(1)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of PhD thesis written by **Bilal Rauf,** Registration No: **00000202462** of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of PhD degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature: [illegible]

Name of Supervisor: **Dr. Haider Abbas**

Date: 18 JAN 2022

Signature (HoD): [illegible]

Date: 18 JAN 2022

Signature (Dean): [illegible]

Date: 24/1/22

Brig
Dean, MCS (NUST)
(Asif Masood, Phd)

2

Form PhD-7
DOCTORAL PROGRAM
OF STUDY
(Must be type written)

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY
REPORT OF DOCTORAL THESIS DEFENCE

Name: **Bilal Rauf** NUST Regn No **00000202462**

School/College/Centre: **Military College of Signals (NUST)**

Title: **A Secure Communication Framework for Enterprise Networks Using SDN**

DOCTORAL DEFENCE COMMITTEE

Doctoral Defence held on **21st December 2021**

		QUALIFIED	NOT QUALIFIED	SIGNATURE
GEC Member 1:	Dr Imran Rashid	✓		[illegible]
GEC Member 2:	Dr. Hammad Afzal	✓		[illegible]
GEC Member 3:	Dr. Abdul Rauf	✓		[illegible]
GEC Member (External):	Dr. Faisal Bashir	✓		[illegible]
Co-Supervisor:	Dr Muhammad Faisal Amjad	✓		[illegible]
Supervisor:	Dr Haider Abbas	✓		[illegible]
Dean:	Dr. Asif Masood	✓		[illegible]
External Evaluator 1: (Local Expert)	Dr. Muhammad Hanif Durad	✓		[illegible]
External Evaluator 2: (Local Expert)	Dr. Syed Hashim Raza Bukhari	✓		[illegible]
External Evaluator 3: (Foreign Expert)	Dr Cliff C. Zou			
External Evaluator 4: (Foreign Expert)	Dr. Peng Liang			

FINAL RESULT OF THE DOCTORAL DEFENCE
(Appropriate box to be signed by HOD)

[illegible] **PASS** **FAIL**

The student **Bilal Rauf** Regn No **00000202462** is accepted for Doctor of Philosophy Degree.

Dated: 23/12/21

[illegible]
Brig
Dean, MCS (NUST)
(Asif Masood, Phd)
Dean/Commandant/Principal/DG

Distribution:
1 x original copy each for PGP Dte, Exam Branch Main Office NUST and Student's dossier at the School/College/Centre. 1x photocopy each for HoD, Supervisor, Co Supervisor (if appointed), sponsoring agency (if any) and 05 copies for insertion in Dissertation.

Note:
* Decision of External Evaluators (Foreign Experts) will be sought through video conference, if possible, on the same date and their decision will be intimated (on paper) to HQ NUST at a later date.

Certificate of Approval

This is to certify that the research work presented in theis titled "**A Secure Communication Framework for Enterprise Networks Using SDN**" was conducted by **Bilal Rauf** , under the supervision of **Assoc Prof Dr. Haider Abbas** No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the Department of Information Security, Military College of Signals in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the field of Information Security, Department of **Information Security** of **Military College of Signals**, **National University of Sciences and Technology, Islamabad**.

Student Name: **Bilal Rauf** Signature: [illegible]

Examination Committee:-
External Examiner Local 1: **Dr. Muhammad Hanif Durad** Signature: [illegible]
Assistant Professor at Computer and Information Science
PIEAS University, Islamabad.

External Examiner Local 2: **Dr. Syed Hashim Raza Bukhari** Signature: [illegible]
Assistant Professor at the Department of
Electrical & Computer Engineering, COMSATS University,
Islamabad. Attock Campus.

External Examiner Local 2: **Dr. Faisal Bashir** Signature: [illegible]
Professor at the Department of
Computer Science, Bahria University, Islamabad.

a) Internal Examiner 1: **Dr. Muhammad Faisal Amjad** Signature: [illegible]
(Associate Professor,
Department of Information Security
Military College of Signals, NUST)
Rawalpindi

Co-Supervisor: **Dr. Muhammad Faisal Amjad** Signature [illegible]

Supervisor: **Assoc Prof Dr. Haider Abbas** Signature [illegible]

Dean: **Dr. Asif Masood** Signature: [illegible]

Brig
Dean, MCS (NUST)
(Asif Masood, Phd)

AUTHOR's DECLARATION

I, Bilal Rauf herby state that my PhD thesis titled "**A Secure Communication Framework for Enterprise Networks Using SDN** " is my own work and has not been submitted previously by me for taking from this University.

NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

Or anywhere else in the country / World.

At any time if my statement is found to be incorrect even after my Graduate the university has the right to withdraw my PhD degree.

[illegible]

Bilal Rauf
00000202462

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in thesis titled "**A Secure Communication Framework for Enterprise Networks Using SDN** " is solely my research work with on significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledge and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and University

NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

towards plagiarism . Therefore I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and my material used as reference is properly referred / cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD degree, the University reserves the rights to withdraw/revoke my PhD degree and that **HEC** and the University has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.

[illegible]

Bilal Rauf

A SECURE COMMUNICATION FRAMEWORK FOR ENTERPRISE NETWORKS USING SDN

By

Bilal Rauf

A thesis submitted to the faculty of Department of Information Security, Military College of Signals, National University of Sciences and Technology, Islamabad, Pakistan, in partial fulfillment of the requirements for the degree of PhD in Information Security

Dec 2021

Abstract

In today's era, large data centers are drawn towards the two popular technologies i.e., Enterprise Integration Patterns (EIP) and Software Defined Networking (SDN). The former is the combination of design patterns that integrates the new and existing business applications in an enterprise environment whereas, the latter is a rapidly evolving networking paradigm that has reshaped the large enterprise network management by introducing programmable planes and centralized control. The SDN-based design provides flexibility in network management which spans over multiple applications e.g., routing, switching, forwarding, and controlling. It reduces the reliance on vendor-specific devices and middlebox solutions like firewalls, IDS, IPS, etc. The promising features of EIP i.e., asynchronous communication, reliability, and that of SDN, namely, robustness, network programmability, agility, and global visibility can be merged, to cope with growing network demands and security.

In this research, we introduce a new communication framework for enterprise networks that incorporates EIP in SDN for asynchronous and reliable message exchange among applications. The proposed communication framework integrates multiple technologies such as Virtual Local Area Networks (VLANs), Address Resolution Protocol (ARP), context-aware services, and anonymous communication, to provide accurate, efficient, and secure network services. Moreover, all the above-mentioned technologies are implemented as application modules of the RYU SDN controller, and communication is only allowed between any two applications/services through EIP Channel.

To provide communication within the same network, the proposed communication framework utilizes the functionality of VLANs by offering an adaptive VLAN Management module. Using this module, the framework supports reactive VLAN creation and deletion mechanisms between the communicating hosts. Additionally, VLANs are only created for the active duration of the communication. Furthermore, to enable communication between applications from different networks in an enterprise environment, this framework also contains a packet forwarding module where hosts IP addresses are concealed from each other.

Furthermore, due to the integration of different technologies, privacy is one of the core issues faced by the enterprise. Host anonymity is one of the techniques to safeguard against privacy attacks; however, the existing anonymization solutions provide better anonymity, but at the cost of higher latency and are most suited for internet traffic. To tackle this issue in an enterprise network, this research offers anonymous communication among hosts in an enterprise environment. Unlike the traditional networks, SDN can modify the header fields of packets as they traverse the network from source to destination. Host anonymity is achieved by replacing the real IP address with the hoax IP address during the transmission of data packets inside the network.

Similarly, we also present a context-aware communication framework by leveraging the global visibility feature of SDN. In this context-aware communication, services are discoverable to the clients without disclosing the addresses of actual application servers. By using these context-aware services, network traffic is routed based on the application layer information rather than the network layer information.

The evaluation is done using multiple scenarios having different host configurations. We conducted series of experiments to test the accuracy, efficiency, computational complexity, and security of the communication framework. In addition, we also highlighted that the proposed framework is more suitable for heterogeneous network environments such as IoT-based solutions.

Dedication

This thesis is dedicated to

MY BELOVED PARENTS,

MY WIFE,

MY CHILDREN (OMER & HAMZA),

MY COLLEAGUES, FRIENDS AND TEACHERS

for their love, endless support and encouragement

Acknowledgements

I am grateful to Allah Almighty who has bestowed me with the strength and the passion to accomplish this thesis and I am thankful to Him for His mercy and benevolence. Without his consent I could not have indulged myself in this task.

