

**DETECTION AND MITIGATION OF ARCHITECTURAL  
VULNERABILITIES IN SOFTWARE DEFINED NETWORKING**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY**

**TINKU ADHIKARI**

**MZU REGN NO:2108257**

**Ph. D REGN NO: MZU/Ph. D/1845 OF 13.09.2021**



**DEPARTMENT OF COMPUTER ENGINEERING  
SCHOOL OF ENGINEERING AND TECHNOLOGY  
SEPTEMBER 2024**

**DETECTION AND MITIGATION OF ARCHITECTURAL VULNERABILITIES  
IN SOFTWARE DEFINED NETWORKING**

BY

TINKU ADHIKARI

Department of Computer Engineering

Supervisor: Prof. Ajoy Kumar Khan

Joint Supervisor: Dr. Malay Kule

Submitted

In partial fulfillment of the requirement of the Degree of Doctor of Philosophy in  
Computer Engineering of Mizoram University, Aizawl.

## **CERTIFICATE**

This is to certify that the thesis entitled “*Detection and Mitigation of Architectural Vulnerabilities in Software Defined Networking*” is a record of the research that Tinku Adhikari completed under our supervision and guidance, and it is submitted to the Mizoram University in the Department of Computer Engineering under the School of Engineering and Technology, in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy in Computer Engineering.

All help received by him from various sources has been duly acknowledged. No part of this thesis has been reproduced elsewhere for the award of any other degree.

Prof. Ajoy Kumar Khan

Supervisor & Professor

Dept. of Computer Engineering

Mizoram University

Aizawl-796004

Place:

Date:



Assistant Professor  
Computer Science & Technology Dept.  
Indian Institute of Engineering  
Science and Technology, Shibpur  
Howrah - 711 103 (India)

Dr. Malay Kule

Joint Supervisor & Assistant Professor

Deprat. of Computer Science and Technology

IIEST Shibpur- 711103

Place:

Date:

## **DECLARATION**

**Mizoram University**

**September, 2024**

I, Tinku Adhikari, hereby declare that the subject matter of this thesis is the record of work done by me, that the contents of this thesis did not form basis of the award of any previous degree to me or to do the best of my knowledge to anybody else, and that the thesis has not been submitted by me for any research degree in any other University/Institute.

This is being submitted to the Mizoram University for the degree of Doctor of Philosophy in Computer Engineering.

(Tinku Adhikari)

(Prof. Ajoy Kumar Khan)  
Supervisor



Assistant Professor  
Computer Science & Technology Dept.  
Indian Institute of Engineering  
Science and Technology, Shibpur  
Howrah - 711 103 (India)

(Dr. V.D Ambeth Kumar)  
Head

(Dr. Malay Kule)  
Joint Supervisor

## Acknowledgement

This thesis represents the culmination of years of dedication, perseverance, and the invaluable support of numerous individuals. It is with profound gratitude that I acknowledge their contributions, encouragement, and unwavering belief in my abilities.

First and foremost, I wish to express my deepest appreciation to my esteemed Ph.D. supervisor, **Prof. Dr. Ajoy Kumar Khan**. His insightful guidance, constructive criticism, and unwavering support have been instrumental in shaping this research. I am deeply indebted to him for his mentorship.

I am equally grateful to my joint Ph.D. supervisor, **Dr. Malay Kule**, for his invaluable contributions and steadfast support. Dr. Kule's expertise, patience, and relentless encouragement have played a pivotal role in the successful completion of this thesis.

I would also like to extend my heartfelt gratitude to **Dr. V.D. Ambeth Kumar**, Head of the Department of Computer Engineering at Mizoram University. Dr. Kumar's leadership and unwavering support have provided a conducive environment for my research.

A special note of appreciation is due to **Dr. Raja Karmakar**, Associate Professor at the Heritage Institute of Technology. Dr. Karmakar's initial support and motivation were crucial during the early stages of my research.

This thesis is dedicated to the memory of my late father, **Sri Ajoy Kumar Adhikari**. His enduring legacy of values, wisdom, and love continues to inspire and guide me. Though he is no longer with us, his presence is felt in every step of this journey.

I am profoundly grateful to my mother, **Smt. Susmita Adhikari**, whose unwavering support, love, and sacrifices have been my pillars of strength. Her endless patience and encouragement have been indispensable throughout this journey.

I wish to express my deepest appreciation to my wife, **Smt. Suparna Adhikari**, for her boundless support, understanding, and love. Her constant encouragement and belief in me have been my greatest sources of motivation.

Lastly, a very special acknowledgment goes to my son, **Master. Shreyangshu Adhikari**. His presence in my life has been a source of joy, inspiration, and motivation, driving me to strive for excellence in all my endeavors.

This dissertation would not have been possible without the contributions and support of these remarkable individuals. I extend my heartfelt thanks to each one of them for their unwavering faith in me and their significant roles in this achievement.

**(TINKU ADHIKARI)**

# Contents

<b>Acknowledgement .....</b>	<b>i</b>
<b>List of Figures .....</b>	<b>vii</b>
<b>List of Tables .....</b>	<b>x</b>
<b>List of Abbreviations .....</b>	<b>xii</b>
<b>1. Introduction .....</b>	<b>1</b>
1.1 Overview of SDN Architecture .....	2
1.2 Security Issues Within SDN Architecture .....	5
1.3 Security Challenges in Different SDN Planes.....	6
1.3.1 Application Plane Security Challenges.....	6
1.3.2 Control Plane Security Challenges.....	9
1.3.3 Data Plane Security Challenges.....	10
1.4 Motivation .....	11
1.5 Objectives .....	12
1.6 Summary of Contributions.....	13
1.7 Thesis Organization .....	14
<b>2. Literature Review .....</b>	<b>16</b>
2.1 Security Issues Related to Southbound Interface and Data Plane .....	17
2.1.1 Attack Classification on Data Plane and SBI .....	18
2.1.2 Defense Classification on Data Plane and SBI .....	19
2.1.3 Relevant Works for MiTM Prevention .....	21
2.1.4 Relevant Works on Compromised Switch Detection and Network Flow Restoration .....	23
2.2 Security Issues Related to Control Plane .....	28
2.2.1 Attack Classification of Control Plane .....	28

2.2.2	Defense Classification of Control Plane .....	31
2.2.3	Existing Solutions of SYN Flood Attack on Controller .....	33
2.3	Security Issues Related to Application Plane and Northbound Interface .....	35
2.3.1	Attack Classification of Application Plane and NBI .....	35
2.3.2	Defense Classification of Application Plane and NBI .....	37
2.3.3	Related Works for Application Authentication .....	39
2.3.4	Related Works for Application Trust Management .....	41
2.4	Security Issues Related to Distributed SDN Architecture via East/Westbound Interface .....	43
2.4.1	Security Issues in Physically Centralized SDN Control Plane	44
2.4.2	Security Issues in Physically Distributed SDN Control Plane	45
2.4.3	Security Issues in Logically Centralized SDN Control Plane	50
2.4.4	Security Issues in Logically Distributed SDN Control Plane	53
2.4.5	Notable Works for Adaptive Network Synchronization and Security .....	57
2.5	Comparative Analysis of Recent Existing Research Works .....	60
2.6	Chapter Summary .....	64
<b>3.</b>	<b>MiTM Prevention and Compromised Switch Detection in Southbound Interface and Data Plane .....</b>	<b>67</b>
3.1	Overview of MiTM Prevention .....	68
3.2	Proposed System Framework .....	69
3.3	Design Strategy .....	71
3.3.1	Elliptic Curve Diffie-Hellman Key Exchange Algorithm .....	72
3.3.2	Advanced Encryption Standard .....	73
3.3.3	Proposed Solution Approach .....	75
3.4	Experimental Setup and Result Analysis .....	76
3.4.1	Execution Logic .....	77
3.4.2	Result of Encryption .....	77

3.5 Security Achievements .....	78
3.6 Overview of Compromised Switch Detection and Restoration of Network Flow .....	79
3.7 Impact of Compromised Switches .....	82
3.8 Proposed Solution Approach .....	83
3.8.1 Proposed Compromised Switch Detection Framework .....	83
3.8.2 Proposed Switch Isolation and Network Flow Reconstruction Framework .....	83
3.8.3 Proposed Algorithms for Compromised Switch Detection and Restoration of Network Flow .....	84
3.9 Simulation Setup and Implementation Result Analysis .....	88
3.9.1 Simulation Testbed Details .....	89
3.9.2 Implementation Result Analysis .....	89
3.10 Security Achievements .....	93
3.11 Chapter Summary .....	95
<b>4. Proactive Detection of SYN Flood Attack on SDN Controller .....</b>	<b>97</b>
4.1 Overview of The Problem .....	98
4.2 Proposed Solution Strategy .....	100
4.3 Issues Addressed by the Proposed Solution .....	101
4.4 Design Details of Proposed Solution: ProDetect .....	101
4.4.1 Threshold Calculation Process .....	103
4.5 Detection of MAC-Spoofing Attack .....	107
4.6 Implementation Simulation Setup and Result Analysis .....	108
4.6.1 Configuration of Network .....	109
4.6.2 Comparative Analysis of Results .....	109
4.7 Security Achievements .....	117
4.8 Chapter Summary .....	118



<b>5. User App Trust Establishment in SDN Application Plane</b>	<b>119</b>
5.1 Background of the Problem .....	119
5.2 Problem Statement .....	121
5.2.1 Obtaining State of the Network .....	122
5.2.2 Create A Network Policy .....	123
5.2.3 Receive Event Notification .....	123
5.2.4 System Call .....	123
5.3 Design of Trust Management Architecture and Working Procedure..	124
5.3.1 Authentication Establishment Unit .....	127
5.3.2 Authorization Establishment Unit .....	130
5.3.3 Trust Establishment Unit .....	132
5.4 Implementation Details .....	134
5.4.1 Authentication Testing .....	135
5.4.2 Authorization Testing .....	137
5.4.3 Testing for Trust Establishment .....	139
5.5 Comparative Analysis .....	141
5.6 Security Achievements .....	146
5.7 Chapter Summary .....	147
<b>6. A Secure and Resilient Distributed Control Plane Architecture for Large Scale SDN Deployments .....</b>	<b>149</b>
6.1 General Overview .....	149
6.2 Problem Statement .....	151
6.3 Proposed System Framework of SRDCP .....	152
6.3.1 Design Goals of the Proposed Architecture .....	153
6.3.2 Interaction Strategy with Different Distributed Controller Scenarios .....	153
6.3.3 Interaction Strategy with Controller Modules by SRDCP and Module Details .....	155
6.4 Constituent Entities of RTPS-DDS Model .....	158

6.4.1	Components of RTPS Model .....	159
6.4.2	Security Attributes of DDS Plugin .....	159
6.5	Prototype Implementation Details .....	160
6.5.1	Execution Logic of SRDCP .....	160
6.5.2	Testbed Details .....	161
6.5.3	Performance Analysis Matrices.....	161
6.5.4	Description of Experiments .....	162
6.6	Comparative Analysis of Experimental Results .....	163
6.6.1	Analysis of Network Confluence Time (NCT) .....	164
6.6.2	Analysis of Topology Change Latency (TCL) .....	168
6.6.3	Analysis of RTPS Throughput (RTPut) .....	172
6.7	Security Threat Modelling of SRDCP Using STRIDE and Achievements .....	173
6.8	Chapter Summary .....	174
<b>7.</b>	<b>Conclusion and Future Scope .....</b>	<b>176</b>
7.1	Summary of the Research Contributions and Novelty .....	177
7.2	Future Research Directions in SDN Security .....	178
<b>REFERENCES .....</b>		<b>180</b>
<b>LIST OF PUBLICATIONS BASED ON THESIS .....</b>		<b>199</b>

## List of Figures

1.1	Traditional SDN Architecture .....	4
1.2	Distributed SDN Architecture .....	4
2.1	Physical Categorization of SDN Controllers .....	50
2.2	Logical Categorization of SDN Controllers .....	57
3.1	Sequence Diagram of MiTM Prevention .....	71
3.2	ECDH Key Exchange Process .....	73
3.3	Block Diagram of AES .....	74
3.4	MiTM Prevention Algorithm .....	75
3.5	Testbed Overview .....	76
3.6	Sniffing Status .....	77
3.7	Result of ECDH Key Exchange .....	78
3.8	Result of AES Encryption .....	78
3.9	SDN Architecture with Compromised Switches .....	81
3.10	Compromised Switch Detection Algorithm .....	86
3.11	Network Flow Reconstruction with 13 Nodes .....	87
3.12	Network Flow Reconstruction Algorithm .....	88
3.13	Compromised Switch Detection Time .....	90
3.14	CPU Overhead Comparison of Compromised Switch Detection Algorithm .....	91
3.15	Execution Time of Network Flow Reconstruction Algorithm .....	92
3.16	CPU Overhead of Network Flow Reconstruction Algorithm .....	93
4.1	Normal TCP Handshaking vs SYN Flood .....	99
4.2	The ProDetect Algorithm .....	103
4.3	Logic Flow Diagram of ProDetect .....	108
4.4	Attack Detection Time .....	111

4.5	Packet Delivery Ratio .....	112
4.6	Bandwidth Usage .....	113
4.7	OpenFlow Switch Entries .....	114
4.8	CPU Usage .....	115
4.9	Memory Usage .....	115
4.10	Server Response Time .....	116
5.1	Untrusted Application Behavior .....	122
5.2	System Architecture .....	125
5.3	System Flowchart .....	126
5.4	Algorithm for Application Authentication .....	128
5.5	Algorithm for Token Generation .....	129
5.6	Algorithm for Application Authorization .....	132
5.7	Algorithm for Trust Establishment .....	134
5.8	Load on Processor .....	142
5.9	Load on Memory .....	143
5.10	NBI Usage .....	144
5.11	Achieved Throughput .....	145
5.12	Network Latency .....	146
6.1	SRDCP Architecture .....	152
6.2	Proposed Hybrid Hierarchical Model of SRDCP .....	155
6.3	Interaction Between SRDCP and Controller Modules .....	156
6.4	LDP Packet Format .....	158
6.5	Analysis of NCT in Homogeneous ODL H2 Cluster .....	165
6.6	Analysis of NCT in Homogeneous ONOS H2 Cluster .....	166
6.7	Analysis of NCT in Heterogeneous ODL H2 Cluster .....	167
6.8	Analysis of NCT in Heterogeneous ONOS H2 Cluster .....	168

6.9	Analysis of TCL in Homogeneous ODL H2 Cluster .....	169
6.10	Analysis of TCL in Homogeneous ONOS H2 Cluster .....	170
6.11	Analysis of TCL in Heterogeneous ODL H2 Cluster .....	171
6.12	Analysis of TCL in Heterogeneous ONOS H2 Cluster .....	172

## List of Tables

2.1	Major Insights of Works on Attack Classifications on Data Plane and SBI .....	19
2.2	Major Insights of Works on Defense Classifications on Data Plane and SBI .....	20
2.3	Major Insights of Works on MiTM Attack Prevention .....	22
2.4	Major Insights of Works on Compromised Switch Detection .....	24
2.5	Major Insights of Works on Network Flow Restoration .....	37
2.6	Major Insights of Works on Attack Classifications on Control Plane .....	30
2.7	Major Insights of Works on Defense Classifications on Control Plane .....	32
2.8	Summary of Main Contributions in SYN Flood Detection .....	34
2.9	Major Insights of Works on Attack Classifications on Application Plane and NBI .....	36
2.10	Major Insights of Works on Defense Classifications on Application Plane and NBI .....	38
2.11	Insights of Major Works in Application Authentication .....	40
2.12	Major Work Insights on Application Trust Management .....	42
2.13	Summary of Main Contributions on Physically Centralized SDN Controllers .....	45
2.14	Summary of Main Contributions on Flat Physically Distributed SDN Controllers .....	47
2.15	Summary of Main Contributions on Hierarchical Physically	

	Distributed SDN Controllers .....	49
2.16	Summary of Main Contributions on Logically Centralized SDN Controllers .....	52
2.17	Summary of Main Contributions on Logically Distributed SDN Controllers .....	55
2.18	Summary of Main Contributions to Adaptive Network Synchronization and Security .....	59
2.19	Comparative Analysis of Recent Security Works in SDN .....	62
3.1	MiTM Attack Categorization .....	69
5.1	Attributes of Load Balancer Application .....	130
5.2	Terminology for Accuracy in Authenticating .....	136
5.3	Application Precision .....	137
5.4	Trust Parameters and Their Volume (Application View) .....	139
5.5	Trust Parameters and Their Volume (Controller view) .....	140
6.1	Testbed Details .....	161

## **List of Abbreviations**

SDN – Software Defined Networking

API – Application Programming Interface

NBI – North Bound Interface

SBI – South Bound Interface

MiTM – Man in the Middle Attack

TCP – Transmission Control Protocol

SYN – Synchronize

DoS – Denial of Service

DDoS- Distributed Denial of Service

MAC – Media Access Control

TLS – Transport Layer Security

ECC – Elliptic Curve Cryptography

ECDH – Elliptic Curve Diffie-Hellman

AES – Advanced Encryption Standard

ARP – Address Resolution Protocol

DNS – Domain Name System

DHCP – Dynamic Host Configuration Protocol

ICMP – Internet Control Message Protocol

HTTP – Hyper Text Transfer Protocol

ODL – OpenDaylight



ONOS – Open Networking Operating System

VM – Virtual Machine

DDS – Data Distribution Service

OMG – Object Management Group

RTPS – Real Time Publish Subscribe

DCN – Data Communication Network

ACK – Acknowledgement

# **Chapter 1**

## **INTRODUCTION**

In this introductory chapter, we sketch the overall flow of the thesis by highlighting the basic motivation and objective of the proposed work. A summary of main contributions is also presented, followed by an outline of the scope of the thesis.

Software-Defined Networking (SDN) revolutionizes traditional network architecture by decoupling the control plane from the data plane, enabling more flexible and efficient network management. While SDN offers numerous advantages, such as enhanced agility, programmability, and centralized control, it also introduces unique security challenges that must be addressed to fully realize its potential. SDN security is a multifaceted domain focusing on safeguarding the network from various threats and vulnerabilities specific to its architecture. One critical aspect is securing the centralized controller, which, if compromised, can grant attackers control over the entire network. Additionally, the communication channels between the control plane and data plane are potential targets for attacks like man-in-the-middle or denial-of-service. Ensuring the integrity, confidentiality, and availability of these channels is paramount. Another concern is the policy enforcement mechanism, which must be robust enough to prevent unauthorized access and configuration changes. Moreover, SDN's programmability can be a double-edged sword, as it allows rapid deployment of security updates but also opens the door for misconfigurations and malicious code injection. To mitigate these risks, comprehensive security frameworks are essential, incorporating authentication, authorization, encryption, and anomaly detection mechanisms. Collaboration between network operators, security experts, and the broader SDN community is crucial in developing standardized security protocols and best practices. As SDN continues to evolve and become more prevalent in various sectors, addressing its security challenges will be vital in ensuring reliable and secure network operations.

## **1.1 Overview of SDN Architecture**

Software-Defined Networking (SDN) architecture fundamentally redefines network management by separating the control plane from the data plane, enabling more dynamic and programmable network operations [1]. This architecture is composed of three primary planes: the data plane, the control plane, and the application plane, each interconnected by various interfaces to ensure seamless communication and functionality.

### **1. Data Plane (Infrastructure Layer):**

The data plane, also known as the forwarding plane, is responsible for handling and forwarding packets based on rules provided by the control plane. It consists of network devices like switches, routers, and other physical or virtual devices that execute these forwarding decisions. The data plane operates at high speeds to ensure efficient packet handling and minimal latency.

### **2. Control Plane:**

The control plane serves as the brain of the SDN architecture. It maintains a global view of the network and makes decisions on how traffic should be managed. This plane is responsible for routing, traffic engineering, and other network policies. The control plane is typically implemented using a centralized SDN controller, which communicates with the data plane devices to install flow rules and manage network behavior. The centralization allows more efficient and consistent policy enforcement across the network.

### **3. Application Plane:**

The application plane consists of various applications and services that leverage the underlying network infrastructure. These applications, which can include security services, load balancing, and network monitoring

tools, interact with the SDN controller to request specific network behaviors. The application plane allows rapid deployment and innovation, as new applications can be developed and integrated without altering the underlying hardware.

## **Interfaces:**

### **1. Southbound Interface:**

The Southbound Interface (SBI) facilitates communication between the control plane and the data plane. Protocols like OpenFlow [2], NETCONF [3], and others are commonly used to allow the SDN controller to configure and manage data plane devices. This interface is crucial for implementing control decisions and ensuring that data plane devices operate according to the controller's instructions.

### **2. Northbound Interface:**

The Northbound Interface (NBI) connects the control plane to the application plane. It provides APIs that enable applications to communicate their network requirements to the SDN controller. These APIs abstract the underlying network complexity, allowing application developers to focus on functionality without worrying about the low-level details of network operations.

### **3. East-West Interface:**

The East-West Interface facilitates communication between multiple SDN controllers in a distributed architecture [4]. This interface is essential for ensuring consistency and coordination across different network segments, particularly in large-scale or geographically dispersed networks. It allows controllers to share information and collaborate on network management tasks, enhancing scalability and reliability. Figure 1.1 and 1.2 depict the

legacy SDN network architecture and distributed SDN architecture, respectively.

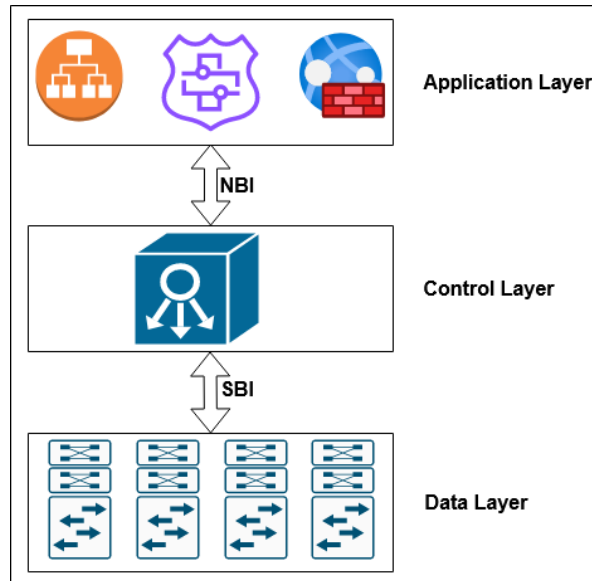


Figure 1.1 Traditional SDN Architecture

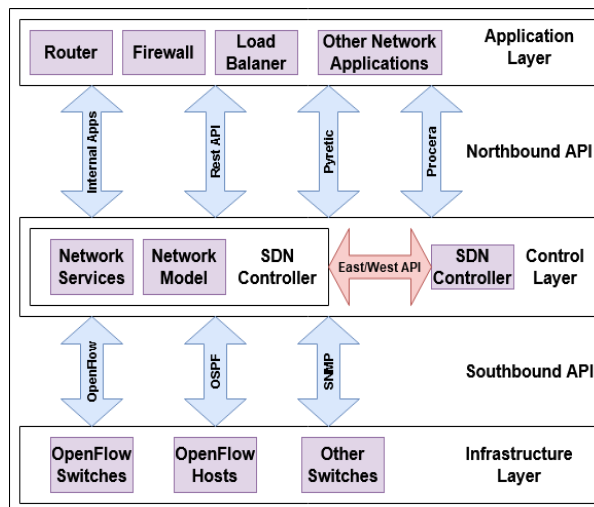


Figure 1.2 Distributed SDN Architecture

## 1.2 Security Issues Within SDN Architecture

Within the periphery of traditional as well as distributed SDN architecture, there are some major security concerns that are cantankerous and can put the entire network down. Therefore, a thorough investigation of these threats and proper mitigation of them is required for the stability and reliability of the entire network. Some of the major attacks are classified below-

**1.2.1 Manipulation of the Network:** A crucial assault on the control plane. An attacker creates fake network data, hacks the SDN controller, and launches additional network-wide assaults.

**1.2.2 Network Traffic Diversion:** The network components at the data plane are attacked in this way. An aspect of the network is compromised by the assault in order to reroute traffic and permit listening in.

**1.2.3 Side Channel Attack:** This type of attack may target the network components at the data plane. Timing details, such as the time it takes for a new network connection to be established, might reveal to an attacker whether or not a flow rule is in place.

**1.2.4 Application Tampering:** The application plane is the target of this assault. An application vulnerability might result in a malfunction, a service interruption, or data eavesdropping. An attacker may be able to access an SDN application with elevated privileges and carry out illicit actions.

**1.2.5 Denial of Service:** One of the most frequent assaults is denial of service, which may impact every aspect of the SDN. An attacker might reduce or completely impair SDN services by implementing a denial-of-service attack.

**1.2.6 ARP Spoofing Attack:** Also known as ARP Cache Poisoning, this is a Man-in-the-Middle attack. ARP spoofing is a technique that hackers can use to enter a network, sniff traffic, and change it if not completely halt it. Attacks of this kind taint topology-

aware SDN applications and network topology data. Poisoning can also occur via other protocols like IGMP or LLDP.

**1.2.7 API Exploitation:** Vulnerabilities in a software component's APIs may make it possible for a hacker to release information without authorization. Additionally possible is API exploitation at the northbound interface, which has the potential to disrupt network traffic.

**1.2.8 Traffic Sniffing:** Sniffing traffic is a common technique used by hackers to gather and examine data about network communications. A hacker can also use sniffing to intercept data from network components or connections and take critical data. Sniffing is possible anywhere there is a continuous flow. Unencrypted communications in SDN allow a hacker to intercept traffic going to and from a central controller. Important details about network flows or permitted traffic may be included in the data collected.

**1.2.9 Brute Force or Password Predicting:** This attack is possible for non-SDN elements. A brute force attack or password guessing might allow an unauthorized person to access the SDN.

### **1.3 Security Challenges in Different SDN Planes**

With SDN technologies being gradually deployed, there will likely be an increase in the list of security problems. These issues need to be addressed in order to fully utilize SDN and enable proactive security measures to be implemented. Thus, this section discusses security concerns and problems that are present in SDN. SDN security flaws may be broadly categorized as focusing on three primary areas: i) applications, ii) control plane, and iii) data plane. As a result, the following describes the security issues that arise in the three planes, or SDN levels.

#### **1.3.1 Application Plane Security Challenges**

SDN possesses two fundamental characteristics that, on the one hand, provide the basis of security issues and, on the other, the cornerstone of networking innovation. The first is

software-based network control, and the second is network intelligence centralization in network controllers [5]. Since the majority of network operations may be implemented as SDN applications, rogue apps have the potential to cause havoc on a network if they are not halted in a timely manner.

Apps can seriously jeopardize the security of network resources, services, and functions as there are no open specifications or standards that enable open APIs for apps to manage network services and functions through the control plane [6]. There aren't many effective OpenFlow security applications, even though OpenFlow [2] makes it possible to implement flow-based security detection algorithms as security apps [7]. Other than that, there aren't any consensus-driven development environments or paradigms for network programming. The multiplicity and diversity of vendor and third-party apps created in various autonomous development environments with disparate programming models and paradigms may lead to security policy collisions and restrictions in interoperability. The following lists some of the dangerous security issues that SDN applications present.

#### **1.3.1.1 Authentication and Authorization**

It is immensely difficult to ensure that only authorized applications can access and control network resources. Weak authentication mechanisms can lead to unauthorized access and potential network breaches. Proper role-based access control (RBAC) frameworks are essential to mitigate this risk [8]

#### **1.3.1.2 Inter-Application Communication**

Applications in the SDN ecosystem often need to communicate and coordinate. However, insecure communication channels can be exploited to inject malicious commands or eavesdrop on sensitive data. Implementing secure communication protocols is vital to safeguard inter-application communication [9].



#### 1.3.1.3 **Application Isolation**

Malicious or buggy applications can interfere with the normal functioning of other applications. Effective isolation mechanisms are necessary to prevent such interactions and ensure that applications operate independently and securely [8].

#### 1.3.1.4 **Resource Management**

Applications may compete for network resources, leading to potential conflicts and inefficiencies. Proper resource management policies and mechanisms are required to ensure fair and secure resource allocation among applications [10].

#### 1.3.1.5 **Data Privacy**

Applications often handle sensitive network data. Ensuring data privacy through encryption and secure data handling practices is essential to prevent data leakage and unauthorized access [9].

#### 1.3.1.6 **Policy Enforcement**

Ensuring that applications adhere to predefined network policies is crucial for maintaining network security and stability. Robust policy enforcement mechanisms are required to verify that applications do not deviate from established policies [8].

#### 1.3.1.7 **Malware Detection**

The open nature of SDN allows third-party applications to be integrated easily, increasing the risk of malware introduction. Effective malware detection systems are needed to identify and mitigate malicious applications [11].

#### 1.3.1.8 **Dynamic Update Handling**

Applications may require updates to improve functionality or address vulnerabilities. Ensuring that updates do not introduce new security risks or disrupt existing services is a significant challenge that requires careful version control and update mechanisms [8].

#### **1.3.1.9 Security Policy Conflicts**

Different applications might enforce conflicting security policies, leading to potential vulnerabilities. Resolving these conflicts through automated policy conflict detection and resolution mechanisms is crucial for maintaining a secure network environment [9].

#### **1.3.2 Control Plane Security Challenges**

The control plane in SDN is a single, centralized decision-making body. Because of its crucial position, the controller might thus be a prime target for network compromise or malicious activity within the network. The following is a description of the primary security risks and difficulties facing the control plane.

##### **1.3.2.1 Controller Overload and DDoS Attack**

As noted in [12], one of the main security risks for the SDN control plane is the possibility of Distributed Denial of Service (DDoS) attacks. Attackers may flood the controller with too many requests, overloading it and making it unable to do proper network management duties, which would interrupt services.

##### **1.3.2.2 Controller Compromise**

According to Kreutz et al. [13], an attacker can cause disastrous outcomes including unapproved access, data breaches, and total network failure by controlling the SDN controller and modifying flow rules and policies throughout the network.

##### **1.3.2.3 Inter-Controller Communication Security**

Several controllers collaborate to govern the network in a distributed SDN architecture [11]. To avoid intercepting, altering, or spoofing control messages—which might result in inconsistent network states and vulnerabilities—it is imperative to ensure the security of these communications.

#### **1.3.2.4 Control Channel Security**

Secure communication channels are required between the data plane devices and the SDN controller. Man-in-the-middle attacks are one type of attack that can compromise network integrity and confidentiality by intercepting and manipulating control signals.

#### **1.3.2.5 Malicious or Faulty Applications**

If the SDN controller's applications are malevolent or have vulnerabilities, they may pose a threat to security [12]. These programs have the ability to interrupt network operations or leak confidential data by taking advantage of their access to the control plane.

#### **1.3.2.6 Flow Rule Modification and Insertion**

Malicious flow rules might be added or modified by attackers and attempt to enter the network. Traffic may be redirected via hacked or malicious nodes, which may result in data interception, traffic analysis, or the denial of service to authorized users [13].

#### **1.3.2.7 Inconsistency in Network Policies**

It can be difficult to establish uniform network rules across all devices because of the dynamic nature of SDN [14]. Inconsistent policies may leave security holes that hackers may take advantage of, perhaps resulting in breaches and illegal network activity.

### **1.3.3 Data Plane Security Challenges**

The OpenFlow controller adds flow rules to the flow tables of the OpenFlow switch in an OpenFlow network. These flow rules can be set up either on the first packet received from a new host or prior to the new host sending any packets. The prominent security challenges are listed below.

#### **1.3.3.1 Flow Rule Manipulation**

The possibility of manipulating flow rules is one of the main security issues in the SDN data plane [15]. Attackers have the ability to add, remove, or alter flow rules in the

switches, which can reroute traffic, interfere with network functions, or open new unapproved channels that might result in data breaches or interruptions of services.

#### **1.3.3.2 Lack of Encryption**

Since the data plane frequently transmits packets without encryption, attacks including data interception and eavesdropping can be launched against it. Sensitive information can be obtained by attackers from network traffic, which presents a serious risk to the security and integrity of data [16].

#### **1.3.3.3 Packet Spoofing and Injection**

Malicious packets can be injected into the data plane or spoofed by attackers. This may result in network reconnaissance, illegal access, or the exploitation of security holes in network equipment. Additionally, more complex assaults like man-in-the-middle (MITM) attacks can be made possible via packet spoofing.

#### **1.3.3.4 Topology Poisoning**

For SDN to function properly, the controller has to have precise topology information. By transmitting erroneous topology information, attackers can seriously impair network performance and dependability by causing improper routing decisions, network loops, or partitioning.

#### **1.3.3.5 Resource Exhaustion**

The resources of SDN data plane devices, like switches, are constrained (e.g., TCAM, RAM). By overloading the switch with several flow entries or requests, attackers can take advantage of this shortcoming and cause resource exhaustion [17]. This may lead to denial of service (DoS) problems when valid traffic is discarded or delayed.

### **1.4 Motivation**

Software-defined networking (SDN) has revolutionized network management by decoupling the control plane from the data plane, offering dynamic network configuration

and flexibility. However, this architectural shift introduces new security challenges that necessitate rigorous academic research. Key motivations driving research in SDN security include:

- **Centralized Control Vulnerabilities:** The separation of control and data planes in SDN centralizes network control, creating a single point of failure that is utterly vulnerable to attacks. Research focuses on mitigating risks such as unauthorized access, data breaches, and denial-of-service (DoS) attacks targeting the controller.
- **Complex Attack Surface:** SDN's programmable nature expands the attack surface compared to traditional networks. Researchers explore threats like flow table manipulation, malicious controller code injection, and covert channels that exploit programmable interfaces.
- **Traffic Visibility and Monitoring:** Traditional network security tools struggle with SDN's dynamic, virtualized environment. Academic efforts concentrate on developing robust monitoring solutions to detect anomalies, enforce policies, and ensure compliance across virtual networks.

Continued research aims to enhance SDN security through innovations in authentication, encryption, anomaly detection, and resilience against evolving cyber threats. By addressing these challenges, academics aim to fortify SDN deployments and enable secure, scalable network management in diverse operational environments.

## 1.5 Objectives

Considering the aforementioned security concerns and difficulties in SDN architecture, the research work's objectives are broken down into the following sections for the proposed work:

- **Identification and characterization of SDN southbound interface and data plane security issues as well as providing solutions to cater to them:** The main goal here is to identify the major security concerns about SDN data plane and

southbound interface and provide efficient and new solution approaches to cater to them.

- **Characterization of SDN control plane security threats and implementation of a novel security architecture to address them.** The main aim here is to classify the main security threats within the SDN control plane and provide a valid and experimentally justified solution to encounter them.
- **Identification and classification of SDN northbound interface and application plane security loopholes and imparting solutions to mitigate them.** Here the main goal is to identify and emphasize major security concerns about the SDN application plane and northbound interface and provide an experimentally verified and valid solution to address them.
- **Security implementation and enhancement in distributed SDN architecture using an east-westbound communication interface.** This objective target to find out the security loopholes within the distributed SDN paradigm which elaborates the security concerns of the east/westbound interface in turn and to provide experimentally established and viable solutions to address the issues.

## 1.6 Summary of Contributions

The major contributions to carry out this research work are mentioned below-

- In the SDN data plane there might be different types of attacks and due to these the entire network might be compromised. One prominent attack is Man in the Middle (MiTM) attack which could lead to DoS in a larger networking prospect. A novel ECDH and AES based SBI encryption technique is suggested to cater this. Another type of fraudulent activity may incur from the different compromised data plane switches by the attackers. Therefore, another approach is suggested to detect the compromised switches and reconstruct the network path after isolating them.
- Control plane is the most vulnerable within the SDN architecture, hence prone to attack. A TCP based attack, SYN flood which can saturate the network resources and eventually lead to a DoS or DDoS is prevented here by proposing a novel

solution which is compared and thoroughly analyzed with purview of another existing state of the art solution.

- A burning issue of deploying SDN in real life scenarios is application trustworthiness which again may incur serious consequences if left unaddressed. Therefore, an application must be trustworthy before communicating with the controller. This issue is addressed here by proposing methods to establish and validate trustworthiness of an application.
- Nowadays SDN is being deployed in distributed environments like DCNs, IoT, EDGE networks and many more, but due to the lack of standardized security protocols these implementations are highly susceptible to attack. A secure, scalable and resilient distributed control plane framework has been proposed by the next works which overcomes these types of limitations.

## **1.7 Thesis Organization**

The following chapters comprise the successive components of this thesis:

Chapter 1 formally introduces the SDN architecture, and its different security concerns in both legacy SDN as well as distributed SDN scenarios. It also briefs different attacks in each layer of SDN. Research motivations and thesis objectives are highlighted here in this chapter.

Chapter 2 provides an in-depth review of the existing security solutions in both legacy and distributed SDN paradigms. This chapter works as a foundation of the entitled research work which identifies different research gaps and probable improvements which formulates the research objectives in the subsequent chapters.

Chapter 3 provides a detailed analytical and novel solution to two individual security concerns that hamper southbound interface security and data plane security. First solution highlights how communication through the southbound interface can be encrypted using Advanced Encryption Standard (AES) and Elliptic Curve Diffie-Hellman (ECDH) key exchange algorithms to prevent Man in the Middle (MiTM) attacks. The second solution

provides a novel solution to detect compromised data plane switches and restoration of network flow after the isolation of defective switches. These two solutions provide significant leverage to southbound interface security and data plane security and hence meet the first thesis objective.

Chapter 4 takes care of a typical control plane security concern, the Syn flood attack. A novel and proactive security solution is proposed in this chapter to detect and mitigate the SYN flood attack in the SDN controller which covers objective two of the thesis.

Chapter 5 addresses security concerns arising from unauthorized and untrusted applications. A trust establishment framework is proposed to create a trusted application repository to address the issue here. This application authentication and trust establishment framework guarantees that no untrusted application is allowed to access the northbound interface and thereby restricts controller access and further malfunctioning, hence it covers objective three of the thesis.

Chapter 6 covers the need for a scalable and standardized method for synchronizing network data across distributed SDN control planes, particularly in diverse and uniformly dispersed networks, to ensure optimal packet routing and comprehensive network state awareness and security. A lightweight and novel solution “SRDCP” has been proposed to address the issue here and it meets objective four of the thesis.

Chapter 7 concludes the research, which summarizes findings, considers objectives and makes recommendations for future work. These include issues with distributed architecture's scalability and reliability, security issues with programmable data planes, and security concerns arising from the integration of SDN with IoT, edge, and fog networks. At the conclusion, there is a list of references.



## **Chapter 2**

### **LITERATURE REVIEW**

Without question, during the past ten years, Software Defined Networking (SDN) has dominated the evolution of the networking paradigms. Researchers looking to get over the constraints of conventional networks that resisted innovation have found the idea of splitting the control and data planes, which was initially proposed by Greenberg et al. [18], to be interesting. Many advances made feasible by the centralized nature of the control plane, also known as an SDN controller, have been made possible that were previously unattainable with individual devices. The industry's increasing acceptance of SDN has been aided by the introduction of blockbuster apps like FlowVisor [19], Open vSwitch [20], and customizable interfaces like OpenFlow [21]. In fact, SDN has garnered more attention in the networking community lately and offers network administrators a host of advantages, including increased security. Because an SDN controller centralizes control, administrators may create native safety measures that are more efficient than conventional ones that depend on middleboxes. Several ideas, including botnet detection [22], mitigation of DDoS [23], and network forensics [24], have shown how to construct security systems with the capacity for preemptive abatement and early discovery. Furthermore, by integrating the supply chain context, SDN may work with middle-boxes to coordinate, resolving issues with backward compatibility with traditional marketplaces [25],[26]. It is important to remember, nevertheless, that the SDN architecture creates new levels that hackers might exploit. As a result, there are now additional attack surfaces available since previously inaccessible levels are now accessible to a larger number of users. It is also discovered that the SDN controller's centralized design is vulnerable to straightforward application-level assaults. Furthermore, even though the centralized control plane may control every linked switch, it is susceptible to saturation assaults [27]–[29]. Apart from the OpenFlow specification's brief explanation of SSL/TLS encryption,

which was published more than 10 years after SDN's inception, there are currently no official standards that offer comprehensive security recommendations [30].

The idea of physical centralization of the controller plane in one customizable software element, termed the controller, is limited by different factors in terms of flexibility, accessibility, validity, and security, according to scientific investigations on the viability of the SDN deployment. Thinking of the control plane as a distributed system, with several SDN controllers managing the entire network but preserving a logically centralized network view, became unavoidable over time [31]. The distributed SDN architecture is susceptible to various new attacks as they grow and are deployed over cloud or fog networks which would eventually lead to DDoS. In this regard, debates within the networking community centered on how best to include distributed SDN designs while considering the increased security threats that these distributed systems bring. Although the distributed SDN controllers that are currently in use can be used to implement decentralized SDN control, choosing a distributed SDN in case of the context of large-scale deployment scenarios proved to be very challenging due to their abundance as well as their unique advantages and disadvantages.

Even with all of this enthusiasm, SDN implementation in the industrial setting is still in its nascent stage. It may take some time for technology to advance and standardization efforts to pay off, which will prevent SDN from reaching its full potential. Here, we emphasize how crucial it is to carefully examine the suggested SDN solutions in order to identify any emerging threats that can inform further study in this area. Although the distributed SDN controllers that are currently in use can be used to implement decentralized SDN control, choosing a distributed SDN framework in relation to large-scale implementations proved to be very challenging due to their abundance as well as their unique advantages and disadvantages.

## **2.1 Security Issues Related to Southbound Interface and Data Plane**

In this section, different attacks on the SDN Data plane and Southbound interface and their defense strategies are categorized thoroughly. This classification is predicated on the

finding that these tiers are the focus of most SDN assaults and thereby corresponding defenses.

### **2.1.1 Attack Classification on Data Plane and SBI**

The TCAM (Ternary Content Addressable Memory) in switches is a crucial resource that must be managed carefully. Controllers must correctly implement flow rules in switch flow tables, which are limited in private devices. However, since southbound connections are not well restricted, flooding flow tables with useless rules can exhaust TCAM. Yoon et al. [32] demonstrate that malicious software can send multiple FLOW MOD messages with varied match fields, overwhelming switches. Malicious servers can also send fake packets to switches.

Switches use OpenFlow, a widely accepted protocol, to communicate with controllers. However, OpenFlow protocol variants can be exploited due to vendor-specific implementations. Yoon et al. [32] describe how unauthorized programs can insert unsupported match fields into flow entries, exploiting switch firmware and reducing packet processing efficiency.

Further, attackers can exploit OpenFlow's dynamic operations. For instance, the OpenFlow Set operation can be used in dynamic tunneling attacks, as shown by Porras et al. [33]. These attacks bypass security measures and modify packet headers within the switch routing chain.

A vulnerability in packet forwarding can lead to serious security issues. Cao et al. [34] propose a buffered-packet abduction technique that exploits the fact that OpenFlow switches rely solely on buffer IDs when sending packets, allowing attackers to steal stored packets.

BigBug [35] and SDNRacer [36] identify race conditions in control messages between switches and controllers. Without using BARRIER REQUEST, OpenFlow messages are handled non-deterministically, potentially causing flow rule installation delays that lead to security gaps.

In cloud systems, SDN is often paired with NFV (Network Function Virtualization) to enhance flexibility. While this supports OpenFlow, it also increases attack surfaces. Bu et al. [37] highlight how compromised switches can alter packet labels used by middle-box communication chains, an attack that remains unaddressed due to the lack of packet integrity checks in current SDN setups.

Table 2.1 Major Insights of Works on Attack Classifications on Data Plane and SBI

<b>Citations</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[33]	Proposed methods to secure the control layer of SDNs against various attacks.	Analysis of security vulnerabilities, design of security mechanisms, and simulation-based validation.	Effectiveness may vary depending on the specific SDN architecture and attack vectors.
[34]	Identified and analyzed buffered packet hijacking vulnerabilities in SDN environments.	Theoretical analysis, experimental validation using SDN testbeds, and real-world scenario simulations.	Practical applicability may depend on specific SDN implementations and network configurations.
[35]	Developed BigBug, a practical tool for concurrency analysis in SDNs to detect and mitigate bugs.	Implementation of static and dynamic analysis techniques, and evaluation of real SDN applications and controllers.	Accuracy and coverage of bug detection may vary with the complexity of SDN applications and network conditions.
[36]	Introduced SDNRacer for detecting concurrency violations in SDN programs.	Static analysis and runtime verification techniques, evaluation with SDN applications and controllers.	The effectiveness may depend on the accuracy of concurrency models and the complexity of SDN program interactions.
[37]	Proposed Flowcloak to mitigate middlebox-bypass attacks in SDNs.	Theoretical analysis, design of evasion-resistant mechanisms, and validation through simulations and experiments.	Real-world deployment and scalability of the proposed defenses may require further investigation.

### 2.1.2 Defense Classification on Data Plane and SBI

Flow rule verification ensures data-plane states align with network policies, preventing attacks or application bugs from violating rules. Several studies have used control message trapping to verify if control messages comply with network regulations. VeriFlow [38] implements real-time invariant testing between the data and control layers, quickly analyzing traffic by modeling address spaces as equivalency classes.

Ropke et al. [39] propose comparing control messages sent to the data plane with control events generated by applications to prevent malicious software from installing unauthorized flow rules.

OpenFlow implementation discrepancies across different switches can create security vulnerabilities. SOFT [40] aims to detect these discrepancies by symbolically executing switch control flows and comparing results from different suppliers.

Compromised SDN switches can allow hackers to capture or drop messages. Kaminski et al. [41] classify malicious switches as packet droppers or swappers using anomaly detection. Chi et al. [42] propose an online detection method that sends fake packets to verify the correct forwarding path. Mohan et al. [43] suggest using node-disjoint control paths to detect compromised origin switches by identifying conflicting control messages.

Table 2.2 Major Insights of Works on Defense Classifications on Data Plane and SBI

Citation	Main Contribution	Methods Used	Limitations
[38]	Verifying network-wide invariants in real time	Introduces Veriflow for real-time verification of network invariants	Scalability challenges with large networks
[39]	Preventing malicious SDN applications from hiding adverse network manipulations	Proposes techniques to detect and prevent malicious manipulations in SDN applications	May not detect all types of malicious activities
[40]	OpenFlow switch interoperability testing	Develops methods for soft OpenFlow switch interoperability testing	Limited to OpenFlow protocol and may miss interoperability issues with other protocols

[41]	Detecting malicious switches in SDNs	Introduces Flowmon for detecting malicious switches in SDNs	Potential false positives and performance overhead
[42]	Detecting compromised SDN switches	Proposes techniques for detecting compromised SDN switches	May require significant monitoring and resources
[43]	Resilient in-band control path routing with malicious switch detection in SDN	Proposes resilient control path routing mechanisms with malicious switch detection	Complexity in deployment and potential performance impact

### 2.1.3 Relevant Works for MiTM Prevention

Current MiTM detection and prevention methods focus on systematizing detection while preserving message integrity and confidentiality. OpenFlow lacks built-in security rule enforcement, prompting various solutions. The authors of [44] discuss TLS security issues and suggest improvements for securing southbound traffic.

Using attack tree modeling and STRIDE, [45] examines OpenFlow security, assuming attackers have data plane access, and suggests countermeasures. The study in [46] assesses the likelihood of SDN attacks, proposing defenses and Context-Based Node Acceptance (CBNA) for mitigating MiTM attacks. A random forest-based detection strategy on CBNA is presented in [47], targeting network intrusions and MiTM attacks.

The authors of [48] offer a MiTM mitigation strategy based on context-based SDNs and HTTPS, though it's vulnerable to SSL-Strip attacks. A multi-controller-based detection approach is proposed in [49], comparing SDN controllers' vulnerability to attacks and offering mitigation strategies.

MiTM attacks, like ARP spoofing [50], [51], are hard to detect but can be addressed with tools like ArpAlert, ArpwatchNG, and Snort [49], [52]. Snort, with an ARP spoof preprocessor and IPS system, is the best option [53]. However, OpenDaylight (ODL) uses HTTP for updates, which should be replaced by HTTPS to enhance security.

