Rao, Shrikant Auti, Akhil Koul, and Gauri Sabnis. High availability and load balancing in sdn controllers. *Int J Trend Res Dev*, 3(2):310–314, 2016.

[84] Danda B Rawat and Swetha R Reddy. Software defined networking architecture, security and energy efficiency: A survey. *IEEE Communications Surveys & Tutorials*, 19(1):325–346, 2016.

[85] René Rietz, Radoslaw Cwalinski, Hartmut König, and Andreas Brinner. An sdn-based approach to ward off lan attacks. *Journal of Computer Networks and Communications*, 2018:1–12, 2018.

[86] RYU. `https://github.com/faucetsdn/ryu`, 2022.

[87] Kshira Sagar Sahoo, Bibhudatta Sahoo, Ratnakar Dash, and Nachiketa Jena. Optimal controller selection in software defined network using a greedy-sa algorithm. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 2342–2346. IEEE, 2016.

[88] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. *Sdn Security: A Survey*. 2013.

[89] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, 2013.

[90] Zawar Shah and Steve Cosgrove. Mitigating arp cache poisoning attack in software-defined networking (sdn): a survey. *Electronics*, 8(10):1095, 2019.

[91] Muhammad Shakil, Alaelddin Fuad Yousif Mohammed, Rajakumar Arul, Ali Kashif Bashir, and Jun Kyun Choi. A novel dynamic framework to detect ddos in sdn using metaheuristic clustering. *Transactions on Emerging Telecommunications Technologies*, 33(3):e3622, 2022.

[92] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th central & eastern european software engineering conference in russia*, pages 1–6, 2013.

[93] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424, 2013.

[94] Leslie F Sikos. Packet analysis for network forensics: A comprehensive survey. *Forensic Science International: Digital Investigation*, 32:200892, 2020.

[95] Daniel Spiekermann and Tobias Eggendorfer. Challenges of network forensic investigation in virtual networks. *Journal of Cyber Security and Mobility*, pages 15–46, 2016.

[96] Vrizlynn LL Thing, Kian-Yong Ng, and Ee-Chien Chang. Live memory forensics of mobile phones. *digital investigation*, 7:S74–S82, 2010.

[97] V Thirupathi, CH Sandeep, Naresh Kumar, and P Pramod Kumar. A comprehensive review on sdn architecture, applications and major benifits of sdn. *International Journal of Advanced Science and Technology*, 28(20):607–614, 2019.

[98] Tyler Thomas, Mathew Piscitelli, Bhavik Ashok Nahar, and Ibrahim Baggili. Duck hunt: Memory forensics of usb attack platforms. *Forensic Science International: Digital Investigation*, 37:301190, 2021.

[99] Tyler Thomas, Mathew Piscitelli, Ilya Shavrov, and Ibrahim Baggili. Memory foreshadow: memory forensics of hardware cryptocurrency wallets–a tool and visualization framework. *Forensic Science International: Digital Investigation*, 33:301002, 2020.

[100] Dave Jing Tian, Kevin RB Butler, Joseph I Choi, Patrick McDaniel, and Padma Krishnaswamy. Securing arp/ndp from the ground up. *IEEE Transactions on Information Forensics and Security*, 12(9):2131–2143, 2017.

[101] Pantelimon-Teodor Tivig, Alexandru Brumaru, and Serban Georgica Obreja. Creating scalable distributed control plane in sdn to rule out the single point of failure. In *2022 14th International Conference on Communications (COMM)*, pages 1–6. IEEE,

2022.

[102] Stefan Vömel and Felix C Freiling. Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition. *Digital Investigation*, 9(2):125–137, 2012.

[103] Daniel Walnycky, Ibrahim Baggili, Andrew Marrington, Jason Moore, and Frank Breitinger. Network and device forensic analysis of android social-messaging applications. *Digital Investigation*, 14:S77–S84, 2015.

[104] Enoch Wang, Samuel Zurowski, Orion Duffy, Tyler Thomas, and Ibrahim Baggili. Juicing v8: A primary account for the memory forensics of the v8 javascript engine. *Forensic Science International: Digital Investigation*, 42:301400, 2022.

[105] Haopei Wang, Guangliang Yang, Phakpoom Chinprutthiwong, Lei Xu, Yangyong Zhang, and Guofei Gu. Towards fine-grained network security forensics and diagnosis in the sdn era. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2018.

[106] Quadri Waseem, Sultan S Alshamrani, Kashif Nisar, Wan Isni Sofiah Wan Din, and Ahmed Saeed Alghamdi. Future technology: Software-defined network (sdn) forensic. *Symmetry*, 13(5):767, 2021.

[107] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.

[108] Lei Xu, Jeff Huang, Sungmin Hong, Jialong Zhang, and Guofei Gu. Attacking the brain: Races in the sdn control plane. In *USENIX Security Symposium*, pages 451–468, 2017.

[109] Yuan Zhang, Lin Cui, Wei Wang, and Yuxiang Zhang. A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications*, 103:101–118, 2018.

[110] Liehuang Zhu, Md M. Karim, Kashif Sharif, Chang Xu, Fan Li, Xiaojiang Du, and Mohsen Guizani. Sdn controllers. *ACM Computing Surveys*, 53(6):1–40, 2021.

[111] Zookeeper. `https://zookeeper.apache.org/`, 2022.

Vita

In 2002, Abdullah Alshaya earned a Bachelor of Science in Electrical Engineering with a concentration in communications from King Fahd University of Petroleum and Minerals in Dhahran, Saudi Arabia. He then earned his Master of Business Administration, MBA, from the same university in 2009. Abdullah enrolled at New York University in New York City, NY in 2010 and earned his second master's degree in Telecommunication Networks in 2013. He enrolled at Louisiana State University, Baton Rouge, to work with Professor Golden G. Richard III on a Ph.D. in Computer Science. software-defined networks, memory forensics, reverse engineering, digital forensics, malware and vulnerability analysis, and penetration testing are among his specialties. Abdullah anticipates graduating with a Ph.D. in Computer Science, Cybersecurity in 2023.