With affection and deep appreciation, I would like to express my heartiest gratitude to my supervisor Dr. Haider Abbas, for his all time guidance, support, and valuable suggestions that lead to the completion of this dissertation. I am also thankful to my co-supervisor Dr. Faisal Amjad (MCS, NUST) and my GEC members, Brig Dr. Imran Rashid (MCS, NUST), Dr. Hammad Afzal (MCS, NUST) and Brig Dr. Abdul Rauf (MCS, NUST), for being supportive during my PhD studies.

I am also very grateful to Dr. Ahmed Muqueem Sheri for his guidance and thought provoking discussions held during the course of this dissertation.

Finally but most importantly, I would like to thank my parents for their support and blessings, My wife for her support, patience, understanding and encouragement while I was completing my PhD studies.

Contents

List of Figures

List of Tables

Acronyms

Address Resolution Protocol	ARP
Advanced Message Queueing Protocol	AMQP
Application	APP
Application Programming Interface	API
Border Gateway Protocols	BGP
Classless Inter Domain Routing	CIDR
Database	DB
Denial of Service	DoS
Destination Address	DA
Enterprise Integration Patterns	EIP
Identification	ID
Internet Protocol	IP
Internet Protocol version 4	IPv4
Local Area Network	LAN
Man-In-The-Middle	MITM
Medium Access Control	MAC
Message Queuing	MQ
Network Operating System	NOS
Northbound Interface	NBI
Open Network Operating System	ONOS
Open Networking Foundation	ONF
Open Service Gateway Initiative	OSGi
Open System Interconnection	OSI
Open VSwitch	OvS
Open vSwitch Database	OvSDB
OpenFlow	OF
Participatory Networking	PANE

Priority Code Point	PCP
Protocol Oblivious Forwarding	POF
Representational State Transfer	REST
Software Defined Networking	SDN
Source Address	SA
Southbound Interface	SBI
Tag Control Information	TCI
Ternary Content Addressable Memory	TCAMs
Transmission Control Protocol	TCP
Transport Layer Security	TLS
Type of Service	TOS
User Datagram Protocol	UDP
Virtual Local Area Network	VLAN

Chapter 1

Introduction

Computer networks have advanced fast in terms of processing, complexity, and diversity during the last few decades. In typical IP networks, routers and switches transport data between hosts, with each router knowing about a subset of the network and forwarding traffic based on its route computations. Due to their vertical integration, intelligence resides within the device, traditional network topologies are complex and difficult to operate. Historically, they have made packet forwarding decisions solely on the basis of the packet's destination address. Due to the heterogeneity of vendor-specific network devices and the quick development of new protocols at each layer of the OSI model, traditional networks are constrained by adaptive configuration and troubleshooting in large-scale networks.

NFV and SDN are two frameworks that were developed to overcome the limitations of conventional networks.NFV leverages virtualization to isolate Network Functions (NFs) such as routing, firewalling, and encryption from their associated hardware middle boxes [3]. On the contrary, the key characteristic of SDN is the separation of administration and control logic from traditional networking hardware such as routers and switches [4]. It reintroduces the centralised network idea, in which the SDN controller is responsible for the whole network's operations, including traffic routing, security, network monitoring, and load balancing. In addition to centralization, SDN introduces the notion of programming network applications for commercial goals. The contrast between these two models is that SDN relies on a centralised controller to interpret application orders into the OpenFlow (OF) language, whereas NFV administers the network totally through NF (applications) housed on virtual machines. In this research, we are focusing entirely on the SDN paradigm and its integration into corporate networks.

Managing a organizational network has never been easy, especially when numerous technologies are involved, vendor-specific solutions are used, and complex monitoring and troubleshooting applications are used. However, in 2008, the introduction of the OpenFlow protocol enhanced the concept of software-based networking solutions, laying the groundwork for the Software Defined Networking paradigm [4]. SDN enables network managers to manage and configure enterprise networks centrally. SDN's central notion is to abstract control logic away from devices on the infrastructure plane and assign it to a centralized controller on the control plane. By limiting the functions of complicated routing and switching devices to merely executing the policy specified by the logically centralized controller, this extraction of control logic reduced them into simple forwarding devices. Additionally, SDN's promising characteristic is the network's programmability, which enables network managers and software developers to respond to changing network demands and conditions source [6].

The SDN now makes traffic routing decisions based on the information of packet header data of multi-layer protocols. This choice on how to forward network data is in contrast to traditional networks, which operate independently of each layer of the protocol stack. Additionally, the SDN controller may create flow rules that alter incoming packet header information, such as IP addresses, port numbers, and MAC addresses. As seen in Fig. 2.1, the architecture of SDN is built of three planes: the Application Plane, the Control Plane, and the Infrastructure Plane. There are two communication channels between these planes: Northbound Interface (NBI) and Southbound Interface (SBI) [7]. Additionally, dispersed controllers employ the East/Westbound Interfaces to communicate with one another and with older networks. Section 2.1 discusses SDN in full.

Apart from the bottom layer communication infrastructure, enterprise networks have integration challenges due to the fact that the majority of services and applications are developed independently, using disparate development platforms, languages, and data formats. These applications provide a range of services to clients and employees through a corporate network. Numerous methods have been offered to provide communication infrastructure amongst various apps that give services to company users.

In [8], the authors identified four primary ways for establishing an inter-application communication infrastructure: Messaging System, Shared Database, File Transfer, and Remote Procedure Invocation. Where messaging systems shown to be a superior implementation when the asynchronous aspect of message exchange is considered.

1.1 Problem Specification

Multiple service (applications) integration is a difficult task in software development. Integrating the two applications is reasonably trivial if applications are of different environment. Integrating apps from diverse backgrounds is fairly challenging. However, in a commercial context, it is essentially difficult for a single application to deliver all operations in a business. The value of reusing previously developed apps is not only practical, but also economical. This necessitates the integration of many apps throughout the company.

Additionally, the SDN paradigm increases the flexibility of the enterprise network's packet forwarding mechanism; however, a critical issue with adopting SDN in an enterprise network is the network's overall security. Furthermore, traditional networkssecurity solutions involves complex and pricey middle-boxes. Numerous encryption techniques, such as IPSec, TLS, and others, are used to protect the data while the travelling of packet over the network. However, information present in the packet header can reveal the names of communicating hosts and their traffic patterns. By abusing the packet header, an attacker can launch multiple attacks on the network.

SDN can assist network managers in tackling issues common in traditional networks, such as vendor-specific devices and solutions, reliance on middleboxes for security solutions, and manual device setup. To address existing problems and to give network administrators with fine-grained control in enterprise networks, the usage of EIP powered by SDN can be beneficial. Where SDN provides the communication infrastructure and EIP ensures the reliable transmission of messages between applications/services. Additionally, the robust qualities of SDN, such as global visibility, programmability, and responsiveness to growing network demands, make SDN an ideal candidate for integrating with EIP in an enterprise network. To our understanding, combining SDN with EIP benefits enterprise networks in a variety of ways. As far as we are aware, this is the first attempt, and no comparable contribution has been identified in the literature.

To address the aforementioned difficulties, we developed a communication framework for enterprise networks that incorporates SDN and Messaging Systems (EIP). This framework enables context-aware services, adaptive VLAN management, and anonymous and secure communication. Additionally, our platform enables enterprises to migrate to an SDN-based network in order to benefit from increased efficiency, built-in security, message reliability, and connection with existing business applications.

1.2 Contribution

In this dissertation, EIP is incorporated in SDN for enterprise networks. The proposed communication framework is segregated into multiple modules and these modules are: *1) Host Registration*: module to register connected hosts; *2) Service Registry*: module to register and maintain the information of all the services offered in the network; *3) IP Address Mapper*: module to generate hoax IP addresses of each hosts and maintain a map of real IP address against the hoax IP address; *4) Adaptive VLAN Management*: module that support communication using VLANS and is responsible for the creation and deletion of VLANs reactively; *5) ARP Request Resolution*: module to support the resolution of Ethernet gateway address when two communicating hosts belongs to different subnets; and finally the last *6) Packet Forwarding*: module that facilitates in path identification between source and destination hosts.

The main contributions of this dissertation are as follows.

1. We incorporated EIP in SDN and proposed a novel communication system for the organizational networks that support reliable message exchange system.

2. A context-aware communication framework is presented by using global visibility characteristic of SDN. Clients can discover services through this context-aware communication without divulging the addresses of the real servers. Network traffic is routed using the -application layer information as compared to the legacy network where routing is done using network layer header information.