Table 2.3 Major Insights of Works on MiTM Attack Prevention

Citation	Main Contribution	Methods Used	Limitations
[44]	OpenFlow communications and TLS security in SDNs	Analysis of OpenFlow communications and implementation of TLS security	Potential performance overhead due to TLS encryption
[45]	Comprehensive security analysis checklist for OpenFlow networks	Development of a security analysis checklist for OpenFlow networks	May not cover all possible security vulnerabilities
[46]	Context-based node acceptance (CBNA) framework for MitM detection in SDN architecture	Proposes CBNA framework for detecting MitM attacks	Framework may have scalability issues in large networks
[47]	MitM detection and defense mechanism CBNA-RF based on machine learning for large-scale SDN context	Uses machine learning-based CBNA-RF for MitM attack detection and defense	Requires extensive training data and may result in false positives
[48]	Mitigating man-in-the-middle attacks within context-based SDNs	Proposes techniques to mitigate MitM attacks in context-based SDNs	Mitigation techniques may not be effective against all types of MitM attacks
[49]	Detection of MitM attack in multi-SDN controller	Develop methods for detecting MitM attacks in multi-controller SDN environments	Complexity in deployment and potential performance impact
[50]	Preventing ARP poisoning attack utilizing SDN paradigm	Proposes techniques for preventing ARP poisoning attacks using SDN	May require significant changes to the existing network infrastructure
[51]	Prevention mechanism for infrastructure-based denial-of-service attack over SDN	Proposes a prevention mechanism for DoS attacks in SDN	May introduce performance overhead and complexity

[52]	New use cases for Snort in cloud and mobile environments	Explores new use cases for Snort IDS in cloud and mobile environments	May require modifications to Snort for optimal performance in new environments
[53]	Enabling SDN-based intrusion prevention system in clouds	Introduces SDNIPS, a software-defined networking-based intrusion prevention system for cloud environments	Potential scalability and performance issues in large cloud deployments

#### 2.1.4 Relevant Works on Compromised Switch Detection and Network Flow Restoration

This work is another part of the first objective of the thesis and some relevant previous works, and their major takeaways are discussed below.

Detecting compromised network switches has been a long-standing issue. Haeberlen et al. [54] proposed that peer nodes request logs from others to assess deviations in behavior. A secure log records messages sent and received, allowing peer review to identify malicious nodes. Zhou et al. [55] introduced Secure Network Provenance (SDP) to trace attacks and identify compromised switches. Al-Shaer and Al-Haj [56] proposed detecting conflicting rules in SDN switches, while Son et al. [57] used the Yices SMT solver to detect property breaches in flow tables.

Khurshid et al. [38] developed VeriFlow to detect network-wide breaches in real-time. Kazemian et al. [58] introduced NetPlumber, a real-time verification tool using Header Space Analysis (HSA) [59]. Dhawan et al. [60] introduced SPHINX to detect attacks that compromise data plane relay and network security policies by analyzing OpenFlow messages. However, these solutions assume a trusted controller, which poses problems if it is compromised.

Chi et al. [42] proposed sending test packets to detect compromised switches, though random selection of switches can delay detection. Khan et al. in [61] and in [62] presented



forensic SDN monitoring with modules C-Watch and S-Watch to monitor the controller and switch, but their work remains theoretical. SDN controllers and apps have been a major focus for enhancing security [63], [64]. Teng et al. [65] used machine learning, including decision trees (DT) and support vector machines (SVM), for intrusion detection systems (IDS), while the Improved SD-WSN Framework [66] offers solutions for node failures and data routing flexibility. However, few studies reliably defend against compromised data plane switches [67], [68].

Despite these efforts, data plane attacks pose significant threats to SDN [69], including DoS, topological poisoning, and side-channel attacks [70]. WedgeTail [71], a method for detecting malicious forwarding rules, identifies and responds to attacks but assumes control plane reliability, which is not always the case. FlowMon [41] detects packet droppers and swappers by analyzing forwarding paths, but can fail if compromised switches provide false data. Finally, a SIEM-based technique [42] detects abnormal switch behaviors and issues security alerts in real-time, though its scope is limited to specific behaviors like improper forwarding and packet manipulation.

Table 2.4 Major Insights of Works on Compromised Switch Detection

Citation	Main Contribution	Methods Used	Limitations
[54]	PeerReview: Practical accountability for distributed systems	Introduces PeerReview for accountability in distributed systems	Scalability challenges in large distributed systems
[55]	Secure network provenance	Proposes techniques for secure network provenance	Potential performance overhead
[56]	FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures	Develops FlowChecker for analyzing and verifying OpenFlow configurations	Limited to OpenFlow and may not generalize to other configurations
[57]	Model checking invariant security properties in OpenFlow	Uses model checking for verifying security properties in OpenFlow	Complexity in model checking large-scale networks

[58]	Real-time network policy checking using header space analysis	Proposes header space analysis for real-time policy checking	Scalability and performance challenges
[59]	Header space analysis: Static checking for networks	Introduces header space analysis for static network checking	Limited to static analysis and may not detect dynamic issues
[60]	Sphinx: Detecting security attacks in software-defined networks	Proposes Sphinx for detecting security attacks in SDNs	Potential false positives and performance overhead
[61]	FML: A novel forensics management layer for software-defined networks	Introduces FML for forensics management in SDNs	Complexity in deployment and integration
[62]	Software-defined network forensics: Motivation, potential locations, requirements, and challenges	Discusses the motivation, potential locations, and challenges of SDN forensics	Broad overview without specific implementation details
[63]	Securing data planes in software-defined networks	Proposes techniques for securing data planes in SDNs	May introduce performance overhead
[64]	Handling malicious switches in software-defined networks	Proposes methods for handling malicious switches in SDNs	Potential false positives and complexity in implementation
[65]	SVM-DT-based adaptive and collaborative intrusion detection	Proposes an SVM-DT-based approach for adaptive and collaborative intrusion detection	Requires extensive training data and may result in false positives
[66]	Methodology for the reliability of WSN based on SDN in an adaptive industrial environment	Proposes a methodology for the reliability of WSN based on SDN	Complexity in deployment and integration
[67]	SDN data plane security: Issues, solutions, and future directions	Reviews issues, solutions, and future directions in SDN data plane security	Broad overview without specific implementation details
[68]	A survey of securing networks using	Surveys methods for securing networks using SDN	Broad overview without specific

	software-defined networking		implementation details
[69]	DSSS-based flow marking technique for invisible traceback	Proposes DSSS-based flow marking for invisible traceback	Potential performance overhead and complexity
[70]	Security threats in the data plane of software-defined networks	Discusses security threats in the data plane of SDNs	Broad overview without specific implementation details
[71]	Wedgetail: An intrusion prevention system for the data plane of software-defined networks	Introduces Wedgetail for intrusion prevention in SDN data planes	Potential performance overhead and complexity

The following considerable work has been done in the restoration of network flow.

SDN controllers use a common protocol to define flows, compute paths, and add flow entries in switches. OpenFlow, the most popular protocol, enables switches to focus on fast traffic forwarding, creating cost-effective networks [72]–[74]. Its flexibility allows dynamic flow restoration during multiple failures, enabling quick path recalculations and deployment for disrupted flows [75], [76]. During link recovery, end nodes automatically find alternative paths [76].

Flow restoration occurs when the origin node creates a new route after failure [77]. IP-based routing protocols like OSPF and RIP [78] restore routes but are slow for large-scale outages [79]. OpenFlow networks enable fast restoration strategies using global knowledge of network resources [80]. A Path Component Element (PCE) at the SDN controller calculates new paths or diversions [81], which can be optimized to reduce latency [82].

Various network flow reconstruction methods have been proposed, such as [83], which classifies single-failure strategies, and [84], which addresses capacity optimization for single-failure recovery. Though some studies [85], [86] consider multiple failures, they do not solve the multi-failure optimization issue, particularly in OpenFlow networks [87], where hundreds of flows may need redirection in disaster scenarios. Processing delays,

such as those for inserting flow entries (0.5–10 ms) [88], are often ignored, yet crucial during widespread failures.

Shortest-path restoration is not always efficient for reducing flow table processing time. As shown in [89], [90], shortest-path algorithms may struggle to restore flows quickly, especially in large networks.

Table 2.5 Major Insights of Works on Network Flow Restoration

Citation	Main Contribution	Methods Used	Limitations
[72]	Cross-stratum resilience for OpenFlow-enabled data center interconnection with Flexi-Grid optical networks	Simulation and experimental evaluation	Limited to specific network configurations and scenarios
[73]	Introduction to OpenFlow as the next generation of networking	Descriptive analysis	Lacks empirical data and quantitative analysis
[74]	Comparison of open standards and proprietary protocols for data center networking	Case studies and comparative analysis	The limited scope of protocols analyzed
[75]	Overview of algorithms for network survivability	Literature review and theoretical analysis	Does not provide empirical validation of algorithms
[76]	Dynamic restoration in wavelength-switched optical networks using PCE	Simulation studies	Limited to wavelength-switched optical networks
[77]	Comparison of path, sub-path, and link restoration for fault management in IP-over-WDM networks	Performance evaluation using GMPLS control signaling	Focused on specific network type, may not generalize to others
[78]	IP protection and restoration mechanisms	Survey of existing techniques	Outdated, newer methods not covered
[79]	QoS-aware fault tolerance for grid-enabled applications	Simulation and performance evaluation	Limited applicability to grid-enabled applications only
[80]	Experience with a globally deployed software-defined WAN (B4)	Real-world deployment and performance metrics	Specific to B4, may not be generalizable
[81]	GMPLS control plane with PCE-based full restoration for translucent WSON	Design and performance evaluation	Focused on translucent WSON, other network types not addressed

[82]	Path selection in user-controlled circuit-switched optical networks	Simulation-based evaluation	Limited to circuit-switched optical networks
[83]	Approaches to failure restoration in optical infrastructure	Review and classification of restoration approaches	Lacks detailed performance evaluation
[84]	Spare capacity reprovisioning for high availability backup path protection	Analytical modeling and simulation	Focused on shared backup path protection, other methods not considered
[85]	Performance study of multiple link failure restorability in shared protection trees	Simulation studies	Specific to shared protection trees, may not generalize
[86]	Network design for tolerating multiple link failures using Fast Re-route (FRR)	Design and simulation-based evaluation	Limited to FRR, other methods not evaluated
[87]	Disaster survivability in optical communication networks	Analytical modeling and simulation	Specific to disaster scenarios, may not apply to other failure types
[88]	Open framework for OpenFlow switch evaluation (OFLOPS)	Design and implementation of the framework	Focused on OpenFlow, limited applicability to other protocols
[89]	Meeting carrier-grade requirements with Software Defined Networking	Performance evaluation and case studies	Focused on carrier-grade networks, limited scope
[90]	Enabling fast failure recovery in OpenFlow networks	Experimental evaluation	Specific to OpenFlow, may not generalize to other network types

## 2.2 Security Issues Related to Control Plane

In this section, different attacks on SDN control and their defense strategies are categorized thoroughly. This classification is conducted on the finding that the controller is the most vulnerable and prone to attack in SDN architecture and therefore, securing the controller access is the most important concern from the security point of view.

### 2.2.1 Attack Classification of Control Plane

It has been proposed that the centralized control plane is structurally weak, even if the separation of the control and data planes allows for the administration of all switches.

More specifically, when switches send out a lot of control signals, one controller may get overloaded. By taking advantage of this, a hacker may launch reflective DDoS assaults that flood control channels with traffic by using SDN switches as reflectors. This assault has the potential to completely take down the control plane and severely impair network performance.

Shin and Gu [27] discuss reflective DDoS attacks exploiting OpenFlow PACKET IN messages. Attackers send packets with varying headers, causing table mismatches and flooding the controller with PACKET IN messages. FloodDefender [91] shows how such attacks can overwhelm a controller's CPU. SWGuard [92] enhances defense by monitoring RTTs to identify specific match fields triggering PACKET IN messages.

SDN controllers maintain a topological view of the network, including host and connection statuses, which is crucial for decision-making. However, attackers can exploit host monitoring and connection detection services to corrupt this view. Link discovery relies on switches sending LLDP packets, and attackers can tamper with connection data as controllers often fail to validate the LLDP packets' origins. TopoGuard [93] and SPHINX [60] highlight host position stealing attacks using fake host IDs, while TopoGuard+ [94] describes port probing attacks, and SecureBinder [95] discusses persona hijacking in DHCP.

Corrupting control-plane constants can impact network efficiency. For example, the ONOS fwd app [96] has a packet\_out\_only option that, when exploited, forces all packets to be sent to the controller, degrading network performance [97]. Additionally, attackers can exploit southbound interface flaws, creating faulty control messages. Shalimov et al. [98] propose a method to check if controllers process faulty OpenFlow messages, while BEADS [99] uses fuzz-testing to reveal vulnerabilities, such as inserting an incorrect length value in an OpenFlow header, leading to crashes. ATTAIN [100] finds that ignoring OpenFlow messages can cause DoS attacks, and BEADS [99] shows that OpenFlow signal mishandling can lead to lost connectivity.

Table 2.6 Major Insights of Works on Attack Classifications on Control Plane

Citations	Main Contribution	Methods Used	Limitations
[91]	Proposed FloodDefender, a solution to protect SDN data and control planes from DoS attacks.	Simulation and experimental evaluation on a testbed.	May not account for all possible variations of DoS attacks in SDNs.
[92]	Identified new control plane reflection attacks and proposed countermeasures for SDNs.	Theoretical analysis and simulations.	May not have been tested in a wide range of real-world scenarios.
[93]	Described new attacks that poison network visibility in SDNs and proposed countermeasures.	Simulation and experimental validation.	The scope of the countermeasures may be limited to specific types of attacks.
[94]	Investigated topology tampering attacks and proposed effective defenses in SDNs.	Simulations and real-world experiments.	Potentially limited by the specific network configurations used in the study.
[95]	Explored identifier binding attacks and proposed defenses in SDNs.	Analysis and experimental validation.	The proposed defenses may not cover all possible identifier-binding attack scenarios.
[96]	Studied the scalability of ONOS reactive forwarding applications in ISP networks.	Performance evaluation through simulations and real network setups.	Results may not generalize to all ISP network environments.
[97]	Demonstrated how a simple malicious application can disrupt an entire SDN environment.	Experimental demonstration and analysis.	The study may not cover all types of malicious applications.
[98]	Provided an advanced study on the performance and reliability of SDN/OpenFlow controllers.	Performance testing and comparative analysis of various controllers.	May not reflect the latest advancements in SDN/OpenFlow controllers.
[99]	Introduced Beads, a system for automated	Automated attack discovery through	The discovery process may miss

	attack discovery in OpenFlow-based SDN systems.	systematic exploration and testing.	some sophisticated attacks.
[100]	Developed Attain, an attack injection framework for SDNs to test network resilience.	Framework implementation and experimental evaluation.	The framework may not cover all possible attack vectors in SDNs.

### 2.2.2 Defence Classification of Control Plane

Proactively installing "wildcard" rules can reduce the controller's workload. DIFANE [101] introduces authority toggles, allowing switches to use wildcard rules for some policy flows, consulting authority switches instead of the controller during table misses. DevoFlow [102] suggests aggressively using wildcard rules for micro-flows, moving most decision-making to the data plane, with minimal changes to existing OpenFlow switches. AVANT GUARD [29] proposes a link migration approach, transmitting traffic only after confirming a TCP session using a syn token, reducing control-plane reliance. SWGuard [94] similarly organizes switch queues by message type, minimizing data-to-control plane communications. To address control-plane saturation and single points of failure, multi-controller platforms like Onix [103] and ONOS [104] have been introduced. OpenDaylight [105] uses a model-driven approach, dividing data into shards synchronized via Raft consensus. Ravana [106] improves replication atomicity with a two-phase protocol, while LegoSDN [107] offers rollback methods using snapshots to restore previous states. TopoGuard [93] tackles topology poisoning by validating port states with host-specific signals like PORT UP and PORT DOWN, preventing attacker ID hijacking. Secure-Binder [95] extends IEEE 802.1x, verifying MAC addresses with certificates, while DFI [108] improves access control using high-level identifiers for host authentication. Fuzzing tools like ATTAIN [100], BEADS [99], and DELTA [109] help identify flaws in protocol handling by testing various input combinations, revealing vulnerabilities in control channel protocols before deployment.



Table 2.7 Major Insights of Works on Defense Classifications on Control Plane

Citation	Main Contribution	Methods Used	Limitations
[29]	Scalable and vigilant switch flow management in SDNs	Introduces Avant-guard for scalable flow management in SDNs	Complexity in deployment and potential performance overhead
[103]	Distributed control platform for large-scale production networks	Develops Onix, a distributed control platform	High complexity and potential issues with fault tolerance
[104]	Towards an open, distributed SDN OS	Proposes ONOS, an open-source distributed SDN operating system	Scalability challenges and initial implementation limitations
[106]	Controller fault-tolerance in software-defined networking	Introduces Ravana, a system for controller fault-tolerance	Potential overhead in maintaining fault-tolerance mechanisms
[107]	Isolating and tolerating SDN application failures	Develop LegoSDN to isolate and tolerate SDN application failures	Performance impact due to additional isolation mechanisms
[102]	Scaling flow management for high-performance networks	Proposes DevoFlow for efficient flow management	May require significant changes to existing network infrastructures
[101]	Scalable flow-based networking	Introduces DIFANE for scalable flow-based networking	Complexity in implementation and potential bottlenecks
[109]	Security assessment framework for SDNs	Develops Delta, a security assessment framework for SDNs	May not cover all potential security vulnerabilities
[93]	New attacks and countermeasures in SDNs	Identifies and mitigates new attacks on SDN network visibility	Potential for evolving attacks that bypass proposed countermeasures
[94]	Effective topology tampering attacks and defenses in SDNs	Demonstrates topology tampering attacks and proposes defenses	Defenses may introduce performance overhead
[95]	Identifier binding attacks and defenses in SDNs	Identifies and mitigates identifier binding attacks in SDNs	Limited to specific types of identifier binding attacks

[99]	Automated attack discovery in OpenFlow-based SDN systems	Proposes BEADS for automated attack discovery	May generate false positives or require constant updates
[100]	Attack injection framework for SDNs	Develops Attain, an attack injection framework for SDNs	Potential to miss novel or evolving attack vectors
[105]	Towards a model-driven SDN controller architecture	Proposes a model-driven architecture for the OpenDaylight SDN controller	Initial implementation may have scalability and performance issues
[108]	Controller oblivious dynamic access control in SDNs	Introduces a system for dynamic access control without controller awareness	Complexity in integrating with existing SDN infrastructures

### 2.2.3 Existing Solutions of SYN Flood Attack on Controller

Sahin et al. [110] propose ConPoolUBF, a function that uses link pooling in configurable switches to handle server requests. This allows routers to perform TCP handshakes, and the P4 switch dynamically allocates resources, facilitating client-server interactions without extra packets. Hwang et al. [112] enhance SLICOTS, originally by Mohammadi et al. [111], by introducing temporary rules to route SYN packets, blocking MACs that exceed a SYN threshold. They also propose S-RST, a lightweight countermeasure.

AEGIS by Ravi et al. [113] mitigates SYN floods by differentiating between malicious traffic and flash crowds. Li et al. [114] introduce SA-Detector, which handles various saturation attacks, including SYN floods and IP spoofing.

Mohammadi et al. [115] propose SYN-Guard, which tracks SYN counts and drops MACs that exceed a threshold. Kim et al. [116] use Round Trip Time (RTT) to detect SYN floods by comparing SYN counts to flexible thresholds. A hashing technique is used by SRL [118] to counter SYN flood attacks by calculating hashable values from packet IP and MSS, replacing rules with lower hash values.

SAFETY [119] detects SYN floods by analyzing entropy in IP addresses, port numbers, and TCP flags. It closes TCP ports if SYN counts exceed adaptive thresholds, distinguishing floods from flash crowds.

Table 2.8 Summary of Main Contributions in SYN Flood Detection

<b>Citation</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[110]	ConPoolUBF: Connection pooling and updatable Bloom filter-based SYN flood defense in programmable data planes	Simulation and performance evaluation	Limited to programmable data planes
[111]	Slicots: An SDN-based lightweight countermeasure for TCP SYN flooding attacks	Simulation and experimental validation	Focused on TCP SYN flooding, may not generalize to other attack types
[112]	Improved lightweight countermeasure scheme to efficiently mitigate TCP attacks in SDN	Simulation studies	Limited to specific attack scenarios in SDN
[113]	Aegis: Detection and mitigation of TCP SYN flood on SDN controller	Experimental evaluation	Specific to SDN controllers, may not apply to other network components
[114]	Detecting saturation attacks based on self-similarity of OpenFlow traffic	Analytical modeling and simulation	Focused on OpenFlow traffic, may not generalize to other types of traffic
[115]	SYN-Guard: An effective counter for SYN flooding attack in software-defined networking	Simulation and performance analysis	Specific to SYN flooding, other attack types not considered
[116]	An effective defense against SYN flooding attack in SDN	Simulation and experimental validation	Limited to SDN, other network architectures not considered
[117]	Effective detection and mitigation of SYN flooding attack in SDN	Experimental evaluation	Focused on SYN flooding, may not generalize to other attack types

[118]	SRL: A TCP SYN Flood DDoS mitigation approach in software-defined networks	Simulation studies	Limited to SDN, may not apply to other network architectures
[119]	SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN	Simulation and performance evaluation	Focused on entropy-based methods, other techniques not considered

### 2.3 Security Issues Related to Application Plane and Northbound Interface

In this section, different attacks emerging from different third-party applications that want to communicate with the controller are thoroughly categorized and discussed. With the emergence of enormous new applications, it is becoming very tedious to judge the trustworthiness of any application. Therefore, threats from these unknown or unverified applications are creating a huge additional security burden. There is as such no security policy in NBI communication and in most cases REST API is used to communicate with the controller which is again highly vendor-specific.

#### 2.3.1 Attack Classification of Application Plane and NBI

SDN applications typically access the controller's northbound APIs, assuming they are benign, but this is not guaranteed. Malicious apps can exploit these APIs to disrupt controller operations. For example, Rosemary [120] shows how System.exit() can forcibly stop Java-based SDN controllers like ONOS and OpenDaylight. Yoon et al. [32] discuss time modification attacks that remove switches from controllers. Additionally, malicious apps can deplete system resources or hijack events to alter payloads, disrupting communication between apps. The lack of access control in SDN controllers allows malicious software to conceal itself. Ropke et al. [121] demonstrate how a rootkit can erase its ID from the controller store and create hidden channels for data theft. Modern SDN controllers use an event-driven framework, enabling complex event chains. Attackers can exploit this complexity to launch indirect chaining attacks. For instance, EventScope [122] highlights how a malicious "HOST ADDED" event with an incorrect IP can cause controller failures.

ConGuard [123] describes TOCTOU (Time-Of-Check to Time-Of-Use) attacks that exploit shared variables, potentially leading to null pointer exceptions. Xiao et al. [124] note that compromised switches can inject unauthorized payloads into OpenFlow messages, allowing attackers to disrupt controllers or extract configuration data. ProvSDN [125] discusses cross-app poisoning, where malicious programs contaminate storage to influence other applications' decisions.

Table 2.9 Major Insights of Works on Attack Classifications on Application Plane and NBI

Citation	Main Contribution	Methods Used	Limitations
[120]	Rosemary: A robust, secure, and high-performance network operating system	Experimental evaluation and performance analysis	Focused on the network operating system, other components not considered
[121]	SDN rootkits: Subverting network operating systems of software-defined networks	Case studies and experimental analysis	Specific to SDN rootkits, may not apply to other types of attacks
[122]	Automated discovery of cross-plane event-based vulnerabilities in software-defined networking	Automated testing and vulnerability discovery	Focuses on event-based vulnerabilities, other types not considered
[123]	Attacking the brain: Races in the SDN control plane	Race condition analysis and experimental validation	Specific to the control plane, other planes not addressed
[124]	Unexpected data dependency creation and chaining: A new attack on SDN	Analytical modeling and experimental validation	Focused on data dependency attacks, other attacks not considered
[125]	Cross-app poisoning in software-defined networking	Experimental evaluation and security analysis	Specific to cross-app poisoning, may not generalize to other vulnerabilities

### 2.3.2 Defense Classification of Application Plane and NBI

Most SDN controllers lack authentication and permission mechanisms for applications, allowing rogue apps to invoke APIs destructively. To enhance security, various access control methods have been proposed. For instance, FortNOX [126] and SEFloodlight [33] utilize digital signatures to track flow rules triggered by specific apps, while Rosemary [120] employs public key infrastructure (PKI) to verify app signatures, preventing rogue program installations.

SM-ONOS [127] and Barista [128] implement role-based access control (RBAC) to limit application behavior based on assigned roles. However, due to the complexity of modern SDN environments, an API-level authorization model is recommended, as suggested by SM-ONOS [129], which combines RBAC with a hierarchical approach for managing rights.

SDNShield [130] presents lightweight authorization filters, allowing operators to select permissions based on a manifest file. AEGIS [131] proposes a natural language processing technique to compare application permissions with the manifest file.

Identifying application-level race conditions during development is challenging. Tools like OFRewind [132] dynamically capture control and data flow to identify issues, while STS [133],[134] uses delta debugging to isolate problematic code fragments. Other tools, such as SDNRacer [36], BigBug [35], and ConGuard [123], analyze recorded event patterns to detect race conditions in multi-threaded applications.

For analyzing complex attack chains, origin graphs help track event paths from the data plane to the control plane. ForenGuard [24] offers origin-based root cause analysis. GitFlow [135] and PicoSDN [136] improve on this by addressing the limitations of earlier methods, particularly regarding interdependence and incomplete origin data.

INDAGO [137] uses machine learning to analyze API call sequences for security policy violations, effectively identifying malicious patterns. Additionally, FLOVER [57], VeriCon [138], model SDN applications as first-order logic for stability verification, while

NICE [139] employs model checking and symbolic operations to manage input space effectively.

Table 2.10 Major Insights of Works on Defence Classifications on Application Plane and NBI

<b>Citation</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[126]	A security enforcement kernel for OpenFlow networks	Prototype implementation and evaluation	Limited to OpenFlow networks, may not generalize to other SDN architectures
[127]	A security-mode for carrier-grade SDN controllers	Design and experimental validation	Specific to carrier-grade controllers, other controller types not considered
[128]	Barista: An event-centric NOS composition framework for software-defined networks	Framework design and experimental validation	Focused on event-centric NOS, other frameworks not addressed
[129]	A security-mode for carrier-grade SDN controllers	Design and experimental validation	A similar type of [127]
[130]	SDNshield: Reconciliation configurable application permissions for SDN app markets	Prototype implementation and evaluation	Focused on SDN app markets, other aspects not considered
[131]	Aegis: An automated permission generation and verification system for SDNs	System design and evaluation	Specific to permission generation and verification, other security aspects not considered
[132]	OFRewind: Enabling Record and Replay Troubleshooting for Networks	Prototype implementation and case studies	Focused on record and replay, other troubleshooting methods not addressed
[133]	Troubleshooting black box SDN control software with	Design and experimental validation	Specific to black box SDN control software, other

	minimal causal sequences		software types not considered
[134]	SDNRacer: concurrency analysis for software-defined networks	Concurrency analysis and experimental validation	Focused on concurrency issues, other aspects not considered
[135]	GitFlow: Flow revision management for software-defined networks	System design and experimental validation	Specific to flow revision management, other management aspects not considered
[136]	Causal Analysis for Software-Defined Networking Attacks	Causal analysis and experimental validation	Focused on causal analysis, other analysis methods not considered
[137]	INDAGO: A new framework for detecting malicious SDN applications	Framework design and experimental validation	Specific to detecting malicious SDN apps, other types of malicious activities not addressed
[138]	Vericon: towards verifying controller programs in software-defined networks	Verification techniques and experimental validation	Focused on controller programs, other components not addressed
[139]	A NICE way to test OpenFlow applications	Testing framework design and experimental validation	Specific to OpenFlow applications, other applications not considered

### 2.3.3 Related Works for Application Authentication

All connectivity levels and SDN layer configurations pose safety issues [140]. These concerns focus on problematic nodes like controllers and northbound interfaces, in addition to network components such as switches and routers. Solutions to the instability of access control for the northbound interface fall into two main categories.

The first approach emphasizes controller security. RoseMary, proposed by Shin et al. [120], manages applications within a controlled environment, monitoring their activities. It enforces authentication to deny illegal access based on the application's identity and



permissions. Ferguson et al.'s PANE [141] introduces an OpenFlow controller that allows users or programs to obtain read/write authorization from network administrators, thereby enhancing efficiency and safety.

The second approach involves developing existing SDN controllers, such as Floodlight and NOX, by integrating security kernel modules. Key examples include PermOF [142], FortNOX [126], and OperationCheckpoint [143]. Cui et al. [144] also present a similar approach. FortNOX emphasizes access authorization auditing, categorizing application functions into administrators, security apps, and non-secure OpenFlow apps. Since network applications vary widely, Wen et al. [142] proposed the PermOF application authorization management system to detail 18 activation privileges and prevent unauthorized access to kernel resources.

While adaptive controllers like FortNOX and SE-Floodlight [145] have evolved, these methods are often tailored to specific controllers and lack transferability. Although substantial work has been done on data and control plane security, challenges related to the northbound interface and application layer remain underexplored.

Table 2.11 Insights of Major Works in Application Authentication

Citations	Main Contributions	Methods Used	Limitations
[140]	A comprehensive survey on SDN security: threats, mitigations, and future directions	Extensive literature review and analysis	The survey is broad but lacks in-depth technical analysis of specific threats or mitigation strategies.
[141]	Developed an API for enabling application-level control of SDNs	API development and experimental evaluation	The focus is primarily on API control, potentially overlooking other critical security aspects.
[142]	Proposed a secure controller platform specifically for	Prototype development and detailed security analysis.	The solution is limited to OpenFlow applications, which

	OpenFlow applications		may not generalize well to other SDN protocols.
[143]	Introduced OperationCheckpoint, a framework for controlling SDN applications	Framework development and case studies for validation	The framework specifically addresses application control but does not consider other security aspects.
[144]	Designed an authentication mechanism tailored for network applications in SDN environments	Mechanism design and experimental validation	The focus on authentication mechanisms may leave other security issues unaddressed.
[145]	Developed a framework to secure the control layer of software-defined networks.	Framework development and comprehensive security analysis	The framework is primarily focused on the control layer, potentially neglecting security concerns related to other layers.

#### 2.3.4 Related Works for Application Trust Management

The authors of [126] propose FortNox as an early authentication mechanism for network applications in SDN environments. It addresses flow-rule collisions and inconsistencies from multiple applications sending requests to the SDN controller. Each flow rule generated by a network application is signed to prevent clashes and to identify the application. FortNox employs a real-time approach to detect and resolve rule inconsistencies.

Rosemary, recommended by the developers of [120], builds on FortNox's principles, focusing on restricting the system call capabilities available to network applications. It uses flow signatures for application identification, while a component called microNOS intercepts requests from applications for access checks.

PermOF [142] offers a fine-grained authorization framework, preventing direct calls to controller memory and minimizing unwanted interactions between network apps and the control plane.

Another study [146] addresses trust issues between the SDN controller and network applications, proposing solutions to mitigate risks introduced by application interactions. When trust is violated, it can lead to various attacks that harm network performance.

Kreutz et al. [10] discuss threats to SDN, including fake traffic streams and control plane vulnerabilities, emphasizing the need for integrated security in SDN design. Chandrasekaran and Benson [147] suggest using fault-tolerant and isolation layers to enhance controller reliability. Betgé-Brezetz et al. [148] recommend a trust assurance technique involving redundant controllers in different operational settings. Yan et al. [149] present a technique for ensuring security and trust in 5G virtual networks through reactive trust management and reliable cloud-based services.

Table 2.12 Major Work Insights on Application Trust Management

<b>Citation</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[126]	Security enforcement kernel for OpenFlow networks	Kernel development, security enforcement analysis	Focused on OpenFlow, limited applicability to other protocols
[120]	Rosemary: A robust, secure, and high-performance network operating system	System design, performance evaluation	Complexity of implementation, potential overheads
[142]	Secure controller platform for OpenFlow applications	Prototype development, security analysis	Limited to OpenFlow applications, generalizability issues
[146]	Trust management framework for SDN applications	Framework design, trust evaluation	Scalability concerns, trust evaluation methods
[10]	Comprehensive survey on software-defined networking	Extensive literature review	Survey-based, may not cover the latest advancements
[147]	LegoSDN: Tolerating SDN application failures	Framework design, failure tolerance evaluation	May not address all types of failures,

			specific to SDN applications
[148]	Trust support for SDN controllers and virtualized network applications	Trust framework development, case studies	Limited to SDN and virtualized applications, potential scalability issues
[149]	Security and trust framework for virtualized networks and SDN	Framework development, security, and trust evaluation	Focused on virtualized networks and SDN, other aspects not addressed

## 2.4 Security Issues Related to Distributed SDN Architecture via East/Westbound Interface

SDN offers centralized control and dynamic configuration, but distributed SDN environments using east/westbound interfaces for controller communication face significant security challenges. These interfaces are prone to unauthorized access, data breaches, and DDoS attacks, which can degrade network performance and cause outages. Trust management between controllers is critical to prevent the propagation of false information and security breaches. Additionally, software vulnerabilities in one controller can quickly spread to others. Robust authentication, encryption, trust mechanisms, and continuous security monitoring are essential to mitigate these risks and ensure SDN reliability. From the distributed SDN perspective controller planes can be categorized into four types, these are-

- Physically Centralized SDN Control Plane
- Physically Distributed SDN Control Plane
- Logically Centralized SDN Control Plane
- Logically Distributed SDN Control Plane

Some relevant and notable works regarding the above-mentioned distributed SDN architecture security are discussed below.

### **2.4.1 Security Issues in Physically Centralized SDN Control Plane**

In terms of straightforwardness, a physically centralized plane of control with just one controller for the whole network makes ideal sense in theory. But when the network expands, just one controller might be insufficient. As it struggles to maintain identical performance assurances while handling a rising number of requests, it is going to get overloaded. Further implementation of security modules as extensions of controller modules creates another significant stress on the controller, but on the other hand, it is a necessity. Common instances of these types of large-scale networks with varying scalability and security needs are data centers and service provider networks. Several thousand switching components are involved in these networks. It is anticipated that a large number of forwarding components with the potential to develop quickly would provide an enormous amount of control events, sufficient to overwhelm a single centralized SDN controller [150]. Research done in [151] demonstrates significant throughput scalability implications for centralized controller techniques. Authors show that to increase the throughput of a centrally deployed controller and satisfy the volume of traffic features found in actual data centers, several controllers need to be employed. Service Provider Networks have a moderate number of nodes in the network, in contrast to data centers. However, because these nodes are typically dispersed geographically, such networks have a relatively vast diameter [152]. For centralized controller systems, this means a distinct kind of controller scaling problem, more precisely, excessive latency.

The study of the behavior of cutting-edge centralized SDN controllers like NOX [153], and Beacon [154] in various networking circumstances has further reinforced the possible flexibility, accuracy, and susceptibility questions associated with centralized controller approaches. Furthermore, Big Switch Networks' highly well-liked Java-based OpenFlow controller Floodlight [155] has significant security and resilience problems. Consequently, SEFloodlight, a follow-up version of Floodlight, has been made available to address these issues by including security apps. Nevertheless, the centralized controller continues to be a possible vulnerability endangering the entire network, even with the security improvements that have been implemented to protect it. Even though it's thought

that none of these centralized architectures can match the scalability and dependability needs of large-scale networks, their widespread usage in research and education has given them more notoriety.

Table 2.13 Summary of Main Contributions on Physically Centralized SDN Controllers

<b>Citation</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[150]	Evaluated the performance of a scalable SDN deployment.	Performance metrics and scalability analysis using various network topologies and traffic patterns.	Limited to specific deployment scenarios; may not generalize to all network environments.
[151]	Analyzed network traffic characteristics in data centers.	Empirical analysis of real-world data center traffic, capturing traffic patterns and volumes.	Focused on data centers; insights may not be directly applicable to other types of networks.
[152]	Discussed the scalability challenges of SDN.	Theoretical analysis and simulation of SDN architectures under various scaling scenarios.	Primarily theoretical; lacks extensive empirical validation.
[153]	Introduced NOX, an operating system for networks to facilitate SDN control.	Development and testing of NOX with various network applications.	Early prototype; performance and scalability in large-scale deployments not fully evaluated.
[154]	Presented the Beacon OpenFlow controller for SDN.	Implementation and benchmarking of Beacon against other controllers.	Focuses on controller performance; does not address broader network-level challenges.
[155]	Compared performance of SDN/OpenFlow controllers POX and Floodlight.	Performance benchmarking using metrics such as latency, throughput, and scalability.	Limited to two controllers; results may not be indicative of all SDN controllers.

#### 2.4.2 Security Issues in Physically Distributed SDN Control Plane

Physically distributed control-plane designs have drawn more interest from researchers recently since they seem to provide a viable remedy for the problems caused by centralized

SDN designs. Consequently, several SDN control plane architectures have been put out in current research. Based on how SDN controllers are physically arranged, there are two basic types of distributed SDN control designs: flat SDN controller designs and hierarchical SDN controller designs.

#### **2.4.2.1 Flat SDN Controllers**

Flat SDN controllers divide the network into horizontal partitions managed by different controllers and their switch clusters. This structure reduces control latencies and enhances resilience. Examples of flat distributed controller platforms include Hyperflow [156], Onix [103], and ONOS [104]. These platforms aim to improve control plane scalability by using interconnected controllers that share a global network view, enabling centrally controlled applications.

Onix partitions the Network Information Base (NIB), assigning each controller instance a specific portion and requiring applications to reduce information accuracy before sharing. Each ONOS cluster manages a subset of network components with a piece of the global view, represented as a graph. In contrast, HyperFlow provides all controllers access to the global view, creating an illusion of complete network management. These systems implement various strategies to ensure reliability and fault tolerance in case of failures or attacks.

Onix [103] is designed to handle infrastructure failures by delegating recovery tasks to a separate management structure following a multi-path protocol. HyperFlow [156] focuses on availability, rerouting affected switches to adjacent controllers when a failure is detected through its publish/subscribe system. Similarly, ONOS [104] prioritizes fault tolerance by connecting each switch to multiple controllers, including standby options for failover.

Recent implementations of SDN controller systems have emphasized enhancing security in distributed control planes [147],[120],[157-159]. While some propose simpler

centralized architectures, their security methods may be beneficial for medium-to-large-scale SDN implementations where management is distributed among multiple controllers.

Table 2.14 Summary of Main Contributions on Flat Physically Distributed SDN Controllers

Citation	Main Contribution	Methods Used	Limitations
[156]	Proposed HyperFlow, a distributed control plane for OpenFlow.	Implemented a distributed control plane using a publish-subscribe mechanism for state sharing.	Limited scalability in extremely large networks.
[103]	Introduced Onix, a distributed control platform for large-scale networks.	Developed a distributed control platform that allows multiple controllers to manage the network state.	Complexity in managing consistency and state synchronization.
[104]	Developed ONOS, an open-source distributed SDN operating system.	Utilized a cluster of controllers to provide high availability and scalability.	Potential challenges in fault tolerance and performance under heavy load.
[120]	Presented Rosemary, a secure and high-performance network operating system.	Incorporated security features and high-performance optimizations within the network OS.	May face issues in adapting to rapidly changing network environments.
[147]	Introduced LegoSDN, a framework to tolerate SDN application failures.	Implemented fault-tolerance techniques for SDN applications using modular composition.	Complexity in ensuring complete fault tolerance across diverse applications.
[157]	Proposed Ravana, a controller fault-tolerance mechanism in SDN.	Used replicated state machines to provide fault tolerance for SDN controllers.	Overhead associated with maintaining consistent state across replicas.
[158]	Introduced Beehive, a simple distributed programming model for SDNs.	Implemented a high-level programming model to simplify distributed SDN applications.	May not fully address the performance overhead of distributed programming.
[159]	Proposed AR2C2, an actively replicated	Used active replication techniques to	Potential performance



	controller framework for SDN resilience.	enhance the resilience of the SDN control plane.	bottlenecks due to replication overhead.
--	--	--	--

#### 2.4.2.2 Hierarchical SDN Controllers

The network controller plane is assumed to be divided vertically into several tiers based on the services that are needed, according to the hierarchical SDN control design. A hierarchical control plane structure can enhance the efficiency and adaptability of SDN, claims [160].

A hierarchical two-layer control framework that divides controller applications into both local and global components is assumed by Kandoo [161]. Kandoo suggests lowering the control plane's total stress without requiring changes to OpenFlow switches. Rather, it creates a two-tier hierarchical controller plane, with the top tier handling non-local events that need a network-wide perspective and the bottom tier handling frequent events that happen close to the data stream. Although there are clear scalability benefits to having local controllers scale linearly without sharing information between them in a control plane arrangement, Kandoo failed to consider strategies for fault tolerance and recovery to safeguard against potential attacks and failures.

However, a two-level hierarchical control structure is suggested by Google's B4 [162], an exclusive intra-domain software-defined wide area network that links its data centers globally, to enhance scalability. Every data center site is managed at the bottom layer by a local site-level control application-hosting Onix SDN controller. The management of these site controllers is carried out via a global SDN gateway which gathers network data from many sites. In this regard, it is noteworthy that the B4 network's scalability is greatly enhanced by the topology abstractions, which entails turning every location into a super-node with an integrated super-trunk to a distant location. To improve the B4 system's availability, B4 uses fault-tolerance and strong reliability methods at every level of the control hierarchy. Particular improvements have been made to these systems following a significant B4 failure.

Another noteworthy SDN addition that stands for the most recent and difficult element of Google's SDN approach is Espresso [163]. Espresso expands the SDN method to the peer edge of Google's network space, where it links to other types of networks globally, building on the first three levels of that technique. Espresso has been manufactured for over two years and is regarded as a massive SDN rollout for the public Web. It distributes more than 23% of Google's total traffic to the Web. Espresso uses a hierarchical controller plane architecture with separate roles for local and global controllers to achieve previously unheard-of scale-out and efficiency. Additionally, Espresso uses commodity servers for scaling by externalizing functionalities into its software according to the programmability design concept. Logical Categorization of SDN Control Plane Distributed SDN control systems can be further classified into theoretically distributed and logically centralized groups based on how information is transferred between controller instances, in addition to the physical grouping.

Table 2.15 Summary of Main Contributions on Hierarchical Physically Distributed SDN Controllers

Citation	Main Contributions	Methods Used	Limitations
[160]	Proposed an optimal hierarchical SDN architecture to enhance scalability and performance.	Used mathematical modeling and simulations to evaluate the performance.	Limited practical implementation and real-world validation.
[161]	Introduced Kandoo, a framework for offloading control applications in SDN to improve scalability.	Implemented a hierarchical control plane and demonstrated scalability improvements using a testbed.	Potential complexity in managing multiple layers of control.
[162]	Discussed B4, Google's SDN WAN, focusing on managing hierarchy, partitioning, and asymmetry for availability and scale.	Describe the architectural design and operational strategies employed in B4, with empirical data from Google's deployment.	Limited details on security aspects and potential vulnerabilities.
[163]	Presented Espresso, a scalable, reliable, and programmable SDN-based	Detailed the architectural principles and deployment experiences,	Challenges in ensuring seamless integration with

	global internet peering solution.	supported by performance evaluations.	existing network infrastructure.
--	-----------------------------------	---------------------------------------	----------------------------------

Figure 2.1 shows the overall layout and physical categorization scheme of distributed SDN controllers.

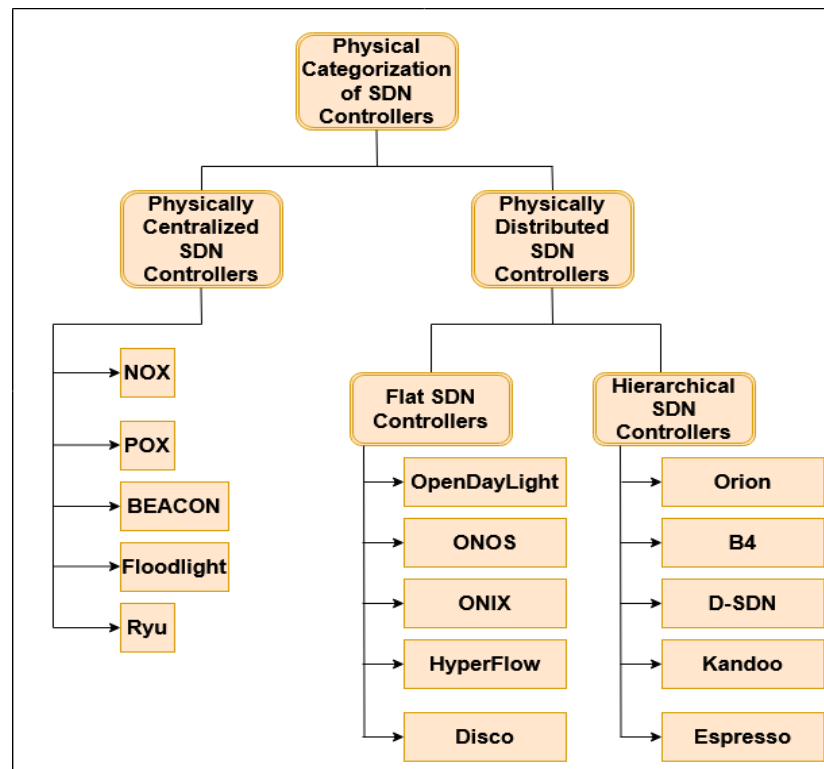


Figure 2.1 Physical Categorization of SDN Controllers

#### 2.4.3 Security Issues in Logically Centralized SDN Control Plane

The global network status is stored by Onix using the NIB data structure as a network graph, which is then dispersed throughout active instances of Onix and synchronized using Onix's state dispersal tools following varying degrees of coherence as required by application needs. Indeed, network apps built on top of Onix build their own data retention and distribution techniques at first by selecting one of two data-store choices that Onix has already implemented in the NIB, in addition to interfacing with the NBI at run-time.

A single controller manages all network choices on the resilient, logically centralized Master-Slave SDN controller architecture known as SMarTLight[164]. In the event of a primary controller collapse, backup controller copies supporting this main controller should be able to take over network administration with a synchronized network viewpoint. Coordinating all controller replicas is done using a common data store maintained by a Replicated State Machine (RSM) architecture that is both robust and resilient.

Logically centralized controller systems HyperFlow [165] and Ravana [106] accomplish state redundancy by employing event repetition. Although the methods by which they construct the application state are similar, there is one distinction: whereas Hyper-Flow necessitates only little alterations to applications, the Ravana interface is invisible to control programs. Furthermore, Ravana provides excellent consistency, even though HyperFlow finally favors accessibility. Hyperflow [165], is a distributed event-based control plane in which a subset of OpenFlow network switches is managed by each NOX-based controller. The distributed file system-based event-propagation publish/subscribe technique it employs is used to propagate specific network events and keep the controllers' shared global network view intact. As a result, the application instance of the Hyperflow controller operating on the top of a single NOX controller which selectively publishes pertinent events that have an impact on the network states and receives events that other controllers have subscribed to.

As a result, each controller instance acts independently and locally (without consulting other controller instances). In other words, they function according to their synchronized, fixed network-wide perspective, functioning as though they are in charge of the whole network. Hyperflow minimizes both flow setup time and network congestion—that is, the cross-site data traffic needed to synchronize controller states—by utilizing this synchronization approach.

The efficiency of the publish/subscribe policy, which can only handle infrequent events, is a possible drawback of Hyperflow. Furthermore, HyperFlow does not deal with

consistency issues or ensure a rigid sequencing of events. As a result, only applications which don't need precise event sequencing with rigorous persistency guarantees can use HyperFlow.

Another group of logically centralized SDN solutions is represented by ONOS[104] and OpenDayLight [166], which differentiate themselves from cutting-edge distributed SDN controller platforms by offering both community-driven open-source designs and the full range of Network Operating System features. These well-known Java-based projects exhibit significant diversity in terms of structure, target audience, emphasis areas, and motivations, despite their evident similarities. OpenDayLight (ODL) [166], supported by the industry and managed by the Linux Foundation, is a universal controller architecture that, in contrast to ONOS, was designed to support a broad range of applications and use cases across several domains. The YANG-based Model-Driven Service Abstraction Layer (MD-SAL) of ODL is a key architectural component that makes it possible to easily and adaptably incorporate network services that are requested by upper layers via the NBI, regardless of the various Southbound protocols that are used among the controllers and the miscellaneous networking devices.