3. To facilitate reactive VLAN formation and deletion across communication hosts, we present a dynamic VLAN management solution.

4. We offer a secure forwarding technique based on application services that makes use of SDN's most important characteristic, global visibility, to deliver secure communication in a business network.

5. We enhance the security of the network by proposing an anonymous communication technique among the communicating hosts by replacing the real IP address with the Hoax address.

6. Implementation of the proposed framework is done in Mininet using a component-based RYU SDN controller. Moreover, Containernet is used on top of Mininet to deploy docker containers as hosts in Mininet.

7. Evaluation of the proposed framework is done using three different scenarios having different host configurations by designing a series of experiments to test the performance, efficiency, computational complexity, and security of our communication framework. The security evaluation is performed using the NMAP network scanning tool.

8. The contribution mentioned in serials 1, 3, and 4 are published in [9], whereas contributions listed in serials 2 and 5 are published in [10]. In addition to the technical articles, in this Ph.D. research, a review article highlighting the security requirements of the northbound interface of SDN is also published in [11].

1.3 Prospective Market

The Internet of Things (IoT) is a concept that promotes heterogeneity. Systems based on the Internet of Things, such as Smart Farming [13], Smart Factory [14], Smart Cities [15], and Smart Healthcare [16], are just a few instances of heterogeneous networks where asynchronous communication over EIP can be advantageous. EIP enables the discovery of services that are utilised by IoT gateways/devices. As a result, a dependable communication structure across all networks is required. These communication pathways generate patterns that may be abstracted primarily as EIP channels. These channels are influenced by properties such as fault tolerance, asynchronous message exchange, and diverse patterns such as publish and consume, translator, and router [8]. In smart cities, IoT networks are built on heterogeneous technologies comprised of a variety of device types [15] that generate data in a variety of forms, necessitating the adoption of the message translator pattern in EIP. Similarly, the publish and consume pattern enables message interchange in EIP, which is advantageous for devices operating on limited power in smart settings. Additionally, for network security in IoT-based networks, safe routing using a trustworthy SDN controller is suggested [17], [18]. Similarly, asynchronous communication can help alleviate resource allocation issues in IoT networks, while SDN enables greater flexibility in network traffic management. As a result, our suggested architecture serves as a viable communication mechanism for Internet of Things-based applications.

1.4 Thesis Outline

The thesis is organized as follows:

- **Chapter 2** presents the brief discussion of technologies used in this research which includes Software Defined Networking, Enterprise Integration Patterns, and Virtual Local Area Networks.
- **Chapter 3** highlights the related literature which is segregated into multiple sections. These sections include SDN in enterprise networks, VLAN management in SDN, handling ARP packets in SDN, anonymous communication using SDN and context aware services using SDN.
- **Chapter 4** provides the detail of all the modules incorporated in the proposed communication framework.
- **Chapter 5** addresses the integration of EIP in the proposed communication framework.
- **Chapter 6** presents the evaluation results and discussions of the proposed communication framework using three different evaluation scenarios.
- **Chapter 7** finally conclude the thesis with some future research directions.

Chapter 2

Background Technologies

This chapter introduces the technologies that are incorporated in this dissertation. The technologies used are: 1) Software Defined Network, 2) Enterprise Integration Patterns, and 3) Virtual Local Area Networks. In the following sections, we elaborate each technology with respect to our research. In the last section of this chapter, we describe a simple example to let the reader comprehend how SDN network communicates using the flow rules.

2.1 Software Defined Networking (SDN)

SDN was introduced as a new networking model in 2008 [4]. The SDN architecture is divided into three planes: the *Data Plane*, the *Control Plane*, and the *Application Plane*. Communication channels exists between the above-mentioned planes: the *Southbound* and the *Northbound* interfaces. Control plane and data planes are connected through SBI, whereas nbusiness applications instructs the controller using the NBI. The other special interfaces are the *East/Westbound Interfaces*, which distributed controllers use to communicate with one another and with legacy networks. The complete SDN architecture is depicted in Fig. 2.1.

2.1.1 Data Plane

The data plane is the most fundamental component of the SDN architecture, consisting of devices like routers and switches. These devices operate as basic forwarding components without inbuilt vendor-specific software or management control, and are remotely controllable via open interfaces. These switches have flow tables, which contain many flow rule entries that are placed by the SDN controller. Various fields such as; the match field, the instructions field, and the counter field are the flow table's most significant fields. The flow table's entire structure is depicted in Fig 2.2. Along with

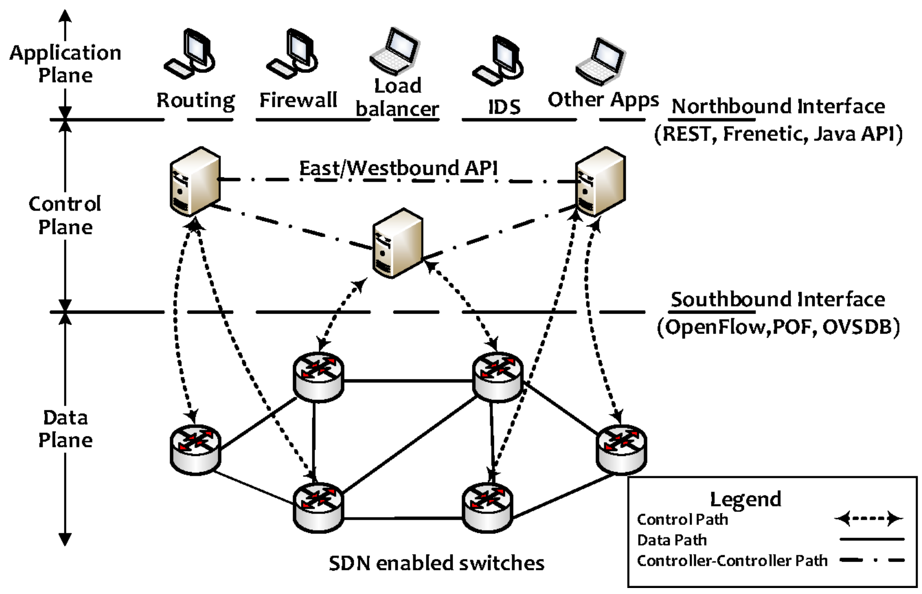
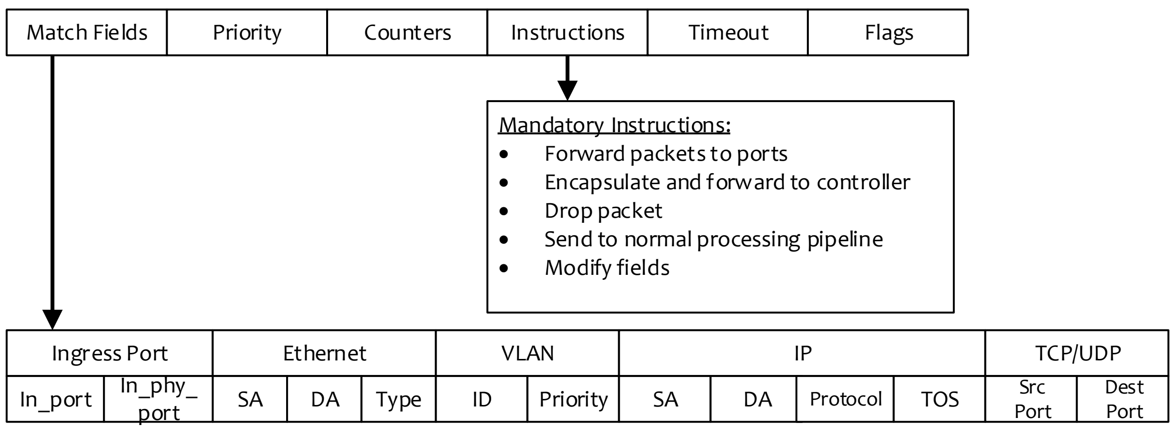


FIGURE 2.1: THE SDN ARCHITECTURE

Flow Tables, SDN switches contain Meter Tables, Group Tables, and Flow Buffers [19].