Table 2.16 Summary of Main Contributions on Logically Centralized SDN Controllers

<b>Citation</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[104]	ONOS: A distributed SDN OS focusing on scalability and high availability.	Implemented a distributed architecture for SDN controllers; evaluated performance using testbeds.	Limited discussion on security aspects and potential attack vectors.
[106]	Ravana: A framework ensuring fault-tolerance in SDN controllers.	Developed a fault-tolerant framework; tested fault tolerance using simulations and real-world scenarios.	Complexity in integrating with existing network infrastructure; potential overhead.
[164]	Smartlight: A practical approach to fault-tolerant SDN controllers.	Designed a fault-tolerant SDN controller; performed empirical evaluations on fault tolerance.	Focuses primarily on fault tolerance; scalability aspects are less explored.
[165]	Hyperflow: A distributed control plane enhancing OpenFlow scalability.	Proposed a distributed control plane; validated using experimental setups	Potential challenges in deployment across

		and performance benchmarks.	diverse network environments.
[166]	Qualitative comparison of open-source SDN controllers.	Conducted qualitative analysis; compared features, performance, and robustness of various controllers.	Limited to qualitative comparison; lacks detailed quantitative performance metrics.

#### 2.4.4 Security Issues in Logically Distributed SDN Control Plane

Within certain operational domains such as data centers, campus networks, business networks, and even wide area networks (WANs), the promise of the SDN model has been thoroughly investigated. The fundamental components of SDN—the separation of the controller and data planes and the ensuing capacity to program the network in a conceptually centralized fashion liberated innovative and productive ideas for the administration of these intra-domain networks. These advantages include the efficient introduction of new domain-dependent services and the enhancement of common control operations based on SDN principles, such as intra-domain routing.

Nevertheless, the primary advantage of logically centralized management, which the majority of SDN solutions have utilized to enhance intra-domain network administration, cannot be completely utilized for managing diverse networks with several autonomous systems operating under various administrative jurisdictions. Recent studies have investigated how to apply the SDN paradigm to these inter-domain networks while still maintaining compatibility with its dispersed design. We discussed various SDN solutions in this part, where we explained how they used a logically dispersed design to interact with older networks. Because of this, we classify them as conceptually dispersed SDN systems rather than logically centralized ones, which are mostly employed in intra-domain scenarios.

One such logically distributed control plane design that functions in such multi-domain diverse environments—more specifically, overlay networks and WANs—is proposed by the DISCO project [167]. Each DISCO controller, which is based on Floodlight [168], manages a separate SDN network area and communicates with different controllers to

deliver end-to-end network functionality. A special lightweight control route to communicate network-wide data summaries ensures this inter-AS connectivity. The distinction between the control plane's inter-domain and intra-domain features—each of which is handled by a separate component of the DISCO architecture—is the most evident aspect of DISCO. The inter-domain components are made to allow message-oriented interaction among neighbor domain controllers, while the intra-domain components are in charge of carrying out the controller's primary duties, which include keeping an eye on the network and responding to problems. The most obvious element of DISCO is the bifurcation of the control plane's inter-domain and intra-domain characteristics, each of which is managed by a different part of the DISCO infrastructure. The intra-domain parts handle the controller's key responsibilities, such as monitoring the whole network and taking action when issues arise, while the inter-domain parts enable message-oriented communication between neighboring domain controllers.

A logical deployment of the SDN control layer based on an order of main controllers and secondary controllers is made possible by INRIA's D-SDN [169], which corresponds to the managerial and organizational structure of both the present and future Internet. D-SDN has advantages over DISCO in terms of control hierarchy management as well as improved security and fault tolerance characteristics.

The idea of Software-Defined eXchanges (SDXes) originated with the consideration of implementing SDN at Internet eXchange Points (IXPs). Members from many fields are connected using these SDXes through a common software-based interface. Typically, the goals of that system are to improve inter-domain traffic administration, facilitate the deployment of customized peer policies, and innovate conventional peering. The SDX design comprises of traditional Edge routers and switches, an OpenFlow-enabled switching material, and a smart SDX controller that manages BGP routes and SDX rules. The primary motto of this solution is to provide participating ASes with the ability to independently and at a high-level draft their own rules, which can subsequently be sent to the SDX controller. It is the latter's responsibility to compile such guidelines to SDN rules

for forwarding while accounting for BGP data. In addition to providing a high-level software-based structure that is simple to incorporate into the present system and maintains good connectivity with its routing protocol, SDX distinguishes itself from other solutions of a similar nature, such as Cardigan [170], with the effective processes that are employed to optimize control and data plane operations. iSDX has specifically looked at the scalability issues that SDX faces in practical situations [171].

One major limitation of SDX is that it only benefits participating ASes connected via the software-based IXP, meaning that non-peering ASes are unlikely to profit from the routing possibilities it offers. In response to this problem, several recent research [172] have explored exporting the control logic to a multi-AS routing controller with a specialized perspective across many ASes, therefore centralizing the entire inter-domain routing plane of control to enhance BGP agreement. It's also important to note that SDX-based devices have some security and dependability issues. Since the SDX controller is the main component of the SDX architecture, security methods must concentrate on safeguarding the SDX infrastructure, which includes authenticating all access to the SDX controller and shielding it from assaults. Specifically, Chung et al. [173] contend that in addition to the basic flaws connected to traditional rules, SDX-based controllers are also susceptible to new flaws introduced by SDN.

Lastly, while designing an SDX architecture, fault acceptance, and resilience mechanisms should be considered since the SDX controller represents a possible single point of breakdown. Although the decentralized peer-to-peer SDN SDX architecture [174] is inherently resilient, fault-tolerance measures have to be included in centralized SDX systems.

Table 2.17 Summary of Main Contributions on Logically Distributed SDN Controllers

Citation	Main Contribution	Methods Used	Limitations
[167]	Proposed Disco, a distributed multi-domain SDN controller to manage multiple domains.	Utilized hierarchical architecture with controllers at different levels to manage different domains.	Scalability issues in extremely large networks.



[168]	Extended the Floodlight controller for improved SDN control capabilities.	Implemented additional modules for security and performance enhancements.	Complexity increases with the addition of new modules.
[169]	Explored decentralizing SDN's control plane to enhance robustness and scalability.	Introduced a distributed control plane using a hierarchical design.	Potential latency issues due to distributed nature.
[170]	Presented Cardigan, a live SDN distributed routing fabric at an Internet exchange.	Deployed a distributed control architecture with fault-tolerant features.	Maintenance and troubleshooting complexity in a live environment.
[171]	Developed an industrial-scale SDX to manage traffic at Internet exchange points.	Employed a combination of software and hardware for robust control.	High initial setup and operational costs.
[172]	Proposed a model for routing centralization across domains via SDN.	Emulation framework for BGP evolution with centralized control.	Interoperability challenges with existing network infrastructure.
[173]	Introduced AtlanticWave-SDX, an international SDX supporting scientific data applications.	Implemented a multi-domain SDX with advanced features for data handling.	Coordination issues among different domain administrators.
[174]	Conducted a qualitative analysis of various SDX architectures.	Evaluated different SDX models based on predefined criteria.	Limited to qualitative assessment without real-world deployment data.

Figure 2.2 shows the overall layout and logical categorization scheme of distributed SDN controllers.

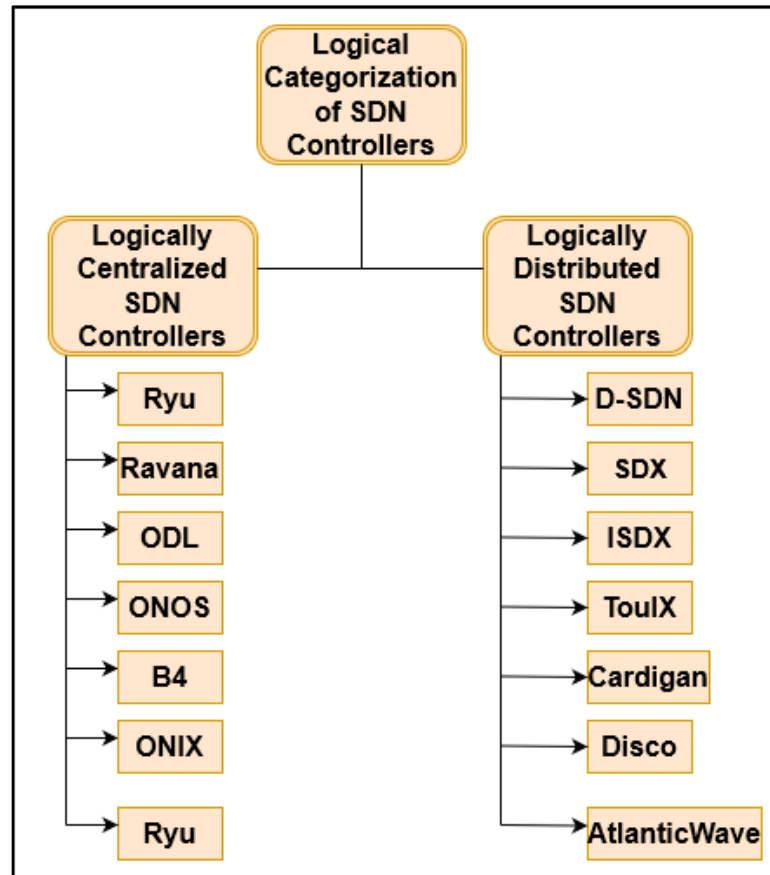


Figure 2.2 Logical Categorization of SDN Controllers

#### 2.4.5 Notable Works for Adaptive Network Synchronization and Security

Logically distributed and logically centralized control planes are the two types of East/West interface options for distributed SDN controllers. A decentralized event-driven control plane for OpenFlow called HyperFlow [165] offers a logically centralized procedure that is applied as a NOX [153] implementation, which is a physically centralized controller. The application uses a publish/subscribe paradigm-based event dissemination mechanism for synchronizing controllers' network-wide views. WheelFS [175], a file system that is distributed and intended to provide wide-area storage for dispersed applications, is used by the system. By adhering to the data, supervision, and controller's channels, monitors keep an eye out for updates in file folders, or channels. The state changes in the network's infrastructure that is utilized to create and preserve the

network's overall perspective make up the subscribed material. HyperFlow strictly requires all controllers to have the same network-wide perspective. Scalable distributed controllers are accomplished via WheelFS. Distributed file systems, on the other hand, are broker-based systems that are increasingly serving as hubs for efficiency bottlenecks. In the event of a breakdown, the danger of network performance deterioration rises with the employment of brokers. WheelFS-distributed data storage devices are called brokers.

Utilizing distributed system concepts, as in Onix [103], a cluster of one or more physical servers offering one or more Onix instances, was another approach that was investigated. Keeping up with a decentralized Network Information Base (NIB), which is a data structure that holds information on the state of the network, is part of the Onix function. The foundation of a distribution system and the heart of Onix control is NIB. A temporal permanent database that is duplicated to spread state changes may be employed for distributing status data, and control logic data for networking behavior is made available to the network using Onix APIs. The approach is only appropriate for networks with slowly changing states due to its severe performance restrictions.

OpenDaylight (ODL), a cluster of collaborating controllers offering network expansion, dependability, and resilience, is suggested by the authors of [105]. ODL controller uses the Model Driven Service Abstraction Layer (MD-SAL) and the Yang modeling dialect to create data structure trees. The four main responsibilities of the MD-SAL are to manage mounts, which are implementations of the MD-SAL, channelize remote procedural calls among procedures, route and organize data reads, and provide a subscription-based message delivery system. The Akka framework, as outlined in [176], is employed to ensure information synchronization among the controller group.

Open Network Operating System (ONOS), an open-source platform managed by The Linux Foundation, is suggested by the contributors of [104] as a substitute for ODL. A cluster of collaborating distributed controllers with excellent efficiency, high reliability, and adaptability is called ONOS. A decentralized data structure that is managed by cluster members preserves comprehensive topology. The ONOS maps each data routing unit to

its master controller using the RAFT [177] agreement mechanism. If a cluster element fails, the RAFT method facilitates replication of the distributed data structure, manages the selection procedure for a new leader, and recovers the controller state after repair.

The authors of [167] suggest DISCO, a distributed SDN control plane for SDN networks with many domains. The agents connected to the domain controllers communicate inter-domain, network-wide data via a distributed control link. For inter-domain controller interaction, the East/West interface procedure uses the Advanced Message Queuing Protocol (AMQP) [178]. Because AMQP is utilized in a client-server manner, its capacity for growth is limited as networks develop. Logically-distributed controllers designed for massive multi-domain systems are studied by the authors of [179]. For interaction among controllers in a logically distributed SDN controller plane, they suggest an East/Westbound interface known as the Communication Interface for Distributed Control plane (CIDC).

Table 2.18 Summary of Main Contributions to Adaptive Network Synchronization and Security

<b>Citation</b>	<b>Main Contribution</b>	<b>Methods Used</b>	<b>Limitations</b>
[175]	Proposed WheelFS, a flexible wide-area storage system for distributed systems.	Utilized a distributed file system to manage storage across multiple nodes.	Potential performance bottlenecks in highly dynamic environments.
[176]	Explored reactive messaging patterns using the Actor model in Scala and Akka.	Implemented actor-based systems to enhance concurrency and fault tolerance.	Complexity in debugging and managing actor systems.
[177]	Studied the response time and availability of RAFT consensus in distributed SDN control planes.	Conducted simulations to evaluate RAFT's performance in SDN environments.	Limited by simulation environment; real-world performance may vary.
[178]	Introduced the Advanced Message Queuing Protocol (AMQP) for reliable	Implemented a messaging protocol to ensure message delivery and reliability.	Overhead associated with ensuring message reliability.

	messaging in distributed systems.		
[179]	Developed an East-West interface for distributed SDN control planes and evaluated its performance.	Implemented and tested an interface to facilitate communication between SDN controllers.	Scalability concerns in extremely large networks.

## 2.5 Comparative Analysis of Recent Existing Research Works

The landscape of SDN (Software-Defined Networking) security has evolved considerably, as evidenced by the numerous studies published in recent years. This comparative analysis seeks to identify common themes, methodologies, and limitations across a selection of research papers published between 2020 and 2024, focusing on SDN security challenges and solutions.

### Common Themes and Problems Addressed

The selected papers collectively address a wide array of security concerns within SDN environments. A recurring theme is the detection and mitigation of Distributed Denial of Service (DDoS) attacks. For instance, Kaur et al. (2024) and Sahin & Demirci (2023) both propose novel methods for DDoS detection and mitigation, highlighting the persistent threat such attacks pose to SDN infrastructure. Similarly, Ravi et al. (2020) and Huang et al. (2022) focus on specific attack types like TCP SYN floods, proposing frameworks to detect and mitigate these threats.

Another significant theme is the security of SDN control planes. The work by Kanwal et al. (2024) provides an in-depth analysis of threats to SDN controllers, underscoring the critical role of the control plane in overall network security. This is complemented by surveys such as those by Han et al. (2020) and Maleh et al. (2023), which offer comprehensive overviews of security threats and mitigation techniques, providing a broader context for understanding specific vulnerabilities and defense mechanisms.

### **Methodologies and Approaches**

The methodologies employed across these studies vary but share some common approaches. Machine learning and automated detection techniques are prominent, as seen in the works of Sebbar et al. (2020) and Ujcich et al. (2021). Sebbar et al. utilize machine learning to detect and defend against Man-in-the-Middle (MitM) attacks, while Ujcich et al. automate the discovery of cross-plane vulnerabilities, demonstrating the effectiveness of AI and automation in enhancing SDN security.

Blockchain technology also emerges as a noteworthy method for securing SDN environments. Ahmad & Mir (2023) and Sandhia et al. (2023) explore the integration of blockchain to decentralize and enhance security, particularly in mobile and distributed SDN contexts. This indicates a growing interest in leveraging blockchain's inherent security features to address the vulnerabilities of traditional SDN architectures.

### **SDN Planes and Their Security Implications**

The research spans all planes of SDN architecture—control, data, and application planes. For example, Wang et al. (2024) focus on constructing secure communication channels within the data plane, while Shaghaghi et al. (2020) provide a comprehensive review of data plane security issues and solutions. This differentiation is crucial as it allows for targeted security measures tailored to the unique challenges of each plane.

### **Limitations and Future Directions**

Despite the advancements, these studies also highlight several limitations and areas for future research. Scalability remains a significant concern, as noted in the works of Kaur et al. (2024) and Kanwal et al. (2024). Both studies emphasize the need for scalable solutions that can handle real-time data processing and large-scale network environments. Similarly, the complexity of implementing blockchain and machine learning solutions is a recurring limitation, as discussed by Ahmad & Mir (2023) and Sebbar et al. (2020).

Future research directions suggested by these studies include enhancing the scalability and efficiency of detection algorithms, exploring new use cases for blockchain in SDN, and developing more robust cross-plane security frameworks. The comprehensive surveys by Han et al. (2020) and Maleh et al. (2023) also call for continuous updates to threat models and mitigation techniques to keep pace with evolving security threats.

This comparative analysis underscores the dynamic and multifaceted nature of SDN security research. By examining a range of studies, it becomes clear that while significant progress has been made in identifying and mitigating various security threats, challenges such as scalability, implementation complexity, and the need for continual adaptation remain. Continued research and innovation are essential to developing robust, scalable, and adaptive security solutions for SDN environments.

Table 2.19 Comparative Analysis of Recent Security Works in SDN

Citation	Publication Year	Problem Addressed	Methods Used	SDN Plane Addressed	Limitations
[186]	2024	Distributed DDoS attack detection in SDN	K-DDoS-SDN approach	Control	Scalability and real-time detection
[182]	2024	Real-time DDoS attack classification in SDN	Kafka streams-based approach	Control	Real-time processing efficiency
[181]	2024	Security processes in SDN-based 5G core networks	Experimental analysis	Data	Limited by specific use cases
[180]	2024	Security channel construction in SDN using SM9	SM9-based security channel	Data	Not generalized for all scenarios
[1]	2024	Security dynamics in SDN controller architectures	Threat landscape analysis	Control	Extensive implementation needed
[110]	2023	SYN flood defense in	ConPoolUBF method	Data	Limited to specific attack types

		programmable data planes			
[140]	2023	Comprehensive survey on SDN security	Survey and classification	All	Broad overview, lacks specific details
[187]	2023	Securing centralized SDN control with blockchain	Distributed blockchain technology	Control	Implementation complexity
[184]	2023	DoS attack protection for smart devices in SDN	SDN-based security framework	Control	Scalability to larger networks
[188]	2023	Security of Software Defined Mobile Networks	Blockchain technology	Data	Specific to mobile networks
[183]	2023	Security enhancement in SDN through routing techniques	Exploration and analysis	Control	Limited to routing techniques
[112]	2022	Mitigation of TCP attacks in SDN	Lightweight countermeasure scheme	Data	Efficiency under heavy load
[220]	2022	Formal safety assessment of DDS protocol in SDN	Formal safety assessment	Data	Specific to DDS protocol
[136]	2021	Causal analysis for SDN attacks	Causal analysis method	All	Complex to implement
[193]	2021	MitM tool analysis for TLS forensics	Analysis of MitM tools	Data	Specific to TLS forensics
[202]	2021	Network traffic analysis using Snort and Wireshark	Snort and Wireshark tools	All	Limited to tool capabilities
[67]	2020	SDN data plane security issues and solutions	Survey and future directions	Data	Broad overview



[47]	2020	MitM detection and defense in large-scale SDN	CBNA-RF based on machine learning	Control	Specific to large-scale environments
[14]	2020	Security threats and mitigation in next-gen SDN controllers	Comprehensive survey	Control	General threats and techniques
[34]	2020	Buffered packets hijacking in SDN	Analysis of match fields	Data	Specific to buffered packets
[122]	2020	Cross-plane event-based vulnerabilities in SDN	Automated discovery	All	Focused on event-based vulnerabilities
[124]	2020	Unexpected data dependency in SDN	Attack analysis and demonstration	All	Limited to specific attack scenario
[113]	2020	TCP SYN flood detection and mitigation in SDN	Aegis framework	Control	Focused on TCP SYN flood
[190]	2020	SDN solutions for sensors in 6G/IoE	SDN-based solutions	Control	Limited to sensor networks

## 2.6 Chapter Summary

This literature review chapter investigates the primary security concerns associated with Software-Defined Networking (SDN) interfaces and planes, categorizing existing threats and proposing solutions to enhance security across the SDN architecture.

### Identification and Characterization of SDN Southbound Interface and Data Plane Security Issues

The literature extensively documents security vulnerabilities in the SDN southbound interface and data plane. Key issues include unauthorized access, data manipulation, and denial-of-service attacks. Proposed solutions involve advanced encryption techniques,

secure channel protocols like SM9, and anomaly detection mechanisms to safeguard communication between the controller and network devices.

### **Characterization of SDN Control Plane Security Threats**

The control plane, being the SDN's brain, faces significant threats, including controller hijacking, malicious applications, and flow rule manipulation. Studies highlight the necessity for robust authentication, authorization, and accounting (AAA) frameworks, and the integration of blockchain technology to decentralize control and enhance security. A novel security architecture leveraging these techniques has shown promise in experimental setups, demonstrating improved resilience against attacks.

### **Identification and Classification of SDN Northbound Interface and Application Plane Security Loopholes**

The northbound interface and application plane are critical for the interaction between the SDN controller and applications. Security concerns here include API exploitation, application-layer attacks, and unauthorized access to network policies. The literature recommends rigorous API security measures, application isolation techniques, and enhanced access control mechanisms. Experimentally verified solutions emphasize the importance of secure coding practices and continuous monitoring to mitigate these vulnerabilities.

### **Security Implementation and Enhancement in Distributed SDN Architecture Using East-Westbound Communication Interface**

In distributed SDN environments, the east-westbound interfaces facilitate communication between multiple controllers, presenting unique security challenges. Identified threats encompass inter-controller communication interception, data breaches, and synchronization issues. Effective solutions involve secure communication protocols, such as TLS, and the implementation of decentralized trust models. Experimental results validate the efficacy of these approaches, ensuring secure and reliable inter-controller interactions.

Overall, this chapter underscores the importance of a holistic approach to SDN security, addressing vulnerabilities across all interfaces and planes. The proposed solutions, supported by experimental evidence, offer a robust framework for enhancing the security and reliability of SDN deployments.

## **Chapter 3**

### **MiTM Prevention and Compromised Switch Detection in SDN Southbound Interface and Data Plane**

Man-in-the-Middle (MiTM) attacks on the Southbound Interface (SBI) of Software-Defined Networking (SDN) pose significant security threats, as they can compromise the communication between the SDN controller and network devices. Detecting MiTM attacks in this context is crucial to maintain network integrity and prevent unauthorized data manipulation. Effective detection strategies often involve monitoring network traffic for anomalies, such as unexpected changes in flow patterns, latency, or packet alterations. Machine learning techniques can be employed to analyze traffic behavior and identify deviations from established baselines, thus flagging potential MiTM activities. Additionally, cryptographic methods like mutual authentication and encryption of SBI communications can enhance security, ensuring that only legitimate controllers and devices can exchange information. Implementing secure key management protocols further mitigates the risk of MiTM attacks by safeguarding the encryption keys. In summary, a combination of anomaly detection, machine learning, and robust cryptographic measures constitutes an effective approach to detecting and mitigating MiTM attacks on the SDN Southbound Interface, thereby protecting the network's operational integrity.

Detecting and mitigating compromised switches in the SDN data plane is vital for maintaining network security and performance. Compromised switches can disrupt network operations by manipulating flow rules, causing unauthorized traffic redirection or denial of service. Effective detection involves continuous monitoring of switch behavior and network traffic for anomalies, such as unexpected flow modifications, traffic spikes, or irregular packet loss. Implementing machine learning algorithms can enhance detection capabilities by identifying deviations from normal traffic patterns. Once a compromised switch is detected, swift restoration of network flow is crucial. This can be achieved by isolating the affected switch and rerouting traffic through alternative paths using the SDN controller's dynamic reconfiguration capabilities. Network administrators

can also employ automated response mechanisms, such as predefined policies that trigger when anomalies are detected, to minimize downtime and maintain service continuity. Additionally, maintaining regular backups of flow tables and configurations allows for quick restoration to a known good state. Combining real-time monitoring, machine learning, and automated recovery processes ensures effective detection and mitigation of compromised switches, preserving the integrity and reliability of the SDN data plane.

### **3.1 Overview of MiTM Prevention**

The telecom industry is being reorganized by the Software Defined Networking (SDN) architecture, which is widely acknowledged as a critical enabler for initiatives like 5G and the Internet of Things (IoT) [190]. Even though SDN has been around for more than 20 years, it is always evolving, and the IT sector is putting more and more pressure on SDN-enabled networks to grow more advanced, flexible, and safe. Nonetheless, as this paradigm expands, a great deal of security concerns are also becoming pertinent and dangerous. One kind of these is called a "Man in the Middle" (MiTM) attack, in which the attacker creates a new communication link with both sides of the communication and alters the data that is received. In actuality, the attacker is in complete control of the chat, but it appears that both sides of the communication are communicating directly to one another over a secure private connection. The catastrophic effect increases with the network's growth. The writers give several examples of MiTM attacks in [191] and [192]. Since Transport Layer Security (TLS) deployment in OpenFlow is not necessary, the protocol is primarily used for interaction between the infrastructure layer and controller layer via the southbound communication interface. In these scenarios, security concerns are not addressed. TLS implementation is entirely vendor-specific; that is, depending on how complicated the implementation is, they may choose not to implement it at all. Consequently, this situation opens a security gap through which attackers may breach the system and cause the network as a whole to malfunction.

The following factors make MiTM the most dangerous:

- An attacker can take control of the controller and deny any connection to it.
- Every conversation between the hosts and controller can be discreetly listened to by the attacker.
- An attacker may choose to discard an outgoing or receiving packet.
- An attacker can reroute any packet for their own malicious purpose.

The goal is to protect the communication between the switches and the SDN controller, hence MiTM must be prevented. A comparative study of attack categorization is shown in Table 3.1.

Table 3.1 MiTM Attack Categorization

Type of MiTM	Attack Process	Reason of Attack	Position of Attacker
Spoofing of ARP	Poisoning of ARP Cache	No authentication mechanism in the ARP protocol.	In between the gateway and target
Spoofing of DHCP	Falsified DHCP Server	No authentication mechanism in the DHCP transmission.	In between the gateway and target
Spoofing of DNS	Poisoning of DNS cache	No authentication mechanism in the DNS server.	In between users and target.
Spoofing of TLS	Injection of false certificate	Take the victim's authentication information and fabricate a TLS channel	In between the web browser and the web server.
Spoofing of ICMP	Forged redirection request	Lack of authentication in ICMP	In between users and target

### 3.2 Proposed System Framework

In this study, AES 256 are used to encrypt data travelling among the SDN controller and the underpinning switches. Mininet has served as the emulator, and OpenDaylight as the SDN controller. The goal of our MiTM attack (ARP and DNS Spoofing) was to get the login credentials from the ODL Dlux interface [192], which is used for authenticating into the controller interface, using Bettercap with SSIStrip [193]. After the assault is effective,

Bettercap may simply capture the Dlux UI login credentials. At this stage, the attacker may take over the controller's access control with ease, contaminate the ARP tables, and do anything they want to do that will eventually cause a denial of service (DoS). This assault will also put the reliability of the shared data in jeopardy.

The Advanced Encryption Standard (AES), a symmetric key encryption cypher, encrypts plaintext to ciphertext using a 256-bit key. The keys will be generated using the Elliptic Curve Diffie-Hellman key exchange (ECDH) method [194]. Without any prior knowledge of one another, two communicating parties can construct a shared secret key via an unsecured communication channel using the Elliptic-curve Diffie-Hellman (ECDH) form of Diffie Hellman. A symmetric key that might be used to encrypt a message is created by merging the public ECC keys of both parties, the sender's private ECC key, and the recipient's public ECC key. The recipient's system receives the ciphertext. A symmetric key that might be used to decrypt the ciphertext is created on the recipient's computer by combining the party data, the sender's public keys, and the recipient's private ECC key. With ECDH (Elliptic-Curve Diffie-Hellman Key Exchange), we will create the AES shared secret key [195].

A timing sequence diagram is presented in figure 3.1 for better understanding of the operations performed. As shown in this diagram, if there is no encryption process then any attacker with the attacking tool can attack and steal the ODL credentials. Therefore, to cater this attack ECDH key exchange algorithm is used to generate public and private key pairs between ODL and Mininet VMs. Now the AES encryption is performed to encrypt the southbound traffic between the controller and underlying data plane switches. Henceforth any attempt of MiTM is not possible because the entire communication channel is encrypted and secured.

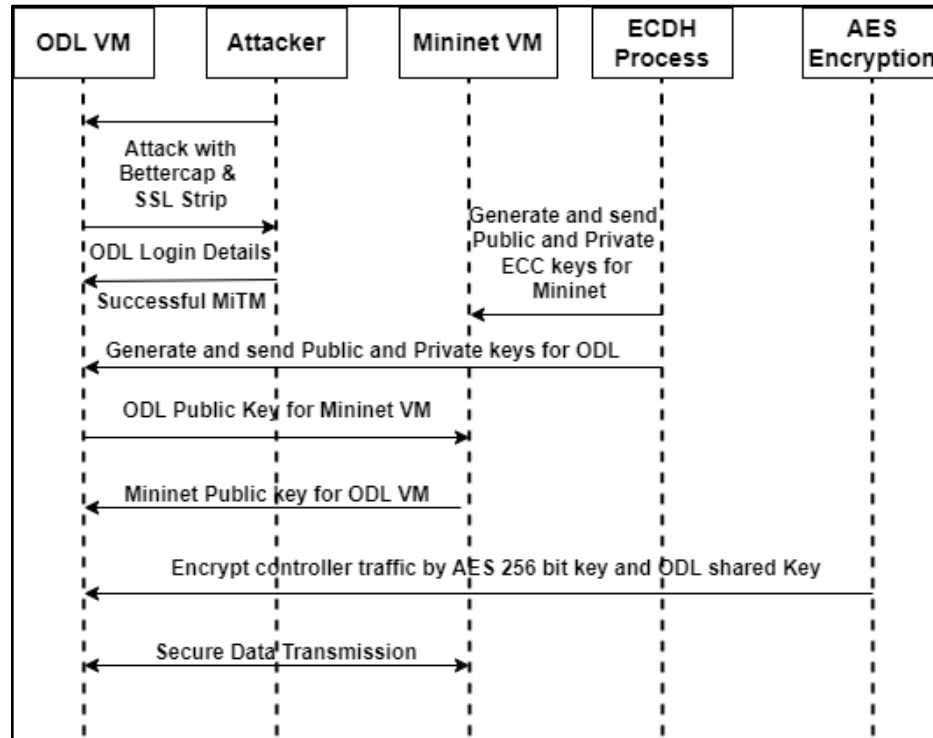


Figure 3.1 Sequence Diagram of MiTM Prevention

### 3.3 Design Strategy

In order to thwart the MiTM attack, the study presented an AES cipher suite based on Elliptic Curve Diffie-Hellman key exchange algorithm. MiTM attackers use the southbound communication interface to spy and mimic any connection the controller has with the OpenFlow switches that make up its underlying data layer. Early prevention of this kind of malicious intention is necessary to reduce the possibility of network and data loss or damage. The suggested approach promises to deliver a completely encrypted Southbound Interface for encrypting controller-to-data layer communication. Because breaking this type of crypto-system is not observable or practical, the combination of AES 256 and ECDH is carefully chosen in this study to integrate private and public key crypto-systems together, increasing the complexity level of the attack. The fundamentals of ECDH and AES are clearly explained using block diagrams in the next two parts, and the suggested solution is provided in the third.



### 3.3.1 Elliptic Curve Diffie-Hellman Key Exchange Algorithm

The Elliptic Curve Diffie-Hellman Key Exchange (ECDH) is an extension of the traditional Diffie-Hellman method that utilizes elliptic curves. It primarily functions as a key-agreement process rather than a direct encryption method. Instead of modular exponentiation, ECDH uses elliptic curve point multiplication, allowing for more efficient and secure key exchanges.

The main goal of ECDH is to enable two parties, such as switches and controllers, to securely share encryption keys. Even if an attacker intercepts the exchanged keys, they cannot decrypt the messages without the private keys.

For instance, when Alice and Bob engage in ECDH, they each generate a private key and a corresponding public key. After exchanging public keys, Alice computes a shared secret using her private key and Bob's public key, while Bob does the same with Alice's public key. This shared secret can then be used for secure communication, ensuring that intercepted keys remain useless to an attacker.

1. Bob and Alice, the first two persons to communicate, will produce their public and private keys. Let  $\mathbf{dA}$  and  $\mathbf{HA=dA*G}$  represent Alice's private and public keys, respectively. Bob's public key is  $\mathbf{HB=dB*G}$ , while his private key is  $\mathbf{dB}$ . ( $\mathbf{G}$ =The elliptic curve's base point)
2. Bob and Alice exchange  $\mathbf{HA}$  and  $\mathbf{HB}$  across an unsecured channel that the attacker may intercept, but he is unable to locate  $\mathbf{dA}$  or  $\mathbf{dB}$ .
3. Bob calculates  $\mathbf{S=dB*HA}$ , and Alice computes  $\mathbf{S=dA*HB}$ .
4. With just knowledge of  $\mathbf{HA}$  or  $\mathbf{HB}$ , the individual in the center would be unable to locate the shared key  $\mathbf{S}$ .

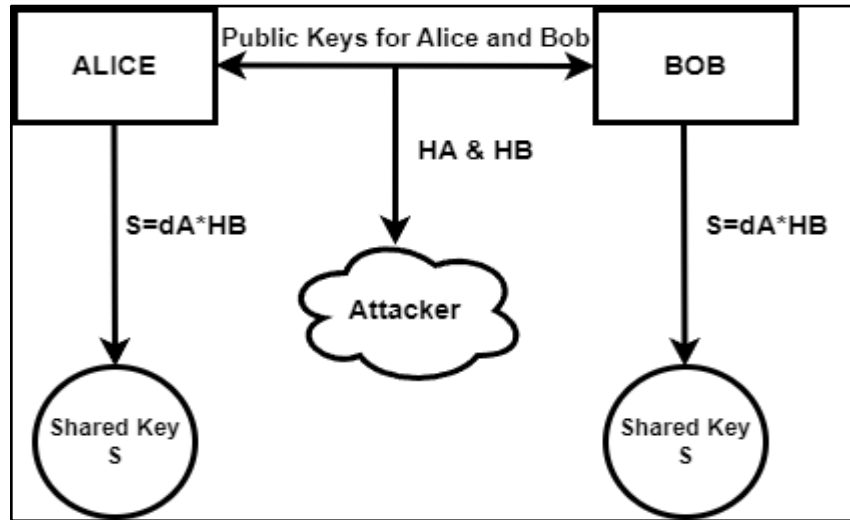


Figure 3.2 ECDH Key Exchange Process

### 3.3.2 Advanced Encryption Standard

AES (Advanced Encryption Standard) is a symmetric key cipher suite, meaning it uses the same secret key for both encryption and decryption. This requires that both the sender and the recipient of the material have a copy of the secret key. One of the primary advantages of symmetric key systems, such as AES, is their speed. Symmetric key algorithms demand fewer computational resources, making them faster and more efficient compared to their asymmetric counterparts. The security key, which is the foundation of all encryption algorithms, is employed to replace each unit of data with a new one. To enhance protection, AES uses a key expansion technique that takes the original key and generates numerous additional keys, known as round keys. Each round key undergoes a series of transformations, making data decryption increasingly complex with each round.

AES operates by taking 128 bits of plaintext and encrypting it using 256-bit keys, ensuring robust security. In our case, the 256-bit key is generated by the Elliptic Curve Diffie-Hellman (ECDH) method, which provides a secure means of key exchange. The ECDH technique ensures that even if an interceptor captures the exchanged key, it cannot be easily deciphered. This 256-bit key is then utilized by AES to encrypt the plaintext, with the process involving multiple rounds of substitution, permutation, and key mixing. Each round further obscures the data, making it extremely difficult for unauthorized

parties to decrypt without the correct key. This layered approach of key expansion and multiple encryption rounds ensures that AES provides a high level of security and efficiency, suitable for a wide range of applications.

The primary key is initially added to the block using an XOR cipher, a feature pre-built into processor hardware. Then, each data byte is replaced with a new one according to a specified table. The next step involves shifting the bytes in the second row of the 4\*4 array one place to the left, the third row's bytes two spaces, and the fourth row's bytes three places. The four bytes in each column are then added together to mathematically mix the columns. Next, adding the round key to the block is done in the same way as adding the first key, and this is done for every round. To get a greater scrambling effect, this step is repeated fourteen times for AES 256. As a consequence, the ciphertext is very different from plaintext. The order of decryption is reversed. Figure 3.2 shows the block diagram for AES encryption.

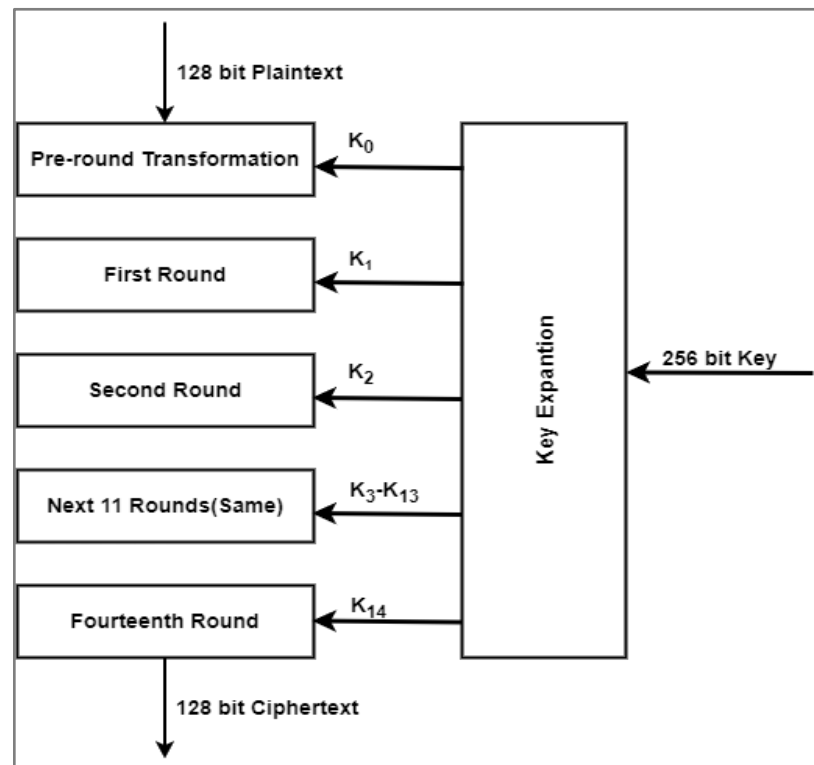


Figure 3.3 Block Diagram of AES

### 3.3.3 Proposed Solution Approach

We are aware that the elliptic curve obeys the formula  $y^2 = x^3 + ax + b$ . One of the safest variations of the Diffie-Hellman key exchange method is created when this ECC is coupled with the current version. The AES variation that we are utilizing, AES 256, is thought to be the most secure of all the AES variants. We have integrated the ideas of symmetric and asymmetric key cryptosystems in our study to offer strong protection against Man-in-the-Middle attacks.

The controller and infrastructure layer switches are connected over an unsecured HTTP link at first, and later on, the secret ECDH key is used for AES encryption. Any traffic between the controller and switches may be sniffed by the attacker, but the traffic cannot be decrypted.

#### Algorithm 1 MiTM Prevention

- 1: Generate random ECC public and private key pairs for ODL controller  
 $\text{ODL\_private\_key}, \text{ODL\_public\_key} \leftarrow \text{generate\_ECC\_key\_pair}()$
- 2: Generate random ECC public and private key pairs for data layer OF switches emulated using Mininet VM  
 $\text{Mininet\_private\_key}, \text{Mininet\_public\_key} \leftarrow \text{generate\_ECC\_key\_pair}()$
- 3: Share the public keys between Mininet VM and ODL VM using port 6633  
 $\text{send}(\text{ODL\_public\_key}, \text{port} = 6633, \text{destination} = \text{Mininet\_VM})$   
 $\text{send}(\text{Mininet\_public\_key}, \text{port} = 6633, \text{destination} = \text{ODL\_VM})$
- 4: Calculate controller's shared key for ODL using controller's private key and Mininet VM's public key  
 $\text{ODL\_shared\_key} \leftarrow \text{calculate\_shared\_key}(\text{ODL\_private\_key}, \text{Mininet\_public\_key})$
- 5: Calculate Mininet VM's shared key using its own private key and controller's public key  
 $\text{Mininet\_shared\_key} \leftarrow \text{calculate\_shared\_key}(\text{Mininet\_private\_key}, \text{ODL\_public\_key})$
- 6: Encrypt outgoing controller traffic using AES and 256-bit shared ECDH key and send it via port 6633  
 $\text{encrypted\_traffic} \leftarrow \text{AES.encrypt}(\text{outgoing\_controller\_traffic}, \text{ODL\_shared\_key})$   
 $\text{send}(\text{encrypted\_traffic}, \text{port} = 6633, \text{destination} = \text{Mininet\_VM})$

Figure 3.4 MiTM Prevention Algorithm

### 3.4 Experimental Setup and Result Analysis

The setup arrangement is deployed across three virtual machines. Using a linear topology and a Mininet emulator, 20 hosts, and 20 switches are installed at the infrastructure layer. The experimental setup is presented as follows:

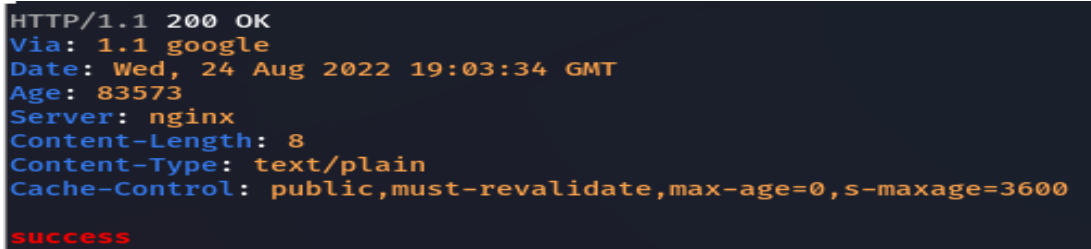
- **Controller:** An Ubuntu 20.04 LTS virtual machine running OpenDaylight 14 (Silicon) with the IP address 192.168.144.133 has been utilized as the controller. In order to interact with Mininet VM, it uses port 6633, and with Dlux UI, it uses port 8181.
- **Emulator:** A set of hosts, OpenFlow switches, and network hosts are replicated via an emulator known as Mininet. It is able to generate simple hardware-like topologies. Utilizing IP address 192.168.144.130, it is installed on a virtual machine running Ubuntu 20.04 LTS.
- **Attacker Machine:** An apparatus designed to initiate a MiTM attack by using tools and other tactics while monitoring certain behaviors in daily tasks. It installs on Kali Linux and uses the IP address 192.168.144.132. Utilizing SSLStrip and the Bettercap tool on top of Kali Linux, we were able to create an ARP spoofing attack in addition to an HTTP spoofing attack on MININET and ODL virtual machines. The testbed overview is displayed in Fig. 3.4.

IP ▲	MAC	Name	Vendor	Sent	Recvd	Seen
192.168.144.132	00:0c:29:a6:13:88	eth0	VMware, Inc.	0 B	0 B	13:40:26
192.168.144.2	00:50:56:e4:95:24	gateway	VMware, Inc.	28 kB	26 kB	13:40:26
192.168.144.1	00:50:56:c0:00:08	DESKTOP-U0I90IM.local	VMware, Inc.	11 kB	14 kB	13:59:12
192.168.144.130	00:0c:29:44:ff:c6		VMware, Inc.	42 kB	73 kB	14:00:54
192.168.144.133	00:0c:29:90:bb:37		VMware, Inc.	43 kB	73 kB	14:00:54
192.168.144.254	00:50:56:e3:64:8a		VMware, Inc.	1.4 kB	15 kB	13:54:13

Figure 3.5 Testbed Overview

### 3.4.1 Execution Logic

Using the Bettercap tool with SSLStrip, an attack is created in Kali Linux from a virtual computer with the IP address 192.168.12.132. This virtual machine is intercepting traffic between the ODL VM (IP 192.168.144.133) and the mininet VM (IP 192.168.144.130). All of these virtual machines are running Windows 10 on VMware 15 Workstation Pro with an Intel Core i5 CPU and 12 GB of RAM. The effectiveness of the MiTM attack is demonstrated by the traceability of the controller credentials and the unambiguous demonstration of Bettercap's communication interception between two virtual machines (VMs) in Figures 3.4 and 3.5. At this point, an attacker may easily get access to the controller, the SBI, and the infrastructure-level hosts and switches.



```
HTTP/1.1 200 OK
Via: 1.1 google
Date: Wed, 24 Aug 2022 19:03:34 GMT
Age: 83573
Server: nginx
Content-Length: 8
Content-Type: text/plain
Cache-Control: public,must-revalidate,max-age=0,s-maxage=3600
success
```

Figure 3.6 Sniffing Status

### 3.4.2 Result of Encryption

The programs for ECDH and AES 256, which are implemented in ODL via RESTful API, were written with Python 3.8.2. The ECDH technique (Fig. 3.6) is used to produce the shared encrypted key first, and then it is utilized for AES 256 encryption (Fig. 3.7). We can verify that the ECDH technique is used to produce a shared key between ODL VM and Mininet VM based on the images in Figures 3.6 and 3.7. Two virtual machines have been sharing this key on port 6633. After that, AES 256 was put into practice using the key produced by ECDH, successfully encrypting all subsequent SBI communications as well as the ODL VM login credentials. If this two-layer encryption is used optionally, the entire SBI connection will adhere to conventional OpenFlow communication guidelines without any security protections.

```
Mininet public key: 0x9941a153aeb864ac5a11b583ffade56460e1424936693da1135bc218fa71550e1
OpenDaylight public key: 0x5318c005e7d44e99afb3458b14ccedf10be8b0ef5af119125120a26884149fd20
Now exchanging the public keys through SBI
Mininet shared key: 0x892e29de7223f644613258e8ae9d0faf116ce27f426207524207aa3e77c4ebd21
OpenDaylight shared key: 0x892e29de7223f644613258e8ae9d0faf116ce27f426207524207aa3e77c4ebd21
Equal shared keys are: True
```

Figure 3.7 Result of ECDH Key Exchange

```
Encrypted Message {'admin': b'70e75bb69404744cde7aead893149f40', 'ciphertext':
b'd43e160717878dacc6d15913e37921e9649b74e7c751a605742de5b85c' }
```

Figure 3.8 Result of AES Encryption

### 3.5 Security Achievements

The eminent security achievements from the work mentioned above are detailed below.

- An innovative encryption-based method is proposed to fortify the security of southbound communications by ensuring that all data transmitted over this link is encrypted. Encrypting these communications eliminates the possibility of interception and tampering, thereby preventing potential attacks and enhancing the overall security of the SDN system.
- The OpenDaylight (ODL) SDN controller was the target of a successful attack, which compromised the Southbound Interface (SBI) communication and the network as a whole. Demonstrating these vulnerabilities provides a clear target for implementing security improvements, helping to prevent similar attacks in the future.
- The study focused on mitigating MITM attacks by encrypting communication between the infrastructure layer and the controller using AES-256 and ECDH algorithms. Implementing strong encryption methods like AES-256 and ECDH shields messages from manipulation and eavesdropping, greatly enhancing the security of the SDN environment.
- The results emphasize the importance of having strong encryption methods to protect SDN infrastructure from potential attackers. Reinforcing the necessity of robust encryption techniques ensures ongoing protection against evolving security threats, thereby securing the SDN infrastructure effectively.