Mandatory Instructions:

- Forward packets to ports
- Encapsulate and forward to controller
- Drop packet
- Send to normal processing pipeline
- Modify fields

Ingress Port		Ethernet			VLAN		IP				TCP/UDP	
In_port	In_phy_port	SA	DA	Type	ID	Priority	SA	DA	Protocol	TOS	Src Port	Dest Port

FIGURE 2.2: OPENFLOW VERSION 1.5 FLOW-TABLE'S STRUCTURE IN [1]. 40 X MATCH FIELD EXISTS IN OPENFLOW 1.3 [2]

2.1.2 Control Plane

A pivotal pillar of SDN architecture is the control plane. The controller is located centrally (logically) and connected to various devices (switches) in the data plane; it also maintains a global view of the entire network [33]. The controller benefits from having a global view because it enables effective network management and rapid response to network changes. Additionally, the controller's security has a direct impact on the data plane's data forwarding devices. If a controller is compromised, it affects the entire SDN network, including multiple switches. As a result, from an attacker's

TABLE 2.1: FEATURE BASED COMPARISON OF SDN CONTROLLERS

SDN Controller	Architecture	Documentation	Language	User Interface	Platform Support	NBI TLS Support
ONOS [21]	Distributed	Good	Java	Web-Based	Windows, Linux and MAC OS	No
OpenDaylight [22]	Distributed	Good	Java	Web-Based	Windows, Linux and MAC OS	Yes
FloodLight [23]	Centralized	Good	Java	Web-Based	Windows, Linux and MAC OS	No
RYU [24]	Centralized	Medium	Python	Web-Based	Mostly supported on Linux	No
POX [25]	Centralized	Poor	Python	GUI	Windows, Linux and MAC OS	No
NOX [26]	Centralized	Poor	C++	GUI	Most supported-on Linux	No
Beacon [27]	Centralized	Medium	Java	Web-Based	Windows, Linux and MAC OS	No
Trema [28]	Centralized	Medium	C/Ruby	CLI	Linux	No
PANE [29]	Distributed	Medium	Java	CLI	Linux	No
Onix [30]	Distributed	Poor	C++	CLI	Linux	No
Kandoo [31]	Hierarchical / Distributed	Good	C++/Python	CLI	Linux	No
MAESTRO [32]	Distributed	Medium	Java	CLI	Linux	No

perspective, the highest priority is to compromise the OpenFlow controller in order to conduct malicious activities on the network.

The controller(s) apply flow rules to the data plane devices in response to instructions from the application plane's third-party applications. The SDN controller communicates with the devices on the data plane via the SBI interface using the OpenFlow protocol [4], [34]. ONOS [21], OpenDayLight [22], Floodlight [23], RYU [24], POX [25], and NOX [26] are just a few of the well-known SDN controllers available. In [35], [36], [37], and [38], comparative studies of famous controllers are presented. Table 2.1 presents a feature-based comparative evaluation of a few famous controllers gathered from the aforementioned studies.

Different techniques can be used to deploy controllers in SDN. [39] and [40] have discussed a variety of topology-related approaches for deploying SDN controllers:

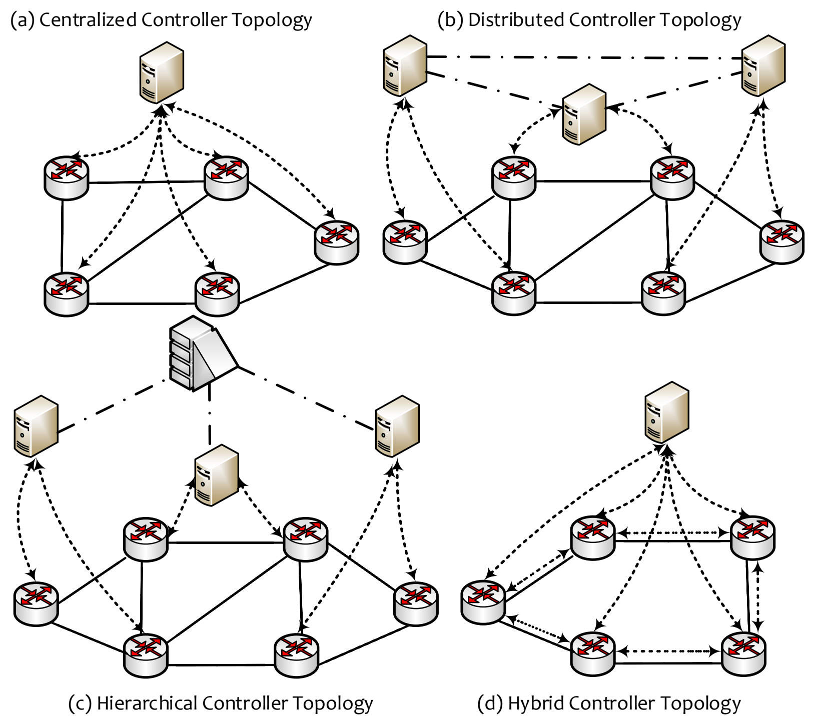

(a) Centralized Controller Topology
(b) Distributed Controller Topology
(c) Hierarchical Controller Topology
(d) Hybrid Controller Topology

FIGURE 2.3: SDN BASED MULTIPLE CONTROLLERS LAYOUTS.

1. Centralised Controller Topology
2. Distributed Controller Topology
3. Hierarchical Controller Topology
4. Hybrid Controller Topology

Fig 2.3 illustrates these SDN-based multiple controller topologies. Each design has some advantages and disadvantages in terms of control plane scalability. Each controller in a multiple controller design (Distributed or Hierarchical) is accountable for overseeing a subset of the network's switches and communicates with the others via an East / Westbound interface.

2.1.3 Application Plane

The ability of third-party enterprises to develop and deploy network applications is one of SDN's core properties. The application plane is where these third-party applications reside. Typically, these applications perform a specific task and operate at a higher level of abstraction than the controller. Through the use of NBIs, applications implement their business logic on data plane devices via controllers. Applications can use the NBIs to collect information from the SDN controller and to send instructions to the SDN controller. The control plane abstracts the data plane's physical resources (routers and switches) from the application plane and applications can only use the services / APIs of the SDN controller to enforce policies on data plane devices via flow rules. The controller's connectivity to applications is two-tiered. The first is the data plane connection that connects user application traffic to the network, while the second is a management-control connection that connects the application to the network. Both tiers are in charge of data traffic transmission across the network. Authorized applications establish management-control sessions with the SDN controller via the NBI in order to invoke services or modify the state of network resources, [41]. Additionally, applications collect information about the network's current state from the controller and make changes in response to the network's updated conditions. Network virtualization, security applications, network monitoring, and traffic engineering are just a few examples of third-party applications.

2.1.4 Southbound Interface

The SBI serves as the communication channel between the data and control planes. By handling flow entries on data plane devices, this interface enables the SDN controller to monitor and manage network behavior. There are numerous southbound APIs available for SDN controllers to communicate with devices, including OpenFlow [4], Protocol Oblivious Forwarding (POF) [42], OpenvSwitch Database (OvSDB) [43], OpFlex [44], OpenState [45], NETCONF [46], and OF-CONFIG [47]. While OpenFlow is considered as the de facto standard for SBI, having been standardized by the Open Networking Foundation (ONF) [34].

According to [48], OpenFlow is by far the most extensively used SBI protocol in SDN. It has evolved, and the most recent version is OpenFlow 1.5, as cited in [1]. It facilitates communication between the controller and the data plane's physical devices. The OpenFlow protocol is designed to be used in conjunction with the Transmission Control Protocol (TCP). Its specification recommends that its communications be encrypted using Transport Layer Security (TLS), which is an optional feature. Controllers listen on TCP port 6653 in order to establish an OpenFlow connection with the switches. The

OpenFlow protocol manages communication between SDN switches and controllers via a variety of messages (s). OpenFlow's most frequently used messages for managing flow rules and switches are listed below

1. ***Features***: Controller queries switch features
2. ***Configure***: Controller queries/set switch configuration parameters
3. ***Modify-state (Flow_Mod)***: Add, delete, modify flow entries in the OpenFlow tables
4. ***Packet_OUT***: Reply of the Packet_IN message
5. ***Packet_IN***: Transfer packet (and its control) to controller.
6. ***Flow-removed***: To delete Flow table entry at particular switch
7. ***Port status***: Information regarding port status

Flow tables contain flow rules in the form of a match field, a priority column, a counter, instructions, a timeout column, and flags. Header matching is performed on incoming packets using the match fields as depicted in Fig 2.2. When all fields in the flow entry matches all fields in the incoming packet's header, a match occurs. Following a successful match, one of the instructions specified against the flow table entry should be executed, along with the associated action(s) over the packet. If an incoming packet does not conform to the switch's flow rules, the switch generates the Packet IN message and forwards it to the SDNcontroller for more instructions. This Packet IN message holds details about the packet's header, the switch's ID, the switch's input port number, and the switch's buffer ID. In response to the Packet IN message, the controller generates a Packet OUT message containing the switch ID, the buffer ID, and the actions that should be performed on the Packet IN message in question (e.g., forward to egress port, push VLAN ID, strip VLAN ID or drop etc.,). To manage the flow of successive packets, the controller generates a Flow Mod message that instructs all switches engaged in the active data transmission path to install flow rules; this mechanism helps to reduce communication between the controller and switches by avoiding unnecessary Packet_IN messages being forwarded to the controller [49].

2.1.5 Northbound Interface

The NBI serves as the link between the control and application planes. Controllers enable applications to communicate with network control in order to manage services and associated resources via NBI. In other words, this interface enables trusted applications to programme the network and to query the network's state, services, and information. In comparison to SBI, there is no standard for NBI. Due to the NBI's lack of standardisation, almost every controller provides its own unique set of NBI services to enable software designers to develop network applications like routing, switching, firewall, and network monitoring. NBI is primarily a software ecosystem, and it is implemented using a variety of methodologies, including the following:

- Establishing fine-grained application interfaces.
- Developing ad-hoc Northbound API for specific controllers.
- Offering intent-based NBI that encapsulates network resources and features,and conveys the application purpose on what to do, but not how to accomplish it.
- Transform application necessities into lower-level service intents.
- SDN programming languages to be used.

With the aforementioned multiple methodologies in mind, various types of NBIs are now available in SDN, which can be classified as *General APIs*, *Programming Languages*, and *Intent-based*NBIs [50], [51]. Due to the lack of a standardised NBI, many controllers have opted for a REST or RESTful NBI [6]. The Open Service Gateway Initiative (OSGi) framework is also available for applications and programmes that run in the same address space as the controller (same physical or virtual machine), whereas the REST API is used by applications that run in the same address space as the controller or on a different machine, [52].

2.1.6 Eastbound and Westbound Interface

In the SDN paradigm, the controller is referred to as the "brain" of the SDN network due to its central location between network devices and applications. However, a single centralised controller is limited in its ability to manage a large number of switches.

Distributed controllers are becoming an essential component of today's large-scale networks, owing to the exponential growth of application services and network devices worldwide. Each controller in a distributed SDN environment is responsible for controlling its own domain via various forwarding devices. These dispersed controllers must share information about their individual domains in order to maintain a global perspective of the network. East/westbound APIs are a specific type of interface that distributed controllers use to communicate with one another and with legacy networks. To be more exact, Eastbound APIs govern inter-controller communication between SDN controllers, whereas Westbound interfaces leverage established networking protocols such as Border Gateway Protocols (BGP) to provide connection between SDN and older networks. Eastbound interfaces include [70] and [30], whereas westbound interfaces include [71] and [72].

In the SDN paradigm, the controller is referred to as the "brain" of the SDN network due to its central location between network devices and applications. However, a single centralised controller is limited in its ability to manage a large number of switches.
Distributed controllers are becoming an essential component of today's large-scale networks, owing to the exponential growth of application services and network devices worldwide. Each controller in a distributed SDN environment is responsible for controlling its own domain via various forwarding devices. These dispersed controllers must share information about their individual domains in order to maintain a global perspective of the network. East/westbound APIs are a specific type of interface that distributed controllers use to communicate with one another and with legacy networks. To be more exact, Eastbound APIs govern inter-controller communication between SDN controllers, whereas Westbound interfaces leverage established networking protocols such as Border Gateway Protocols (BGP) to provide connection between SDN and older networks. Eastbound interfaces include [70] and [30], whereas westbound interfaces include [71] and [72].

2.2 Enterprise Integration Patterns (EIP)

The concept of EIP was introduced in 2004 [8], which develops standard design outcomes to connect different applications running in enterprise. These patterns are frequently used to connect business processes with enterprise systems. The demand for these patterns arises because there is no single application in a business network that provides all of the services an organisation want to offer its clients and workers. Furthermore, the vast majority of applications are spread by nature. As a result, multiple services are offered via a variety of applications written in a different language and platforms (operating systems), which may be distributed across multiple locations. When one of these application requests a service from another, a formal inter-application communication protocol must exist.

In the book [8], the authors identified four primary techniques for establishing the inter-application communication infrastructure: Messaging System, Shared Database, File Transfer, and Remote Procedure Invocation. Where messaging systems shown to be a superior implementation when the asynchronous aspect of message exchange is considered.

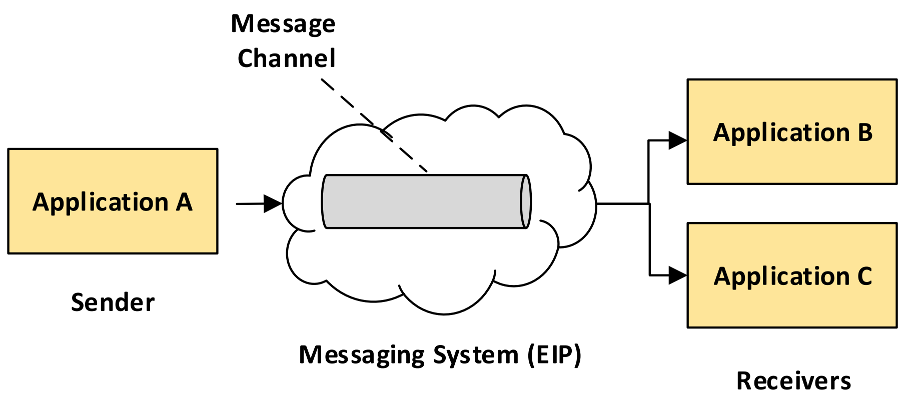


FIGURE 2.4: A GENERIC MESSAGING SYSTEM

Because message-oriented design is based on asynchronous message exchange, it promotes loose coupling and increases communication reliability by eliminating the requirement for two programmes to run concurrently. The messaging system is in charge of data transmission between applications, which means that the programme may concentrate entirely on the content of the data being communicated without being distracted by data transmission functionality. Fig. 2.4 represents a simple messaging system, where an application (sender) using a messaging system transfers packets to the other applications (receivers) reliably, frequently, and asynchronously. The communicating applications can have different messaging formats and it is the job of a message translator (one of the concepts of the messaging system) to convert the message from one format to another so that each application can interpret the message according to its own format.

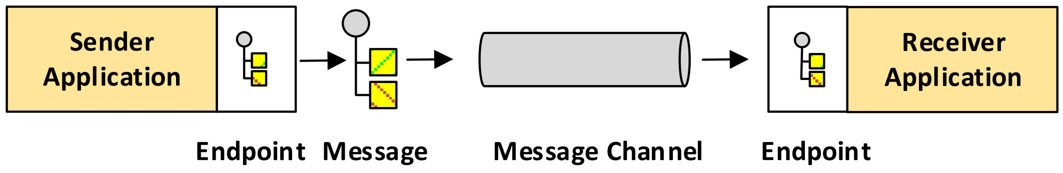


FIGURE 2.5: A SIMPLE MESSAGING SYSTEM COMPRISING OF THREE CORE CONCEPTS (MESSAGE, MESSAGE CHANNEL, AND ENDPOINT)

Messaging systems are composed of a number of fundamental ideas, including message channels, message,pipesand filters, message routers, message translators, and message endpoints. Each of the main principles is subdivided into various subcategories based on the demand for message exchange. We used only three fundamental elements into this research:

- *Message*; a data packet that must be delivered to the receiver(s).
- *Message Channel*; applications exchange messages by sending and receiving data via a channel that connects the sender and receiver.
- *Message Endpoint*; an interface that enables an application to send and receive messages via the messaging system.

Apart from their asynchronous character, message channels are distinct from applications and ensure the delivery of reliable packets. Fig. 2.5 illustrates a Messaging System composed of a Message Endpoint, a Message Channel, and a Message. In this Fig. 2.5, an application (sender) uses the message endpoint to relay a message on the message channel. A receiving application consumes a message from the message channel via its message endpoints on the receiver side.

2.3 Virtual Local Area Networks (VLANs)

Local Area Network (LAN) is a collection of hosts that span over a limited geographical area to share resources (data, information or hardware) between hosts and are commonly deployed in offices, universities and homes. The network connection between the hosts is through a data link layer switch having a single broadcast domain (broadcast frame sent from one host will be received by all other host connected on LAN). A single large broadcast domain can be partitioned into smaller domains using a router to route packets between LANs and a switch(es) for each LAN. However, this partitioning would leads to the requirement of extra financial resources.