### 3.6 Overview of Compromised Switch Detection and Restoration of Network Flow

A compromised switch has the ability to modify its own flow entries in order to carry out nefarious operations such as packet loss or unapproved forwarding. It can also establish a connection with a rogue controller and transmit fictitious OpenFlow messages, which is one way an attacker may manipulate the control flow to their benefit. A further attack goal is to use the topology discovery function to fool the controller into believing links exist that do not.

As such, it is imperative to quickly detect SDN switches that have been exploited. If not, network traffic passing via these compromised switches might result in a DDoS or DoS assault, which would eventually impair the entire network. Another crucial issue here is the isolation of hacked SDN switches and the restoration of network traffic flow. After the compromised SDN switches are isolated, maintaining the correct path from sender to receiver becomes a difficult operation since it takes a lot of resources and there may be similar types of attacks in the newly formed path as well. As a result, eliminating the compromised device alone might not be enough to preserve network integrity; instead, we need to eliminate all of the links that are connected to it. Although it places a significant load on the controller and CPU, finding corrupted switches, isolating them, and restoring the original network flow is crucial from a security standpoint. To work in tandem with the primary or master controller, in which all the aforementioned features will be deployed, the deployment of a secondary or backup controller in addition to the first controller has been suggested in our method. For the prototype implementation and performance assessment of the suggested algorithms, OpenDaylight (ODL)[196] was employed as the master or major controller, and ONOS [104] was used as the slave or secondary controller in the study. OpenFlow networks provide users with a single point of contact from which they can quickly determine the optimal restoration procedures and obtain information about participating switches, existing paths, and flows. The additional paths or diversions are computed on the Path Computing Element (PCE), a dedicated computing platform in the SDN controller that can account for many constraints for



sophisticated route calculation. By adding or removing flow entries from/to the connected switches using the flow tables of that path, the SDN controller alters the configuration of the hacked switches. Transmitting data packets to the controller and other switches is all that data plane switches perform. They are intelligent devices. This conduct greatly increases the likelihood that an attacker may exploit it. A rogue or broken SDN switch might seriously impair network connectivity. The anomalous behavior at an SDN switch may be categorized into five groups.

1. **Traffic loss**, or the inability of a hacked or corrupted switch to pass network traffic, can occur either randomly or selectively.
2. Sending traffic from a hijacked switch to the wrong address is referred to as **traffic misrouting**.
3. The technique of changing the packet order on a compromised switch is known as **traffic reshaping**.
4. A hacked switch may modify the contents of the traffic, a phenomenon referred to as **traffic modification**.
5. A hacked switch may cause **traffic delays**, which is a potentially catastrophic hazard in time-sensitive communications.

The SDN architecture with a few compromised switches is shown in Figure 3.8. The graphic illustrates how an SDN controller functions as a central liaison between the data plane switches and the applications that are housed in the application layer in a typical SDN architecture. The northbound interface (NBI) is used by applications to connect with the controller, while the southbound interface (SBI) is used by the controller for interaction with the data plane switches. Therefore, in order to establish flow rules in the data plane switches, applications require controller interaction. As seen in the accompanying figure, the entire network can become unstable or disconnected if an attacker manages to hack one or more switches. These switches are vulnerable to any traffic, which might have a major effect on the entire network. As a result, it is critical to identify hacked switches of this kind as soon as possible and to isolate and recreate the flow thereafter.

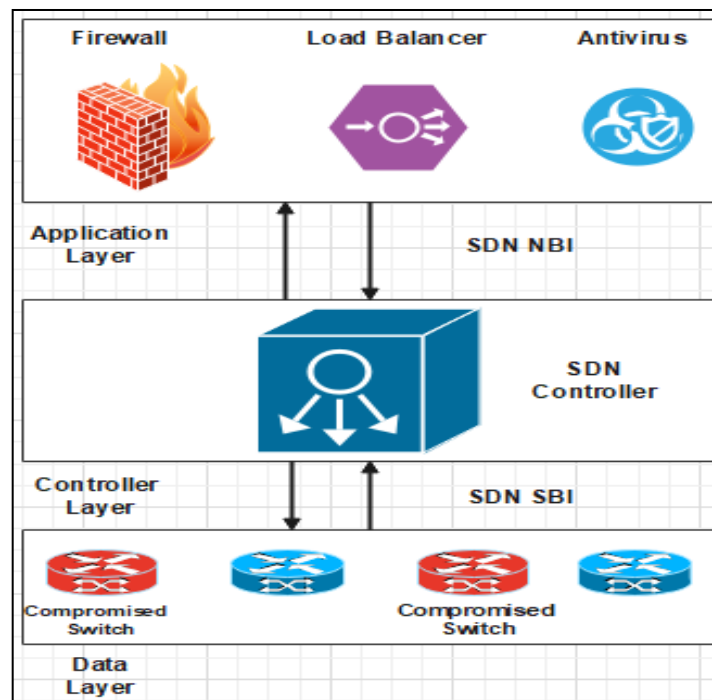


Figure 3.9 SDN Architecture with Compromised Switches

### 3.7 Impact of Compromised Switches

Based on the routing-related data in the packet, when a packet is transferred from a host to its neighboring SDN switch, the switches will first look for a similar item in the flow table. The specific switch will encapsulate the packet content as an OpenFlow [2] PACKET\_IN message and transmit it to the controller if it cannot find a matching entry inside the flow table. The controller will then extract the packet from the PACKET\_IN message and obtain the routing data. Any packet's transmission path will be deployed by the controller programs, and switches will then get the proper instructions to deploy the required data flow via the southbound interface.

The malevolent influence of exploited SDN switches extends to the controller and switches and finally affects the entire network. Since the controller is the primary element of the SDN design, any negative impact on it will also have an impact on the entire network. The following is a summary of the main points of attack for hacked switches on the controller and the whole network:

- An attacker may direct a controller that uses a hacked switch to erase every entry in its flow table in order to disable the switches.
- The attacker can seize and utilize system resources that belong to authorized users at will by exploiting the compromised switches.
- The attacker may continue taking controllers to control additional switches that are tied to them until the attack aim is accomplished.
- The network topology and flow table entries may be changed by the attacker, which would interfere with normal network operations.
- If the attacker tampers with the controller's log, operators might not be aware of the attempted hijacking.

### **3.8 Proposed Solution Approach**

The recommended technique is divided into two main sections: switch isolation and network path rebuilding are carried out when compromised switches have been detected. The following lists the guiding ideas for each of these pursuits.

#### **3.8.1 Proposed Compromised Switch Detection Framework**

Based on the investigation, it is essential to gather data on each network update event and the associated master controller and switch handling data in order to detect possibly compromised SDN switches in a reliable and timely manner. This is the first step to performing. Therefore, the following data are required to be collected primarily-

- Each request for network update.
- Network update instruction received by switches from both controllers.
- Execution log of that request by the switches.

To collect this information as a whole set, an idiosyncratic ID is assigned for each network update request. The actions performed by the primary controller are mirrored in the secondary controller. The execution log of the secondary controller and switches for any particular network update request ID contains information on network state update. A similar type of information is generated from the master controller but in case of any compromization of any switch its flow table entries do not match with what is obtained from the secondary controller. Therefore, these execution results from primary and secondary controllers can be compared to find any inconsistency in the execution log of the switches. If any such inconsistency is detected, then the corresponding switch is treated as a compromised switch, and it is subject to be isolated from the network.

#### **3.8.2 Proposed Switch Isolation and Network Flow Reconstruction Framework**

Once detected, compromised switches can be isolated logically by deleting their corresponding flow table entries from the primary controller. As we are maintaining similar copies of flow table entries for both the primary and secondary controller, flow

table entries in the master controller get updated too. At this point, the network is fully or partially unreachable due to the absence of the switches or links and therefore network reconstruction is essential at this point to maintain the normal operational flow.

To reconstruct the network flow, there are two major considerations- i.e. the link cost factor or link weightage between two switches and the flow table alteration cost (internal processing time) if any such link is inserted or removed from the existing flow table. In this study, these two parameters are important because normally a network can be thought of as a connected directed graph where switches are like graph vertices and link between them are the edges. Removal or insertion of any edge means altering the existing flow table which incurs flow table alteration cost because subsequent flows must have to be altered to accommodate these changes. If there is a huge flow table modification then a considerable amount of processing power and time is wasted to perform this job, so flow table alteration cost should be minimized as far as possible.

Another important concern is link weightage which might be thought of as any feasible metric like distance, time, etc. As we all know, that spanning tree is a tree that connects all the vertices of a graph, and minimum cost spanning tree incurs minimum cost among them, MST is a good choice for network flow reconstruction. Therefore, from the set of all possible paths from source to destination, the path that will incur minimum link cost has been considered in this study assuming the link cost factor as a constant, because internal processing time will pose serious time overhead on overall network and CPU performance if not tactfully managed and demands more research highlights.

### **3.8.3 Proposed Algorithms for Compromised Switch Detection and Restoration of Network Flow**

To detect compromised SDN switches four types of information are collected for each network update request under a single ID-

1. The request for network update.
2. The execution report by the primary controller-  $E_M$

3. The execution report by the secondary controller-  $E_S$
4. The execution report by the switches-  $E_T$

Based on the above-collected information, the secondary controller can make the decision whether a switch is compromised or not by comparing the execution log of both controllers. This is informed to the master controller immediately to perform the subsequent actions of switch isolation and path reconstruction. The following might be observed by the secondary controller against each ID-

- $E_M=E_S=E_T$  which implies all the network update operations have been performed successfully without any issue.
- $E_M \neq E_S$  which implies execution reports of master and secondary controller are not the same and switch tampering may occur.
- $E_S \neq E_T$  which implies execution report of the secondary controller and switches are not matching and that the switch has been compromised

Let  $CON_M$  denote the master controller and  $CON_S$  denote the secondary controller.  $Req_i$  is the network update request for any ID denoted by  $ID_i$  where  $i=1,2,3,\dots,n$ .  $S$  is the set of all switches controlled by  $CON_M$ ,  $S_i$  is the set of all switches participating in  $Req_i$ .

The detection algorithm is presented in Figure 3.9 which takes a set of network request updates from all legitimate sources and provides a set of compromised SDN switches. It can be observed from Figure 3.9 that the compromised switch detection algorithm takes the parameters mentioned above and checks the execution report of the primary controller, secondary controller, and the switches and loosely predicts any anomaly if the execution report of the primary and secondary controller does not match. Afterwards, the algorithm finally detects the compromised switch if there is a mismatch between the secondary controller log and the same of the switches.

```

Input: A pool of network update requests from legitimate sources
Output: A set of compromised SDN switches
1 if Controller is  $CON_M$  then
2   Send  $Req_i$  ( $i=1,2,3 \dots N$ ) to  $CON_S$ ;
3   Execute  $Req_i$ 
4 end
5 else if Controller is  $CON_S$  then
6   Generate  $Record_i$  by using  $ID_i$ ;
7   Add  $Req_i$  to  $Record$ ;
8   if  $E_M \neq E_S$  then
9     The switch might be compromised.
10  end
11 end
12 else if  $E_M = E_S$  then
13    $\forall E_T$  if  $Req_i = ID_i \neq S_i$  and  $E_S \neq E_T$  then
14     The switch has been compromised.
15   end
16 end

```

Figure 3.10 Compromised Switch Detection Algorithm

At this point, all the compromised SDN switches are detected and hence comes the subsequent part of the process i.e. isolation and network flow/path reconstruction. For isolation of the compromised SDN switches corresponding flow table entries are removed from the master controller as well as from the secondary controller too.

Now at this point, we can consider the whole network as a weighted directed graph  $G(v, e)$  where  $v$  is the set of switches,  $e$  is the set of links between them, and  $e_{ij}$  is a link between node  $v_i$  to  $v_j$ . Here the term switch and node are operationally synonymous and used interchangeably in this literature. The cumulative cost of link  $e_{ij}$  is denoted by  $cost_{ij} > 0$  which may be any metric like weightage, delay, hop distance, etc. We denote the  $n$ th source-target pair by  $S_n$  and  $D_n$  respectively.

The following points are considered to reconstruct the network flow-

- A link was there between  $S_n$  and  $D_n$  or can be established.
- $P$  is a path between any two pairs of nodes, and it contains no self-loop.
- The path cost denoted by  $cost_{ij}$  is 1 if it traverses  $e_{ij}$  and 0 otherwise.
- Path cost is additive.

An exemplary snapshot is presented in Figure 3.10, illustrating a network of 13 nodes. Initially, there was a path between node N1 and node N6 that passed through nodes N11, N12, and N13, with a total link cost of 5, as the cost between nodes  $cost_{ij}$  is additive. However, nodes N11 and N7 were compromised, leading to the deletion of their subsequent links from the controller's flow table.

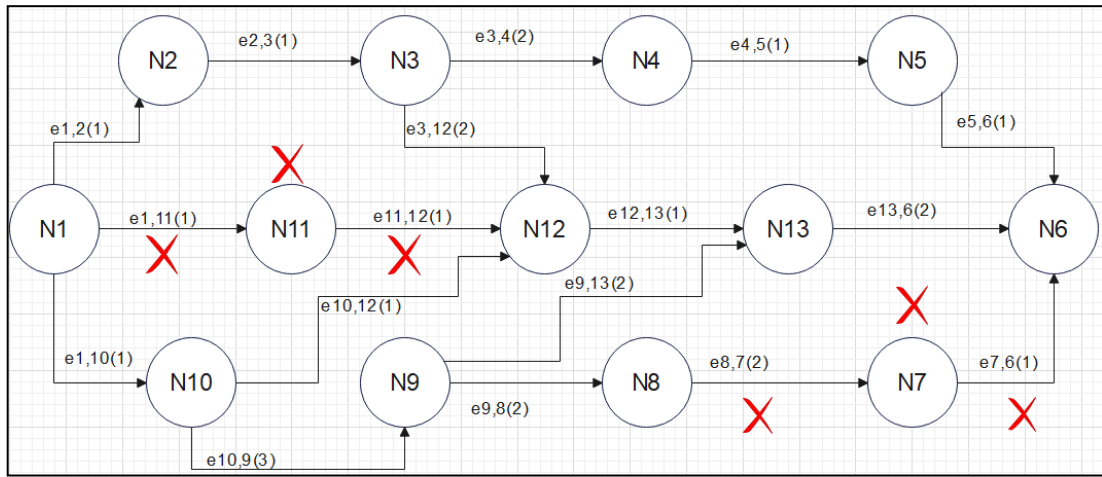


Figure 3.11 Network Flow Reconstruction with 13 Nodes

As a result, four alternative paths to reach node N6 from node N1 were identified:

1. Path  $P1$  :  $N1 \rightarrow N2 \rightarrow N3 \rightarrow N4 \rightarrow N5 \rightarrow N6$ , with a total cost  $cost_{ij}$  of 6.
2. Path  $P2$ :  $N1 \rightarrow N2 \rightarrow N3 \rightarrow N12 \rightarrow N13 \rightarrow N6$ , with a total cost  $cost_{ij}$  of 7.
3. Path  $P3$ :  $N1 \rightarrow N10 \rightarrow N12 \rightarrow N13 \rightarrow N6$ , with a total cost  $cost_{ij}$  of 5.
4. Path  $P4$ :  $N1 \rightarrow N10 \rightarrow N9 \rightarrow N13 \rightarrow N6$ , with a total cost  $cost_{ij}$  of 8.

Among these alternatives, Path P3 was selected as the newly constructed path due to its minimal link cost of 5 and the minimal flow table alteration required. Corresponding flow table entries for this path were updated in both the master and secondary controllers, ensuring efficient and secure network operation despite the initial compromise.

Figure 3.11 depicts the network flow restoration algorithm which receives all the compromised SDN switches and associated links between them thereby providing a path by which switches can reconnect again after eliminating the compromised ones. It depicts that if there is a connection between two distinct nodes, then there might be different paths



from these two nodes which may incur different cost values. Therefore, the minimum cost-valued path has been chosen here as a path between two such nodes.

**Input:** Set of compromised switches and links  
**Output:** Path between any two switches ( $S_n$ ,  $D_n$ )

1. For each source-destination pair ( $S_n$ ,  $D_n$ ):
2.   If any switch is compromised:
3.       Initialize  $\text{minCost} = \infty$
4.       Initialize  $\text{bestPath} = \text{None}$
5.       For each path available from  $S_n$  to  $D_n$ :
6.       Calculate  $\text{Cost}_{ij}$  for the path
7.       If  $\text{Cost}_{ij} < \text{minCost}$ :
8.            $\text{minCost} = \text{Cost}_{ij}$
9.            $\text{bestPath} = \text{currentPath}$
10.       Else if  $\text{Cost}_{ij} == \text{minCost}$ :
11.            $\text{bestPath} = \text{currentPath}$
12.   Return  $\text{bestPath}$

Figure 3.12 Network Flow Reconstruction Algorithm

### 3.9 Simulation Setup and Implementation Result Analysis

As per the prototype implementation of the proposed algorithm for the detection of compromised switches, a random switch MAC and link flooding attack have been initiated from the Macof tool of Kali Linux. Detection time and overhead on CPU performance or utilization have been measured for the compromised switch detection algorithm and flow path construction time along with overhead posed on the CPU for running the second algorithm has been measured for the network flow reconstruction algorithm. Similar types of matrices are used for both algorithms to achieve a better symmetrical view. After

successful detection, a network flow reconstruction algorithm has been implemented on a virtual linear network of 25 hosts and 25 switches.

### 3.9.1 Simulation Testbed Details

Three virtual machines were deployed for the setup configuration and switches were deployed using a Mininet emulator using a linear topology in the data layer. All the VMs were implemented on a Windows 10 host running on Intel core i5 CPU 3.8 GHz with 16 GB of RAM. The snapshot of the testbed setup is given below-

- **Controllers-** OpenDaylight version 14 (Silicon) has been used as a primary controller installed on Ubuntu 20.04 LTS VM. The same virtual machine has been used for the secondary controller i.e. ONOS version 2.6.0 (woodpecker).
- **Emulator-** An emulator called Mininet has been used to simulate a total of 25 hosts, 25 switches, and 625 internal paths among them in a linear topology. It is installed on another virtual machine based on a similar type of VM.
- **Attacker Machine-** A device that is used to initiate the switch MAC flooding and link flooding attack by Macof tool and has been installed on Kali Linux.

### 3.9.2 Implementation Result Analysis

In the implementation scenario attack traffic has been generated by the Macof tool installed in Kali Linux which has generated both MAC flood and link flood traffic to overwhelm the switches and links in between. Afterwards, both algorithms were implemented in the similar simulation environment described above to measure two major parameters i.e., **running time** and **CPU overhead** imposed by the algorithms. These two parameters are extremely important because a larger detection time in a compromised switch detection process increases the risk of overall DoS in the network and a larger time in network restoration makes the whole network unusable. The major computing resource is CPU time therefore both the algorithms should be lightweight enough so that they can easily run and do not impose any major overhead on the overall network performance. Both of the algorithms were compared with the existing state of the art and noticed

significant improvement for both the parameters for both the algorithms which are described below.

**Compromised Switch Detection Time:** In the experiment, the PACKET\_IN message has been selected as the initiator of the network update. Upon receiving the PACKET\_IN message the master and secondary controllers execute this request and update their flow table. The switches send the execution reports to both controllers. A switch whose report is different from master or secondary controller has been chosen as a compromised one. We measured the average detection time after two successful packet and link flooding attacks and obtained 600 different values of the same. As per the experimental findings, we found that all switches compromised by the attack have been detected successfully and the average time taken to achieve the highest performance is 4.85 milliseconds. A similar parametric value achieved in [197] was 7.4867 milliseconds, therefore our algorithm has provided faster detection time which has been depicted in figure 3.13.

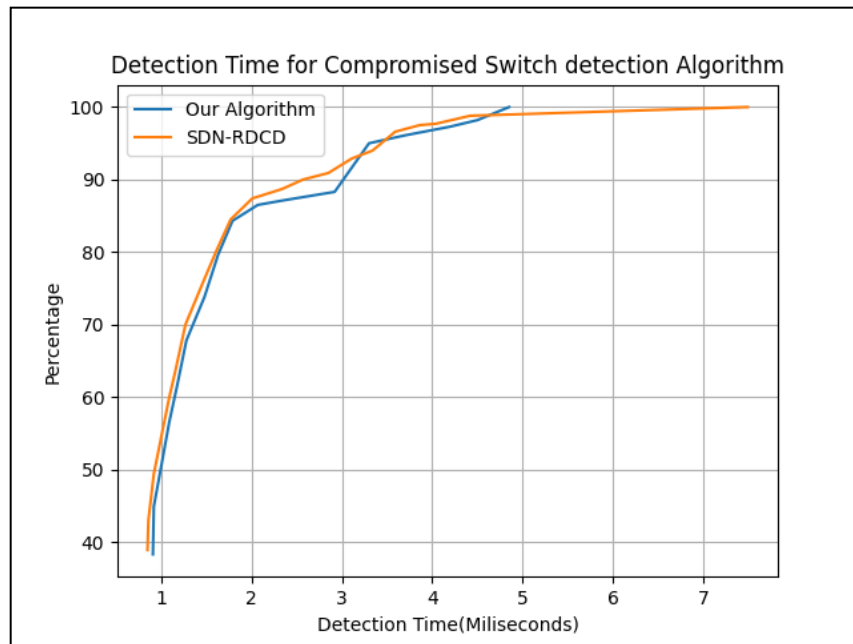


Figure 3.13 Compromised Switch Detection Time

### ***CPU Overhead of Compromised Switch Detection Algorithm:***

In the next experimental study, the impact of extra overhead on CPU usage has been studied with our algorithm and without our algorithm and it is found that 117.85% CPU overhead is achieved without the algorithm and 134.44% with our algorithm. The average overhead is therefore 17.24% approx. which is the resultant cumulative impact from all the switches. The same parameters are compared with the existing state-of-the-art SDN-RDCD[197] which has an overhead of 136.34% approx. when the algorithm is running. Therefore, our algorithm imposes 1.9% less overhead on overall CPU performance which has been depicted in Figure 3.14.

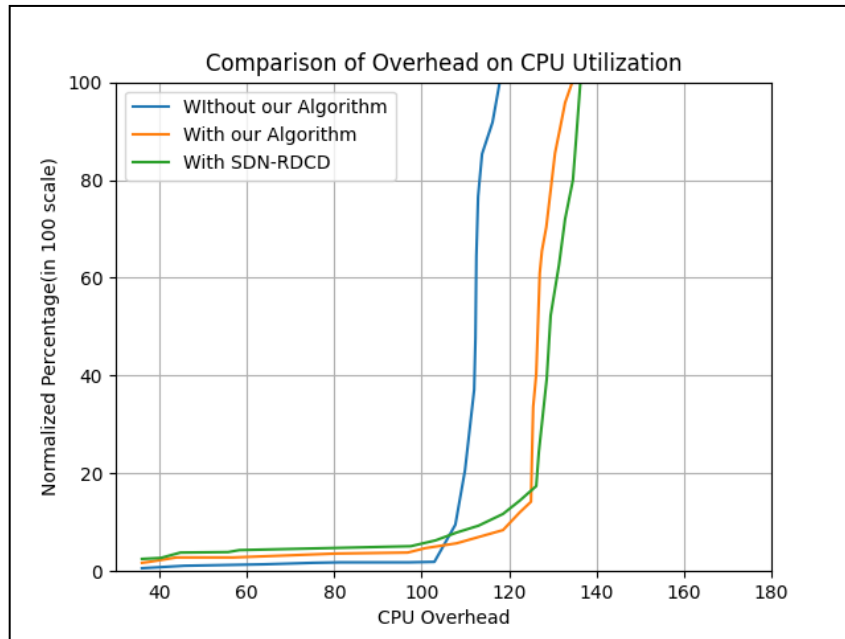


Figure 3.14 CPU Overhead Comparison of Compromised Switch Detection Algorithm

**Network Flow Reconstruction Time:** For the construction of the network flow path, a sample of 25 switches and 25 hosts has been taken into consideration along with 625 links among them. The proposed algorithm completed its execution within 5.56 milliseconds which is an average taken from 500 different execution cycles under a normal operational environment. In a similar simulation environment, the simulation result of the proposed algorithm is compared with Astaneh et al. [198] which gives an execution time of 7.25 milliseconds. Therefore, the proposed algorithm achieves a significant 1.69 millisecond

faster execution time which is very crucial in real-time scenario. Figure 3.15 presents a graphical representation of the aforesaid claim.

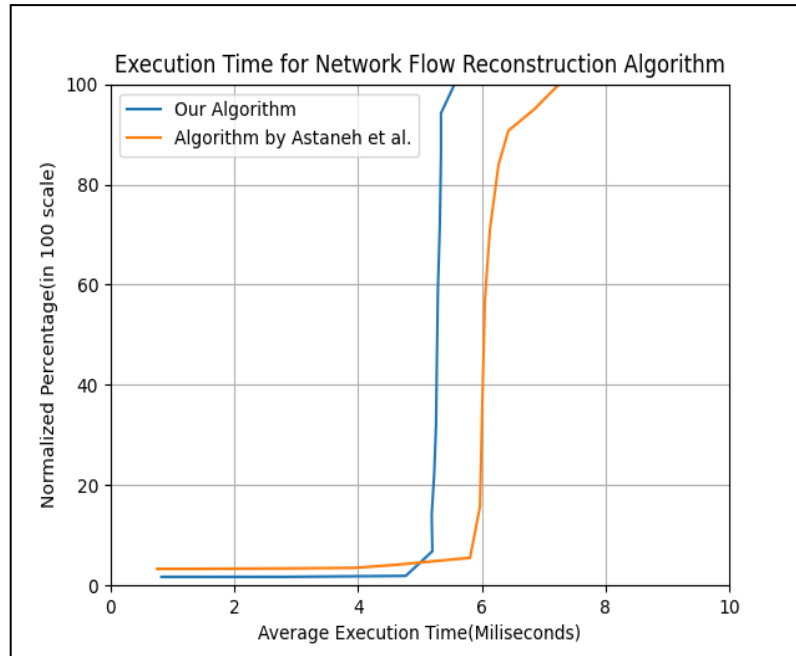


Figure 3.15 Execution Time of Network Flow Reconstruction Algorithm

**CPU Overhead for Network Flow Reconstruction Algorithm:** A similar approach has been followed for testing the CPU overhead utilization of the network flow reconstruction algorithm like the previous one i.e. compromised switch detection algorithm. Therefore in this approach, the extra burden or load on the CPU has been measured. It is found that 135.33% CPU overhead is achieved without the algorithm and 142.47% with the proposed algorithm. Similarly, this algorithm is also compared with Astaneh et al.[198] with the same simulation testbed and it provides 144.33% overhead on the CPU with the algorithm. So to conclude that the proposed algorithm has a very marginal amount of load addition on the CPU performance which is illustrated in Figure 3.16.

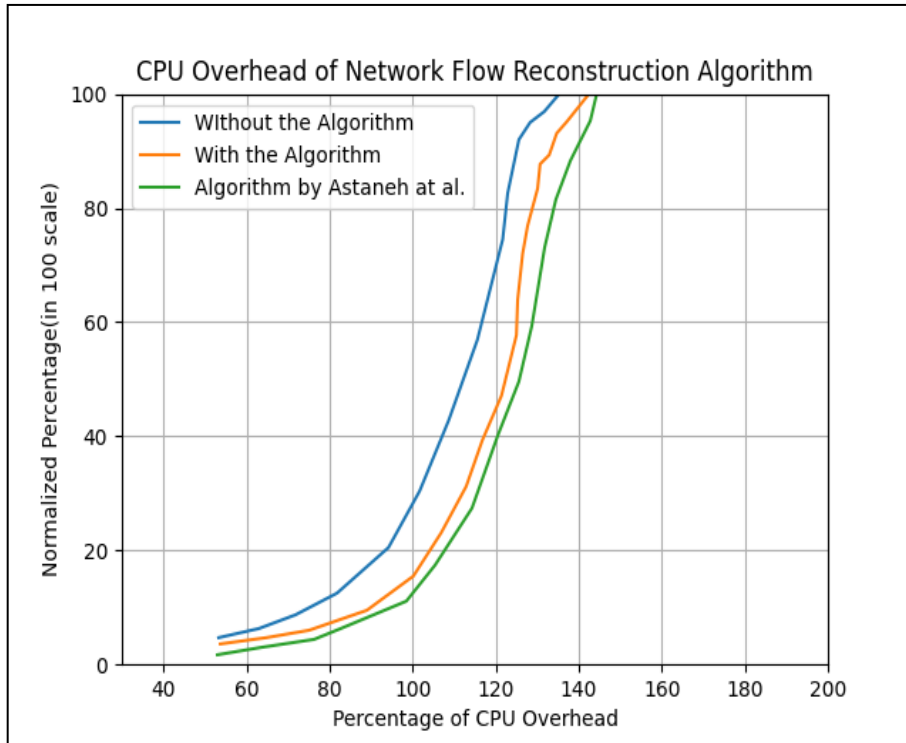


Figure 3.16 CPU Overhead of Network Flow Reconstruction Algorithm

### 3.10 Security Achievements

#### 1. Detection and Isolation of Compromised Switches:

- **Work Done:** The algorithm detects switches that have been attacked and compromised by an attacker and isolates them from the network successfully.
- **Achievement:** This detection and isolation prevent compromised switches from causing further harm to the network, enhancing the overall security and integrity of the SDN infrastructure.

#### 2. Removal of Compromised Flow Table Entries:

- **Work Done:** The subsequent flow table entries associated with the compromised switches are removed.
- **Achievement:** Removing these compromised entries ensures that malicious data flows are eradicated, preventing any further exploitation of the network through these entries.

**3. Restoration of Network Flow:**

- **Work Done:** After the elimination of compromised switches, the entire network flow is restored.
- **Achievement:** Restoring network flow ensures that the network can return to normal operations quickly, minimizing downtime and maintaining service continuity.

**4. Two-Controller Approach:**

- **Work Done:** A two-controller approach is proposed where the secondary controller handles the extra load of the detection algorithms.
- **Achievement:** This approach ensures that the overall network load remains the same as before, maintaining performance while adding security measures.

**5. Holistic Approach for Threat Detection and Mitigation:**

- **Work Done:** The work proposes a two-fold approach—detecting the threat and mitigating the ill effects caused by that threat.
- **Achievement:** This comprehensive approach not only identifies, and isolates compromised switches but also ensures that their impact on the network is neutralized, thereby enhancing the resilience and robustness of the SDN environment.

**How It Enhances SDN Security**

- **Proactive Threat Detection:** By detecting compromised switches early, the network can prevent attacks from spreading and causing more damage.
- **Effective Mitigation:** Isolating compromised switches and removing malicious flow entries ensures that threats are contained and neutralized effectively.
- **Performance Maintenance:** The two-controller approach maintains network performance while implementing security measures, ensuring that security enhancements do not degrade network efficiency.
- **Rapid Recovery:** Restoring network flow quickly after a threat is mitigated ensures minimal disruption to network services, maintaining reliability and trust in the SDN infrastructure.
- **Comprehensive Security Solution:** The holistic approach addresses both detection and mitigation, providing a robust solution to enhance the overall security of the SDN environment.

### 3.11 Chapter Summary

This chapter addresses critical security issues in the SDN data plane and SBI, focusing on controller vulnerabilities and compromised switch detection. The first part of the chapter highlights the necessity for SDN designers to tackle controller flaws. Evidence from the study shows that perfect security for SDN controllers is achievable. A successful attack on the OpenDaylight (ODL) SDN controller demonstrated that the communication between the controller layer (ODL) and the infrastructure layer (Mininet) via the Southbound Interface (SBI) is insecure, allowing attackers to corrupt the network.

The research outlined here in this chapter has two main objectives: demonstrating the vulnerability in the ODL platform due to a Man-in-the-Middle (MITM) attack and preventing such attacks by encrypting communications using Elliptic-Curve Diffie-Hellman (ECDH) and Advanced Encryption Standard (AES-256) algorithms. The proposed encryption method enhances the security of SDN environments by protecting communications from interception and tampering, underscoring the need for robust encryption mechanisms to safeguard SDN infrastructure.

The second part of the chapter introduces a novel approach for detecting compromised SDN switches and reconstructing network flow. This approach was developed, implemented, and evaluated based on speed in running time and overhead on computing resources. The first algorithm demonstrated a 35.22% increase in speed and imposed 1.9% less overhead compared to existing solutions. The second algorithm achieved a 23.31% increase in speed and a reduction of 1.29% in resource overhead.

These results indicate that both algorithms are lightweight and efficient, enabling prompt detection of compromised switches and efficient restoration of network flow. This minimizes disruption and maintains network integrity, ensuring that the overall network performance remains stable despite additional security measures.



In conclusion, the proposed solutions in this chapter significantly enhance the security and efficiency of the SDN data plane and SBI confirming the intention of research objective one of the thesis.

## **Chapter 4**

### **Proactive Detection of SYN Flood Attack on SDN Controller**

A SYN flood attack is a type of Denial of Service (DoS) attack that exploits the TCP handshake process. In a typical TCP handshake, a client sends a SYN (synchronize) packet to a server to initiate a connection. The server responds with a SYN-ACK (synchronize-acknowledge) packet, and the client completes the handshake by sending an ACK (acknowledge) packet. In a SYN flood attack, the attacker sends a large number of SYN packets to the target server but does not respond to the SYN-ACK packets from the server. This leaves the server with numerous half-open connections, consuming its resources and preventing legitimate connections from being established.

In the context of Software-Defined Networking (SDN), detecting and mitigating SYN flood attacks is crucial in maintaining network performance and security. The SDN controller, which has a centralized view of the network, can play a pivotal role in detecting such attacks. Detection methods often involve monitoring network traffic for anomalies, such as an unusually high number of SYN packets from a single source or multiple sources in a short period.

One effective detection technique is to implement a threshold-based mechanism within the SDN controller. The controller continuously monitors the rate of incoming SYN packets. If the rate exceeds a predefined threshold, it triggers an alert indicating a potential SYN flood attack. Additionally, flow rules can be dynamically updated to mitigate the attack. For instance, the controller can instruct switches to drop packets from suspicious sources or redirect traffic to a firewall for further inspection.

Another advanced method involves machine learning algorithms that analyze traffic patterns and identify deviations from normal behavior. By training models on normal traffic patterns, the SDN controller can detect SYN flood attacks more accurately and adaptively. This approach enhances the controller's ability to detect sophisticated and evolving attack strategies.

Therefore, SYN flood attacks pose a significant threat to network availability. Leveraging the centralized intelligence of SDN controllers, combined with threshold-based mechanisms and machine learning techniques, provides a robust framework for detecting and mitigating these attacks, ensuring the resilience and reliability of the network.

#### **4.1 Overview of The Problem**

The data plane switches handle the forwarding of packets in accordance with the flow rules set by the SDN controller. A flow rule includes both an action (forwarding or deleting) and header match fields (receiver IP address, recipient MAC address, and TCP port; and sender IP address, sender MAC address, and TCP port). Each packet that gets to the OpenFlow switch must be checked against the data in its flow table. If an OpenFlow switch is unable to find a matching rule for an incoming packet, it will forward the packet to the controller, asking for further action. Because of this, the OF switch needs to ask the controller for new rules for each packet that comes in. SDN offers a lot of advantages, but it also includes several security vulnerabilities. Because the SDN controller is centralized, it may be the target of a Distributed Denial of Service (DDoS) assault, such as a SYN flood attack [199], which might bring down the whole network.

The most notable result of a SYN flood assault is DDoS. In this kind of assault, the attackers target a server and flood it with SYN packets, decreasing the availability of the service. The sender and the recipient engage in a three-way handshaking procedure while establishing a traditional TCP connection. When a TCP client sends a SYN packet, a TCP server listens and responds with a SYN-ACK packet. Furthermore, server resources are frozen until the client responds with an ACK. After a set amount of time passes without receiving an ACK message, the server will make the content accessible. An adversary launches many TCP connections that are only half open without sending a SYN-ACK message in a SYN flooding attack. This behavior exhausts the resources of the targeted server, making the service inaccessible to legitimate traffic [200]. Attackers produce a large number of new packets when they utilize SYN flooding since OF switches must connect with the controller in order to seek the required actions. Attackers that send a lot

of SYN packets slow down the controller's processing power and performance. The TCP port can sustain a good number of half-open TCP connections. As a result, once the number of SYN packets reaches the designated limit, any further incoming SYN requests are rejected. The partly established connections are closed once the 75-second SYN Received timeout [201] has elapsed. Figure 4.1 shows both a typical TCP handshaking session and a SYN flood situation.

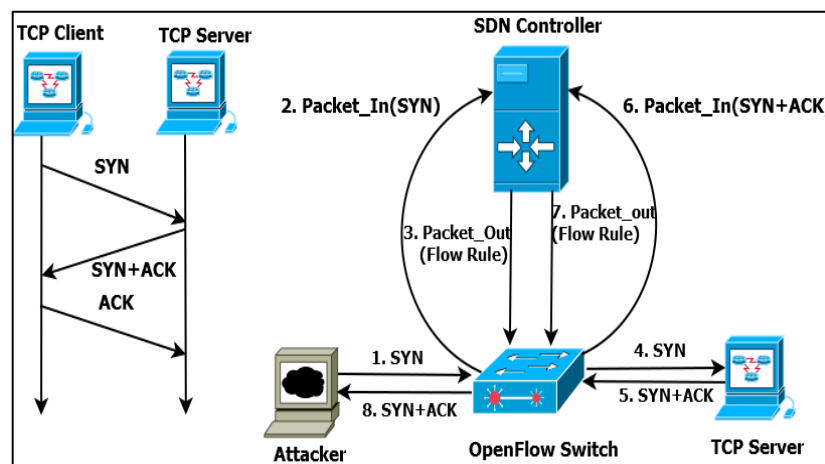


Figure 4.1 Normal TCP Handshaking vs SYN Flood

The following are three popular and effective ways to create a SYN flood attack:

1. By faking the IP addresses of several hosts, an attacker affects them all and launches a SYN flood assault.
2. A malicious party forges both the MAC and IP addresses of many hosts to cause a SYN flood attack.
3. A DDoS situation might be brought on by a flash mob with many incoming SYN requests.

The huge volume of SYN requests is too much for the network with limited resources to handle, even when the clients try to send ACK in response to the legitimate SYN requests generated by the flash mob. The SYN flood attack not only takes down the compromised server but also overloads the controller and switch plane resources. When the switch lacks rules, a flood of SYN packets with distinct headers is generated, which leads to a packet in flood toward the SDN controller. Elevated SYN flood rates hinder the processing capability and throughput of the controller. Due to the switch's little memory, flow rules

required for the SYN flood assault cannot be stored, which commonly leads to flow rule overflow and resource exhaustion on the switch plane. The SYN flood in SDN disrupts the network because the controller is resistive to the incoming packet\_in.

## 4.2 Proposed Solution Strategy

ProDetect is a state-of-the-art technique designed to detect and eradicate SYN flooding attacks. It is intended to address SYN flooding and its associated malicious effects on SDN networks. ProDetect is deployed as a security enhancement module at the OpenDaylight controller, monitoring all incoming SYN requests. Every incoming TCP SYN request is inspected by ProDetect, which then filters out any illegitimate requests. Only the genuine requests are then sent to the controller for OpenFlow switches to construct further rules. Every malicious request is eliminated, and MACs that relate to it are stopped in advance. As a result, it successfully stops control plane saturation attacks and TCAM memory fatigue attacks. In a typical TCP 3-way handshaking process, a TCP client makes a TCP SYN request to a server, and the server responds with an SYN ACK packet. The server now allots the required resources to the recently established flow and waits in listening mode, leaving the TCP port partially open, for a predetermined period of time. The client sends back an ACK packet to finish the three-way handshaking engagement. When there is SYN flooding, the attacker sends a continuous stream of SYN requests with the purpose of wasting the intended server's resources rather than receiving a SYN-ACK packet in response. As a result, the server is overloaded and stops responding to legitimate TCP requests. When an OF switch in an SDN architecture receives a new connection request from a host, it needs the data forwarding policies from the SDN controller, which necessitates a TCP handshaking among the server and its clients. A typical SYN flooding scenario involves an attacker initiating an attack with the malicious goal of overloading and draining SBI's bandwidth as well as the controller's data processing capabilities. There are three major methods by a standard SYN Flood assault might be conducted.

- By impersonating many hosts and corrupting them.

- By impersonating many hosts and gaining access to their MAC addresses.
- By creating a flurry of SYN requests along with valid ones.

#### 4.3 Issues Addressed by the Proposed Solution

The major issues which are addressed by the proposed solution are described below-

- A dynamic threshold-based SYN flood detection algorithm, ProDetect, is proposed, which is capable of filtering out malicious traffic beforehand, thereby eliminating the chance of data or control plane saturation.
- By adjusting its threshold, the system may effectively manage DDoS scenarios without interfering with regular network operations.
- The suggested approach functions as a lightweight controller extension while resolving all of the fundamental shortcomings of the earlier studies.
- The suggested method does not take fake MACs into account since it can identify MAC spoofing.
- By employing traffic filtering, the suggested method removes the possibility of congestion in the SDN southbound interface.

#### 4.4 Design Details of Proposed Solution: ProDetect

Major steps of the proposed approach ProDetect are discussed below-

- The following eight parameters are fetched from the TCP SYN header packet and saved in the Incoming\_Requests table.
  - $MAC_{SRC}$ ,
  - $MAC_{DEST}$ ,
  - $TCP_{SRC}$ ,
  - $TCP_{SRC\_PORT}$ ,
  - $TCP_{DEST}$ ,
  - $TCP_{DEST\_PORT}$ ,

- $SYN\_COUNT$ ,
- $MAC_{ACK}$
- Each item in this table is uniquely recognized by  $MAC_{SRC}$ , and  $SYN\_COUNT$  is increased by 1 each time a new connection request is received from the same  $MAC_{SRC}$ .
- If  $SYN\_COUNT$  and  $MAC_{SRC}$  both go over the predetermined threshold  $P$  and there isn't a matching  $MAC_{ACK}$  item against  $MAC_{SRC}$ ,  $MAC_{SRC}$  will be blocked and removed from the Incoming\_Requests table.
- The entries whose  $SYN\_COUNT$  is less than  $P$  and for which there is a valid  $MAC_{ACK}$  entry are now stored in the Incoming\_Requests table.
- Add the flow rules and designate them as valid or benign connection requests in the modified Incoming\_Requests table for the forwarding tables of OpenFlow switches.

Figure 4.2 illustrates the main computational approach used by ProDetect. It illustrates how, after storing the previously given parameters in the Incoming\_Requests table, ProDetect examines each incoming TCP source. Every time it finds the same source,  $SYN\_COUNT$  is increased by 1. As a logical separator, the dynamic threshold  $P$  acts as follows: if  $SYN\_COUNT$  is more than  $P$  and no valid  $MAC_{ACK}$  has been received for this  $MAC_{SRC}$ , then this MAC is blocked. As a result, the Incoming\_Requests table contains only benign MAC information, for which entries may be added to flow tables. This approach inhibits SYN flood before it happens, defending against DDoS assaults.

In a DDoS scenario, the value of the threshold  $P$  can be continuously adjusted to filter out traffic that might be damaging while allowing legitimate traffic to get through. The sudden increase in incoming traffic may be examined using network flow analyzers like Wireshark [202] to look for patterns or similar types of incoming packets coming from just one source, which indicates packet flood and can result in DDoS. As a result, as we shall discuss in the next section, the baseline value of threshold  $P$  is often kept at 55.

The module continuously monitors incoming traffic using Wireshark and matches it to the preset threshold. In order to mitigate DDoS, the value of  $P$  is lowered to filter out

anomalous request type patterns, surges, and notable increases in traffic volume. The identification of a single request from one source may also be done using this technique for blocking valid traffic.

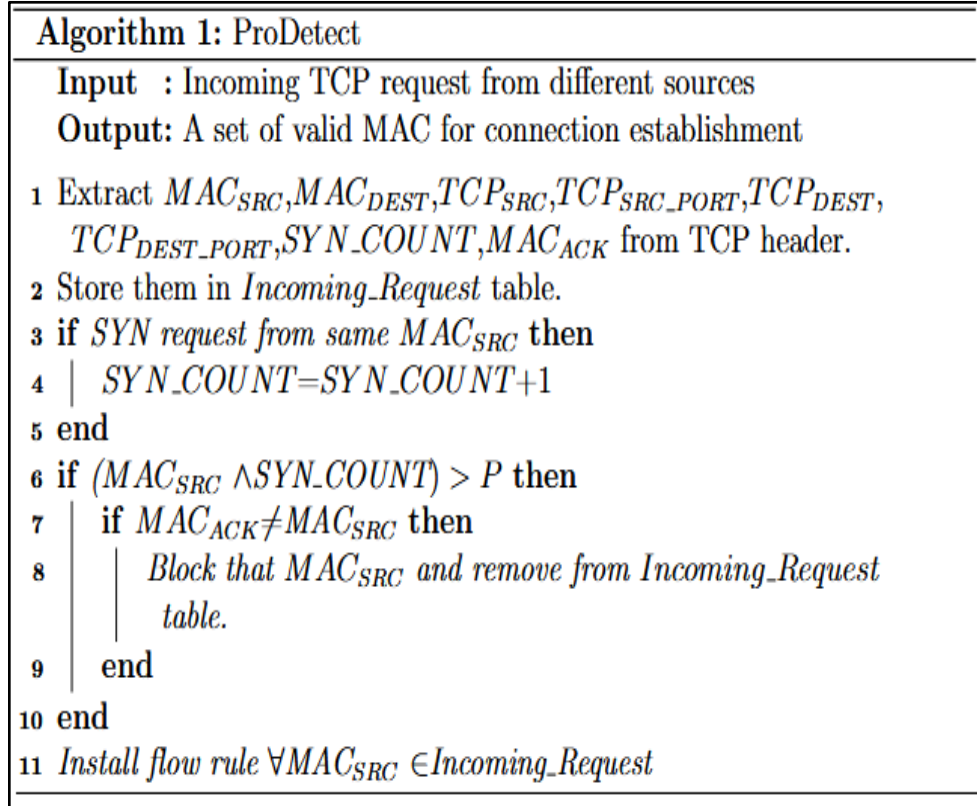


Figure 4.2 The ProDetect Algorithm

#### 4.4.1 Threshold Calculation Process

Because early detection of the SYN flood is crucial for the controller's continued operation, setting the optimal period value is essential. Thus, a method to select the optimal value for the given time frame is provided. This computation is conducted well in advance, and SYN requests are permitted or denied based on the determined threshold since SYN floods are proactively identified and removed by the proposed solution. A threshold's ideal value should be considered since too loose a threshold would raise the likelihood of a SYN flood and deteriorate controller performance, while too tight a threshold will limit harmless SYN requests.



Therefore, six parameters are considered for the calculation of the threshold value P. Lower values are expected for the first three parameters, and higher values are appreciated for the remaining three. The parameters are-

1. **SYN Flood Probability (P\_SYN):** This lower value metric, which is represented by P\_SYN, indicates that minimal values are preferable. The number of those messages that the controller does not handle indicates the total number of Packet\_In messages that it is unable to process. P\_SYN is a number that ranges from 0 to 100 and is determined using the formula below.

$$P\_SYN = \frac{Packet_{in} not\ handled}{Packet_{in} received} * 100 \text{ --- (1)}$$

2. **SYN and SYN-ACK Delay (D\_SYN):** D\_SYN, a lower value measure, is also used to represent this. When there is a SYN flood, the server cannot respond to the SYN flood with SYN\_ACK; so, the SYN-ACK Packet\_In number needs to be lower than the SYN Packet\_In number. The following formula is used to determine D\_SYN, which has a range of 0 to 100.

$$D\_SYN = \frac{SYN\ Packet\_In}{Total\ Packet\_In} - \frac{SYN - ACK\ Packet\_In}{Total\ Packet\_in} * 100 \text{ --- (2)}$$

3. **Hellinger Distance of SYN (HD):** Once more, HD indicates a lower value measure. HD is the degree to which the Packet\_In flow distribution seen in both the current and previous eras is similar to one another. The Hellinger distance lies in the interval [0, 1]. Up to the [0-100] range, we scale it. HD is the outcome of equation 3. The current Packet\_In in this instance is P\_j, while the prior Packet\_In is Q\_j.

$$HD = \left(\frac{1}{\sqrt{2}}\right) * \left(\sqrt{\sum_{j=1}^4 (\sqrt{P\_j} - \sqrt{Q\_j})^2}\right) * \frac{SYN - ACK\ Packet\_In}{Total\ Packet\_In} * 100 \text{ --- (3)}$$

4. **Throughput (TP):** It is a metric with a higher value, indicated by TP. The SYN flood exceeded the controller's processing capacity, causing the Packet\_In queue to overflow. Equation 4 determines TP, which is in the [0-100] range.

$$TP = \frac{\text{Number of Packet\_In Processed}}{\text{Total Packet\_In}} * 100 \text{ --- (4)}$$

5. **CPU Load (CL):** This metric expects a higher value and is represented by CL. The CPU utilization increases because of SYN flood managing the Packet\_In flood. The range of CPU load is [0-100]. As CPU demand drops, so does the controller's capacity to handle incoming Packet\_In.

$$CL = \text{Number of Packet\_In} * \text{Time to process each Packet\_In} \text{ --- (5)}$$

6. **Memory Load (ML):** This metric is expected to have a higher value and is represented by ML. It is the fraction of the controller's processing memory that is available. SYN flood utilizes more memory than Packet-In flood because of its high memory use. The range of ML is [0-100].

$$ML = \frac{\text{Memory Free}}{\text{Total Physical Memory}} * 100 \text{ --- (6)}$$

The six indications need to be added together into a single number in order to calculate the controller's effectiveness. To do this, Merrill's Figure of Merit (FM) [203] is employed. FM is a limited measure of system performance. To analyze FM, two equally sized sets of observations (lower and higher values) with values occurring within the same range are needed. The ranges of measurements with lower and higher values align with the boundaries of the FM. There are the same number of signs in the lower-value and higher-

value categories, and all of the measures' values lie between 0 and 100. Consequently, FM's range is [0-100]. FM has been used to establish the threshold P by the following equation.

$$FM = 1/6\{(100 - TP)(100 - P_{SYN}) + (TP + CL)(100 - D_{SYN}) + (CL + ML)(100 - HD)\} \text{ --- (7)}$$

The cutoff value is considered as 55 for the following reasons-

- Since the controller oversees network operations, it should not be overloaded with unnecessary processing.
- Since the SDN controller is the primary processing unit of the SDN, SYN flood should not overwhelm it.
- Setting a threshold higher than 55 will cause the server to get overloaded with SYN queries.
- If a threshold below 55 is selected, it will limit valid SYN queries.

Certain matrices, such as TP (Throughput), CL (CPU Load), and ML (Memory Load), frequently show anomalies during a Distributed Denial of Service (DDoS) assault. In particular, TP may drop to almost nothing, which would signify a dramatic reduction in the precision of classifying valid traffic. On the other hand, CL and ML are probably going to rise, getting close to 100%, which indicates that hostile traffic categorization is more confidently believed to be true and that misleading data is more common, respectively.

A strategy that makes use of Jain's fairness index [204] is used to resolve these abnormalities and restore the integrity of the system. Rebalancing the parameters and restoring equilibrium is done with the help of this index, which is well-known for its use in evaluating the fairness of resource distribution. The various results are harmonized and scaled up to a standardized range, usually ranging from 0 to 100, by using Jain's fairness index to TP, CL, and ML.

By ensuring that every indicator contributes proportionately to the total scenario assessment, this method helps to improve the accuracy of reaction mechanisms and decision-making. The system can more effectively distinguish between malicious and legal traffic by re-establishing balance and fairness across the metrics. This will lessen the impact of the DDoS assault and allow network operations to resume as usual.

#### **4.5 Detection of MAC-Spoofing Attack**

To counter the MAC spoofing attempt, a MAC spoofing attack detector is utilized, constructed in compliance with the techniques outlined in [205]. Each communication stream's numerical sequence is logged by the proposed technique to detect MAC spoofing attempts. If it is a resend frame, the current frame's sequence number may equal or differ from the preceding frame's sequence number. If the sequence number of a frame falls within the replicated range, it is considered resent. If the current frame is a resent frame, the receiver node can verify that it is a resent frame by using the copy it already has. When the current frame is different from the stored copy, it must be a counterfeit frame.

The procedure calculates the distance ( $G$ ) between the frame's sequential number and the preceding frame that originated from the same place when a frame arrives. In contrast,  $G = 1$  or  $G = 2$  implies that this frame is the right one;  $G = 0$  indicates that this frame is a repeated frame. Nonetheless, the sequence number is deemed abnormal if there is a discrepancy between the current and previous frames of 3 to 4096 frames. The main advantages of this approach are that it doesn't require any adjustments at the end hosts and only utilizes the sequence number element found in the IEEE 802.11 packet's link-layer preamble. The ProDetect logical flow diagram is displayed in Figure 4.3.

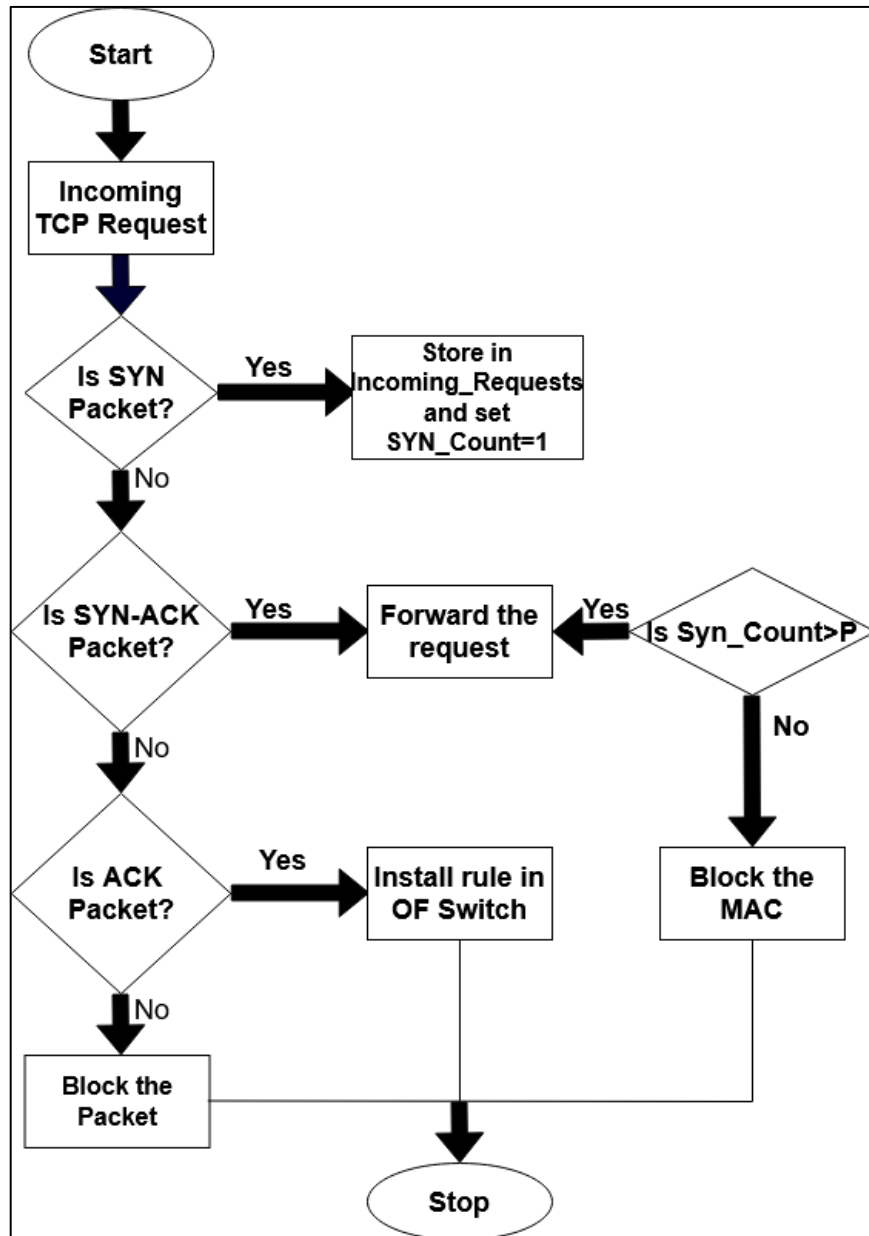


Figure 4.3 Logic Flow Diagram of ProDetect

#### 4.6 Implementation Simulation Setup and Result Analysis

This section describes in detail how an experimental SDN scenario was put into practice and how well ProDetect performed in terms of identifying SYN flooding assaults. ProDetect is deployed in the OpenDayLight [105] controller as a security plugin. Mininet [206] is used to emulate the SDN scenario.

For every result shown below, ProDetect's efficiency is compared to that of SYN-Guard [115]. Because lower values raise false positive rates and larger values increase resource usage, the range of P is arbitrarily selected between 8 and 80. As previously stated, threshold P's value is maintained at 55 throughout regular network operations. In order to lessen its influence on the general operation of the network, it can also be updated dynamically during the assault. The threshold's value may also be manually adjusted to observe how the network performs overall. If P is set to 0, all incoming TCP connection requests will be blocked, and if P is set to 100, all incoming TCP requests will be allowed. These are two extreme terminal scenarios. Thus, in order to produce various parametric test cases, the range of P is preserved between 8 and 80, ranging from a very low range to a much greater one, in order to assess the overall network performance.

#### 4.6.1 Configuration of Network

The target SDN testbed consists of five network entities: an OpenFlow switch, an OpenDaylight controller with ProDetect integrated, a forged host that engages in SYN flooding attack, a web server, and a reliable host that delivers benign HTTP requests every two seconds. Using separate virtual machines running Ubuntu 20.04 LTS and equipped with an Intel Core i5 CPU and 8 GB of RAM, the web server, SDN emulator, and controller are hosted on Windows 11. The SDN emulation is a mininet environment, and the attacker machine is a different virtual machine (VM) running Kali Linux 2023.2.

The 120-second simulation is conducted. A reputable host uses the tool Curl [207] to generate HTTP requests. The Hping [208] tool is used by the attacker to carry out a SYN flooding attack using fictitious IP addresses. Using the Hping program, the attacker may generate fraudulent SYN packets at a rate of 4000–40000 packets per second.

#### 4.6.2 Comparative Analysis of Results

The proposed cure and the attack behavior are evaluated at different attack rates using the following seven relevant metrics.

1. **Attack Detection Time:** This is the amount of time that ProDetect needs to detect a SYN Flood assault after it has started.
2. **Delivered Packet Ratio:** The total of all packets sent by the sender divided by all packets received by the recipient is this ratio.
3. **Bandwidth Use:** Bandwidth use is calculated by summing up the bandwidth utilized by all of the active data flows on the network.
4. **OpenFlow switch entry:** This parameter shows how an OpenFlow switch's forwarding policy table would be affected by a memory attack.
5. **CPU Utilisation:** This indication shows the average CPU usage compared to the total amount of resources available to the controller.
6. **Memory Utilisation;** The average amount of RAM used relative to the controller's entire resource pool is shown by this indication.
7. **Response Time by Server:** This metric indicates how long it takes for a web page to be returned from the server following a request.

Figure 4.4 shows that when attack rates rise in both SYN Guard and ProDetect, the attack detection time repeatedly reduces. Because no flow rules are established and all incoming SYN requests are stopped from that particular MAC when threshold condition P is met, ProDetect is superior to SYN Guard in situations where there is an elevated attack rate. It can detect the SYN flood extremely rapidly. This threshold P is fulfilled sooner in a high attack scenario, which results in a notable decrease in attack detection time. In ProDetect, flow rules are implemented instantly and prevented as soon as the threshold is reached, but in SYN Guard, the controller adds flow rules only after the threshold has been exhausted. It so offers more time efficiency than SYN Guard.

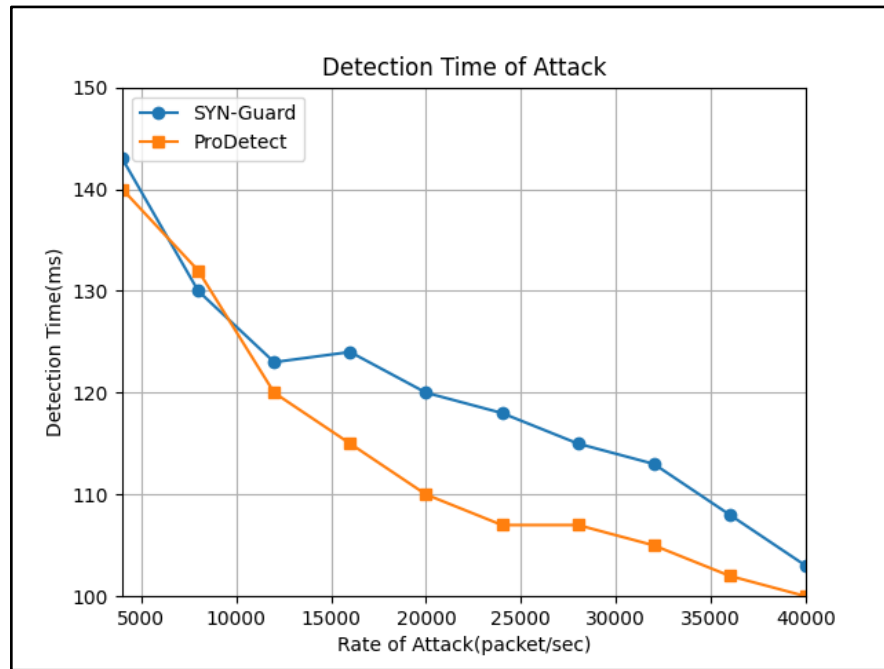


Figure 4.4 Attack Detection Time

Figure 4.5 illustrates that in an attack scenario, both SYN Guard and ProDetect yield nearly identical results. However, ProDetect demonstrates slightly superior performance because, once the P threshold is reached, forged requests are effectively blocked. This capability enables the controller to deliver packets more efficiently. Furthermore, the simulation results indicate that in high attack scenarios, ProDetect excels, facilitating the successful delivery of nearly 97% of packets. This enhanced performance is critical for maintaining network integrity and ensuring that legitimate traffic is prioritized even under heavy attack conditions. The ability of ProDetect to mitigate forged requests contributes significantly to its overall effectiveness in managing network resources during cyber threats.



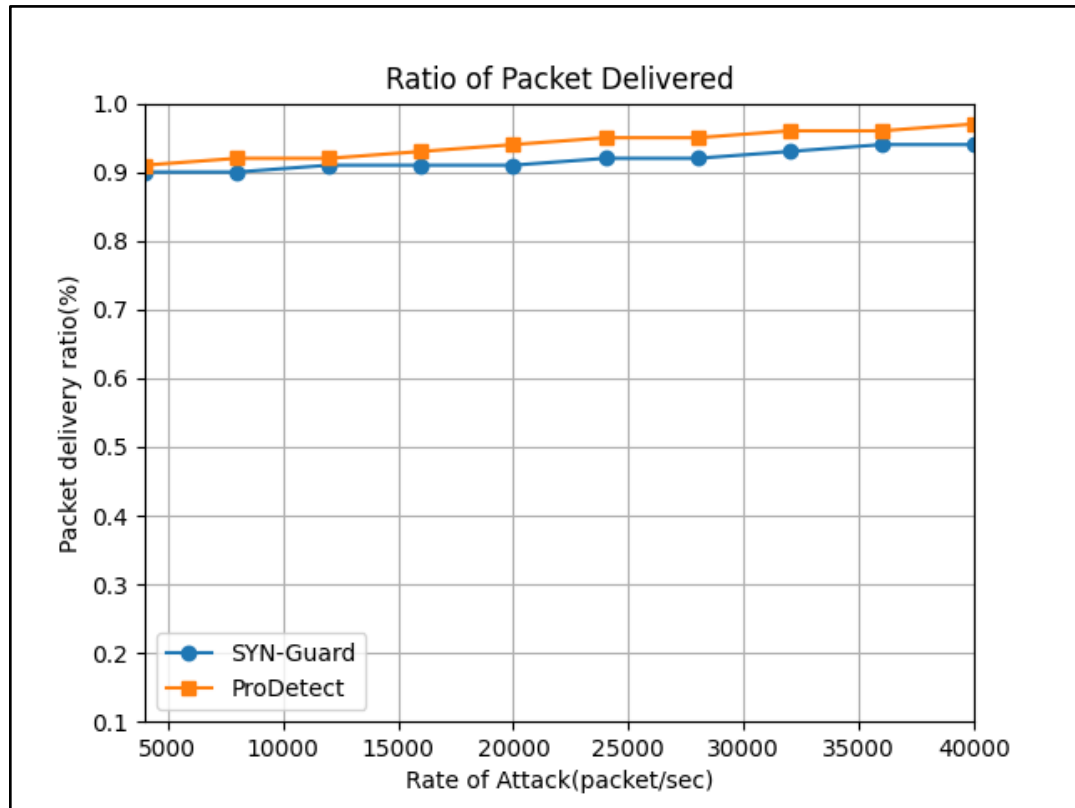


Figure 4.5 Packet Delivery Ratio

No false requests may be submitted to the controller after the attacker has been recognized, as all requests of this kind are destroyed at the OpenFlow switch. ProDetect thus consumes less bandwidth than SYN-Guard (as seen in Figure 4.6). Figure 4.6 illustrates that the use of bandwidth in ProDetect grows linearly with the rate of attack, but in SYN-Guard, the use of bandwidth increases somewhat with the rate of assault. ProDetect thereby makes more efficient use of the available bandwidth and guards against bandwidth exhaustion, which may otherwise happen to the controller. By preventing unnecessary traffic from reaching the controller, ProDetect not only conserves bandwidth but also ensures that legitimate traffic is prioritized and processed without interruption. This capacity to maintain bandwidth efficiency under attack scenarios is crucial for sustaining overall network performance and security.

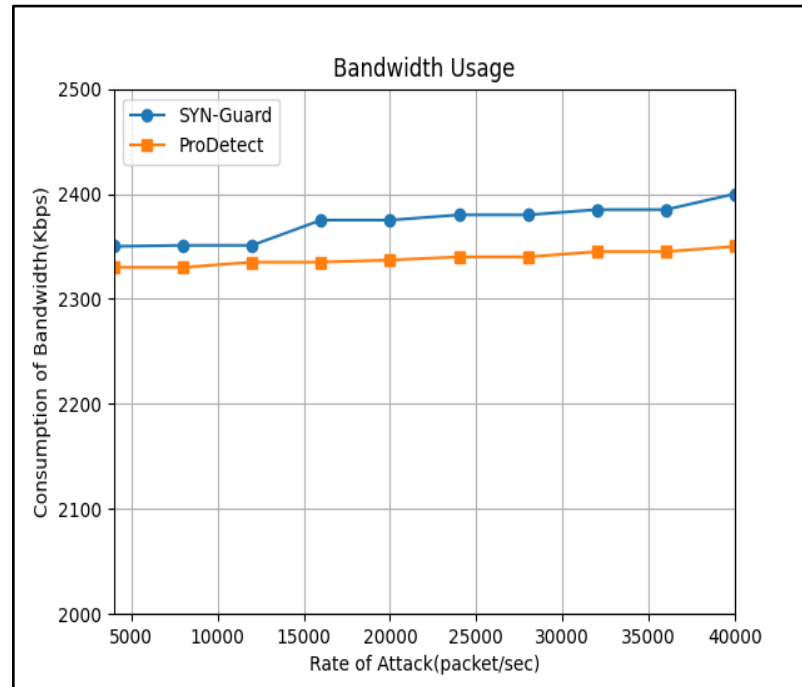


Figure 4.6 Bandwidth Usage

Figure 4.7 illustrates the effect of a SYN flooding attack and the behavior of ProDetect on the average number of deployed entries in the OpenFlow switch forwarding rule table. The graph clearly shows that ProDetect has a more significant impact in this area compared to SYN Guard. This disparity arises because SYN Guard prevents the attacker as soon as it identifies a malicious activity, thereby stopping the attacker's fictitious requests from reaching the controller early on. In contrast, ProDetect operates differently, allowing requests to pass through until the predefined threshold  $P$  is reached. Once this threshold is met, ProDetect enforces its filtering mechanism. As a result, ProDetect implements a more stringent strategy to prevent controller saturation, which is reflected in its higher impact on the forwarding rule table entries.

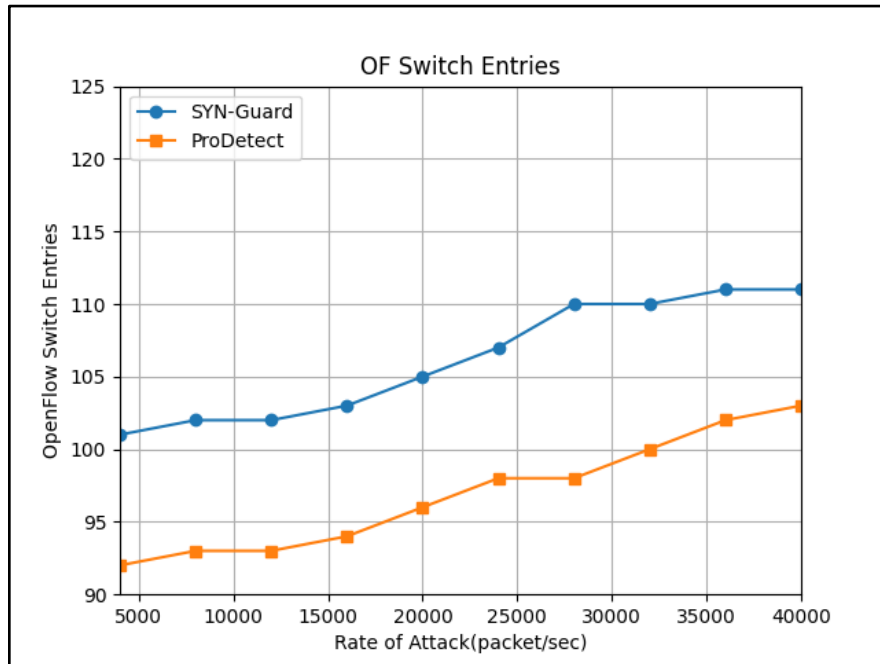


Figure 4.7 OpenFlow Switch Entries

Figure 4.8 illustrates the CPU usage of both comparison methods in an attack scenario. ProDetect demonstrates its effectiveness in preventing control plane overloading by optimizing the utilization of the controller's CPU. This is evident in the figure, where ProDetect maintains a balanced CPU load while ensuring that resources are not overwhelmed. A key factor contributing to this effectiveness is the capability of OpenFlow switches to reject malicious requests. When an attacker is identified, their ability to send further fake requests to the controller is effectively nullified. Moreover, the figure reveals an interesting distinction between ProDetect and SYN-Guard: ProDetect's CPU usage notably increases after reaching a certain threshold. This increase is significant because it indicates that ProDetect actively monitors and halts all SYN requests that exceed this threshold, resulting in a heightened CPU load as the controller processes these requests. This proactive approach not only enhances the resilience of the control plane but also showcases ProDetect's ability to manage resources efficiently, even under adverse conditions. The dynamic handling of CPU resources underscores ProDetect's robust architecture in mitigating potential overloads while maintaining effective network security.

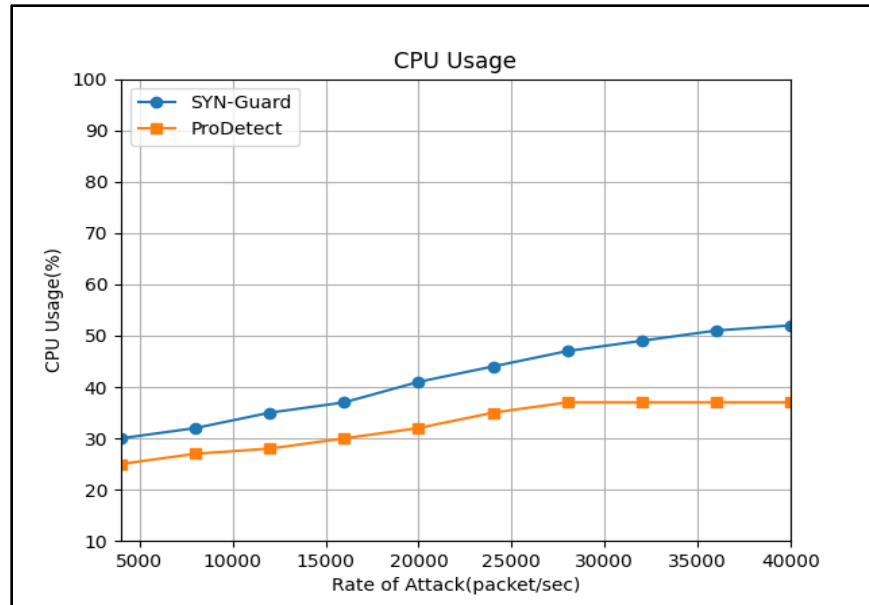


Figure 4.8 CPU Usage

The memory used for the two methods under comparison is shown in Figure 4.9, where it is evident that ProDetect uses less memory than SYN Guard. Even if there isn't much of a difference, in real-world situations, it might have disastrous effects.

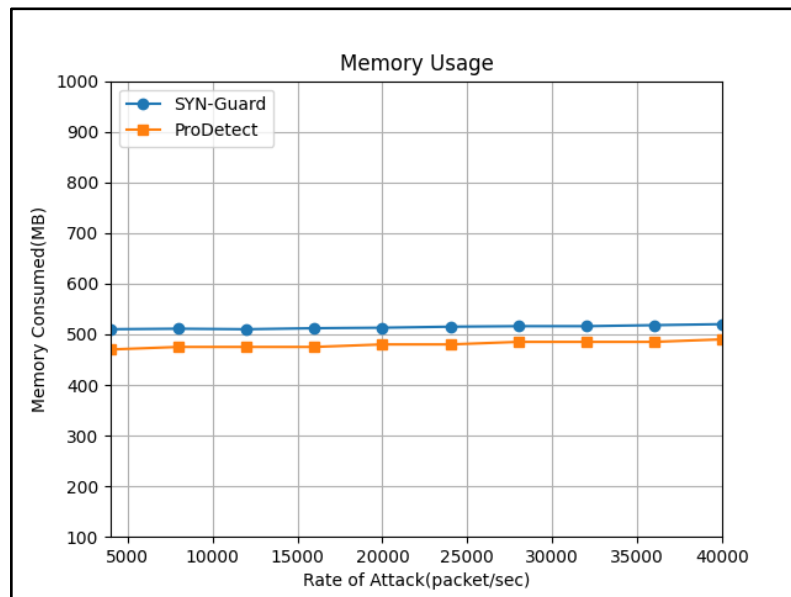


Figure 4.9 Memory Usage

As attack rates increase, Figure 4.10 shows server response times for safe sites. ProDetect effectively prevents attacker nodes from continuing their attack for prolonged periods of time and speeds up the response time for real hosts, as seen in Figure 4.10. All of the intruder's packets will be halted once they have been produced, ending the saturation attack on the targeted web server and enabling it to promptly respond to HTTP requests from genuine users. ProDetect outperforms in terms of server response time, as Figure 4.10 illustrates, because the controller sends the SYN request to the target host promptly. When a SYN packet is received in SYN Guard, the controller adds a forwarding rule across the route. It causes a little lag as compared to ProDetect, which causes a minor increase in reaction time.

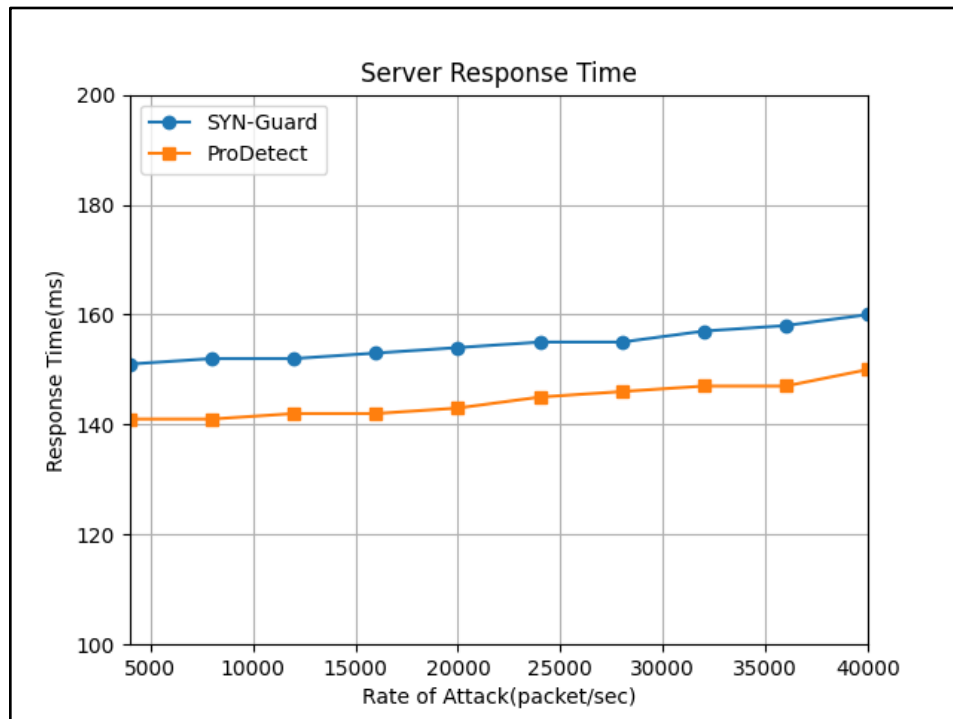


Figure 4.10 Server Response Time

#### 4.7 Security Achievements

The study proposes a proactive lightweight SYN flooding attack detection method that limits the assault. This is accomplished using a dynamic threshold technique, which may also be manually regulated in adversarial scenarios. As a result, the solution offers a proactive countermeasure to the SYN flood attack from a security perspective, and its deployment does not overload the controller. The main security objectives are listed below, along with their accomplishments:

1. **Early Identification:** The suggested method is able to recognize unusual traffic patterns that point to the beginning of a SYN flooding assault. Early detection of the assault allows for the quick implementation of mitigation strategies to lessen the server's damage.
2. **Resource Preservation:** ProDetect aids in the preservation of server resources by detecting and thwarting SYN flooding assaults. rather than permitting harmful SYN packets to use up system resources. By adjusting the threshold, the algorithm can refuse or restrict the processing of these packets, protecting resources for valid traffic.
3. **Preserving Uptime:** By keeping servers from being inundated with unauthorized connection requests, ProDetect helps to preserve the availability of services. The method makes sure that genuine users may continuously use the server's resources by identifying and thwarting SYN flooding assaults.
4. **Decreased Downtime:** Quick identification and mitigation of a SYN flooding assault might result in less downtime for the services that are impacted. Organizations may lessen the impact on their operations and shorten the duration of service outages by promptly recognizing the attack and putting in place the necessary remedies.

5. **Preventing Service Deterioration:** By proactively detecting and reducing harmful traffic while preserving the level of service for authorized users, it helps to avoid a decrease in service.

#### 4.8 Chapter Summary

In this study, ProDetect, a defensive mechanism within SDN networks specifically designed to counteract SYN flooding attacks, is introduced. ProDetect is implemented on the OpenDaylight controller, operationalizing the proposed solution. Its core function involves the continuous monitoring of inbound TCP connection requests, which are subjected to validation procedures based on dynamically adjusted thresholds. Legitimate requests that pass validation criteria prompt the controller to establish forwarding policies, effectively facilitating their passage through the network.

Experimental findings affirm the efficacy of ProDetect in detecting and neutralizing malicious TCP requests. Additionally, the solution offers a defense mechanism against MAC cloning or spoofing attacks, enhancing overall network security. By leveraging dynamic thresholds and early detection mechanisms, the system effectively identifies and counters potential threats, conserving resources and maintaining service availability. Rapid response capabilities minimize downtime and prevent service degradation, ensuring uninterrupted access for legitimate users. The proactive approach presented for SYN flooding attack detection and mitigation offers a robust solution to enhance network security.

To bolster network resilience during attack scenarios, ProDetect incorporates a manual threshold control mechanism, safeguarding both the controller and overall network operations. Evaluation results validate the practicality of ProDetect as a robust defense module against SYN flooding attacks, surpassing existing state-of-the-art solutions.

## **Chapter 5**

### **User App Trust Establishment in SDN Application Plane**

Application trust establishment is a critical aspect of Software-Defined Networking (SDN) security, essential for ensuring the integrity and reliability of the network. In SDN, the control plane is separated from the data plane, allowing network behavior to be programmatically managed through applications running on the controller. This separation introduces new security challenges, particularly concerning the trustworthiness of these applications. Establishing trust involves verifying that the applications interacting with the controller are authentic, have not been tampered with, and are authorized to perform their intended functions. This can be achieved through several mechanisms, such as digital signatures, secure coding practices, and rigorous authentication protocols. Digital signatures ensure that the application code has not been altered and originates from a trusted source. Secure coding practices mitigate vulnerabilities within the applications themselves, reducing the risk of exploits. Authentication protocols, such as mutual TLS, help verify the identities of both the applications and the controller, establishing a trusted communication channel. Additionally, runtime monitoring and anomaly detection can be employed to continuously assess the behavior of applications, identifying and mitigating potential threats in real time. By implementing these measures, SDN environments can maintain a high level of security, preventing malicious or compromised applications from disrupting network operations. Overall, application trust establishment is a fundamental component of SDN security, protecting the network from internal and external threats while enabling dynamic and flexible network management.

#### **5.1 Background of the Problem**

Network management is being transformed by Software Defined Networking (SDN), a new concept that makes networks configurable and separates the control plane from the data plane [1]. Adaptability automation, coordination, and cost savings in terms of both capital and operating expenses are provided by this separation. The primary issue with SDN arises from its advantages [209], so the programming capability of the network at



the application plane poses some critical security loopholes. These provide new vulnerabilities and attack scenarios, which in turn create new dangers that weren't there previously or were challenging to capitalize on [210]. The trust relationship between the controller and network apps has been characterized as a significant hazard or vulnerability in the discussion [10]. The controller in an SDN architecture is conceptually centralized and has a global perspective of the network's conditions, including connections, communicating devices, linked devices, and topologies. The network apps send instructions that govern the data plane's forwarding activity and ask the conceptually centralized controller for knowledge of the network state and views. From the security perspective of the network, this type of association between the controller and the network apps is considered as dangerous. For instance, depending on the adversary's motivation, a network application with a malicious purpose may use the global network status information to alter traffic flow. Furthermore, a poorly designed application may create security flaws that would otherwise jeopardize the smooth running of the network.

The SDN architecture encounters trust difficulties with existing network applications while supporting attempts by third parties to create applications. A breach of trust can result in a variety of assaults, each with serious repercussions that affect the network as a whole. As a result, there is a risk of trust breach between the application and the controller. SDN networks lack standard verification methods or processes to evaluate the trustworthiness of the network apps since their rules are expressly designed to let network apps implement changes inside the network [148],[211]. These rights can be abused by malicious apps, which would disrupt network functionality. Inbuilt network applications instantly acquire all access privileges to alter, control, or modify the state of the network [97] initiated with the controller. This is a severe hazard that is rarely discussed and gets little coverage in the available research. Since network programs have full access to vital network assets like flow tables, device settings, CPU, RAM, I/O ports, and so on, it becomes difficult to assess the dependability and credibility of a program module. When using third-party apps, network administrators often presume the same sort of confidence as the controller on network apps [142]. This is due to the fact that controller

units have to pass a number of tests and verifications before they can be used in operation or study. However evaluating a third-party application's dependability and credibility can be challenging [212]. Various host or network-based attacks may use a hacked or malicious application as a source. Control plane assaults that result in information leakage may then be possible. The many network applications that alter, modify, and influence network activities are contained at the application layer of the SDN architecture due to its design [213]. Every application is created with a certain logic and purpose in mind. SDN apps use the Northbound API to interface with the network while operating on top of the controller. Utilizing the controller's API, the main role is handling data flows that are included inside the forwarding components [214].

Utilizing the NBI API the apps can do:

- Configure flows to send packets along the best possible path from one destination to another.
- Multiple path load distribution and traffic consolidation.
- Respond to topological flexibility when a device fails, moves, joins, or leaves the network.
- Reroute traffic for security-related operations such as extensive packet inspection, verification, and separation in preparation for additional evaluation.

## **5.2 Problem Statement**

The northbound communication interface is used to facilitate communication between the controller and the network apps. This northbound interface only permits authorized network apps to program and demand services and resources from the network, in accordance with the SDN paradigm [9]. Access rights allow an application running on the network to read and write data, receive notifications, and perform system functions to influence the current network state to facilitate network interactions and the execution of network changes.

In Figure 5.1 a scenario with the untrusted applications within SDN architecture is shown where an adversary can disrupt the entire network by pushing one or more forged applications.

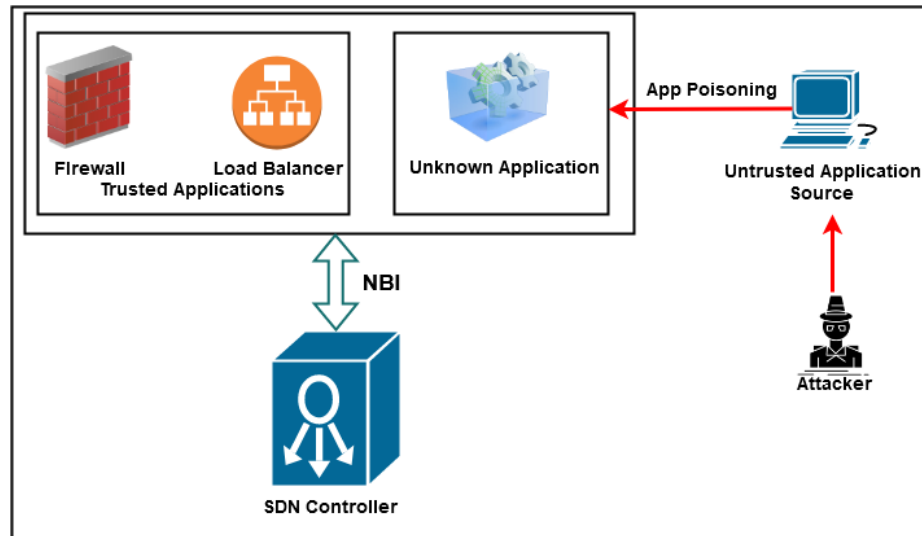


Figure 5.1 Untrusted Application Behavior

Networking applications use event receptions and services API calls to communicate with the common SDN control plane. One of the controller units may be read from and written to by a network application using a specific service mechanism [154]. An application running on the network may associate with the controllers and listen to events of choice for the event callbacks. Control, which includes defining network policies, accessing network status, receiving event notifications, and gaining access to resources through system calls, is essentially the focus of the interaction. The problem statement can be subdivided into four parts, they are-

### 5.2.1 Obtaining State of the Network

Any network application can obtain the state of the network by sending an HTTP GET request to retrieve device settings or network topology data. Using any accessible southbound API, mostly OpenFlow [215], the controller acts as a middleman to transmit the request made by the network app to the data plane switches. After receiving the requested data from the data layer switches, the controller will analyze it and provide an

HTTP response back to the network application. If the native controller API is utilized rather than a REST API call, the inquiry proceeds in a similar fashion.

### **5.2.2 Create A Network Policy**

The network application sends the controller an HTTP POST request to effect an alteration in the network state. By using the southbound API, the controller would interpret the request as a packet-out instruction or flow adjustment that needed to be applied in the appropriate switch's forwarding table in the data plane. The write access authorization is what allows the HTTP POST technique to alter the network state [143]. Network applications have an advantage when it comes to network control and management since they may create network regulations that have an impact on the network's overall status. Knowing exactly the access right a program needs to perform successfully on a network without affecting security or network functioning is crucial when setting it.

### **5.2.3 Receive Event Notification**

Network applications have the ability to subscribe to particular events inside the network and get notifications. A network application may subscribe to as many events as it needs to complete its assigned responsibilities; there are no limitations on the number or type of events it may subscribe to [125]. An interested network event may be joined or subscribed to by a network app; a suspicious network can do the same to listen in on, watch over, and monitor network traffic for the purpose of espionage.

### **5.2.4 System Call**

The operating system provides the abstraction that the controller sits atop, therefore access to resources such as RAM, CPU, Storage, processor threads, file system, and kernel modules should be controlled. The kernel code of the operating system is where network applications may tamper with system characteristics and affect network activities and the controller as a whole. Applications on the network that manage the states of the network are equipped with read, write, notify, and system call rights. There is no such access control procedure to cater to the detrimental intention, so if a network app designed by a third-party developer determines to launch a detrimental application that can use these

rights to affect network operations and hinder network resources, the attack would remain undiscovered.

### **5.3 Design of Trust Management Architecture and Working Procedure**

The design and structure of the trust management system are presented in this part. The suggested architecture for trust management aims to alleviate the security issues mentioned in the previous section. Figure 5.2 provides an illustration of the architecture. In a technical sense, the controller integrates the framework as a module. The modules are developed using Python programs, which are then directly integrated into the controller. The suggested design's viability is illustrated through the utilization of the OpenDaylight controller. The numerous controller classes, methods, and data are accessible to the constructed modules. The architecture for managing trust consists of many sub-modules that collaborate with each other to offer a reliable and secure communication channel via NBI among the controller and the network applications.

The trust management framework's system architecture will meet the security requirements required to lessen the hazard between the controller and the network application. This will be accomplished by

- supplying a means of using a credential for recognizing a network app.
- Setting the permissions that network programs can use.
- Analyze network applications' dependability and reliability in light of their interactions with the controllers.

In the proposed framework there are three main units or modules (as seen in Fig 5.2) to establish the trust relationship among the controller and the applications. These are-

1. Authentication Establishment Unit
2. Authorization Establishment Unit
3. Trust Establishment Unit.

The framework is primarily composed of another four additional components and interfaces, including the controller essential service modules, in addition to the three major modules. These are-

1. Application interface that enables the controller and related network applications to be chained together.
2. The interface for authentication, that links to the module for authentication. It uses an application instance and a unique token to inspect and verify apps.
3. The authorization interface links to the authorization module, which limits access to apps that have been verified.
4. The trust interface helps determine the belief and dependability of a network application.

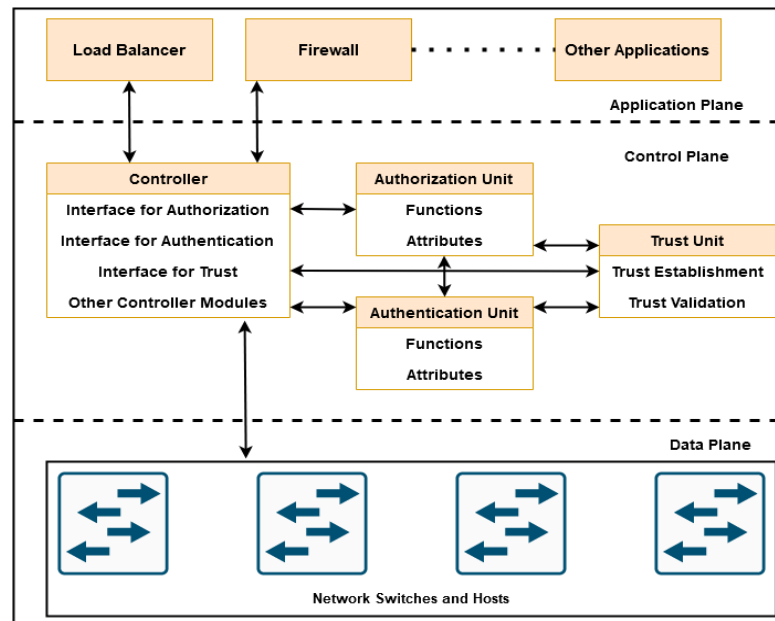


Figure 5.2 System Architecture

The three modules under the architecture works in tandem i.e. any unknown application will pass through the authentication establishment module and if it passes the authentication process then it will go to the next module i.e. the authorization establishment module for the authorization. Finally if it passes the authorization process then the application passes through trust establishment module for becoming a trusted

application. Now if the application fails in any stage i.e. suppose if it fails in authentication establishment stage, it will not go further i.e. in authorization establishment stage or in trust establishment stage, it will be treated as a harmful application. Similarly, after passing the authentication procedure if the application fails in authorization stage it will not go to the trust establishment module, it will be again treated as a harmful application. Any authenticated and authorized application should go to the final module i.e. trust establishment module and if it fails there then again it is not considered as a trusted application, hence not permitted for controller access via NBI. The applications passed through all the modules are considered as trusted applications and granted controller access. The entire process is depicted as a process flowchart in figure 5.3.

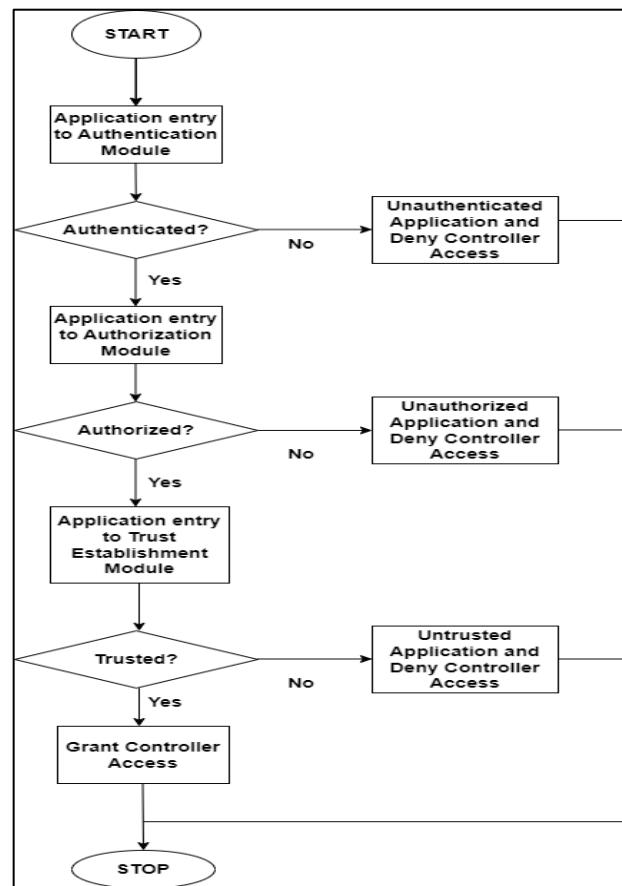


Figure 5.3 System Flowchart

### 5.3.1 Authentication Establishment Unit

One of the framework's primary units, the authentication unit, validates, checks, and either permits or prohibits SDN applications based on whether or not they pass the network application authentication tests. Every network application is given a unique Credential identification or token, which is utilized by the application rights management system throughout the authorization phase. When an application requests to use the control plane to perform network changes, the authentication process begins. Every running instance of an application has a distinct token, which forms the foundation for checks and verifications. Authentication always occurs first, prior to any other code being permitted to run or permissions to be checked for authorization.

This unit has two sub-units which are described below-

#### 5.3.1.1 Application Accuracy and Validation

This module checks for accuracy and confirmation to provide a seamless authentication procedure. The Application accuracy and validation unit handles this. In our study, we have considered two applications, namely Firewall and Load Balancer as seen in Fig 5.2. They are assigned two IDs APP\_1 and APP\_2 respectively for the purpose of demonstration (any further application will become APP\_3 and so on). The authentication establishment unit takes two arguments for its functioning- the application ID and token credentials which are used to uniquely identify the application. After the applications are successfully authenticated, they are placed in a table named Authenticated\_ applications; this segregates them from the other unauthenticated applications.