Virtual Local Area Networks (VLANs) were created to circumvent the broadcast domain issue. VLANs enable the grouping of hosts on distinct physical networks into a single broadcast domain (logically) or the segmentation of a single physical network into numerous broadcast domains. The obvious advantage of adopting VLAN is that it provides security for hosts within an organisation. All ports on the switch are required to be configured either of the following type.

- ***Access Port*** A port that is directly linked to the host and to be configured on a single VLAN..

- ***Trunk Port***A port that connects different switches and must be a member of every VLANs, and forwards traffic from all VLANs defined among switches. [73].

IEEE 802.1Q standard [74] Specifies the VLAN management. Moreover, VLAN membership can be obtain using following four different ways

- *Port-based classification:* Configuration of VLANs based on switch ports.

- *MAC-address classification:* Configuration of VLAN ID based on MAC addresses.

- *Protocol-based classification:* Configuration of VLAN ID based on layer 3 / 4 protocols.

- *IP subnet based classification:* Configuration of VLAN based on IP address / subnet.

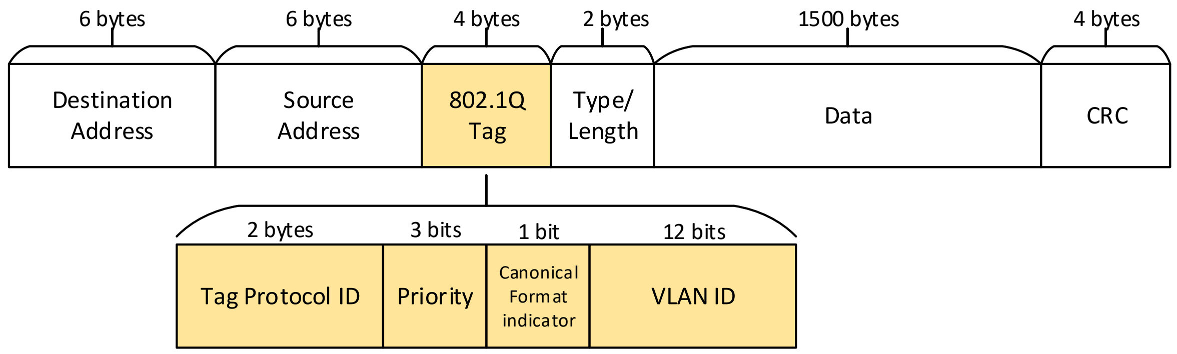


FIGURE 2.6: IEEE 802.1Q FRAME FORMAT

2.4 Understanding Simple SDN Topology

To understand how SDN network communicates, we explain it through a simple SDN topology illustrated in Fig. 2.7. Here Host-1 having IP address 192.168.10.1/24 initiates communication with Host-2 with IP address 192.168.10.2/24. Both hosts are connected to separate switches which support OpenFlow communication. Fig. 2.7 illustrate When Host-1 begins communication with Host-2, a series of activities is performed to install rules on OF switches. The green arrows (1, 4, 7) indicate the data path message route, the red arrows (2, 5) indicate the Packet IN message, and the blue arrows (3, 6) indicate the control path's Flow Mod message, which is used to set flow rules.. The whole communication is divided into following steps.

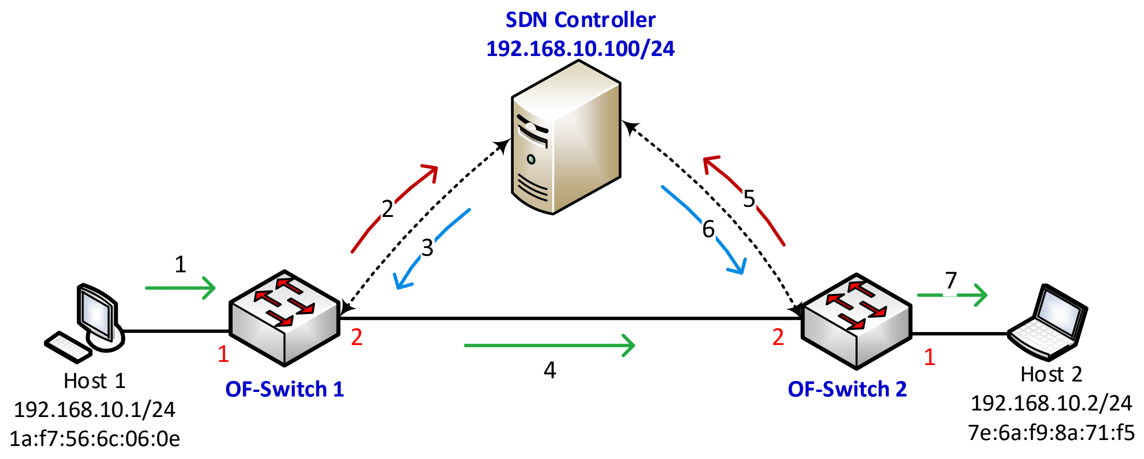
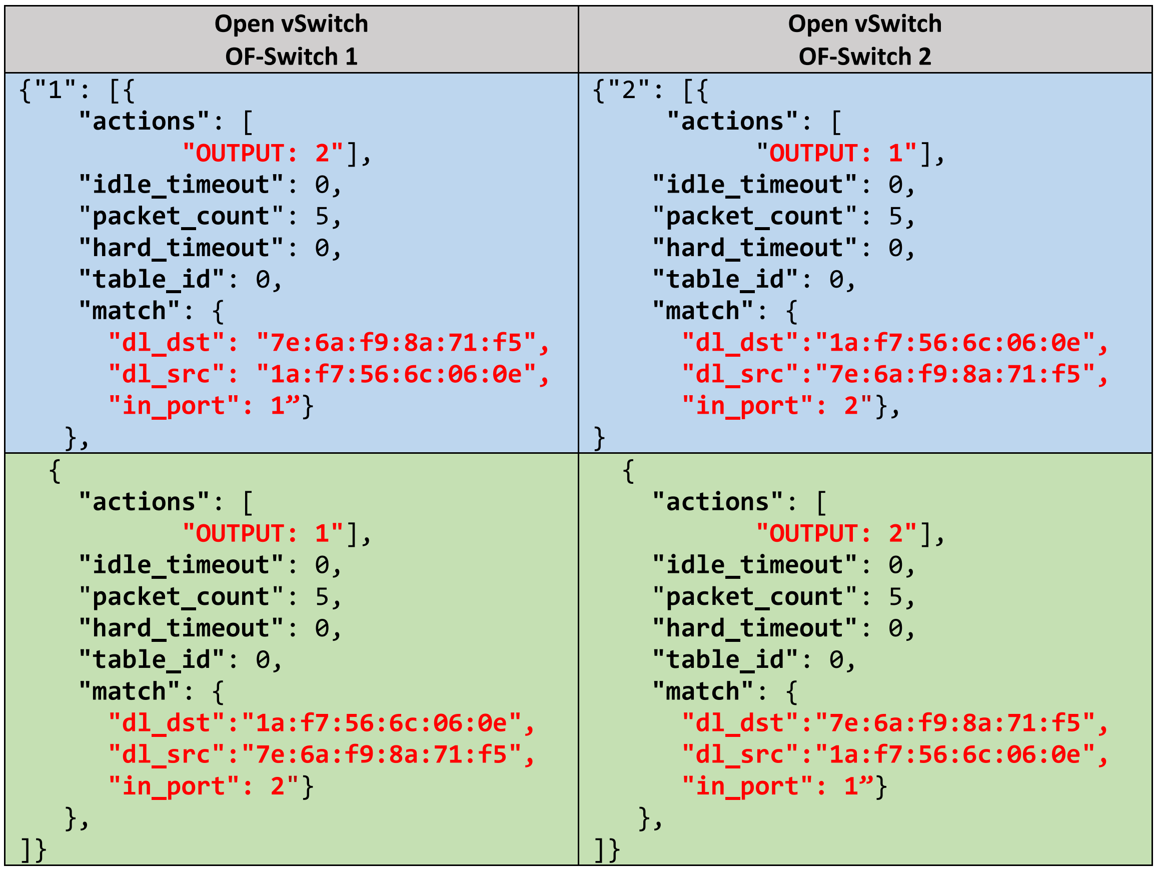


FIGURE 2.7: A SIMPLE EXAMPLE TO HIGHLIGHT THE COMMUNICATION BETWEEN TWO HOSTS IN SDN ENVIRONMENT.

Step 1: When Host-1 send data packets having Host-2 as destination, these packets are received at OF-Switch 1 through port 1.



Open vSwitch
OF-Switch 1

```
{"1": [{
    "actions": [
           "OUTPUT: 2"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst": "7e:6a:f9:8a:71:f5",
      "dl_src": "1a:f7:56:6c:06:0e",
      "in_port": 1”}
   },
  {
    "actions": [
           "OUTPUT: 1"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst":"1a:f7:56:6c:06:0e",
      "dl_src":"7e:6a:f9:8a:71:f5",
      "in_port": 2"}
   },
]}
```

Open vSwitch
OF-Switch 2

```
{"2": [{
     "actions": [
           "OUTPUT: 1"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst":"1a:f7:56:6c:06:0e",
      "dl_src":"7e:6a:f9:8a:71:f5",
      "in_port": 2"},
}
  {
    "actions": [
           "OUTPUT: 2"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst":"7e:6a:f9:8a:71:f5",
      "dl_src":"1a:f7:56:6c:06:0e",
      "in_port": 1”}
   },
]}
```

FIGURE 2.8: FLOW RULES INSTALLED DURING THE COMMUNICATION OF HOST-1 AND HOST-2 ILLUSTRATED IN FIG. 2.7. HERE RULES MENTIONED IN THE BLUE BOXES REPRESENTS THE FORWARDING PATH AND RULES FOR REVERSE PATH IS LISTED IN GREEN BOXES.

Step 2: As OF-Switch 1 has no prior knowledge of Host-2, it generates a control messages known as *Packet_IN message* to SDN controller that contains the data packet sent by Host-1.

Step 3: Upon receiving the Packet_IN message, the SDN controller through application, for example *switching application* can decide that on which port it has to forward this packet. After calculating the path, SDN controller installs a flow rule on OF-Switch 1 using another control message known as *Flow Mod message*. Rule listed in Blue Box under OF-Switch 1 of Fig. 2.8.

Step 4: Once the flow rule is installed on OF-Switch 1 regarding the destination Host-2, all traffic is forwarded to port 2.

Step 5: Upon receiving the data packets from OF-Switch 1, the OF-Switch 2 also generates Packet_IN message containing the data packet sent by Host-1 for Host-2.

Step 6: Once again, SDN controller through its application decides the destination port number and installs a flow rule through Flow Mod message on OF-Switch 2. Rule listed in Blue Box under OF-Switch 2 of Fig. 2.8.

Step 7: After the installation of flow rule, the OF-Switch 2 now forwards the traffic of Host-2 coming from port 2 to port 1.

Step 8: When Host-2 receives data packets from Host-1, it then generates a response packet. This response packet will also follow the same procedure to reach Host-1 using flow rule installed by SDN controller. Flow Rules installed on both switches for reverse path between Host-2 and Host-1 are listed in Fig. 2.8 inside green boxes.

2.5 Summary

In this chapter, we have provided the overview of the technologies used in this dissertation. We used multiple technologies such as, Software Defined Networks, Enterprise Integration Patterns, and Virtual Local Area Networks to propose a framework that has built-in security. In the last sections, we present a basic example to help the reader understand how an SDN network interacts using flow rules.In this chapter, we have provided the overview of the technologies used in this dissertation. We used multiple technologies such as, Software Defined Networks, Enterprise Integration Patterns, and Virtual Local Area Networks to propose a framework that has built-in security. In the last sections, we present a basic example to help the reader understand how an SDN network interacts using flow rules.

Chapter 3

Literature Review

This chapter highlights the existing work related to the adoption of SDN in enterprise network. The proposed communication framework incorporates multiple technologies such as VLANs, ARP protocol, service registry, and anonymous communication. We have organized this literature review into multiple sections where each section presents the related work concerning to each technology incorporated in the research. The summary of the existing work pertaining to each technology using in this proposed framework is depicted in Fig. 3.1.

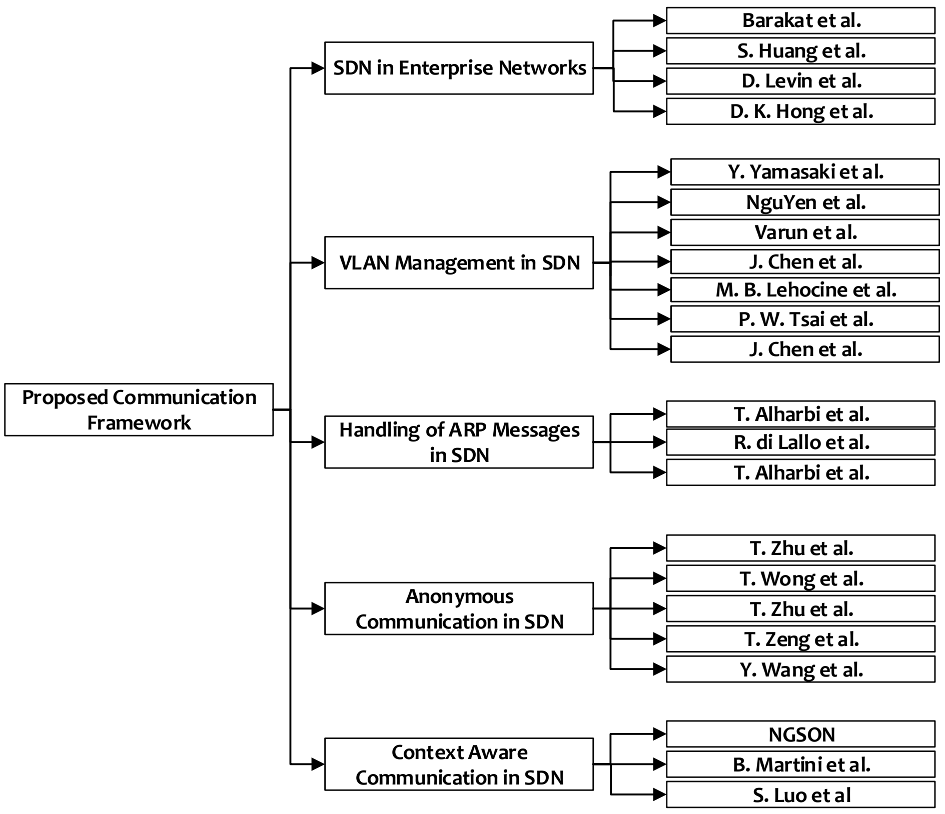


FIGURE 3.1: SUMMARY OF THE EXISTING LITERATURE PERTAINING TO THE DEPLOYMENT OF SDN PARADIGM IN ENTERPRISE NETWORKS

TABLE 3.1: EFFORTS TO DEPLOY SDN IN ENTERPRISE NETWORKS

Literature	**Research Motivation**	**Proposed Solution**	**Cons**
Barakat et al. [75]	Different methodologies for converting traditional networks to SDN-based networks are compared	Proposed four different strategies to integrate SDN with fewer upgrades. This approach is extension of on the work done by [76].	Integration issues between legacy and SDN switches
S. Huang et al. [77]	Migration from a traditional network to an SDN network	Proposed a scheme to integrate traditional Networks and SDN	Their plan necessitates careful integration, routing, and network administration.
D. Levin et al. [76]	Stage wise deployment of SDN in enterprise networks with fewer updates of switches	Proposed an approach where transition of network from legacy to SDN switches only requires handful of switches to be replaced so that at least one SDN switch is placed between every source and destination hosts.	No experimentation is done to support their approach
D. K. Hong et al. [78]	Transitional issues in adopting SDN in enterprise and ISP networks	In a hybrid SDN network, gradual deployment of SDN switches is proposed, with just 20% of the switches required to be converted to SDN-based switches	Requires careful planning to deploy SDN switches among legacy switches

3.1 Deployment of SDN in Enterprise Networks

Recently, researchers and academicians have proposed different solutions to adopt the SDN paradigm in enterprise networks. One of the most prominent solutions to deploy SDN in existing networks is the hybrid or incremental approach. The hybrid or incremental approach, where few switches are upgraded to SDN-based switches and remaining switches are left as traditional switches, poses multiple challenges such as integration of SDN switches in traditional networks, routing decisions and designs, and management. This approach is presented by [75, 77, 76] and [78] in their respective articles.

Barakat et al. [75]did a study in which they compared several ways for converting traditional networks to SDN. They offered a strategy based on the comparison of techniques in which SDN network processes the traffic of legacy networks with minimal changes. Likewise, [77] abstracts a hybrid solution to incorporate SDN network and legacy network. Both SDN and traditional switches are included in the logical SDN network.