The authentication establishment unit is implemented by the following function-

authenticateApplication(ID,Token) where  $ID \in \{APP\_1, \dots, APP\_n\}$  and  $Token \in \{TOK\_1, \dots, TOK\_n\}$ . The following test cases may appear throughout the authentication process-



1. If no parameter is passed in `authenticateApplication()` i.e. `Param=0` (where `param=number of parameters`) it asks for the proper parameter i.e. ID and Token by “Enter Two Parameters” message.
2. If `Param=1` and `ID ∉ {APP_1....APP_n}` then the authentication process fails and terminates with the message “Authentication Failure”.
3. If `Param=1` and `ID ∈ {APP_1....APP_n}` but `Token ∉ {TOK_1...TOK_n}` then “Insufficient Argument” is flashed.
4. If `Param=2`, `ID ∈ {APP_1....APP_n}` and `Token ∈ { TOK_1...TOK_n }` then the `authenticateApplication()` succeeds with the message “Authentication Successful”

The algorithm is listed in Figure 5.4

Algorithm 1 Application Authentication Process	
1:	<b>procedure</b> AUTHENTICATEAPPLICATION( <i>ID,Token</i> )
2:	<i>ID</i> ← { <i>APP_1,APP_2,...,APP_N</i> }
3:	<i>Token</i> ← { <i>TOK_1,TOK_2,...,TOK_N</i> }
4:	<b>if</b> <i>Param</i> == 0 <b>then</b>
5:	Print(“Enter Two Parameters”)
6:	<b>else if</b> <i>Param</i> == 1 <b>AND</b> <i>ID</i> ∉ { <i>APP_1,...,APP_N</i> } <b>then</b>
7:	Print(“Authentication Failure”)
8:	<b>else if</b> <i>Param</i> == 1 <b>AND</b> <i>ID</i> ∈ { <i>APP_1,...,APP_N</i> } <b>AND</b>
	<i>Token</i> ∉ { <i>TOK_1,...,TOK_N</i> } <b>then</b>
9:	Print(“Insufficient Argument”)
10:	<b>else if</b> <i>Param</i> == 2 <b>AND</b> <i>ID</i> ∈ { <i>APP_1,...,APP_N</i> } <b>AND</b>
	<i>Token</i> ∈ { <i>TOK_1,...,TOK_N</i> } <b>then</b>
11:	Print(“Authentication Successful”)
12:	<b>end if</b>
13:	Update Authenticated_Application with <i>APP_N</i>
14:	<b>end procedure</b>

Figure 5.4 Algorithm for Application Authentication

### 5.3.1.2 Token Generation Process

The token used in this study is a distinct 16-character key for each network application, and it must be altered after a certain amount of time. Within architecture, these locally created tokens have a local importance. They cannot be duplicated or exported to another program. A collection of defined-boundary natural integers and alphanumeric variables are permuted, combined, and randomly generated to initiate the token. Their key space

and power against brute force are defined by their 16-character length. The key is derived from the following set-

$$X = \{A | a \in a \dots z | A \dots Z\}$$

$$Y = \{0 \dots 9\}$$

$$\text{Therefore, } XY = \{X \cup Y\}$$

$$\text{rand} = \text{makeRandom}(P, Q) \text{ where } P = \text{length}(XY), Q = 16 \text{ character key}$$

$$\text{Tok}_n = \text{createToken}(\text{rand})$$

The makeRandom(P, Q) function takes two parameters to generate a random sequence where P is the length of XY and Q is a 16-character key. This function generates a random sequence which is fed as input to createToken() which finally creates a 16-character token for the application. The algorithm is illustrated in Fig 5.5.

Algorithm 2 Token Generation Process	
1:	<b>procedure</b> CREATE_TOKEN( <i>rand</i> )
2:	$X \leftarrow \{A   a \in a \dots z   A \dots Z\}$
3:	$Y \leftarrow \{0 \dots 9\}$
4:	$XY \leftarrow X \cup Y$
5:	$\text{length\_XY} \leftarrow \text{length}(XY)$
6:	$\text{key\_length} \leftarrow 16$
7:	$\text{token\_list} \leftarrow \text{empty list}$
8:	<b>for</b> $i = 1$ <b>to</b> $\text{key\_length}$ <b>do</b>
9:	$\text{token\_character} \leftarrow XY[\text{rand}(\text{length\_XY})]$
10:	Append $\text{token\_character}$ to $\text{token\_list}$
11:	<b>end for</b>
12:	$\text{token} \leftarrow \text{concatenate all characters in token\_list}$
13:	<b>return</b> $\text{token}$
14:	<b>end procedure</b>
15:	
16:	<b>procedure</b> TOKEN_GENERATION
17:	$\text{tokens} \leftarrow \text{empty list}$
18:	<b>for</b> $i = 1$ <b>to</b> $2$ <b>do</b> <span style="float: right;">▷ Generating two tokens</span>
19:	$\text{token} \leftarrow \text{createToken}(\text{rand})$
20:	Append $\text{token}$ to $\text{tokens}$
21:	<b>end for</b>
22:	<b>return</b> $\text{tokens}$
23:	<b>end procedure</b>

Figure 5.5 Algorithm for Token Generation

### 5.3.2 Authorization Establishment Unit

After successful authentication, authorization deals with the particular rights that a network application is permitted to run on the network. It acts as the second step before authorization privileges are granted to apps that have successfully completed the authentication process.

Depending on how an application functions and behaves, different applications have different permission sets. Table 5.1 displays several characteristics of the load balancing application (APP\_2) that have been identified. Access control and permission are implemented by applying a wrapper to the characteristics from the network application's list of attributes.

Table 5.1 Attributes of Load Balancer Application

<b>Application Attributes</b>	<b>Corresponding Functions</b>
<i>Read_Port</i>	<i>To list all switch's linked ports, including virtual and actual.</i>
<i>Select_Route</i>	<i>Determine the best path for the flow of traffic using the predefined policy.</i>
<i>Flow_Count</i>	<i>Make a flow table with the total flows that are transmitted or retrieved.</i>
<i>Read_Buffer</i>	<i>To access ports' and interfaces' buffers.</i>

Each network application's set of rights is specified during the authorization phase. An application's distinctive ID is used for binding its rights, and it is specific for every network application. These authorizations consist of the associated instructions that a network application uses to create network policies, register events, read the network status, and initiate system calls. The following components made up the authorization model-

- Set of Applications= {APP\_1, APP\_2,.....APP\_n}
- Set of Objects={Obj\_1, Obj\_2.....Obj\_n}
- Set of Allotted Read Permissions Rd= {Rd\_1, Rd\_2....Rd\_n}
- Set of Allotted Write Permissions Wr= {Wr\_1,Wr\_2.....Wr\_n}

- Set of Allotted Notify Permissions  $Nt = \{Nt\_1, Nt\_2, \dots, Nt\_n\}$
- Set of Allotted System Call Permissions  $Sc = \{Sc\_1, Sc\_2, \dots, Sc\_n\}$

Therefore, the set of all permissions is defined as-

- $All\_Per = \{ Rd \cup Wr \cup Nt \cup Sc \}$

One example of an application say APP\_1 with different permissions may look like this-

- $APP\_1 = \{ Rd, Wr, Nt, Sc \}$

Applications are only allowed to have a subset of All\_Per at any given time, which implies that no application is allowed to read, write, receive alerts, or perform system calls to any state of the network. After comparing the application permission in a boolean fashion, the authorization module returns a value of 0, which indicates that the application does not have permission, and 1, which indicates that it does.

The authorization method is used to ascertain if a network action that is linked to permission that an application has established is permitted. The action will proceed if the permission is granted; but, to safeguard the network from harmful or unauthorized access, the request will be rejected if the application lacks the necessary authorization for that request. The authorization module consists of the authorization permissions that are permitted for that application as well as the authentication module. Thus, before authorization powers are granted to an application, the application must first authenticate using its token. The algorithm is presented in Fig 5.6

<p><b>Algorithm 3</b> Application Authorization Process</p> <pre> 1: <b>procedure</b> AUTHORIZEDAPPLICATION(<i>ID</i>, <i>Token</i>, <i>All_Per</i>) 2:   <b>Input:</b> 3:     <i>ID</i> = List of application IDs 4:     <i>Token</i> = List of corresponding tokens 5:     <i>All_Per</i> = List of all permissions 6:   <b>for</b> <i>i</i> = 1 <b>to</b> <i>length</i>(<i>ID</i>) <b>do</b> 7:     <b>if</b> AUTHENTICATEAPPLICATION(<i>ID</i>[<i>i</i>], <i>Token</i>[<i>i</i>], <i>All_Per</i>) == False 8:       <b>then</b> 9:         PRINT("Authorization Denied") 10:      <b>else</b> 11:        PRINT("Authorization Successful") 12:        <i>n</i> ← Number of allotted permissions for <i>ID</i>[<i>i</i>] 13:      <b>end if</b> 14:    <b>end for</b> 15:  <b>end procedure</b> </pre>
---

Figure 5.6 Algorithm for Application Authorization

### 5.3.3 Trust Establishment Unit

A directed graph relation between two entities, referred to as the trustee and the trustor, can be used to represent trust. The controller acts as the trustor and the corresponding network application as the trustee in this trust structure. The trustor must be able to make decisions that are rational and competent, and they must base their assessments on facts that they have access to and can use to influence their decisions. Because it is the network's fundamental logic that manages network functioning, these requirements fall into the controller model. In contrast, the trustee serves as the agent that the trustor depends on to deliver essential services or knowledge that will help the trustor make a big choice. Since the network program has tasks to do within the network, this works effectively for it as a trustee.

The next tier is the trust value, which offers network applications a more comprehensive recommendation based on how well the application behaves in accordance with the policy criteria established once the application has successfully completed the authentication and authorization phase. Although the decision of whether or not to trust an application is quite obvious and unambiguous, there are situations in which the application's trust value is somewhere in the middle of these two extremes.

There are five functions to establish a trust relationship between the controller and the applications. These are-

***Request Receiver:*** This function communicates with the authorization module and oversees transmitting each network application's current state of execution. It is the duty of this function to notify the subsequent function of any anomalies or breaches that may be noticed during execution. The output from this function is fed as input to the next function.

***Decision:*** Using probability calculus and the belief operations of belief (b), disbelief (d), and uncertainty (u), this function operates on the previous function's output to determine the trust equivalency. Utilizing state execution, this provides a report on the dependability and reliability of a network application when it is executed. The module offers a foundation for graph, triangulation, and distribution of probability function visualization of the trust.

***Probability Distribution:*** The results of the preceding function are reported on an ongoing range, and using the probability calculus is the most exact and accurate model for illustrating the comparable trust. This function will display the trust value at any given time, which can assist the framework in making an informed judgment about how trustworthy the application is.

***Trust Logger:*** Following the evaluation process according to the network application's current state of operation, all conclusions obtained from it are saved in the Trust Logger. This acts like a trust repository and facilitates expediting the trust process for subsequent evaluations of the same network application.

***Sign-in Log:*** All illegal attempts and network status events are recorded in a login database or log. Timestamps and the ID of the network apps to which the event is connected are kept in this log. Reports on network administration are compiled for examination and analysis. This log contains information about the anomalies and danger that were found. To further protect the network perimeter, network administrators can use this log report.

The operations are algorithmically explained in Figure 5.7.

Algorithm 4 Trust Establishment Process	
1:	<b>procedure</b> REQUESTRECEIVER
2:	<b>Input:</b> None
3:	<b>Output:</b> Current state of execution for each network application, anomalies or breaches noticed during execution
4:	<b>Process:</b>
5:	- Communicate with authorization module
6:	- Transmit current state of execution for each network application
7:	- Notify subsequent functions of any anomalies or breaches noticed during execution
8:	<b>end procedure</b>
9:	<b>procedure</b> DECISION(previousOutput)
10:	<b>Input:</b> Output from REQUESTRECEIVER()
11:	<b>Output:</b> Trust equivalency report
12:	<b>Process:</b>
13:	- Use probability calculus and belief operations (belief, disbelief, uncertainty)
14:	- Determine trust equivalency based on <i>previousOutput</i>
15:	- Provide reliability and dependability report
16:	- Offer foundation for trust visualization
17:	<b>end procedure</b>
18:	<b>procedure</b> PROBABILITYDISTRIBUTION(previousOutput)
19:	<b>Input:</b> Output from DECISION(PREVIOUSOUTPUT)
20:	<b>Output:</b> Trust value at any given time
21:	<b>Process:</b>
22:	- Report results on an ongoing range
23:	- Utilize probability calculus for accuracy
24:	- Assist framework in making informed judgments
25:	<b>end procedure</b>
26:	<b>procedure</b> TRUSTLOGGER(evaluationResults)
27:	<b>Input:</b> Conclusions from evaluating network application's state
28:	<b>Output:</b> Saved conclusions acting as a trust repository
29:	<b>Process:</b>
30:	- Save conclusions obtained from evaluation
31:	- Facilitate expedited trust process for subsequent evaluations
32:	<b>end procedure</b>
33:	<b>procedure</b> SIGNINLOG(illegalAttempts, networkEvents)
34:	<b>Input:</b> Illegal attempts and network status events
35:	<b>Output:</b> Reports for network administration examination and analysis
36:	<b>Process:</b>
37:	- Record illegal attempts and network events
38:	- Keep timestamps and network application IDs
39:	- Compile reports for examination and analysis
40:	- Aid in network perimeter protection
41:	<b>end procedure</b>

Figure 5.7 Algorithm for Trust Establishment

## 5.4 Implementation Details

This section deals with the implementation of the proposed framework which is installed on top of the OpenDaylight controller installed in Ubuntu 20.04 VM with 8GB of RAM and AMD Ryzen 5 5600X CPU. SDN environment is emulated using the Mininet emulator with a similar type VM with 6 GB of RAM. We will first test the authentication

module then the authorization module and finally the trust establishment module with the help of two applications i.e. Firewall (APP\_1) and Load Balancer (App\_2).

#### 5.4.1 Authentication Testing

To test the working of the authentication module we first incorporated this module on the Firewall application with ID APP\_1 and then the behavior of the Load Balancer (APP\_2) application was noticed. The findings are listed below.

The first application i.e. firewall application is assigned ID as APP\_1 and the second application i.e. load balancer application is assigned ID as APP\_2. Let the tokens generated for them are TOK\_1 and TOK\_2 respectively. Therefore, to access their authenticity these two parameters are passed to the AuthenticateApplication module as parameters, and hence Param=2. To authenticate the applications let-

$\alpha_1$  denotes the first argument ID as APP\_1

$\beta_1$  denotes the second argument TOKEN as TOK\_1

$\alpha_2$  denotes the first argument ID as APP\_2

$\beta_2$  denotes the second argument TOKEN as TOK\_2

$\gamma_1$  Successful authentication for APP\_1

$\delta_1$  unsuccessful authentication for APP\_1

$\gamma_2$  successful authentication for APP\_2

$\delta_2$  unsuccessful authentication for APP\_2

Therefore

$$(\alpha_1 \wedge \beta_1) \rightarrow \gamma_1 \text{ -----(1)}$$

Which denotes successful authentication of APP\_1 iff ID and TOKEN parameters are correct and both are present. Another situation might be

$$\neg (\alpha_1 \wedge \beta_1) \rightarrow \delta_1 \text{ -----(2)}$$



Which denotes unsuccessful authentication of APP\_1 if ID and TOKEN are not valid. A similar behavior is noted for APP\_2.

Therefore

$$(\alpha_2 \wedge \beta_2) \rightarrow \gamma_2 \text{-----}(3)$$

and

$$\neg (\alpha_2 \wedge \beta_2) \rightarrow \delta_2 \text{-----}(4)$$

A tabular structure based on the authentication reality is provided in Table 5.2 as an additional means of verifying the accuracy and integrity of the network applications. Truly Positive (TP), Truly Negative (TN), False Positives (FP), and False Negatives (FN) are the terms used in the matrix's nomenclature.

Table 5.2 Terminology for Accuracy in Authenticating

TP	A valid network application requests authentication, and the application successfully completes the authentication process.
TN	An unauthorized application requests authentication, however, the application fails because the credentials are incorrect.
FP	unsuccessful attempt at authentication by an accredited network application using the right credentials.
FN	A harmful app that completes its authentication.

The two programs were run 100 times against the authentication module using the right credentials in order to assess their accuracy and validity. All 100 trials in the test were successful, and only Truly Positive (TP) hits and no anomalies were noted. A similar test is conducted for a threatening Hping application as well; however, on this occasion, all the hits are on True Negatives (TN), and the malicious Hping application's authentication would fail due to the incorrect token which is elaborated in table 5.3.

Table 5.3 Application Precision

Application	Truly Positive	Truly Negative	False Positive	False Negative
Firewall	100	0	0	0
Load Balancer	100	0	0	0
Hping	0	100	0	0

#### 5.4.2 Authorization Testing

The authorization process is the second step after authentication for the network application. An application's logic and intended use must be understood beforehand in order to be authorized. The requirement is that in the event of authentication failure, the application cannot go on to authorization.

A firewall is a critical component of network security infrastructure, acting as a separator between an intranet and an extranet, such as the Internet. It operates by inspecting and controlling the flow of data packets based on predefined security rules. One of its primary functions is packet filtering, where each packet entering or leaving the network is examined. This examination involves scrutinizing attributes such as source and destination IP addresses, port numbers, and protocol types. By comparing these attributes against predetermined rules, the firewall decides whether to allow or block the packet. Firewalls may analyze data payloads at the application layer of the OSI model, a process known as application layer filtering, in addition to packet filtering. They can recognize and manage certain programs or services, such as HTTP, FTP, or DNS, thanks to this capacity. Application layer filtering helps reduce hazards related to unauthorized software or services by enforcing regulations governing which apps are allowed or banned within the network.

The firewall application must first successfully complete authentication before authorization can begin. It receives the authorization privileges after successful authentication.

Therefore the AuthorizeApplication module is loaded with the following parameters

ID=APP\_1

TOKEN=TOK\_1

All\_Per={Rd,Wr,Nt,Sc}

Now at this point AuthenticateApplication module is already true for ID=APP\_1 and TOKEN=TOK\_1. The application APP\_1 is already contained in the Authenticated\_application table after a successful authentication process therefore it has been authorised further and Read (Rd), Write (Wr), Notify (Nt), and System call (Sc) all are allocated to it.

Load balancer applications play a vital role in ensuring high availability, scalability, and reliability of distributed systems by efficiently distributing incoming network traffic among multiple servers or resources. They form a crucial component of modern web architectures, cloud deployments, and microservices environments, enabling organizations to deliver responsive and resilient applications to their users.

In the case of a load balancer application (APP\_2) it is also a part of the Authenticated\_application table because it was authenticated beforehand. From Table 2 we can see that a load balancer application exhibits some attributes like *Read\_Port*, *Select\_Route*, *Flow\_Count*, and *Read\_Buffer*. So here also it needs the Read (Rd), Write (Wr), Notify (Nt), and System call (Sc) permissions to function properly. Therefore the AuthorizeApplication module is loaded with the following parameters

ID=APP\_2

TOKEN=TOK\_2

All\_Per={Rd,Wr,Nt,Sc}

Here also the load balancer application was successfully authenticated previously and with the aforementioned parameters it is successfully authorized. The allocated permissions for it are Read (Rd), Write (Wr), Notify (Nt), and System call (Sc).

### 5.4.3 Testing for Trust Establishment

Once prescribed functions are executed successfully and there are no anomalies or deviations from the list of stated tasks, applications can be trusted. Applications are given numerical trust levels according to how well they follow the network policy. A high trust volume will be achieved by the application if the tasks it has been assigned are successfully completed. Consequently, this will enable the controller to communicate freely with the network application. If the network application does not adhere to these specified sets of functions, the controller will notify it and assign a low trust rating, which may ultimately result in the program's total denial of access to all network resources. Table 5 lists the default values for trust that are allocated to each network application itself that runs successfully. These values are determined by the architecture of the trust establishment framework.

According to Table 5.4 the different trust parameters are Belief(B), Disbelief(D), and Uncertainty(U) whose volumes are 0.8, 0.2, and 0 respectively for the authenticated and authorized applications and the volume of Disbelief is negligible and no Uncertainty is there.

Table 5.4. Trust Parameters and Their Volume (Application view)

Trust Parameter	Trust Parameter Volume
Belief (B)	0.8
Disbelief (D)	0.2
Uncertainty (U)	0

The network application computes these values of trust (B, D, and U) for itself. The controller also maintains a similar table for these parameters which is shown in table 6. It will use a trust recalculation procedure by multiplying two values to assess the final trust values when it is associated with the application. Additionally, the controller has some faith in the program.

Table 5.5. Trust Parameters and Their Volume (Controller view)

Trust Parameter	Trust Parameter Volume
Belief (B)	1
Disbelief (D)	0
Uncertainty (U)	0

For the assessment of trust to be effective, the first two phases need to be a true value. The trust validation stage would not move further in the event that either the authorization or the authentication were false, which would ultimately result in trust failure. The trust validation (TV) is given by

$$TV = \forall \text{Applications} \in (\text{APP}_1 \dots \text{APP}_n) \exists (\text{TOK}_1 \dots \text{TOK}_n \wedge \text{functions})$$

$$\therefore TV = (\text{Authentication} \wedge \text{Authorization} \wedge \text{Trust})$$

For the firewall application (APP\_1) the trust volume for B D and U is 0.8, 0.2, and 0 respectively. The trust volume on the controller is B=1, D=0, and U=0 for this application bearing the ID which is APP\_1. Therefore, the final values are-

$$B \text{ is } 0.8 * 1 = 0.8$$

$$D = 0.2 * 0 = 0$$

$$U = 0 * 0 = 0$$

Hence the final value of belief (B) is 0.8 and disbelief (D) is 0 which implies that the firewall application with ID=APP\_1 and Token=TOK\_1 is a trusted application.

For the load balancer application (APP\_2) the trust volume for B D and U is 0.8, 0.2, and 0 respectively too. The trust volume on the controller is B=1, D=0, and U=0 for this application bearing the ID which is APP\_2. Therefore, the final values are-

$$B \text{ is } 0.8 * 1 = 0.8$$

$$D = 0.2 * 0 = 0$$

$$U = 0 * 0 = 0$$

Hence the final value of belief (B) is 0.8 and disbelief (D) is 0 which implies that the load balancer application with ID=APP\_2 and Token=TOK\_2 is a trusted application.

### 5.5 Comparative Analysis

This section offers a thorough comparison of the suggested framework with various models that are currently in use and aims to address issues ranging from network application security to controller problems. It is essential to assess and validate any model or framework developed in a research project.

To evaluate the behavior of the suggested model and provide a comparable comparison with Rosemary [120], we employ the following five relevant indicators.

1. **Load on Processor:** This indicator shows the usage of the CPU or burden imposed on it due to the framework and the status of it without the framework. As we know, the module is implemented on top of OpenDaylight architecture therefore, by running the module some additional amount of processing capacity will be consumed.

The number of instances are plotted on the X-axis and the time required to successfully complete the computational framework on the Y-axis, is measured in milliseconds (ms). Analysis of Figure 5.8 reveals that Rosemary [120] completed the task in 59.63 milliseconds, whereas our proposed framework completed it in 54.71 milliseconds, indicating a 4.92 milliseconds reduction in time compared to Rosemary [120]. This reduction suggests a significantly lighter load on the processor. Furthermore, it's noticeable that in the absence of any trust establishment framework, the curve follows an almost linear pattern, requiring only 26.81 milliseconds.

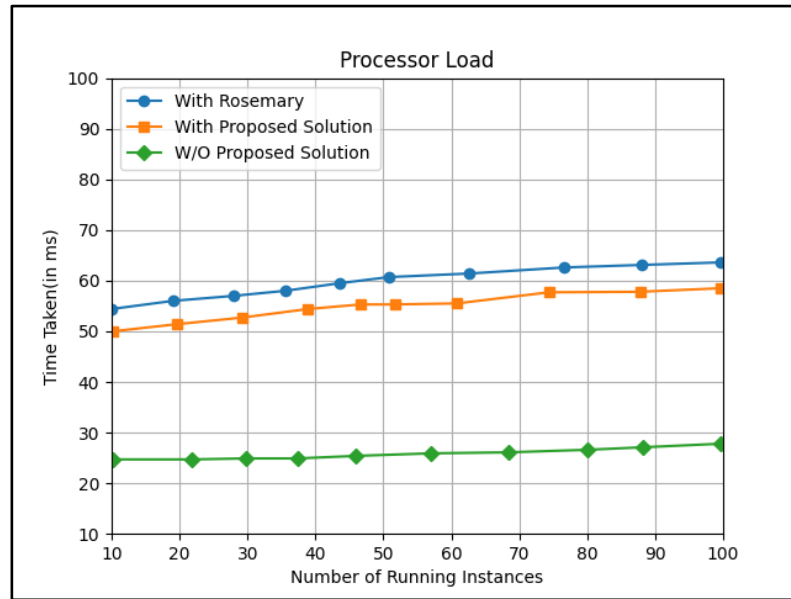


Figure 5.8 Load on Processor

2. **Load on Memory:** This is another key indicator that depicts the amount of memory consumed when the framework is running and when it is not running. Again, in this case, memory consumption will be higher when the framework is running. For this examination, we've depicted the number of running instances on the X-axis and memory usage in megabytes on the Y-axis. Figure 5.9 illustrates that Rosemary[120] consistently utilizes a greater amount of memory in each run compared to our suggested framework. When no such application is active, memory usage remains relatively constant across various instances.

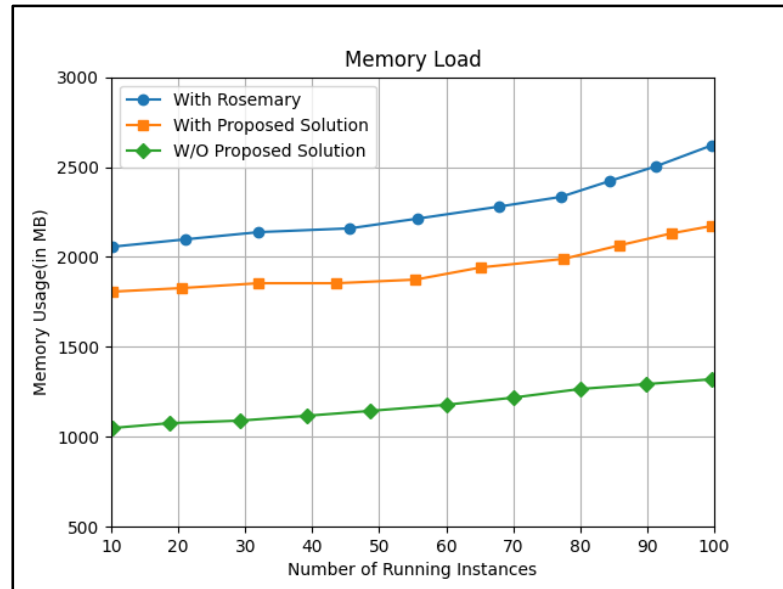


Figure 5.9 Load on Memory

3. **NBI Usage-** When applications communicate with the controller via the NBI, activating a filtration mechanism slows down communication. However, without such restrictions, communication is faster. Our proposed solution establishes trust in 4.87 milliseconds, consuming 10.5 Kbps of NBI bandwidth. After trust is established, communication proceeds without delay, increasing NBI bandwidth usage. Assuming an application typically requires 20 milliseconds to communicate with the controller, our solution completes the entire process, including trust establishment and data transfer, in 12.99 milliseconds. In contrast, Rosemary [120] takes 7.13 milliseconds for trust establishment—2.26 milliseconds longer than our approach—and consumes 11.8 Kbps of bandwidth, 1.3 Kbps more than our solution. Rosemary [120] finishes the entire process in 15.48 milliseconds, 2.49 milliseconds longer than our method. Thus, our solution demonstrates greater efficiency in both NBI bandwidth consumption and overall performance, as shown in figure 5.10.



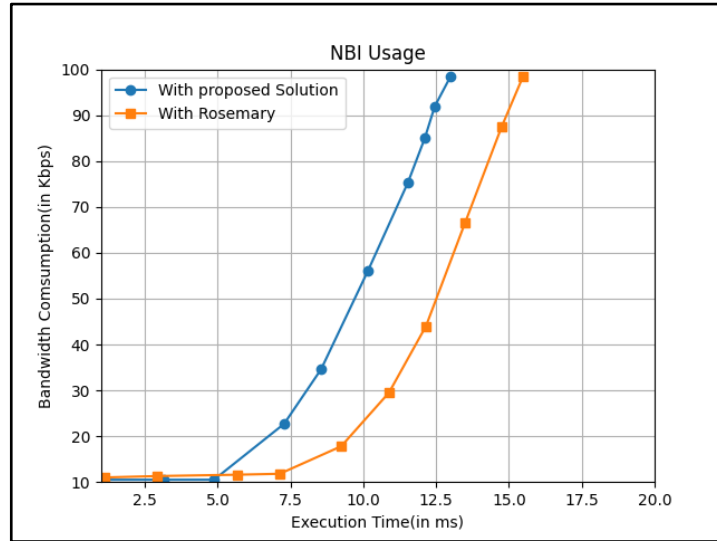


Figure 5.10 NBI Usage

4. **Throughput:** It is the amount of data transferred between the controller and the applications. When the authentication, authorization, and trust establishment modules are running there will be no or negligible communication between the controller and the applications. Once the applications are trusted, throughput will increase drastically. It can be observed from Figure 5.11 that our proposed solution is gaining higher throughput values over time and initially both approaches perform similarly. In the case of Rosemary [120] the quantity of achieved throughput got stuck on 38.8 in 10<sup>th</sup> instance whereas our proposed approach achieves the value of 49.7 on 10<sup>th</sup> instance. Therefore, after certain instances, Rosemary [120] will exhibit linear throughput which implies it is somehow frozen at that point.

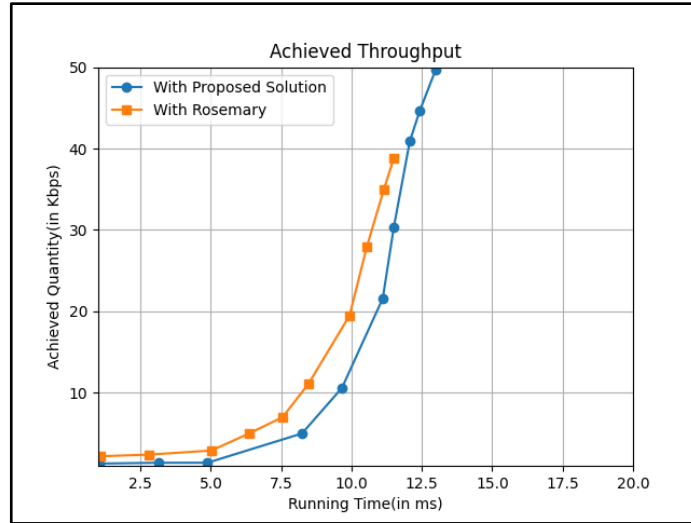


Figure 5.11 Achieved Throughput

5. **Network Latency**- Latency is delay in communication. Therefore, before the trust establishment, it will be very high and once the applications are trusted it will reduce gradually. It can be observed from Figure 5.12 that from the initial stage of the trust establishment process the latency in the proposed approach is 8.74 ms and in Rosemary [120] it is 8.77 ms. Both the values are very close to the maximum value, which is 10. The network latency decreases over time in both approaches as the trust establishment process is over and normal NBI access and controller communication is active. Our proposed approach achieves lower latencies as compared to Rosemary [120] in all time instances which implies a better communication efficiency with the controller after the framework has completed its task.

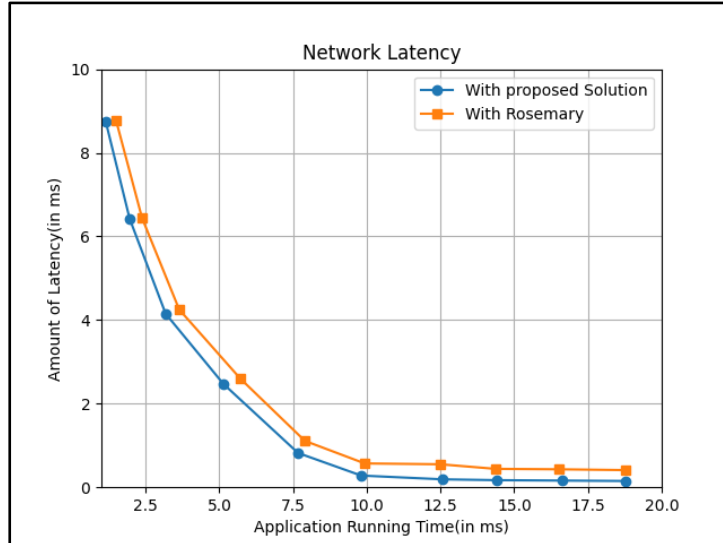


Figure 5.12 Network Latency

## 5.6 Security Achievements

The key security achievements are listed below-

### Authentication Establishment:

- Every network application is assigned a unique credential token, ensuring only verified applications can interact with the control plane.
- Authentication occurs before any code execution, preventing unauthorized applications from initiating any actions.

### Authorization Establishment:

- Specific permissions are assigned to each authenticated application, controlling their access to network functions.
- The framework uses a fine-grained permission model that limits applications to only the necessary actions, reducing the risk of malicious activities.

**Trust Establishment:**

- Continuous monitoring of application behavior allows for dynamic assessment of trustworthiness.
- A trust logger maintains records of all applications' actions, enabling quick detection and response to anomalies or breaches.
- Probability-based trust evaluation provides a nuanced understanding of application reliability, enhancing decision-making for network security.

**Robust Authentication:** Ensures only verified applications gain access, preventing unauthorized entities from interacting with the control plane.

**Granular Authorization:** Assigns precise permissions to each application, limiting their capabilities to essential functions and minimizing security risks.

**Dynamic Trust Evaluation:** Continuously assesses application behavior, maintaining a trust log and using probability-based evaluation to ensure reliable and secure application interactions.

By implementing these security measures, the framework significantly enhances the protection of the SDN environment, ensuring robust control over application interactions and safeguarding network integrity.

## 5.7 Chapter Summary

This chapter addresses the security challenges inherent in the northbound communication interface and application plane within Software-Defined Networking (SDN). These components are crucial for the interaction between the controller and network applications, facilitating the programming of demanding network services. However, these interactions are susceptible to significant security risks due to inadequate access

control mechanisms. Potential threats include unauthorized applications gaining access to sensitive network state data, altering network policies, eavesdropping on network events, and exploiting system resources, all of which can result in severe security breaches.

To mitigate these risks, the chapter introduces a comprehensive trust management framework integrated into the SDN controller. This framework enhances security through three primary components: authentication, authorization, and trust evaluation.

1. **Authentication:** The framework assigns unique credential tokens to verified applications, ensuring that only authenticated entities can access the network.
2. **Authorization:** Specific permissions are allocated to authenticated applications, restricting their access to critical functions and preventing unauthorized modifications and monitoring.
3. **Trust Evaluation:** Continuous behavior assessment and a dynamic trust evaluation process further reinforce security by continuously monitoring application activities and adjusting trust levels accordingly.

By implementing these components, the framework ensures robust security and maintains the integrity and reliability of the SDN environment. This approach significantly strengthens the security of application interactions within SDN, ensuring that only trusted applications can communicate with the controller and access network resources.

## **Chapter 6**

### **A Secure and Resilient Distributed Control Plane Architecture for Large-Scale SDN Deployments**

The rapid evolution of cloud data centers has led to the proliferation of large-scale, distributed architectures, necessitating advanced management and optimization strategies. Software-Defined Networking (SDN) has emerged as a critical enabler for efficient, scalable, and flexible network management in these environments. The aim of this chapter is to present a comprehensive analysis of SDN's role in enhancing the performance and scalability of distributed cloud data centers. It explores the inherent challenges associated with traditional network architectures, particularly in terms of scalability, resource utilization, and security and proposes a novel Secure and Resilient Distributed Control Plane (SRDCP) which is based on real time publish/subscribe model to address these issues. The chapter highlights how SDN, through its centralized control and programmability, addresses these challenges by enabling dynamic resource allocation, efficient traffic management, and streamlined network configuration. Furthermore, the chapter examines the impact of SDN on improving network agility and reducing operational complexity, making it an ideal solution for large-scale deployments. However, the factors for further investigation and improvement like increased latency and the need for reliable SDN controllers, proposing strategies to optimize deployment and enhance network resilience are also identified. On the note of the security aspects, this chapter provides a comprehensive security threat analysis using Microsoft STRIDE threat modelling approach to establish the security exposure of the proposed solution. The key findings are technically compared with existing state-of-the-art solutions and proven better for large scale distributed SDN deployments in data center networks, offering insights into best practices and future research directions.

#### **6.1 General Overview**

Fast and dependable connection is becoming more and more necessary for today's and tomorrow's networks to handle a growing number of network-capable devices, which is

predicted to reach 36.7 billion by 2026 [216]. Concern is heightened by the requirement for Enterprise and Data Centre Networks (DCNs) to meet Quality-of-Service requirements for high-bandwidth capacity, scalability, resilience, and security. Programmable networks are thought to offer a way to accomplish the above goals and transform ways of interacting.

A programmable network idea called software-defined networking (SDN) [217] was created to get around some of the drawbacks of traditional networks, such vendor lock-in, difficulty in configuration and maintenance, and lack of flexibility. A customizable network control that is independent of relaying is suggested by SDN. The two primary tenets of the idea are the control plane's flexibility and the separation of its control and data planes [218]. Data forwarding modules cede control responsibilities to the controller, a conceptually central networking unit. The commands sent towards the controller through the different APIs like northbound or southbound API are the source of SDN's programmability. The control system may establish network-wide traffic forwarding choices, making network configuration, policy enforcement, and evolution simpler [219]. Centralized and distributed designs are the two classifications into which controllers fall. Fundamentally centralized, the first class of controllers provided a simpler design choice.

Network design became a focus point for single points of failure and performance constraints as they grew [15]. A single physically centralized controller became overwhelmed by large-scale network installations, including Data Centre Networks, which included tens of thousands of network components [152]. A physically centralized controller called NOX [153] can handle 32,000 flow queries per second with an 11 ms delay. Concerns about controller flexibility and safety gained front stage in the adoption of SDN [220], [221].

Multiple geographically dispersed controllers working together to settle a single, congruous picture of the entire network make up the logically centralized control plane. Precise synchronization between controllers is required by the design. Logically

distributed control planes were developed to enable massive, dispersed networks wherein a conceptually centralized controller manages an area of routing components inside a huge broad zonal system and interacts with various controllers across domains. Globally dispersed data centers and wide area networks are the main applications for this class of controllers. Because of its dispersed form, the control plane's scalability problem endures throughout all SDN classifications. An important issue covered in the literature is the East/West interface, or the technique for communication across dispersed entities. One aspect that stands out is the lack of a communication standard among inter-domain SDN entities [222].

This study proposes the Secure and Resilient Distributed Control Plane (SRDCP), an adaptive architecture for the East-West-bound interface. SRDCP employs the Real-Time Publish/Subscribe (RTPS) model [223], which is standardized and data-driven. SRDCP's features in comparison to earlier East-West bound methods are discussed. A series of experiments are conducted to evaluate the efficiency of the proposed interface in many SDN controller systems across diverse network types.

## **6.2 Problem Statement**

To retain a comprehensive picture of the entire network and enable path improvement for data streams between source and destination in massive, dispersed networks, distributed SDN controllers need close coordination of network policies. The research describes a variety of methods for transferring network status information across an extensive array of rules for East-West bound interface. While a few approaches are scalable, others are not capable of coordinating across platforms. Moreover, earlier studies mainly show how networks with uniformly distributed control planes may synchronize their network states. To describe and illustrate the use of a standardized approach in the east-west bound communication interface, this work provides a flexible data-driven RTPS framework for significant dispersed SDN control planes in both uniform and diverse networks.



### 6.3 Proposed System Framework of SRDCP

This section describes the overall functional structure of the proposed SRDCP framework. For diverse and distributed control planes, SRDCP is an adaptable solution for the East-West bound interface that synchronizes topologies using a common interaction mechanism. The SRDCP interface's flexible and modular design makes it possible to apply it across a wide range of SDN controllers. For the East-West bound interfaces, the suggested framework makes use of the Data Distribution Services (DDS) standard [224] which is a data-centric RTPS paradigm.

Figure 6.1 shows a structural overview of the SRDCP architecture. The architecture shows an interaction procedure among diverse controller types via a shared global space and a shared east-west bound interface. Basically, each controller domain creates its own cluster of underlying data plane forwarding devices and connected among several controllers using east-west bound interface while sharing common information using a RTPS model. This work describes the three types of distributed control plane techniques that the proposed interface allows: the Flat, Hierarchical, and the new proposed Hybrid Hierarchical (H2) model. Control plane entities use domain agents provided by SRDCP elements to implement communication inside the network domain. The framework is flexible to the controller platform's programming dialect and vendor-neutral when it comes to choosing the DDS implementations.

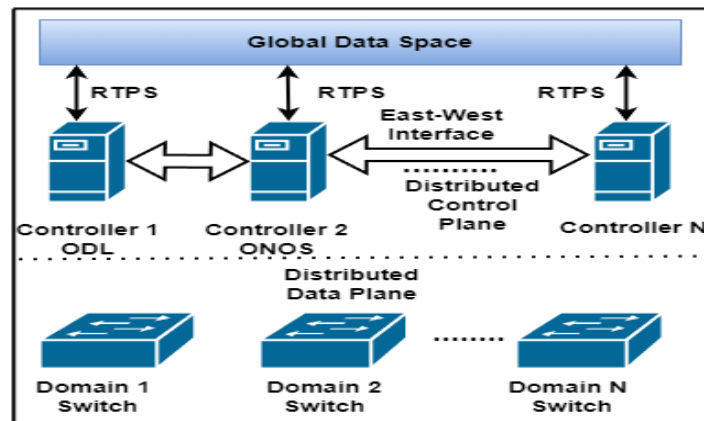


Figure 6.1: SRDCP Architecture

### 6.3.1 Design Goals of the Proposed Architecture

The proposed architecture has five primary design goals which are the major and obvious intentions of this design approach. These are highlighted below-

- **Expandability-** Providing an expandable inter-domain control plane framework for corporate and data-center networks is one of this framework's core goals. Furthermore, networks operating in a large group of nodes under stringent latency limits have significant challenges in terms of expansion.
- **Diversity-** A wide range of network users should be able to access the proposed framework by showcasing the control plane entity's vendor and platform atheism.
- **Trustworthiness-** The framework must need to endure and surpass worst-case situations, ensuring uninterrupted functionality under extreme or harsh conditions.
- **Security-** For the structure to remain intact, security is essential. Encrypted end-to-end data flow between verified parties should be made possible by it. This feature is made possible by the suggested framework via the DDS security extensions, described later in the text.
- **Punctuality-** Certain networks have time constraints and demand that the design provide quality of service for information sharing between controllers.

### 6.3.2 Interaction Strategy with Different Distributed Controller Scenarios

The distributed SDN controller architecture is mainly divided into two segments in case of inter-domain communications: flat model, where controllers are placed horizontally and hierarchical model where the controllers are placed vertically. This solution proposes a mixture of both the approaches which combines the essence of both the above-mentioned models, the Hybrid-Hierarchical (H2) model, depicted in figure 6.2. This segment describes the interaction mechanism of the SRDCP architecture with all of the distributed controller architectures i.e. flat, hierarchical and H2 model.

### ***Interaction with Flat Model***

The SDRCP architecture shown in figure 1 also depicts the interaction among different controllers in flat model using the proposed architecture. In this model, distributed controllers act as peer nodes. Nodes with predefined structure broadcast modifications to the local network state into the global shared data space. The nodes subscribe to updates of the network status data from every other peer. The link updates are kept at each node, from where a global overview of the network's management is constructed and kept up to date. Every node in the flat model keeps its overall perspective on the network.

### ***Interaction with Hierarchical Model***

In a tree-topology form, hierarchically distributed controllers obey a series of commands. Using a publish/subscribe mechanism, child nodes inform parent nodes about their local state data. The parent creates a comprehensive perspective by storing and integrating the state data from the child's domains. The conjugated snapshot is thereafter disseminated to global controllers, and the parent nodes function as the offspring of larger organisational realm entities.

### ***Interaction with Hybrid Hierarchical (H2) model***

In order to support large-scale dispersed networks that spread across many geographical regions, this study offers a hybrid hierarchical architecture known as the H2 model. In massive global networks, this model makes advantage of the horizontal dispersion of state data for global flat networks and its vertical dispersion for local hierarchical networks. From child to parent controllers in the hierarchy, the network status data travels vertically until it reaches the geographic organisational domain's root. When the general layout of the appropriate realm is established at the root, this layout is then spread horizontally across neighboring nodes for each global controller to get and construct the entire image.

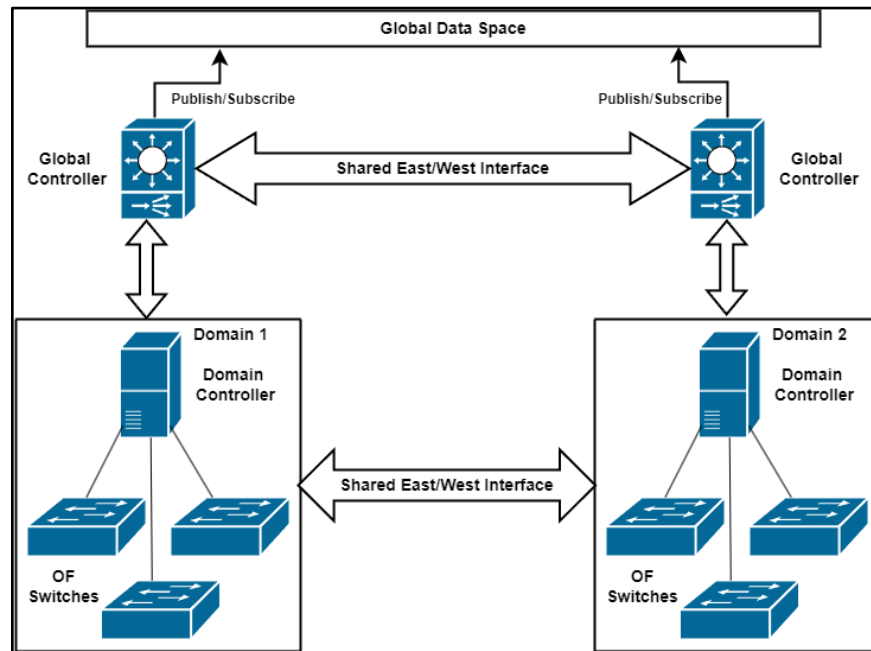


Figure 6.2 Proposed Hybrid Hierarchical Model of SRDCP

### 6.3.3 Interaction Strategy with Controller Modules by SRDCP and Module Details

This section highlights the controller modules to which the proposed solution SRDCP interacts with and how this is done. The proposed architecture works like a controller plugin or extension which is made up with a publisher or data writer and subscriber or data reader with DDS plugins.

An installed controller structure with an SRDCP layout is shown in Figure 6.3. handling data plane sending entities, like the OpenFlow protocol, via the southbound API is the responsibility of controller modules. In response to connection modifications within the local and global domains, SRDCP modules communicate with internal controller modules for synchronizing typologies. The details of SRDCP modules are elaborated in the next sub-sections.

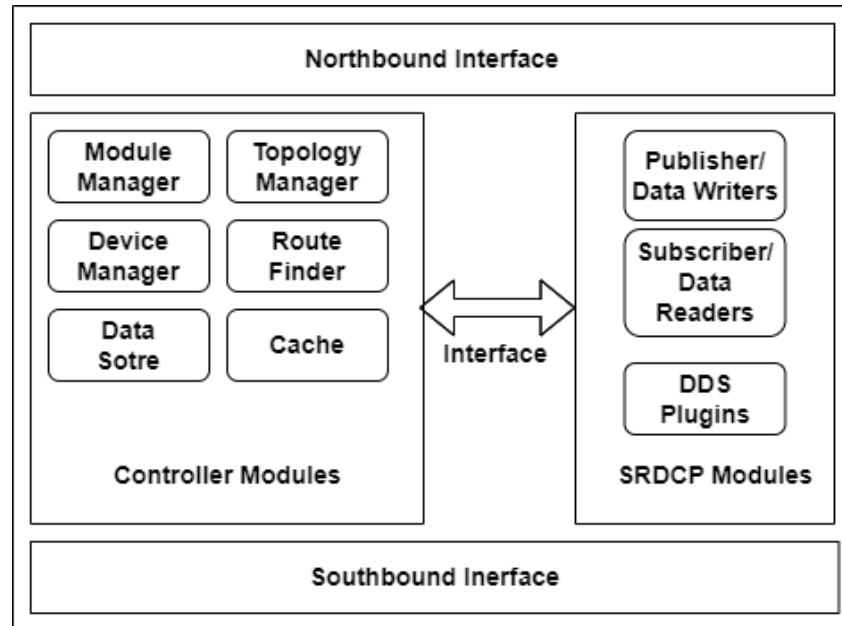


Figure 6.3 Interaction Between SRDCP and Controller Modules

#### 6.3.3.1 Publisher Module

The publisher module's job is to use the interface to keep an eye on the controller's topology management module. Each time something happens at the Topology Manager, a link update is created. This link update is received by the publisher module, which then transmits it to the appropriate data writer component in the SRDCP module. Next, an instance of the link change is created by the entity. After then, this notice is released into the designated domain's data space.

#### 6.3.3.2 Subscriber Module

Alerts of link updates are sent to the Subscriber module by the global domain that corresponds to its subscription. Subscriber data readers read data samples from the data space that correspond to specified subscribers. The module then uses the SRDCP Interface to update the Topology Manager with the samples that it obtained.

### 6.3.3.3 Topology Manager Module

This module manages the network's local and global views. It does this by using local domain activities to generate link updates and using the Subscriber module to receive changes from global domains. Through the northbound API, this module grants connectivity to network management applications while maintaining the topology version of the network.

### 6.3.3.4 Interface Module

The Interface module is responsible for facilitating synchronization between SRDCP modules and controller inner modules. The Interface consists of two main sub-components: listener and service. The service component alerts all other modules to apply the event listener in case of a generic link update event that reaches the subscriber, thereby informing the Topology Manager regarding the update.

### 6.3.3.5 Link Discovery Protocol (LDP)

The Link Discovery Protocol (LDP) is a 64 bit packet sized protocol proposed along with the SRDCP architecture which is used by SRDCP for informing about new link updates or discovery. It acts with the current SBI protocol implementation of the controller i.e. OpenFlow [21] or any other standard protocol. The protocol consists of eight 8-bit components mentioned below. The packet format is depicted in figure 6.4.

- **ControllerID**- This field specifies the unique identification of the controller willing to communicate.
- **Task**- This field specifies the actual operation to be performed upon link discovery or establishment, which might be one of these four operations: Link up, Link Down, Link Deleted or Link Modified.
- **MACSource**- This field specifies the MAC address from where the link discovery process initiated.

- PortSource- This field specifies the Port information via which the link discovery process initiated.
- MACDestination- This field keeps the info about the destination MAC address to which the link should be established.
- PortDestination- This 8-bit field records the port information of the destination via which the link has been established.
- Delay- This field keeps the information regarding the communication latency if any.
- LinkType- This field keeps the information regarding the type of link (mentioned above) that has been established between source and destination.

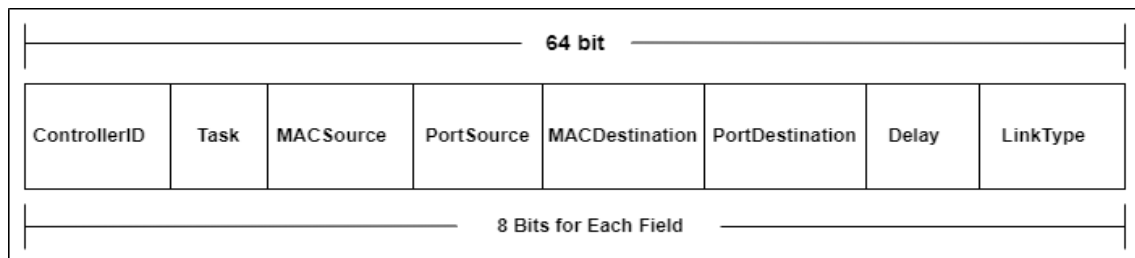


Figure 6.4 LDP Packet Format

#### 6.4 Constituent Entities of RTPS-DDS Model

The SRDCP framework is constructed on purview of Real Time Publish/Subscribe (RTPS) model [223] which is data centric along with the Data Distribution Services (DDS) [224] plugins, conjugatively said as RTPS-DDS model. The components and deliverables of this model is detailed in this section.

The RTPS-DDS paradigm offers Quality of Service qualities and can provide dependable, secure, and timely data sharing. Adopting the RTPS model for the control plane has the primary advantage of enabling immediate and independent interaction between an extensive set of attendees, which is not possible with client/server architectures, which typically limit the number of concurrent read and write activities,

create a risk of single point errors, and incur additional processing obstacles. Furthermore, DDS plugin guarantees secure end-to-end data communication by means of security plugins that provide encryption, access control, logging, and authentication.

#### 6.4.1 Components of RTPS Model

The following entities constructs a data-centric RTPS model-

- **Publisher-** A group of Data Writers who update fresh samples of the subjects that Data Readers have subscribed to is called a publisher. It removes unnecessary created samples and shares only accurate data.
- **Subscriber-** A group of data readers that receive current versions of the data subjects they find interesting in is known as a subscriber.
- **Domain-** Domains are practically isolated from one another, and data posting and subscription are managed on a per-domain level. Data cannot be shared between individuals in different domains.
- **Topic-** The samples of data that specify the information to be shared are called topics. By way of the common subject samples, a data writer and data reader are implicitly coupled.

#### 6.4.2 Security Attributes of DDS Plugin

The official security specifications for DDS [224] is provided by the Object Management Group (OMG), the organization that formalized the DDS specification. It outlines the security layout for DDS implementations and represents security-related components into a series of plugins that are explained below.

- **Cryptography-** Encryption, decryption, digests, message authentication rules, and public/private key transfers are all made possible by the plugin.
- **Access Control-** For participants who have verified their identity, this plugin manages and keeps track of access control rights for DDS subjects, domains, etc.



- Authentication- Using pre-established information, this plugin guarantees that the framework is authenticated. The intention is to prevent unauthenticated individuals from contaminating the data.
- Logger- All safety incidents, including unintended behavior, security policy breaches, and failures, can be logged with the help of this plugin.

## 6.5 Prototype Implementation Details

The purpose of the implementation is to access the suggested interface's adaptability and efficiency in multi-domain control plane situations. The OpenDaylight (ODL) and Open Network Operating System (ONOS) controllers are used to implement the SRDCP framework. Test cases cover ODL and ONOS based implementations in homogeneous and heterogeneous cluster of H2 based DCN, traffic evaluation of RTPS packet transfers, and performance assessment of ODL and ONOS controllers implemented using the SRDCP framework. The controllers synchronize their topologies using the LDP.

### 6.5.1 Execution Logic of SRDCP

Initially, the controllers directly connected with data plane routers or switches receive 64-bit Link Discovery Protocol (LDP) packets on a regular basis, enclosed in a Packet-out message. After reading the instructions contained in the packet-out message, the data plane switch or router forwards the data packet to a particular neighbor node. Afterwards, that neighbor sends that packet towards the controller by generating a Packet-In message. The controller uses its data to confirm as the relationship between the two peers is new, revised, or removed. The controller creates a link discovery upgrade list if there has been an alteration.

Two modules—the SRDCP Publisher and the Topology Manager under controller observe the updated lists by link discovery process. The Publisher notifies the DDS Plugin component inside the SRDCP framework of the changes made to the controller's topology instance by the Topology Manager. The update is transformed into a data sample by the DDS Plugin, which then uses the Data Writer to write it to the global data space. The

controller continuously scans the global data space for relevant instances of link discovery upgrades that peer controllers have published via the SRDCP Subscriber module. The DDS Plugin filters the incoming samples based on the QoS feature criteria and the structure of the link discovery process. The filtered samples are sent towards the controller's topology manager for updating the network topology when it matches.

### 6.5.2 Testbed Details

SRDCP is incorporated into two controllers OpenDaylight v19 (Potassium) and ONOS v2.7.0 (X-Wing), both of them are installed in two separate Ubuntu 20.04 VM instances with 8GB of physical memory each, 40GB of HDD each and NAT adapter. SDN environment is emulated through Mininet [225] in a separate Ubuntu 20.04 instance with similar specs. All VMs are hosted in a Windows 11 64 bit OS with 1TB of SSD, 24 GBs of RAM and Intel core i7 12<sup>th</sup> generation CPU having 4.9 GHz of clock speed. For network traffic analysis Wireshark [226] tool is used. The testbed details are summarized in table 6.1.

Table 6.1: Testbed Details

Item	Description
CPU	Intel® Core™ i7-12 <sup>th</sup> Gen Processor with 4.9GHz.
RAM	24 GB DDR4, 8GBs for each VM
HDD	40 GB for each VM, 1 TB for Host OS
Host OS	Windows 11 64 Bit
VMs	Ubuntu 20.04 LTS
NW Adapter	NAT and host based adapter
SDN Controllers	ODL v19 and ONOS v 2.7.0
SDN Emulator	Mininet
RTPS Implementor	DDS v 6.0.1

### 6.5.3 Performance Analysis Matrices

The following three metrics are measured under different circumstances which are described below-

- **Network Confluence Time (NCT):** The amount of time that passes following the publication of a link discovery updating sample into a given data space before the network merges into a single, coherent shape.
- **Topology Change Latency (TCL):** The time it takes for a link update packet to be received by the peer controller and change a network's overall topology is known as the Topology Change Latency (TCL).
- **RTPS Throughput(RTPut):** The quantity of RTPS packets created and sent in a second.

#### 6.5.4 Description of Experiments

Two sets of experiments are conducted and the performance of the above-mentioned three parameters are measured under different circumstances. The proposed H2 model of controller orientation is used for both cases. Firstly, the SRDCP architecture is tested in homogeneous DCN and then it has been tested on heterogeneous DCN. The performance of SRDCP is then compared with another similar type of existing solution named DSF [227].

##### 6.5.4.1 SRDCP on Homogeneous Networks

The intention of this experiment is to evaluate the performance of SRDCP on ODL and ONOS based homogeneous H2 networks. Observations of topology change latency (TCL) and network confluence time (NCT) following a link update event are captured as part of the performance assessment. In this experiment number of controller VMs are increased gradually till the capacity of the host machine. Both controllers are hosted in two virtual machines (VMs), while the Mininet emulator is housed in a third. Each remote controller is assigned four data plane switches by Mininet, along with a host device per switch in a tree topology. To synchronize network topology, ICMP ping tests are conducted across local domains. Total of 14 ODL controller instances are created successfully over two VMs with one controller at a time in each virtual machine is added. Total 14 repetitions are carried out in each configuration.

In case of ONOS, total 10 instances are created successfully over two VMs and similar type of tree topology is emulated by Mininet. Total 14 repetitions are considered in each configuration which successfully created a substantially large network for conducting the test in a hybrid hierarchical (H2) model. This configuration is compared with a similar topology with DSF which is Floodlight and ONOS based solution. Therefore, to maintain the similarity in testbed a separate VM is configured with FL which is used to simulate the test case with respect to ODL of the solution.

#### **6.5.4.2 SRDCP on Heterogeneous Networks**

In this set of experiments, we assume that a data exchange contract exists between two inter-domain SDN deployed DCNs. The controller of each and every SDN deployment may vary, and it is quite expected too. For the obvious simplistic reason and hardware limitation, we have limited the controller deployment to ODL and ONOS only. To synchronize network topology between diverse SDN controller domains, data plane elements use flow rule synchronization policy induced by the respective controller. To facilitate topology synchronization process, each controller platform implements the SRDCP interface. The experiment's design comprises of host devices allocated via a Mininet VM and controllers hosted on different virtual machines (VMs) with data forwarding capability. ODL and ONOS are the two SRDCP implemented controllers per platform, and there are fourteen repetitions of the experiments.

### **6.6 Comparative Analysis of Experimental Results**

This segment presents an analytical comparison of the above-mentioned three parameters i.e. NCT, TCL and RTPut under different test cases mentioned in section 6.5.4.1 and 6.5.4.2. In each analysis each of these three parameters are judged under four different circumstances- i.e. under homogeneous ODL cluster, under homogeneous ONOS cluster, under heterogenous ODL cluster and under heterogenous ONOS cluster. Similar types of parameters are considered for experimental evaluation in the case of DSF also to provide a comparative overview.

### 6.6.1 Analysis of Network Confluence Time (NCT)

The time it takes for networks to come together into a single, comprehensive topology following the publication of a link update packet by a controller is known as Network Confluence Time (NCT). In the evaluation of NCT, four different test cases are considered and NCT is measured in each test case against SRDCP and DSF to provide a comparative point of view.

#### *Case 1- NCT in Homogeneous ODL H2 Cluster*

The variation and comparison of NCT is shown in Figure 6.5. In figure 6.5 the number of ODL controller instance is considered along x axis which is 14 in this case and time is taken along y axis which ranges from 100 milliseconds to 1000 milliseconds with a gap of 100 milliseconds. The proposed solution SRDCP takes 0.32 seconds to converge in the first test case whereas DSF records 0.35 seconds in this first test. The value of NCT has been recorded for 14 different instances in both SRDCP and DSF where the proposed solution shows a better result i.e. lesser amount of time taken in each of the instances than DSF. Finally in the last test instance SRDCP records 0.59 second and DSF records 0.65 second, higher by 0.06 seconds. After the 8th iteration a steady increase of NCT in DSF implementation on homogeneous ODL is noticed. The average NCT noticed in SRDCP is 0.52 seconds and in DSF it is found as 0.56 which is 0.04 seconds higher than the proposed solution and thereby the proposed solution is obtaining 7.14% lesser value in NCT which might result a substantial gain in real life large scale homogeneous ODL cluster deployment.

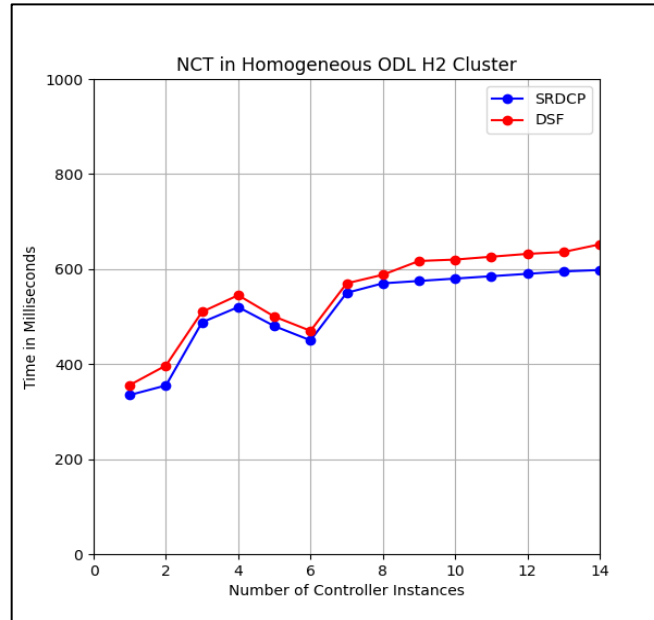


Figure 6.5 Analysis of NCT in Homogeneous ODL H2 Cluster

### ***Case 2- NCT in Homogeneous ONOS H2 Cluster***

In the case of homogeneous ONOS H2 cluster network 10 network instances are considered for testing the value of NCT for both SRDCP and DSF over 14 iterations. Along the x axis number of controller instances is considered which is 10 in this case and execution time is considered along the y axis which is 100 milliseconds to 1000 milliseconds with a gap of 1000 milliseconds. Figure 6.6 shows the findings of this analysis. In this study also SRDCP performs better than DSF by a substantial amount. As shown in Figure 6 SRDCP takes 0.33 seconds to converge in the first iteration and DSF takes 0.36 seconds gaining a difference of 0.03 seconds. This tradition is maintained till the 14<sup>th</sup> iteration where SRDCP takes 0.8 seconds and DSF takes 0.85 seconds, again gaining a difference of 0.05 seconds. The average NCT in SRDCP is therefore 0.58 seconds and in DSF it is 0.62 seconds which is 0.04 seconds higher than SRDCP and thus the proposed solution is gaining 6.9% of betterment in NCP value over DSF which might be considered a substantial among of gain in real time implementation of Homogeneous ONOS H2 cluster implementation.

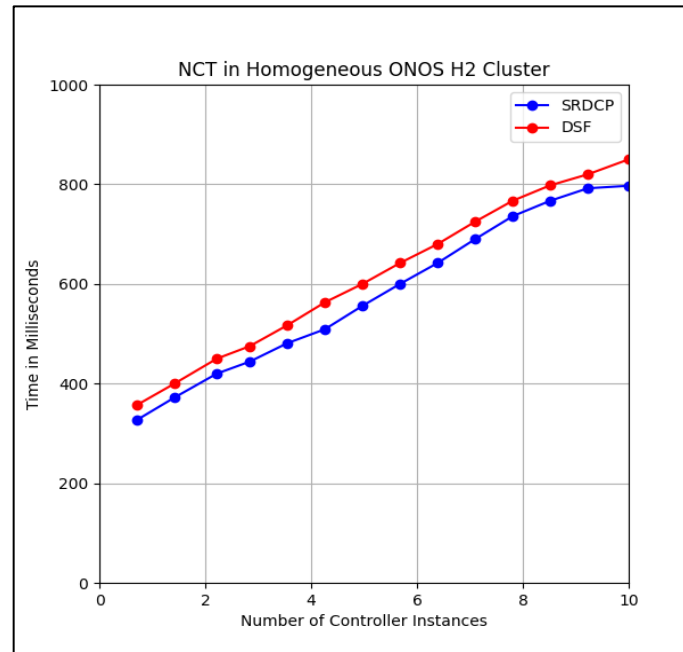


Figure 6.6 Analysis of NCT in Homogeneous ONOS H2 Cluster

### ***Case 3- NCT in In Heterogeneous ODL H2 Cluster***

In the next type of analysis, first SRDCP and DSF implementations in heterogeneous ODL cluster of DCN has been considered. The number of ODL controller instance is considered along x axis distributed in two separate clusters of controllers, having 7 instances in each cluster and time is taken along y axis which ranges from 100 milliseconds to 1000 milliseconds with a gap of 100 milliseconds. Fourteen repetitions are considered in both clusters with SRDCP and DSF. The experimental results are shown in figure 6.7. It has been observed that in this study SRDCP has outperformed DSF by a margin of 0.13 to 0.16 seconds on an average in each iteration. Both the implementations record a steady rise in NCT with the increment of controllers. In the first iteration SRDCP achieves 0.22 seconds of NCT whereas DSF achieves 0.3 seconds and in the 14<sup>th</sup> iteration the NCT value for SRDCP is 0.51 seconds and in DSF it is 0.65 seconds. The average NCT in SRDCP is 0.40 seconds and in DSF it is 0.53 seconds thereby SRDCP gains 24.5% of betterment in NCT value when compared in similar type of heterogeneous ODL based H2 cluster. Now,

this is a substantial percentage of betterment in NCT values when deployed in real time heterogeneous ODL based DCNs.

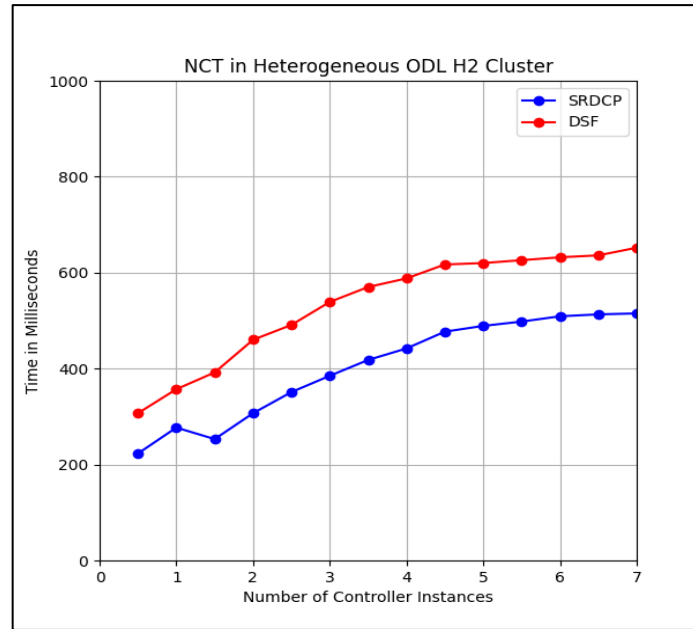


Figure 6.7 Analysis of NCT in Heterogeneous ODL H2 Cluster

#### ***Case 4- NCT in Heterogeneous ONOS H2 Cluster***

To test the NCT values in heterogeneous ONOS based H2 cluster, 10 instances of controllers are considered in two segments, resulting in 5 controllers in each segment. A total of 14 repetitions of the experiment are conducted by implementing SRDCP and DSF in the controller architecture. Figure 8 highlights the experimental achievements. The controller instances are plotted along x axis ranging between 1-5 and time is plotted along y axis ranging between 100 to 1000 milliseconds with a gap of 100 milliseconds. As seen in figure 6.8 the value of NCT in SRDCP in 1<sup>st</sup> run is 0.29 seconds and in DSF it is 0.36 seconds which again achieves 0.07 seconds increment. In SRDCP, NCT value decreases in the next two iterations but increases after that which stabilizes after the 9<sup>th</sup> iteration. In case of DSF, NCT value gains almost linear growth up to 6<sup>th</sup> iteration, then decreases slightly and almost maintains this trend up to 14<sup>th</sup> iteration achieving NCT of 0.57 seconds finally. The average NCT of SRDCP is 0.4 seconds whereas it is 0.51 seconds in DSF



which shows 0.11 seconds of gain in NCT value by SRDCP which is 21.5% lesser than DSF and impactful in real time implementation of heterogeneous ONOS based H2 cluster. Therefore, in the analysis of NCT values of SRDCP and DSF the proposed solution achieves better results in each of the four types of implementation scenarios.

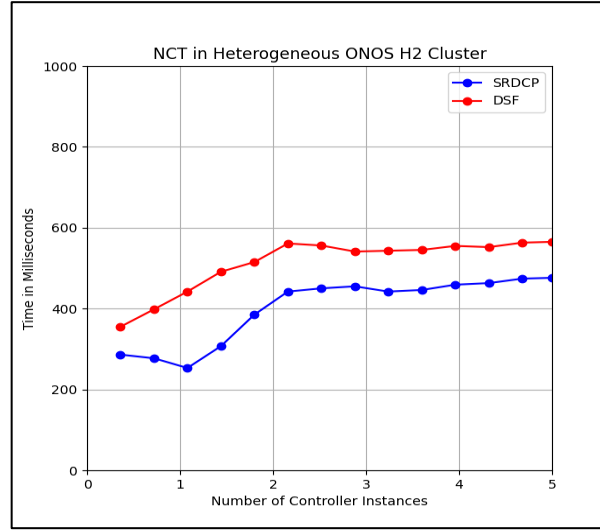


Figure 6.8 Analysis of NCT in Heterogeneous ONOS H2 Cluster

**Conclusion-** It is clear from the above mentioned four test cases that in all the cases of measurement of NCT, the proposed solution SRDCP outperforms DSF by a substantial range, hence a better and obvious choice of real-time large-scale implementation of SDN based data center networks.

#### 6.6.2 Analysis of Topology Change Latency (TCL)

The latency is the delay or time taken by a controller to receive a topology change event and act on that. In the analysis of TCL four test cases are considered again and in each test case TCL of SRDCP is compared the same of DSF to provide a comparative analysis. Figure 6.9-6.12 depict the graphical representation of all the test cases. In each diagram the controller instances are considered along x axis and time in milliseconds are considered along y axis.

##### *Case 1- TCL in Homogeneous ODL H2 Cluster*

As mentioned in section 6.5.4.1 the number of ODL controller instance is 14 which is considered along x axis in this case and time is taken along y axis which ranges from 0-

200 milliseconds with a gap of 25 milliseconds. A total of 14 repetitions in this test case are performed and the experimental results are depicted in figure 6.9.

In the first iteration of this test the latency observed in SRDCP is 36 milliseconds and in DSF it is 40 milliseconds. For both frameworks the latency increases with the increment of iterations but in the proposed solution the increment is more stable whereas in DSF it is bit aggressive and in all the iterations, SRDCP achieves either almost similar or lesser latency value as compared with DSF.

The average TCL in SRDCP noted as 49.36 milliseconds and in DSF it is 60.07 milliseconds, confirming that the proposed solution achieves 10.71 milliseconds lesser latency compared to DSF which is 21.7% lower than DSF and might prove impactful in real-time deployments.

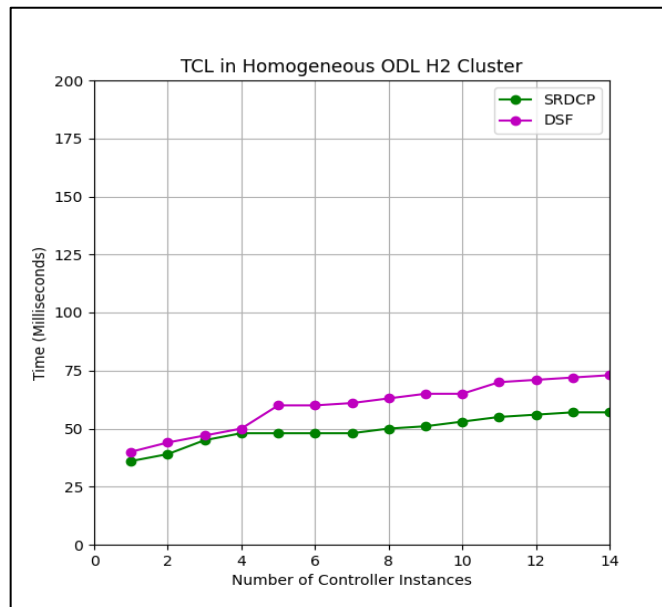


Figure 6.9 Analysis of TCL in Homogeneous ODL H2 Cluster

#### ***Case 2- TCL in Homogeneous ONOS H2 Cluster***

In the case of homogeneous ONOS H2 cluster network 10 network instances for testing the value of TCL for both SRDCP and DSF over 14 iterations are considered. Along the x axis number of controller instances is considered which is 10 in this case and execution time is considered along the y axis which is 0-200 milliseconds with a gap of 25

milliseconds. The experimental findings are depicted in figure 6.10. SRDCP records latency of 40 milliseconds and DSF records 43 milliseconds in the first run. Surprisingly in all other iterations except the last one DSF performed as well as SRDCP and records almost similar values and in 5<sup>th</sup> and 6<sup>th</sup> iterations DSF outperformed SRDCP! The average TCL in SRDCP is 44.21 and it is 45.12 in DSF, confirming the proposed solution has achieved 0.91 milliseconds lesser latency than DSF which is the increment of 2.05%. Therefore, TCL in homogeneous ONOS based implementation of both the frameworks are almost identical and they should perform almost identically in real-time implementations also.

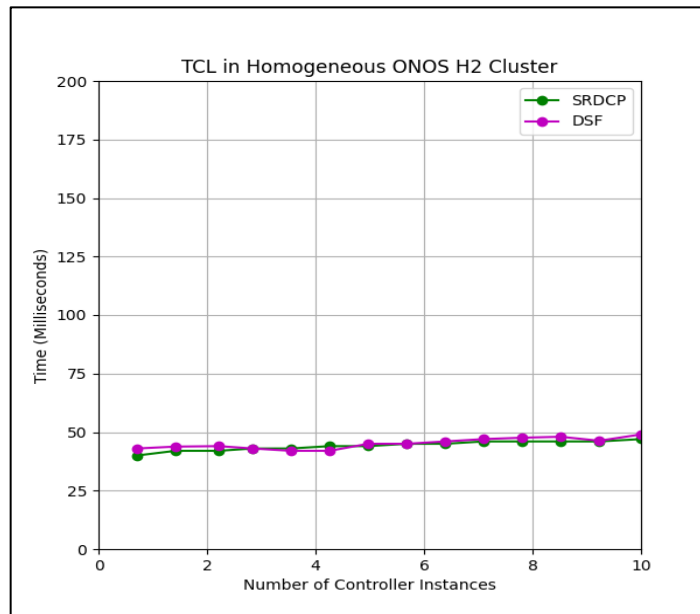


Figure 6.10 Analysis of TCL in Homogeneous ONOS H2 Cluster

### ***Case 3- TCL in Heterogeneous ODL H2 Cluster***

In the next experiment, SRDCP and DSF implementations in heterogeneous ODL cluster of DCN has been considered. The number of ODL controller instance is considered along x axis distributed in two separate clusters of controllers, having 7 instances in each cluster and time is taken along y axis which ranges from 0-200 milliseconds with a gap of 25 milliseconds. Fourteen repetitions are considered in both clusters with SRDCP and DSF. The experimental outcomes are shown in figure 11. In this test case SRDCP records

latency of 42.3 milliseconds where DSF records 47.2 milliseconds of latency which is 4.9 milliseconds higher than SRDCP. This trend continues till the 14<sup>th</sup> iteration and there SRDCP achieves 61 milliseconds and DSF achieves 66 milliseconds of latency. The average TCL in SRDCP is therefore 55.16 milliseconds and in DSF it is 60.51 milliseconds which is 5.35 milliseconds higher than SRDCP confirming 9.7% of hike in performance over DSF which is impactful in real-time and large-scale deployments.

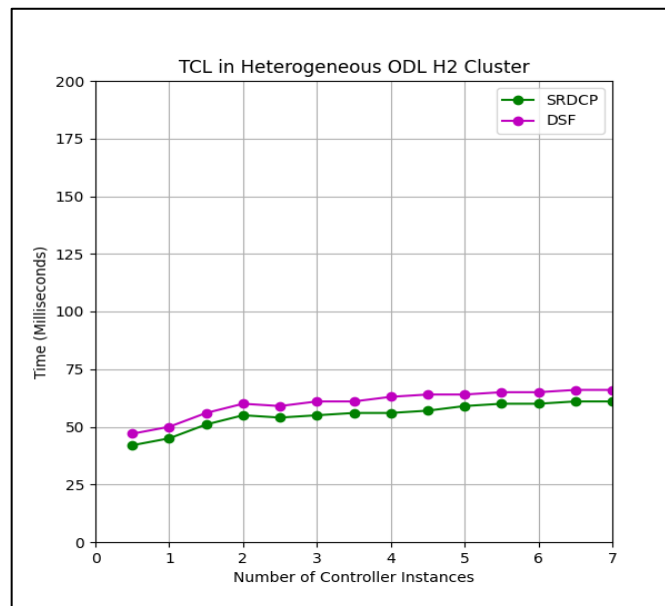


Figure 6.11 Analysis of TCL in Heterogeneous ODL H2 Cluster

#### ***Case 4- TCL in Heterogeneous ONOS H2 Cluster***

To test the TCL values in heterogeneous ONOS based H2 cluster, 10 instances of controllers are considered in two segments, resulting in 5 controllers in each segment. A total of 14 repetitions of the experiment are conducted by implementing SRDCP and DSF in the controller architecture. Figure 6.12 shows the experimental achievements. The controller instances are plotted along x axis ranging between 1-5 and time is plotted along y axis ranging between 0-200 milliseconds with a gap of 25 milliseconds. In the heterogeneous ONOS implementation both SRDCP and ONOS performed similarly again by obtaining TCL values 40.7 and 42.5 respectively. In the 7<sup>th</sup> and 8<sup>th</sup> iteration DSF outperformed SRDCP marginally again but thereafter SRDCP continues its lead towards

the end. The average TCL achieved by SRDCP and DSF is 40.18 and 43.4 milliseconds respectively which confirms the lead of 3.22 milliseconds again by SRDCP. This performance gain by SRDCP over DSF is 8.01% which is moderate in this case.

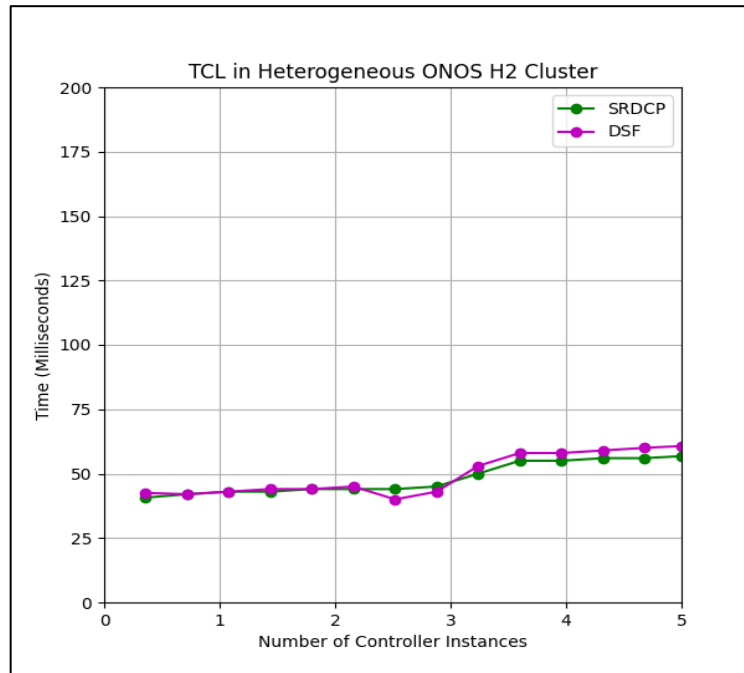


Figure 6.12 Analysis of TCL in Heterogeneous ONOS H2 Cluster

### Conclusion

The analysis of TCL in all four test cases reveals that SRDCP outperformed DSF in all the cases but in ONOS based implementations in both homogeneous and heterogeneous DSF achieved almost similar latency values. Therefore, DSF can be considered in ONOS based deployments whereas SRDCP is potent in all four types of deployments which makes the proposed solution SRDCP as a more impactful choice of real-life large-scale SDN controller deployment in data center networks.

### 6.6.3 Analysis of RTPS Throughput (RTPut)

Wireshark network traffic analyzer tool is used to trace the RTPS packet transfer among the different controller segments. In homogeneous ODL and ONOS clusters the statistics of RTPS packet transmission is analyzed up to 180 seconds and in heterogeneous ODL and ONOS clusters it is analyzed up to the same time duration. An increase in RTPS

packet volume in the network during initialization is noted, which is caused by controllers own previously existing data plane links. Following the initialization state, the RTPS throughput significantly decreased before stabilizing. The experiment's simulation of a link failure causes a link update, which causes RTPS throughput to rise once again before migrating to the single and comprehensive view.

At about 35 RTPS packets per second, the networks with the homogenous set of controllers achieve the maximum throughput value. Following the initialization state, there is a notable decrease in RTPS events, which then stabilize after 50 seconds. A little portion of the packets sent during the network's initialization procedure which takes almost 18 seconds. Before the network returns to steady-state, RTPS events occur for most of the packet activity during a link update event, which happens every 125 seconds or so.

In case of heterogeneous set of controllers, the RTPS throughput reaches almost 39 packets/second in the initialization state, stabilizes after and in case of any link failure event rises again to almost 46 packets/second and stabilizes further again. This trend of the RTPS throughput explores three main points of concern. Firstly, in both types of network configurations it is not stable either till its initialization. Secondly, after the topology converges into a single holistic view, the throughput remains almost constant and thirdly, in case of any link failure or network update event it rises again abruptly indicating the fact of network topology change and /or alteration in link status. This characteristic event is almost similar in all iterations of SRDCP implementations.

### **6.7 Security Threat Modelling of SRDCP Using STRIDE and Achievements**

The Secure and Resilient Distributed Control Plane (SRDCP) aims to improve East-West communication between these controllers using a Real-Time Publish/Subscribe (RTPS) model. Here, the security capability of SRDCP is analysed using the Microsoft STRIDE [228] threat model.

Using the STRIDE threat model, SRDCP shows significant potential in addressing key security concerns for large-scale distributed SDN deployments. It aims to tackle *Spoofing* by integrating robust authentication which help mitigate unauthorized access

and false updates. ***Tampering*** is addressed through data integrity checks and secure transmission protocols, although ongoing improvements are needed to enhance resilience against sophisticated attacks. The framework incorporates comprehensive logging and auditing to manage ***Repudiation***, but there is room for further refinement in traceability and accountability measures.

While SRDCP is designed to protect against ***Information disclosure*** with access controls, continual updates are essential to keep pace with emerging threats. The system's defences against ***Denial of Service (DoS)*** attacks, such as rate limiting and filtering, are effective but may require optimization to handle increasingly large-scale disruptions. Additionally, although SRDCP applies the principle of least privilege to combat ***Elevation of privilege***, further enhancements in access control mechanisms could strengthen its security posture. Overall, while SRDCP demonstrates a robust approach to securing distributed SDN environments, continuous improvements are necessary to ensure it remains more secure and effective compared to DSF.

## 6.8 Chapter Summary

This chapter discusses the challenges associated with distributed SDN (Software-Defined Networking) control planes, particularly in multi-domain and heterogeneous networks. One of the primary issues identified is the absence of a standardized communication protocol for east-west bound interfaces, which hampers effective communication between different controllers.

To address these challenges, the study introduces the Secure and Resilient Distributed Control Plane (SRDCP). SRDCP leverages a data-centric RTPS (Real-Time Publish-Subscribe) communication framework integrated with a DDS (Data Distribution Service) security plugin. The SRDCP framework is designed with a Hybrid Hierarchical (H2) controller arrangement that organizes control plane entities both vertically and horizontally. This configuration ensures synchronization of network topology and efficient packet routing across various network domains.

A key feature of SRDCP is its novel LDP (Link Discovery Protocol) message format, which facilitates the sharing of link discovery updates among the control plane entities. The effectiveness of the SRDCP framework was evaluated by implementing it on the ODL (OpenDaylight) and ONOS (Open Network Operating System) controllers and comparing its performance with the Distributed SDN Framework (DSF), a leading solution in the field.

The evaluation was conducted in both homogeneous and heterogeneous network environments, where controllers were required to share link updates to maintain an accurate and real-time view of the network topology. The results demonstrated that SRDCP performed reliably, especially in scenarios involving increased controller engagement, maintaining real-time coordination and synchronization of network topology across different domains.



## **Chapter 7**

### **Conclusion and Future Scope**

The research undertaken on the detection and mitigation of architectural vulnerabilities in Software-Defined Networking (SDN) highlights critical insights and contributions to enhancing the security and resilience of SDN environments. Through comprehensive analysis, several key vulnerabilities inherent in the SDN architecture have been identified, including the centralization of the control plane, potential for Denial of Service (DoS) attacks, and vulnerabilities in the communication channels between the control and data planes.

A multi-faceted approach was employed to address these vulnerabilities, integrating advanced detection mechanisms and robust mitigation strategies. The detection mechanisms leverage anomaly-based monitoring to identify potential threats in real-time, ensuring proactive security measures in all SDN layers and interfaces. Additionally, the research developed and validated mitigation techniques that include distributed control planes, secure communication protocols, and dynamic resource allocation to enhance fault tolerance and minimize attack surfaces.

The proposed solutions have been rigorously tested in simulated environments, demonstrating significant improvements in the security and efficiency of SDN deployments. These solutions not only mitigate existing vulnerabilities but also provide a scalable framework adaptable to evolving threats.

In conclusion, the research provides a substantial contribution to the field of SDN security, offering practical solutions that can be readily implemented in real-world scenarios. Future work will focus on refining these techniques and exploring their applicability in broader network environments, ensuring the continued evolution of secure and resilient SDN architectures.

## 7.1 Summary of the Research Contributions and Novelty

The major research contributions of this thesis can be summarized as follows:

- 1. Survey of Existing Literature to Define Research Objectives:** The literature review identifies critical security issues in Software-Defined Networking (SDN), categorizing threats and proposing solutions to enhance security across SDN's architecture. It emphasizes different vulnerabilities in the southbound interface, data plane, control plane, and northbound interface. The objectives and the problem statements of this thesis are coined from this survey.
- 2. MiTM Prevention and Compromised Switch Detection:** This chapter focuses on preventing Man-in-the-Middle (MiTM) attacks on the Southbound Interface (SBI) and detecting compromised switches in the data plane. New and efficient techniques like cryptographic methods and dynamic threshold based lightweight solutions are proposed to maintain network integrity and secure communications.
- 3. Proactive Detection of SYN Flood Attack:** The thesis addresses the detection and mitigation of SYN flood attacks on SDN controllers, using threshold-based mechanisms to monitor and adapt to traffic anomalies, ensuring network resilience against such attacks. The efficiency of the proposed solution is analyzed against existing solutions and found adequate to be installed as a controller module as a lightweight solution.
- 4. Application Trust Establishment in the SDN Application Plane:** The chapter introduces a trust management framework to secure northbound communications, involving authentication, authorization, and continuous trust evaluation to prevent unauthorized access and ensure controlled interactions within the SDN environment. A trust management module is proposed here which will authenticate and authorize any unknown applications before the trust establishment process.

### **5. Secure and Resilient Distributed Control Plane for Large-Scale SDN Deployments:**

The final chapter presents a secure control plane framework for large-scale distributed SDN deployments, focusing on synchronization, fault tolerance, and secure inter-controller communication in similar and diverse networks, with case studies demonstrating the effectiveness of the proposed architecture.

### **7.2 Future Research Directions in SDN Security**

This section outlines recent research trends and critical pathways for enhancing security in both traditional and distributed SDN architectures. The key research directions include:

1. **Advanced Encryption Techniques:** Develop sophisticated methods to secure communications between the SDN controller and devices.
2. **Real-Time Anomaly Detection:** Utilize machine learning for real-time monitoring to identify and mitigate unauthorized access and data manipulation.
3. **Robust Authentication Protocols:** Create stronger mechanisms to ensure only legitimate devices interact with the SDN controller.
4. **Controller Redundancy:** Improve redundancy strategies to prevent single points of failure in SDN controllers.
5. **Blockchain Integration:** Explore blockchain technology for decentralizing control and enhancing security.
6. **Intrusion Detection Systems:** Develop advanced systems to detect and respond to malicious activities targeting the control plane.
7. **API Security:** Strengthen security for northbound APIs to prevent exploitation and unauthorized access.
8. **Application Isolation:** Enhance techniques for isolating applications to protect overall network performance.

9. Dynamic Trust Management: Implement systems that continuously assess trust levels of applications interacting with the controller.
10. Fault Tolerance: Investigate techniques to ensure network reliability and performance despite controller failures.

## REFERENCES

- [1] A. Kanwal, M. Nizamuddin, W. Iqbal, W. Aman, Y. Abbas and S. Mussiraliyeva, "Exploring Security Dynamics in SDN Controller Architectures: Threat Landscape and Implications," in *IEEE Access*, vol. 12, pp. 56517-56553, 2024, doi: 10.1109/ACCESS.2024.3390968.
- [2] Jarschel, M., Oechsner, S., Schlosser, D., Pries, R., Goll, S. and Tran-Gia, P., 2011, September. Modeling and performance evaluation of an OpenFlow architecture. In *2011 23rd International Teletraffic Congress (ITC)* (pp. 1-7). IEEE.
- [3] Swapna, A.I., Huda, M.R. and Aion, M.K., 2016, December. Comparative security analysis of software defined wireless networking (SDWN)-BGP and NETCONF protocols. In *2016 19th International Conference on Computer and Information Technology (ICCIT)* (pp. 282-287). IEEE.
- [4] Bannour, F., Souihi, S. and Mellouk, A., 2017. Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1), pp.333-354.
- [5] Chowdhary, A., Dixit, V.H., Tiwari, N., Kyung, S., Huang, D. and Ahn, G.J., 2017, November. Science DMZ: SDN based secured cloud testbed. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (pp. 1-2). IEEE.
- [6] Nadeau, T. and Pan, P., 2011. Software driven networks problem statement. *Network Working Group Internet-Draft*, Sep, 30.
- [7] Shin, S.W., Porras, P., Yegneswaran, V. and Gu, G., 2013. A framework for integrating security services into software-defined networks. In *Open Networking Summit*. Open Networking Summit.
- [8] Mijumbi, R., Serrat, J., Gorricho, J.L., Bouten, N., De Turck, F. and Boutaba, R., 2015. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1), pp.236-262.
- [9] Scott-Hayward, S., Natarajan, S. and Sezer, S., 2015. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1), pp.623-654.
- [10] Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2014. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), pp.14-76.
- [11] Khan, M.A. and Salah, K., 2018. IoT security: Review, blockchain solutions, and open challenges. *Future generation computer systems*, 82, pp.395-411.
- [12] sJafarian, T., Masdari, M., Ghaffari, A. and Majidzadeh, K., 2021. A survey and classification of the security anomaly detection mechanisms in software defined networks. *Cluster Computing*, 24, pp.1235-1253.
- [13] Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2014. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), pp.14-76.

- 
- [14] Han, T., Jan, S.R.U., Tan, Z., Usman, M., Jan, M.A., Khan, R. and Xu, Y., 2020. A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers. *Concurrency and Computation: Practice and Experience*, 32(16), p.e5300.
- [15] Kreutz, D., Ramos, F.M. and Verissimo, P., 2013, August. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 55-60)
- [16] Lara, A., Kolasani, A. and Ramamurthy, B., 2013. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, 16(1), pp.493-512.
- [17] Hu, F., Hao, Q. and Bao, K., 2014. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, 16(4), pp.2181-2206.
- [18] Greenberg, A., Hjalmtysson, G., Maltz, D.A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J. and Zhang, H., 2005. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5), pp.41-54
- [19] Sherwood, R., Gibb, G., Yap, K.K., Appenzeller, G., Casado, M., McKeown, N. and Parulkar, G., 2009. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep, 1*, p.132.
- [20] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P. and Amidon, K., 2015. The design and implementation of open {vSwitch}. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)* (pp. 117-130).
- [21] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J., 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2), pp.69-74. LNCS Homepage, <http://www.springer.com/lncs>, last accessed 2016/11/21.
- [22] Shin, S.W., Porras, P., Yegneswara, V., Fong, M., Gu, G. and Tyson, M., 2013. Fresco: Modular composable security services for software-defined networks. In *20th annual network & distributed system security symposium*. Ndss.
- [23] Fayaz, S.K., Tobioka, Y., Sekar, V. and Bailey, M., 2015. Bohatei: Flexible and elastic {DDoS} defense. In *24th USENIX security symposium (USENIX Security 15)* (pp. 817-832).
- [24] Wang, H., Yang, G., Chinprutthiwong, P., Xu, L., Zhang, Y. and Gu, G., 2018, October. 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* (pp. 3-16).
- [25] Qazi, Z.A., Tu, C.C., Chiang, L., Miao, R., Sekar, V. and Yu, M., 2013, August. SIMPLE-fying middlebox policy enforcement using SDN. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (pp. 27-38).
- [26] Fayazbakhsh, S.K., Sekar, V., Yu, M. and Mogul, J.C., 2013, August. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions.

- In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 19-24).
- [27] Shin, S. and Gu, G., 2013, August. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 165-166).
- [28] Wang, H., Xu, L. and Gu, G., 2015, June. Floodguard: A dos attack prevention extension in software-defined networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 239-250). IEEE.
- [29] Shin, S., Yegneswaran, V., Porras, P. and Gu, G., 2013, November. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 413-424).
- [30] Chaves, L.J., Garcia, I.C. and Madeira, E.R.M., 2016, June. Ofswitch13: Enhancing ns-3 with openflow 1.3 support. In *Proceedings of the 2016 Workshop on ns-3* (pp. 33-40).
- [31] Canini, M., De Cicco, D., Kuznetsov, P., Levin, D., Schmid, S. and Vissicchio, S., 2014. STN: A robust and distributed SDN control plane.
- [32] Yoon, C., Lee, S., Kang, H., Park, T., Shin, S., Yegneswaran, V., Porras, P. and Gu, G., 2017. Flow wars: Systemizing the attack surface and defenses in software-defined networks. *IEEE/ACM Transactions on Networking*, 25(6), pp.3514-3530.
- [33] Cheung, S., Fong, M., Porras, P., Skinner, K. and Yegneswaran, V., 2015, February. Securing the software-defined network control layer. In *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS)*.
- [34] Cao, J., Xie, R., Sun, K., Li, Q., Gu, G. and Xu, M., 2020, January. When match fields do not need to match: Buffered packets hijacking in SDN. In *Proc. of the Network and Distributed System Security Symposium (NDSS'20)*.
- [35] May, R., El-Hassany, A., Vanbever, L. and Vechev, M., 2017, April. BigBug: Practical concurrency analysis for SDN. In *Proceedings of the Symposium on SDN Research* (pp. 88-94).
- [36] Miserez, J., Bielik, P., El-Hassany, A., Vanbever, L. and Vechev, M., 2015, June. SDNRacer: Detecting concurrency violations in software-defined networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (pp. 1-7).
- [37] Bu, K., Yang, Y., Guo, Z., Yang, Y., Li, X. and Zhang, S., 2018, April. Flowcloak: Defeating middlebox-bypass attacks in software-defined networking. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (pp. 396-404). IEEE.
- [38] Khurshid, A., Zhou, W., Caesar, M. and Godfrey, P.B., 2012, August. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 49-54).
- [39] Röpke, C. and Holz, T., 2018, August. Preventing malicious SDN applications from hiding adverse network manipulations. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges* (pp. 40-45).

- 
- [40] Kuzniar, M., Peresini, P., Canini, M., Venzano, D. and Kostic, D., 2012, December. A soft way for openflow switch interoperability testing. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies* (pp. 265-276).
  - [41] Kamisiński, A. and Fung, C., 2015, October. Flowmon: Detecting malicious switches in software-defined networks. In *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense* (pp. 39-45).
  - [42] Chi, P.W., Kuo, C.T., Guo, J.W. and Lei, C.L., 2015, April. How to detect a compromised SDN switch. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)* (pp. 1-6). IEEE.
  - [43] Mohan, P.M., Truong-Huu, T. and Gurusamy, M., 2018, January. Towards resilient in-band control path routing with malicious switch detection in SDN. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)* (pp. 9-16). IEEE.
  - [44] Agborubere, B. and Sanchez-Velazquez, E., 2017, June. Openflow communications and tls security in software-defined networks. In *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCoM) and IEEE Smart Data (SmartData)* (pp. 560-566). IEEE.
  - [45] Hori, Y., Mizoguchi, S., Miyazaki, R., Yamada, A., Feng, Y., Kubota, A. and Sakurai, K., 2017. A comprehensive security analysis checklist for OpenFlow networks. In *Advances on Broad-Band Wireless Computing, Communication and Applications: Proceedings of the 11th International Conference On Broad-Band Wireless Computing, Communication and Applications (BWCCA-2016) November 5-7, 2016, Korea* (pp. 231-242). Springer International Publishing.
  - [46] Sebbar, A., Zkik, K., Boulmalf, M. and El Kettani, M.D.E.C., 2019. New context-based node acceptance CBNA framework for MitM detection in SDN Architecture. *Procedia Computer Science*, 160, pp.825-830.
  - [47] Sebbar, A., Zkik, K., Baddi, Y., Boulmalf, M. and Kettani, M.D.E.C.E., 2020. MitM detection and defense mechanism CBNA-RF based on machine learning for large-scale SDN context. *Journal of Ambient Intelligence and Humanized Computing*, 11(12), pp.5875-5894.
  - [48] Gonzaga, R. and Sampaio, P.N.M., 2020, January. Mitigating man in the middle attacks within context-based sdn. In *8th international workshop on ADVANCES in ICT infrastructures and services (ADVANCE 2020)* (pp. 1-8).
  - [49] Sebbar, A., Boulmalf, M., El Kettani, M.D.E.C. and Baddi, Y., 2018, October. Detection MITM attack in multi-SDN controller. In *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)* (pp. 583-587). IEEE.
  - [50] Masoud, M.Z., Jaradat, Y. and Jannoud, I., 2015, November. On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm. In *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)* (pp. 1-5). IEEE.



- 
- [51] Singh, S., Khan, R.A. and Agrawal, A., 2015, May. Prevention mechanism for infrastructure based denial-of-service attack over software defined network. In *International Conference on Computing, Communication & Automation* (pp. 348-353). IEEE.
  - [52] Erturk, E. and Kumar, M., 2018. New use cases for snort: Cloud and mobile environments. *arXiv preprint arXiv:1802.02359*.
  - [53] Xing, T., Xiong, Z., Huang, D. and Medhi, D., 2014, November. SDNIPS: Enabling software-defined networking based intrusion prevention system in clouds. In *10th international conference on network and service management (CNSM) and workshop* (pp. 308-311). IEEE.
  - [54] Haeberlen, A., Kouznetsov, P. and Druschel, P., 2007. PeerReview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review*, 41(6), pp.175-188.
  - [55] Zhou, W., Fei, Q., Narayan, A., Haeberlen, A., Loo, B.T. and Sherr, M., 2011, October. Secure network provenance. In *Proceedings of the twenty-third ACM symposium on operating systems principles* (pp. 295-310).
  - [56] Al-Shaer, E. and Al-Haj, S., 2010, October. FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration* (pp. 37-44).
  - [57] Son, S., Shin, S., Yegneswaran, V., Porras, P. and Gu, G., 2013, June. Model checking invariant security properties in OpenFlow. In *2013 IEEE international conference on communications (ICC)* (pp. 1974-1979). IEEE.
  - [58] Kazemian, P., Chang, M., Zeng, H., Varghese, G., McKeown, N. and Whyte, S., 2013. Real time network policy checking using header space analysis. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)* (pp. 99-111).
  - [59] Kazemian, P., Varghese, G. and McKeown, N., 2012. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (pp. 113-126).
  - [60] Dhawan, M., Poddar, R., Mahajan, K. and Mann, V., 2015, February. Sphinx: detecting security attacks in software-defined networks. In *Ndss* (Vol. 15, pp. 8-11).
  - [61] Khan, S., Gani, A., Wahab, A.W.A., Abdelaziz, A. and Bagiwa, M.A., 2016, January. FML: A novel forensics management layer for software defined networks. In *2016 6th international conference-cloud system and big data engineering (confluence)* (pp. 619-623). IEEE.
  - [62] Khan, S., Gani, A., Wahab, A.W.A., Abdelaziz, A., Ko, K., Khan, M.K. and Guizani, M., 2016. Software-defined network forensics: Motivation, potential locations, requirements, and challenges. *IEEE Network*, 30(6), pp.6-13.
  - [63] Chao, T.W., Ke, Y.M., Chen, B.H., Chen, J.L., Hsieh, C.J., Lee, S.C. and Hsiao, H.C., 2016, June. Securing data planes in software-defined networks. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)* (pp. 465-470). IEEE.

- 
- [64] Ghannam, R. and Chung, A., 2016, April. Handling malicious switches in software defined networks. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (pp. 1245-1248). IEEE.
  - [65] Teng, S., Wu, N., Zhu, H., Teng, L. and Zhang, W., 2017. SVM-DT-based adaptive and collaborative intrusion detection. *IEEE/CAA Journal of Automatica Sinica*, 5(1), pp.108-118.
  - [66] Duan, Y., Li, W., Fu, X., Luo, Y. and Yang, L., 2017. A methodology for reliability of WSN based on software defined network in adaptive industrial environment. *IEEE/CAA Journal of Automatica Sinica*, 5(1), pp.74-82.
  - [67] Shaghaghi, A., Kaafar, M.A., Buyya, R. and Jha, S., 2020. Software-defined network (SDN) data plane security: issues, solutions, and future directions. *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*, pp.341-387.
  - [68] Ali, S.T., Sivaraman, V., Radford, A. and Jha, S., 2015. A survey of securing networks using software defined networking. *IEEE transactions on reliability*, 64(3), pp.1086-1097.
  - [69] Yu, W., Fu, X., Graham, S., Xuan, D. and Zhao, W., 2007, May. DSSS-based flow marking technique for invisible traceback. In *2007 IEEE Symposium on Security and Privacy (SP'07)* (pp. 18-32). IEEE.
  - [70] Gao, S., Li, Z., Xiao, B. and Wei, G., 2018. Security threats in the data plane of software-defined networks. *IEEE network*, 32(4), pp.108-113.
  - [71] Shaghaghi, A., Kaafar, M.A. and Jha, S., 2017, April. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (pp. 849-861).
  - [72] Yang, H., Zhang, J., Zhao, Y., Li, H., Huang, S., Ji, Y., Han, J., Lin, Y. and Lee, Y., 2014. Cross stratum resilience for OpenFlow-enabled data center interconnection with Flexi-Grid optical networks. *Optical Switching and Networking*, 11, pp.72-82.
  - [73] Vaughan-Nichols, S.J., 2011. OpenFlow: The next generation of the network?. *Computer*, 44(08), pp.13-15.
  - [74] Decusatis, C.J.S., Carranza, A. and DeCusatis, C.M., 2012. Communication within clouds: open standards and proprietary protocols for data center networking. *IEEE Communications Magazine*, 50(9), pp.26-33.
  - [75] Kuipers, F.A., 2012. An overview of algorithms for network survivability. *International Scholarly Research Notices*, 2012(1), p.932456.
  - [76] Giorgetti, A., Valcarengi, L., Cugini, F. and Castoldi, P., 2010, May. PCE-based dynamic restoration in wavelength switched optical networks. In *2010 IEEE International Conference on Communications* (pp. 1-6). IEEE.
  - [77] Wang, J., Sahasrabuddhe, L. and Mukherjee, B., 2002. Path vs. subpath vs. link restoration for fault management in IP-over-WDM networks: Performance comparisons using GMPLS control signaling. *IEEE Communications Magazine*, 40(11), pp.80-87.

- 
- [78] Metz, C., 2000. IP protection and restoration. *IEEE Internet Computing*, 4(2), pp.97-102.
  - [79] Valcarenghi, L., Cugini, F., Paolucci, F. and Castoldi, P., 2008. Quality-of-service-aware fault tolerance for grid-enabled applications. *Optical Switching and Networking*, 5(2-3), pp.150-158.
  - [80] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M. and Zolla, J., 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4), pp.3-14.
  - [81] Liu, L., Tsuritani, T., Casellas, R., Morita, I., Martínez, R. and Muñoz, R., 2013. Design of a GMPLS control plane with PCE-based impairment-aware full restoration capability for translucent WSON: Enabling techniques, service demonstration, and performance evaluation. *Optical Switching and Networking*, 10(1), pp.16-31.
  - [82] Golab, W. and Boutaba, R., 2008. Path selection in user-controlled circuit-switched optical networks. *Optical Switching and Networking*, 5(2-3), pp.123-138.
  - [83] Tapolcai, J., Ho, P.H., Babarczi, P., Rónyai, L., Tapolcai, J., Ho, P.H., Babarczi, P. and Rónyai, L., 2015. Failure restoration approaches. *Internet Optical Infrastructure: Issues on Monitoring and Failure Restoration*, pp.15-31.
  - [84] Guo, Q., Ho, P.H., Yu, H.F., Tapolcai, J. and Mouftah, H.T., 2010. Spare capacity reprovisioning for high availability shared backup path protection connections. *Computer communications*, 33(5), pp.603-611.
  - [85] Shah-Heydari, S. and Yang, O., 2007, September. Performance study of multiple link failure restorability of shared protection trees. In *2007 Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS'07)* (pp. 594-600). IEEE.
  - [86] Sinha, R.K., Ergun, F., Oikonomou, K.N. and Ramakrishnan, K.K., 2014, April. Network design for tolerating multiple link failures using Fast Re-route (FRR). In *2014 10th International Conference on the Design of Reliable Communication Networks (DRCN)* (pp. 1-8). IEEE.
  - [87] Habib, M.F., Tornatore, M., Dikbiyik, F. and Mukherjee, B., 2013. Disaster survivability in optical communication networks. *Computer Communications*, 36(6), pp.630-644.
  - [88] Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R. and Moore, A.W., 2012. OFLOPS: An open framework for OpenFlow switch evaluation. In *Passive and Active Measurement: 13th International Conference, PAM 2012, Vienna, Austria, March 12-14th, 2012. Proceedings 13* (pp. 85-95). Springer Berlin Heidelberg.
  - [89] Staessens, D., Sharma, S., Colle, D., Pickavet, M. and Demeester, P., 2011, October. Software defined networking: Meeting carrier grade requirements. In *2011 18th IEEE workshop on local & metropolitan area networks (LANMAN)* (pp. 1-6). IEEE.
  - [90] Sharma, S., Staessens, D., Colle, D., Pickavet, M. and Demeester, P., 2011, October. Enabling fast failure recovery in OpenFlow networks. In *2011 8th*

- International Workshop on the Design of Reliable Communication Networks (DRCN)* (pp. 164-171). IEEE.
- [91] Shang, G., Zhe, P., Bin, X., Aiqun, H. and Kui, R., 2017, May. FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (pp. 1-9). IEEE.
  - [92] Zhang, M., Li, G., Xu, L., Bi, J., Gu, G. and Bai, J., 2018. Control plane reflection attacks in SDNs: New attacks and countermeasures. In *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21* (pp. 161-183). Springer International Publishing.
  - [93] Hong, S., Xu, L., Wang, H. and Gu, G., 2015, February. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Ndss* (Vol. 15, pp. 8-11).
  - [94] Skowyra, R., Xu, L., Gu, G., Dedhia, V., Hobson, T., Okhravi, H. and Landry, J., 2018, June. Effective topology tampering attacks and defenses in software-defined networks. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 374-385). IEEE.
  - [95] Jero, S., Koch, W., Skowyra, R., Okhravi, H., Nita-Rotaru, C. and Bigelow, D., 2017. Identifier binding attacks and defenses in {Software-Defined} networks. In *26th USENIX Security Symposium (USENIX Security 17)* (pp. 415-432).
  - [96] Bianco, A., Giaccone, P., Mashayekhi, R., Ullio, M. and Vercellone, V., 2017. Scalability of ONOS reactive forwarding applications in ISP networks. *Computer Communications*, 102, pp.130-138.
  - [97] Lee, S., Yoon, C. and Shin, S., 2016, March. The smaller, the shrewder: A simple malicious application can kill an entire sdn environment. In *Proceedings of the 2016 ACM international workshop on security in software defined networks & network function virtualization* (pp. 23-28).
  - [98] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V. and Smeliansky, R., 2013, October. Advanced study of SDN/OpenFlow controllers. In *Proceedings of the 9th central & eastern european software engineering conference in russia* (pp. 1-6).
  - [99] Jero, S., Bu, X., Nita-Rotaru, C., Okhravi, H., Skowyra, R. and Fahmy, S., 2017. 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* (pp. 311-333). Springer International Publishing.
  - [100] Ujcich, B.E., Thakore, U. and Sanders, W.H., 2017, June. Attain: An attack injection framework for software-defined networking. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 567-578). IEEE.
  - [101] Yu, M., Rexford, J., Freedman, M.J. and Wang, J., 2010. Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computer Communication Review*, 40(4), pp.351-362.

- 
- [102] Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P. and Banerjee, S., 2011, August. DevoFlow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference* (pp. 254-265).
- [103] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T. and Shenker, S., 2010. Onix: A distributed control platform for large-scale production networks. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*.
- [104] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W. and Parulkar, G., 2014, August. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking* (pp. 1-6).
- [105] Medved, J., Varga, R., Tkacik, A. and Gray, K., 2014, June. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE international symposium on a world of wireless, mobile and multimedia networks 2014* (pp. 1-6). IEEE.
- [106] Katta, N., Zhang, H., Freedman, M. and Rexford, J., 2015, June. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research* (pp. 1-12).
- [107] Chandrasekaran, B., Tschaen, B. and Benson, T., 2016, March. Isolating and tolerating SDN application failures with LegoSDN. In *Proceedings of the Symposium on SDN Research* (pp. 1-12).
- [108] Gomez, S.R., Jero, S., Skowrya, R., Martin, J., Sullivan, P., Bigelow, D., Ellenbogen, Z., Ward, B.C., Okhravi, H. and Landry, J.W., 2019, June. Controller-oblivious dynamic access control in software-defined networks. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 447-459). IEEE.
- [109] Lee, S., Yoon, C., Lee, C., Shin, S., Yegneswaran, V. and Porras, P.A., 2017, February. Delta: A security assessment framework for software-defined networks. In *NDSS*.
- [110] Şahin, M.E. and Demirci, M., 2023. ConPoolUBF: Connection pooling and updatable Bloom filter based SYN flood defense in programmable data planes. *Computer Networks*, 231, p.109802.
- [111] Mohammadi, R., Javidan, R. and Conti, M., 2017. Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, 14(2), pp.487-497.
- [112] Huang, Y.F., Chang, S.W., Lin, C.B., Chen, C.J., Li, S.J. and Chen, C.M., 2022, December. An Improved Light Weight Countermeasure Scheme to Efficiently Mitigate TCP Attacks in SDN. In *International Computer Symposium* (pp. 501-511). Singapore: Springer Nature Singapore.
- [113] Ravi, N., Shalinie, S.M., Lal, C. and Conti, M., 2020. Aegis: Detection and mitigation of tcp syn flood on sdn controller. *IEEE Transactions on Network and Service Management*, 18(1), pp.745-759.

- 
- [114] Li, Z., Xing, W., Khamaiseh, S. and Xu, D., 2019. Detecting saturation attacks based on self-similarity of OpenFlow traffic. *IEEE Transactions on Network and Service Management*, 17(1), pp.607-621.
- [115] Mohammadi, R., Conti, M., Lal, C. and Kulhari, S.C., 2019. SYN-Guard: An effective counter for SYN flooding attack in software-defined networking. *International Journal of Communication Systems*, 32(17), p.e4061.
- [116] Kim, D., Dinh, P.T., Noh, S., Yi, J. and Park, M., 2019, October. An effective defense against SYN flooding attack in SDN. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 369-371). IEEE.
- [117] Oo, N.H. and Maw, A.H., 2019, September. Effective detection and mitigation of SYN flooding attack in SDN. In *2019 19th International Symposium on Communications and Information Technologies (ISCIT)* (pp. 300-305). IEEE.
- [118] Ubale, T. and Jain, A.K., 2018, March. SRL: An TCP SYN FLOOD DDoS mitigation approach in software-defined networks. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)* (pp. 956-962). IEEE.
- [119] Kumar, P., Tripathi, M., Nehra, A., Conti, M. and Lal, C., 2018. SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN. *IEEE Transactions on Network and Service Management*, 15(4), pp.1545-1559.
- [120] Shin, S., Song, Y., Lee, T., Lee, S., Chung, J., Porras, P., Yegneswaran, V., Noh, J. and Kang, B.B., 2014, November. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security* (pp. 78-89).
- [121] Röpke, C. and Holz, T., 2015. Sdn rootkits: Subverting network operating systems of software-defined networks. In *Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan, November 2-4, 2015. Proceedings 18* (pp. 339-356). Springer International Publishing.
- [122] Ujcich, B.E., Jero, S., Skowrya, R., Gomez, S.R., Bates, A., Sanders, W.H. and Okhravi, H., 2020, January. Automated discovery of cross-plane event-based vulnerabilities in software-defined networking. In *Network and Distributed System Security Symposium*.
- [123] Xu, L., Huang, J., Hong, S., Zhang, J. and Gu, G., 2017. Attacking the brain: Races in the {SDN} control plane. In *26th USENIX Security Symposium (USENIX Security 17)* (pp. 451-468).
- [124] Xiao, F., Zhang, J., Huang, J., Gu, G., Wu, D. and Liu, P., 2020, May. Unexpected data dependency creation and chaining: A new attack to SDN. In *2020 IEEE symposium on security and privacy (SP)* (pp. 1512-1526). IEEE.
- [125] Ujcich, B.E., Jero, S., Edmundson, A., Wang, Q., Skowrya, R., Landry, J., Bates, A., Sanders, W.H., Nita-Rotaru, C. and Okhravi, H., 2018, October. Cross-app poisoning in software-defined networking. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security* (pp. 648-663).

- 
- [126] Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M. and Gu, G., 2012, August. A security enforcement kernel for OpenFlow networks. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 121-126).
  - [127] Yoon, C., Shin, S., Porras, P., Yegneswaran, V., Kang, H., Fong, M., O'Connor, B. and Vachuska, T., 2017, December. A security-mode for carrier-grade sdn controllers. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (pp. 461-473).
  - [128] Nam, J., Jo, H., Kim, Y., Porras, P., Yegneswaran, V. and Shin, S., 2018, April. Barista: An event-centric nos composition framework for software-defined networks. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (pp. 980-988). IEEE.
  - [129] Yoon, C., Shin, S., Porras, P., Yegneswaran, V., Kang, H., Fong, M., O'Connor, B. and Vachuska, T., 2017, December. A security-mode for carrier-grade sdn controllers. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (pp. 461-473).
  - [130] Wen, X., Yang, B., Chen, Y., Hu, C., Wang, Y., Liu, B. and Chen, X., 2016, June. Sdnshield: Reconciliating configurable application permissions for sdn app markets. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)* (pp. 121-132). IEEE.
  - [131] Kang, H., Shin, S., Yegneswaran, V., Ghosh, S. and Porras, P., 2018, August. Aegis: An automated permission generation and verification system for sdns. In *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges* (pp. 20-26).
  - [132] Wundsam, A., Levin, D., Seetharaman, S. and Feldmann, A., 2011. {OFRewind}: Enabling Record and Replay Troubleshooting for Networks. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*.
  - [133] Scott, C., Wundsam, A., Raghavan, B., Panda, A., Or, A., Lai, J., Huang, E., Liu, Z., El-Hassany, A., Whitlock, S. and Acharya, H.B., 2014, August. Troubleshooting blackbox SDN control software with minimal causal sequences. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (pp. 395-406).
  - [134] El-Hassany, A., Miserez, J., Bielik, P., Vanbever, L. and Vechev, M., 2016, June. SDNRacer: concurrency analysis for software-defined networks. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 402-415).
  - [135] Dwaraki, A., Seetharaman, S., Natarajan, S. and Wolf, T., 2015, June. GitFlow: Flow revision management for software-defined networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (pp. 1-6).
  - [136] Ujcich, B.E., Jero, S., Skowyra, R., Bates, A., Sanders, W.H. and Okhravi, H., 2021. Causal Analysis for {Software-Defined} Networking Attacks. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 3183-3200).
  - [137] Lee, C., Yoon, C., Shin, S. and Cha, S.K., 2018, September. INDAGO: A new framework for detecting malicious SDN applications. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)* (pp. 220-230). IEEE.

- 
- [138] Ball, T., Bjørner, N., Gember, A., Itzhaky, S., Karbyshev, A., Sagiv, M., Schapira, M. and Valadarsky, A., 2014, June. Vericon: towards verifying controller programs in software-defined networks. In *Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation* (pp. 282-293).
- [139] Canini, M., Venzano, D., Perešini, P., Kostić, D. and Rexford, J., 2012. A {NICE} way to test {OpenFlow} applications. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (pp. 127-140).
- [140] Maleh, Y., Qasmaoui, Y., El Gholami, K., Sadqi, Y. and Mounir, S., 2023. A comprehensive survey on SDN security: threats, mitigations, and future directions. *Journal of Reliable Intelligent Environments*, 9(2), pp.201-239.
- [141] Ferguson, A.D., Guha, A., Liang, C., Fonseca, R. and Krishnamurthi, S., 2013. Participatory networking: An API for application control of SDNs. *ACM SIGCOMM computer communication review*, 43(4), pp.327-338.
- [142] Wen, X., Chen, Y., Hu, C., Shi, C. and Wang, Y., 2013, August. Towards a secure controller platform for openflow applications. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 171-172).
- [143] Scott-Hayward, S., Kane, C. and Sezer, S., 2014, October. Operationcheckpoint: Sdn application control. In *2014 IEEE 22nd International Conference on Network Protocols* (pp. 618-623). IEEE.
- [144] Cui, H., Chen, Z., Yu, L., Xie, K. and Xia, Z., 2017, December. Authentication mechanism for network applications in SDN environments. In *2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC)* (pp. 1-5). IEEE.
- [145] Porras, P.A., Cheung, S., Fong, M.W., Skinner, K. and Yegneswaran, V., 2015, February. Securing the software defined network control layer. In *Ndss*.
- [146] Aliyu, A.L., Bull, P. and Abdallah, A., 2017, March. A trust management framework for network applications within an SDN environment. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (pp. 93-98). IEEE.
- [147] Chandrasekaran, B. and Benson, T., 2014, October. Tolerating SDN application failures with LegoSDN. In *Proceedings of the 13th ACM workshop on hot topics in networks* (pp. 1-7).
- [148] Betgé-Brezetz, S., Kamga, G.B. and Tazi, M., 2015, April. Trust support for SDN controllers and virtualized network applications. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)* (pp. 1-5). IEEE.
- [149] Yan, Z., Zhang, P. and Vasilakos, A.V., 2016. A security and trust framework for virtualized networks and software-defined networking. *Security and communication networks*, 9(16), pp.3059-3069.
- [150] Azodolmolky, S., Wieder, P. and Yahyapour, R., 2013, October. Performance evaluation of a scalable software-defined networking deployment. In *2013 Second European Workshop on Software Defined Networks* (pp. 68-74). IEEE.



- 
- [151] Benson, T., Akella, A. and Maltz, D.A., 2010, November. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (pp. 267-280).
- [152] Yeganeh, S.H., Tootoonchian, A. and Ganjali, Y., 2013. On scalability of software-defined networking. *IEEE Communications Magazine*, 51(2), pp.136-141.
- [153] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N. and Shenker, S., 2008. NOX: towards an operating system for networks. *ACM SIGCOMM computer communication review*, 38(3), pp.105-110.
- [154] Erickson, D., 2013, August. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 13-18).
- [155] Bholebawa, I.Z. and Dalal, U.D., 2018. Performance analysis of SDN/OpenFlow controllers: POX versus floodlight. *Wireless Personal Communications*, 98, pp.1679-1699.
- [156] Ganjali, Y. and Tootoonchian, A., 2010. {HyperFlow}: A Distributed Control Plane for {OpenFlow}. In *2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN 10)*.
- [157] Katta, N., Zhang, H., Freedman, M. and Rexford, J., 2015, June. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research* (pp. 1-12).
- [158] Yeganeh, S.H. and Ganjali, Y., 2016, March. Beehive: Simple distributed programming in software-defined networks. In *Proceedings of the Symposium on SDN Research* (pp. 1-12).
- [159] Spalla, E.S., Mafioletti, D.R., Liberato, A.B., Ewald, G., Rothenberg, C.E., Camargos, L., Villaca, R.S. and Martinello, M., 2016, April. AR2C2: Actively replicated controllers for SDN resilient control plane. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (pp. 189-196). IEEE.
- [160] Liu, Y., Hecker, A., Guerzoni, R., Despotovic, Z. and Beker, S., 2015, June. On optimal hierarchical SDN. In *2015 IEEE international conference on communications (ICC)* (pp. 5374-5379). IEEE.
- [161] Hassas Yeganeh, S. and Ganjali, Y., 2012, August. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 19-24).
- [162] Hong, C.Y., Mandal, S., Al-Fares, M., Zhu, M., Alimi, R., Bhagat, C., Jain, S., Kaimal, J., Liang, S., Mendelev, K. and Padgett, S., 2018, August. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (pp. 74-87).
- [163] Yap, K.K., Motiwala, M., Rahe, J., Padgett, S., Holliman, M., Baldus, G., Hines, M., Kim, T., Narayanan, A., Jain, A. and Lin, V., 2017, August. Taking the edge off with espresso: Scale, reliability and programmability for global internet

- peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (pp. 432-445).
- [164] Botelho, F., Bessani, A., Ramos, F. and Ferreira, P., 2014. Smartlight: A practical fault-tolerant SDN controller. *arXiv preprint arXiv:1407.6062*.
  - [165] Tootoonchian, A. and Ganjali, Y., 2010, April. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (Vol. 3, No. 10).
  - [166] Bondkovskii, A., Keeney, J., van der Meer, S. and Weber, S., 2016, April. Qualitative comparison of open-source sdn controllers. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (pp. 889-894). IEEE.
  - [167] Phemius, K., Bouet, M. and Leguay, J., 2014, May. Disco: Distributed multi-domain sdn controllers. In *2014 IEEE network operations and management symposium (NOMS)* (pp. 1-4). IEEE.
  - [168] Morales, L.V., Murillo, A.F. and Rueda, S.J., 2015, September. Extending the floodlight controller. In *2015 IEEE 14th International Symposium on Network Computing and Applications* (pp. 126-133). IEEE.
  - [169] Santos, M.A., Nunes, B.A., Obraczka, K., Turletti, T., De Oliveira, B.T. and Margi, C.B., 2014, September. Decentralizing SDN's control plane. In *39th Annual IEEE Conference on Local Computer Networks* (pp. 402-405). IEEE.
  - [170] Stringer, J., Pemberton, D., Fu, Q., Lorier, C., Nelson, R., Bailey, J., Corrêa, C.N. and Rothenberg, C.E., 2014, June. Cardigan: SDN distributed routing fabric going live at an Internet exchange. In *2014 IEEE Symposium on Computers and Communications (ISCC)* (pp. 1-7). IEEE.
  - [171] Gupta, A., MacDavid, R., Birkner, R., Canini, M., Feamster, N., Rexford, J. and Vanbever, L., 2016. An {Industrial-Scale} Software Defined Internet Exchange Point. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)* (pp. 1-14).
  - [172] Kotronis, V., Gämperli, A. and Dimitropoulos, X., 2015. Routing centralization across domains via SDN: A model and emulation framework for BGP evolution. *Computer Networks*, 92, pp.227-239.
  - [173] Chung, J., Cox, J., Ibarra, J., Bezerra, J., Morgan, H., Clark, R. and Owen, H., 2015, November. AtlanticWave-SDX: An international SDX to support science data applications. In *Software Defined Networking (SDN) for Scientific Networking Workshop, SC* (Vol. 15, pp. 1-7).
  - [174] Chung, J., Owen, H. and Clark, R., 2016, March. SDX architectures: A qualitative analysis. In *SoutheastCon 2016* (pp. 1-8). IEEE.
  - [175] Stribling, J., Sovran, Y., Zhang, I., Pretzer, X., Li, J., Kaashoek, M.F. and Morris, R.T., 2009, April. Flexible, Wide-Area Storage for Distributed Systems with WheelFS. In *NSDI* (Vol. 9, pp. 43-58).
  - [176] Vernon, V., 2015. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka. Addison-Wesley Professional.

- 
- [177] Sakic, E. and Kellerer, W., 2017. Response time and availability study of RAFT consensus in distributed SDN control plane. *IEEE Transactions on Network and Service Management*, 15(1), pp.304-318.
  - [178] Vinoski, S., 2006. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), pp.87-89.
  - [179] Benamrane, F. and Benaini, R., 2017. An East-West interface for distributed SDN control plane: Implementation and evaluation. *Computers & Electrical Engineering*, 57, pp.162-175.
  - [180] Wang, D., Dong, L., Tang, H., Gu, J., Liu, Z. and You, X., 2024, April. SDN Security Channel Constructed Using SM9. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS)* (pp. 1-5). IEEE.
  - [181] Kalafatidis, S., Agrafiotis, G., Giapantzis, K., Lalas, A. and Votis, K., 2024, March. Experiments with Digital Security Processes over SDN-Based Cloud-Native 5G Core Networks. In *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)* (pp. 97-99). IEEE.
  - [182] Kaur, A., Rama Krishna, C. and Patil, N.V., 2024. KS-SDN-DDoS: A Kafka streams-based real-time DDoS attack classification approach for SDN environment. *Journal of Intelligent & Fuzzy Systems*, (Preprint), pp.1-12.
  - [183] Levterov, A., Pliekhova, H., Kostikova, M., Berezhna, N. and Okun, A. (2023) "ENHANCING SECURITY IN SOFTWARE-DEFINED NETWORKING THROUGH ROUTING TECHNIQUES EXPLORATION", *Bulletin of National Technical University "KhPI". Series: System Analysis, Control and Information Technologies*, (1 (9)), pp. 10–18. doi: 10.20998/2079-0023.2023.01.02.
  - [184] Jenny, R.S. and Sugirtham, N., 2023. SDN-Based Security for Smart Devices Against Denial of Service Attacks. *Indian Journal of Science and Technology*, 16(3), pp.181-189.
  - [185] David, A.O. and OMOTOSHO, O.I., Scalable Flow based Management Scheme in Software Define Network (SDN) using sFlow.
  - [186] Kaur, A., Krishna, C.R. and Patil, N.V., 2024. K-DDoS-SDN: A distributed DDoS attacks detection approach for protecting SDN environment. *Concurrency and Computation: Practice and Experience*, 36(3), p.e7912.
  - [187] Ahmad, S. and Mir, A.H., 2023. Securing centralized sdn control with distributed blockchain technology. *Computer Science*, 24(1).
  - [188] Sandhia, G.K., Nithyaselvakumari, S., Saidulu, V., Sulaiman, N.H.B. and Salameh, A.A., 2023. Enhancing the Security of Software Defined Mobile Networks (SDMN) Based on Blockchain Technology. *International Journal of Interactive Mobile Technologies*, 17(4).
  - [189] Priyadarsini, M., Bera, P., Das, S.K. and Rahman, M.A., 2022. A security enforcement framework for SDN controller using game theoretic approach. *IEEE Transactions on Dependable and Secure Computing*, 20(2), pp.1500-1515.
  - [190] Lv, Z. and Kumar, N., 2020. Software defined solutions for sensors in 6G/IoE. *Computer Communications*, 153, pp.42-47.

- 
- [191] Hao, G. and Tao, G., 2009. Principle of and Protection of Man-in-the-middle Attack Based on ARP Spoofing. *Journal of Information Processing Systems*, 5(3), pp.131-134.
  - [192] Brooks, M. and Yang, B., 2015, September. A Man-in-the-Middle attack against OpenDayLight SDN controller. In *Proceedings of the 4th annual ACM conference on research in information technology* (pp. 45-49).
  - [193] Kim, M., Shin, Y. and Shon, T., 2021, August. Mitm tool analysis for tls forensics. In *2021 International Conference on Platform Technology and Service (PlatCon)* (pp. 1-4). IEEE.
  - [194] Haakegaard, R. and Lang, J., 2015. The elliptic curve diffie-hellman (ecdh). Online at [https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+ Lang. pdf](https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf).
  - [195] Yusfrizal, Y., Meizar, A., Kurniawan, H. and Agustin, F., 2018, August. Key management using combination of Diffie–Hellman key exchange with AES encryption. In *2018 6th International Conference on Cyber and IT Service Management (CITSM)* (pp. 1-6). IEEE.
  - [196] Badotra, S. and Singh, J., 2017. Open Daylight as a Controller for Software Defined Networking. *International Journal of Advanced Research in Computer Science*, 8(5).
  - [197] Zhou, H., Wu, C., Yang, C., Wang, P., Yang, Q., Lu, Z. and Cheng, Q., 2018. SDN-RDCD: A real-time and reliable method for detecting compromised SDN devices. *IEEE/ACM transactions on networking*, 26(5), pp.2048-2061.
  - [198] Astaneh, S.A. and Heydari, S.S., 2016. Optimization of SDN flow operations in multi-failure restoration scenarios. *IEEE Transactions on Network and Service Management*, 13(3), pp.421-432.
  - [199] Wang, H., Zhang, D. and Shin, K.G., 2002, June. Detecting SYN flooding attacks. In *Proceedings. Twenty-first annual joint conference of the IEEE computer and communications societies* (Vol. 3, pp. 1530-1539). IEEE.
  - [200] Chen, W. and Yeung, D.Y., 2006, April. Defending against TCP SYN flooding attacks under different types of IP spoofing. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)* (pp. 38-38). IEEE.
  - [201] Harris, B. and Hunt, R., 1999. TCP/IP security threats and attack methods. *Computer communications*, 22(10), pp.885-897.
  - [202] Jain, G., 2021, March. Application of snort and wireshark in network traffic analysis. In *IOP Conference Series: Materials Science and Engineering* (Vol. 1119, No. 1, p. 012007). IOP Publishing.
  - [203] Morris, M.F., 1974. Kiviat graphs: conventions and" figures of merit". *ACM SIGMETRICS Performance Evaluation Review*, 3(3), pp.2-8.
  - [204] Schwarz, S., Mehlhruer, C. and Rupp, M., 2011, June. Throughput maximizing multiuser scheduling with adjustable fairness. In *2011 IEEE International Conference on Communications (ICC)* (pp. 1-5). IEEE.

- 
- [205] Guo, F. and Chiueh, T.C., 2006. Sequence number-based MAC address spoof detection. In *Recent Advances in Intrusion Detection: 8th International Symposium, RAID 2005, Seattle, WA, USA, September 7-9, 2005. Revised Papers* 8 (pp. 309-329). Springer Berlin Heidelberg.
- [206] Wang, S.Y., 2014, June. Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet. In *2014 IEEE Symposium on Computers and Communications (ISCC)* (pp. 1-6). IEEE.
- [207] Zalila, F., Challita, S. and Merle, P., 2017. A model-driven tool chain for OCCI. In *On the Move to Meaningful Internet Systems. OTM 2017 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I* (pp. 389-409). Springer International Publishing.
- [208] Agrawal, N. and Tapaswi, S., 2017, November. A lightweight approach to detect the low/high rate IP spoofed cloud DDoS attacks. In *2017 IEEE 7th international symposium on cloud and service computing (SC2)* (pp. 118-123). IEEE.
- [209] Sood, K., Karmakar, K.K., Varadharajan, V., Tupakula, U. and Yu, S., 2019. Analysis of policy-based security management system in software-defined networks. *IEEE Communications Letters*, 23(4), pp.612-615.
- [210] Gray, N., Zinner, T. and Tran-Gia, P., 2017, May. Enhancing SDN security by device fingerprinting. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (pp. 879-880). IEEE.
- [211] Lara, A. and Quesada, L., 2018, November. Performance analysis of SDN northbound interfaces. In *2018 IEEE 10th Latin-American Conference on Communications (LATINCOM)* (pp. 1-6). IEEE.
- [212] Shi, Y., Dai, F. and Ye, Z., 2017, November. An enhanced security framework of software defined network based on attribute-based encryption. In *2017 4th International Conference on Systems and Informatics (ICSAI)* (pp. 965-969). IEEE.
- [213] Aliyu, A.L., Bull, P. and Abdallah, A., 2017, March. Performance implication and analysis of the OpenFlow SDN protocol. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (pp. 391-396). IEEE.
- [214] Suárez-Varela, J. and Barlet-Ros, P., 2018, March. Sbar: Sdn flow-based monitoring and application recognition. In *Proceedings of the Symposium on SDN Research* (pp. 1-2).
- [215] Yoon, C., Park, T., Lee, S., Kang, H., Shin, S. and Zhang, Z., 2015. Enabling security functions with SDN: A feasibility study. *Computer Networks*, 85, pp.19-35.
- [216] Cisco, 2020. Cisco Annual Internet Report—Cisco Annual Internet Report (2018–2023) White Paper.
- [217] Singh, S. and Jha, R.K., 2017. A survey on software defined networking: Architecture for next generation network. *Journal of Network and Systems Management*, 25, pp.321-374.

- 
- [218] Nisar, K., Jimson, E.R., Hijazi, M.H.A., Welch, I., Hassan, R., Aman, A.H.M., Sodhro, A.H., Pirbhulal, S. and Khan, S., 2020. A survey on the architecture, application, and security of software defined networking: Challenges and open issues. *Internet of Things*, 12, p.100289.
- [219] Kim, H. and Feamster, N., 2013. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2), pp.114-119.
- [220] Sezer, S., Scott-Hayward, S., Chouhan, P.K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M. and Rao, N., 2013. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications magazine*, 51(7), pp.36-43.
- [221] Karakus, M. and Durresi, A., 2017. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks*, 112, pp.279-293.
- [222] Shah, N., Giaccone, P., Rawat, D.B., Rayes, A. and Zhao, N., 2019. Solutions for adopting software defined network in practice. *International Journal of Communication Systems*.
- [223] Khare, S., An, K., Gokhale, A., Tambe, S. and Meena, A., 2015, June. Reactive stream processing for data-centric publish/subscribe. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems* (pp. 234-245).
- [224] Pardo-Castellote, G., 2003, May. Omg data-distribution service: Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops*, 2003. *Proceedings.* (pp. 200-206). IEEE.
- [225] Lantz, B. and O'Connor, B., 2015. A mininet-based virtual testbed for distributed SDN development. *ACM SIGCOMM Computer Communication Review*, 45(4), pp.365-366.
- [226] Chappell, L. and Combs, G., 2010. *Wireshark network analysis: the official Wireshark certified network analyst study guide*. Reno, NV: Protocol Analysis Institute, Chappell University.
- [227] Almadani, B., Beg, A. and Mahmoud, A., 2021. Dsf: A distributed sdn control plane framework for the east/west interface. *IEEE Access*, 9, pp.26735-26754.
- [228] Scandariato, R., Wuyts, K. and Joosen, W., 2015. A descriptive study of Microsoft's threat modeling technique. *Requirements Engineering*, 20, pp.163-180.

## BIO DATA OF THE CANDIDATE

Name : Tinku Adhikari

Date of Birth : 18/04/1983

Phone : 9836360673

email : tinku.adhikari@gmail.com, mzu2103577@mzu.edu.in

Permanent Address : Flat 2B, Susanta Apartment, Agarpara, Kolkata-109

Married : Yes

Educational Details

1. MCA : Indira Gandhi National Open University (IGNOU)
2. M.Tech : NITTTR, Kolkata, WB
3. Ph.D Course Work : Mizoram University

Present Occupation : Teaching

Organization : Techno International Newtown

## List of Publications Based on Thesis

### Journals

1. **T.Adhikari**,AK.Khan,M.Kule,S.Das“An Efficient Approach for Detection of Compromised SDN Switches and Restoration of Network Flow” in International Journal of Computer Network and Information Security (IJCNIS) (**Accepted**) Scopus
2. **T Adhikari**, AK Khan, M Kule (2024) ‘ProDetect: A Proactive Detection Approach of the TCP SYN Flooding Attack in the SDN Controller’, *IETE Journal of Education*, pp. 1–12. doi: 10.1080/09747338.2024.2379267. (**Published**) Scopus
3. **T Adhikari**, AK Khan, M Kule “A Secure Framework for Trust Management to Mitigate Application Threats in Software Defined Networking” in Journal of Network and Systems Management (**Under Review**) SCIE IF-4.1
4. **T Adhikari**, AK Khan, M Kule “An Analytical Review of Security Issues in Centralized and Distributed SDN Environments” in Information Security Journal: A Global Perspective (**Communicated**)ESCI IF-1.6
5. **T Adhikari**, AK Khan, M Kule “SRDCP: A Secure and Resilient Distributed Control Plane Architecture for Large Scale SDN Deployments” in International Journal of Parallel, Emergent and Distributed Systems (**Communicated**) SCIE IF-0.6

### Conferences

1. **T. Adhikari**, M. Kule and A. K. Khan, "An ECDH and AES Based Encryption Approach for Prevention of MiTM in SDN Southbound Communication Interface," *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2022, pp. 1-5, doi: 10.1109/ICCCNT54827.2022.9984509
2. **T. Adhikari**, A. K. Khan and M. Kule, “Design of An Application Authentication Framework to Secure Controller Access via Northbound Interface in SDN” in International Conference on Computational Technologies and Electronics (ICCTE) 2023
3. **T. Adhikari**, A. K. Khan and M. Kule, “ADSA: An Adaptive Distributed Architecture for SDN East/West Interface to Ensure Network Synchronization and Security” in International Conference on Intelligent Systems and Security (**Communicated**)



### **Book Chapters**

1. **T. Adhikari**, A. K. Khan and M. Kule “An Indirect Trust Establishment Framework for Network Applications in Software Defined Network (SDN)” as a chapter in “Emerging Artificial IoT Technologies: Security and Communication in Industrial Applications” by Wiley Publishers, SCOPUS indexed (**Accepted**)

### **PARTICULARS OF THE CANDIDATE**

NAME OF THE CANDIDATE : Tinku Adhikari

DEGREE : Ph.D.

DEPARTMENT : Computer Engineering

TITLE OF THE THESIS : Detection and Mitigation of Architectural  
Vulnerabilities in Software Defined  
Networking

DATE OF ADMISSION : 13/09/2021

APPROVAL OF RESEARCH PROPOSAL

1. DRC : 26/04/2022

2. BoS : 24/05/2022

3. SCHOOL BOARD : 03/06/2022

MZU REGISTRATION NUMBER : 2108257

Ph.D REGISTRATION NUMBER : MZU/Ph.D/1845 of 13.09.2021

EXTENSION : NA

(DR. V.D. AMBETH KUMAR)

Head

Dept. of Computer Engineering

# ABSTRACT

## DETECTION AND MITIGATION OF ARCHITECTURAL VULNERABILITIES IN SOFTWARE DEFINED NETWORKING

AN ABSTRACT SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

TINKU ADHIKARI

MZU REGISTRATION NO.:2108257

Ph.D REGISTRATION NO.:MZU/Ph.D/1845 OF 13.09.2021



DEPARTMENT OF COMPUTER ENGINEERING  
SCHOOL OF ENGINEERING AND TECHNOLOGY

SEPTEMBER 2024

**DETECTION AND MITIGATION OF ARCHITECTURAL VULNERABILITIES  
IN SOFTWARE DEFINED NETWORKING**

BY

TINKU ADHIKARI

Department of Computer Engineering

Supervisor: Prof. Ajoy Kumar Khan

Joint Supervisor: Dr. Malay Kule

Submitted

In partial fulfillment of the requirement of the Degree of Doctor of Philosophy in  
Computer Engineering of Mizoram University, Aizawl.

## **Abstract**

The evolution of Software-Defined Networking (SDN) has revolutionized the management and configuration of network infrastructures by decoupling the control plane from the data plane, allowing centralized control and programmability. However, this paradigm shift also introduces significant security challenges across various interfaces and planes of the SDN architecture. This Ph.D. thesis addresses critical security concerns across the data plane, southbound interface, control plane, northbound interface, application plane, and distributed SDN architectures, offering innovative solutions to enhance the security and resilience of SDN deployments.

The southbound interface, which connects the SDN controller to network devices like switches, is crucial but vulnerable to threats such as unauthorized access and flow rule manipulations. This research focuses on man-in-the-middle attacks and compromised switch detection, analyzing their impact on the data plane. A novel security framework is proposed with encrypted communication and real-time flow restoration after isolating faulty switches to secure the southbound interface and maintain data plane integrity.

The control plane, acting as the network's brain, orchestrates traffic flows and enforces policies across the entire SDN environment. Given its central role, the control plane is a prime target for various security threats, including Distributed Denial of Service (DDoS) attacks, controller hijacking, and policy injection attacks. This thesis thoroughly characterizes one of these threats, SYN flood attack and analyzing its potential to disrupt network operations and compromise sensitive information. To address this challenge, a novel security architecture is developed, to cater SYN flood attack and minimize the risk of underlying DoS attack. The proposed architecture aims to enhance the control plane's resilience, ensuring continuous and secure network operations.

The northbound interface, which enables communication between the SDN controller and network applications, is another critical area of concern. Applications which communicate through this interface may be vulnerable and attack might be launched from these types of untrusted applications. Security loopholes in this interface can lead to unauthorized access, data leakage, and application-level attacks, potentially compromising the entire network. This research identifies and classifies untrusted applications, providing a detailed taxonomy of threats based on their origins and impact. Solutions are proposed to create a trustworthy environment where only the authenticated and trusted applications can communicate with the controller via the northbound interface. These measures are designed to fortify the northbound interface and protect the application plane from malicious activities.

With the increasing adoption of distributed SDN architectures, ensuring secure communication between multiple controllers via the east-westbound interface has become paramount. This thesis identifies security challenges specific to distributed SDN deployments, such as inter-controller trust issues, synchronization attacks, and cross-domain policy enforcement. A security-enhanced, robust and resilient distributed controller interface is designed and simulated, incorporating features like security and scalability in large scale homogeneous and heterogeneous networks

This Ph.D. thesis makes significant contributions to the field of SDN security by addressing critical vulnerabilities across all layers and interfaces of the SDN architecture. The proposed solutions not only enhance the security posture of SDN deployments but also pave the way for more resilient and trustworthy network infrastructures. Through rigorous analysis, innovative security frameworks, and comprehensive testing, this research provides a solid foundation for future advancements in SDN security, ensuring the safe and reliable operation of next-generation networks.

**Keywords - SDN Security, Data Plane Security, Control Plane Security, Application Plane Security, Distributed SDN Environment, Compromised Switch Detection, SYN Flood Attack, Application Trust.**