In [76], authors have proposed an approach to integrate SDN and legacy switches. The main idea behind this approach is that for every communication between any source and destination, the path of packet flow must include at least one SDN switch. For that purpose, they introduce cost aware optimization tool for operators to determine the topological location of SDN switches in enterprise network. This approach of at least one SDN switch between source and destination path is used to control traffic forwarding decisions through SDN controller.

Hong et al. [78] proposes a hybrid SDN Network by introducing incremental placement of SDN switches. In their approach, authors have replaced a subset of traditional switches by OF operated switches without updating the configurations of existing traditional switches. Their proposed solution comprises of four components, which are SDN deployment planner, global topology viewer, Traffic Engineering (TE) module, and fail-over module. The SDN deployment planner component decides which switch be upgraded from legacy to OpenFlow enabled switch. By Using the global visibility module, authors maintain the global view of hybrid network which includes both SDN and legacy switches. Forwarding paths are controlled and monitor by TE module by minimizing the link utilization to avoid congestion. In their approach, the legacy switches uses the Link-State protocol to forward traffic and once the traffic reaches SDN based switch, the SDN controller installs flow rules using the TE module. The authors claim that using their approach only 20% of the switches are required to be upgraded on SDN based switches.

3.2 VLAN Management in SDN

VLAN tagging can be advantageous for security and network administration in an enterprise network. The studies cited in [79], [80], [81], [82], [83], [84], and [85] all included VLANs in their SDN implementations. [79] researched one of the earliest

TABLE 3.2: LITERATURE RELATED TO VLAN MANAGEMENT IN SDN

Literature	**Research Motivation**	**Proposed Solution**	**Cons**
Y. Yamasaki et al. [79]	VLAN configuration in Campus Networks is laborious	Similar to VLAN IDs they created group IDs (GIDs) and assigned every host a GID. The communication is only allowed if the GIDs of both communicating hosts are matched.	Their approach does not use the VLAN Tag information in packet header. So their solution is not purely based on VLANs
NguYen et al. [80]	VLANs configuration is complex, time-consuming and error-prone process	Configuration of Dynamic and Static VLANs are done through a GUI based application	The focus of their research is more concerned regarding the user mobility than to a dynamic installation of VLANs.
Varun et al. [81]	Implementation of VLANs in SDN	Developed an application that allows SDN controller to provide communication using VLAN Tagging	Their solution only support static VLANs and requires prior knowledge of topology information like access ports, trunks ports and VLAN IDs assigned to each port.
J. Chen et al. [82], [83]	Link and switch failure recovery using VLANs	A unique VLAN ID along with the backup path is calculated for each link.	Their mechanism is efficient with respect of link failures only.
M. B. Lehocine et al. [84]	Segmentation of single VLAN into multiple isolated VLANs for the purpose of complete isolation among hosts on same subnets	Used double VLAN tagging to identify isolated VLAN ID and Primary VLAN ID	Their study was to compare flexibility in deploying network application in SDN as compared with Autonomic Network Management (ANM) solutions designed for traditional networks

solutions for VLANs in SDN. They discovered that administering several VLANs in campus networks is time consuming and has limitations due to the limited number of VLAN IDs. They presented an open-source management solution called OpenFlow. They used Group IDs (GIDs) rather than VLANs to analyse incoming traffic and determine whether or not the connection should be allowed. When a packet IN message is received at the controller for a flow rule request, the source and destination group IDs are compared against those recorded in the GID database. Communication is permitted only if the GIDs of the communicating hosts match. Their solution circumvents the limitation on the number of VLANs and simplifies network configuration. Additionally, their approach does not make advantage of the VLAN information contained in the packet header.

Nguyen et al. [80] presented an SDN-based architecture for enterprise and campus networks. Their solution enables communication over VLANs and includes an application for managing VLANs in an enterprise network. Network administrators can configure static and dynamic VLANs using their application. Two hosts with the same VLAN ID can connect with one another regardless of their physical location, thanks to their dynamic VLAN configuration approach. However, their dynamic VLAN configuration is more concerned with the user's mobility than with the dynamic configuration of the VLANs. Similarly, [81] built a RYU controller-based application to handle VLAN tagging in SDN. They validated their technique by running it through three distinct network topologies.

Chen et al. presented a mechanism for recovering from connection failures based on VLAN-tag in [82] and then in [83]. Their mechanism determines a backup path and assigns a unique VLAN ID to each network link. Prior to the link loss, a backup path is computed. They switched backup paths automatically, without involving the controller, by employing group tables. Their mechanism is effective only in the case of link failures.

The authors of [84] exploited the concept of private VLANs to segment a single VLAN into numerous isolated segments and provided a method for segmenting larger SDN networks via VLANs. They employed double VLAN tagging in their technique, with the inner VLAN Tag defining the isolated VLAN and the outside VLAN Tag identifying the principal VLAN.

A dynamic VLAN mapping enables traffic to be sent across OpenFlow networks and non-OpenFlow networks. VLAN tags on the inside and outside of the network were used to establish a connection between two OpenFlow networks.

TABLE 3.3: LITERATURE RELATED TO HANDLING ARP MESSAGES IN SDN

Literature	**Research Motivation**	**Proposed Solution**
Di Lallo et al. [86]	Handling ARP in SDN	Fake MAC address is send by controller in ARP reply message to avoid the ARP broadcast. Later installs correct flow rule with correct MAC address
T. Alharbi et al. [87]	Handling ARP request using OF switches	Using the payload of ARP request message, OF switches convert request message into ARP response message by rewriting the request message fields

3.3 Handling of ARP in SDN

To handle the ARP requests in this research, the articles mentioned in Table. 3.3 were reviewed. The first approach of handing ARP packets was proposed by Di Lallo et al. [86]. In their approach, to avoid the ARP broadcast storm the SDN controller replies with the fake MAC address of the destination as soon as it receives the ARP request message from the SDN switch. After sending the fake MAC address, the SDN controller then finds the actual MAC address of the destination by requesting the switches. Upon receiving the actual MAC address, the SDN controller deploys the flow rule between the source and destination hosts.

In the article [87], authors have presented an efficient approach to resolve ARP requests directly from the SDN switches. In their approach, the SDN switch generates an ARP response message without contacting the SDN controller. The ARP response message is generated by examining the payload of the ARP request message and installing a rule that converts an ARP request to an ARP response message.

TABLE 3.4: LITERATURE RELATED TO SECURING ENTERPRISE NETWORKS USING ANONYMOUS COMMUNICATION

Literature	**Research Motivation**	**Proposed Solution**	
T. Zhu et al. [88],[89]	Providing anonymous communication in data centers using SDN	During the packet transmission, each switches replaces the source and destination address.	Requires collision avoidance mechanism to overcome the flow rules conflicts.
Y. Wang et al. [90]	Providing untraceable and anonymous packet forwarding mechanism	Proposed URI mechanism to conceal the information such as MAC, network and transport layer of network traffic.	-
Wong et al. [91]	Anonymous communication in SDN	Provide anonymity of both IP and MAC addresses between hosts	Their solution requires collision avoidance mechanism. As collision between multiple flow rules is possible.
T. Zeng et al. [92]	SDN introduces additional challenges of host anonymity due to global view	Their approach is based on mixnets in which clients send data using multiple mix nodes	Their solution more suited for internet traffic and is not suitable for enterprise networks

3.4 Anonymous Communication in SDN

To address the issue of identity or information theft, numerous anonymous communication methods have been introduced, including TOR [93], I2P [94], and Crowds [95]. For instance, TOR employs an onion routing method to conceal the sender and receiver's identities, with each router knowing only its successor and predecessor. Additionally, people are attracted toward identity hiding via cellular communication and are utilizing smartphone programs that enable anonymity [96]. Sadly, these technologies were developed for Internet traffic and are therefore ineffective for time-sensitive applications and organizational networks.

Zhu et al. originally presented their work on anonymous communication in data centres using SDN in [88] and again in [89]. They began by establishing the threat model, which describes how an attacker affects an SDN switch, a host, or network traffic. They change the source and destination IP addresses of packet, as packets travels from source to destination host. Once packet arrives at the target host switch, the fake address is replaced with the genuine one. Their approach is throughput efficient ; nonetheless, it necessitates the use of a collision-avoidance system to handle flow rule conflicts, as two flow rules may use the same fake address to carry packets from distinct hosts.

The developers of [90] proposed the U-TRI strategy for disguising the network's genuine traffic. Not only MAC layer information is obfuscated by U-TRI, but also network layer information. By substituting unstructured random numbers for MAC addresses and keeping the mapping between the real address and the fake information, MAC layer IDs are concealed.

In 2018, Wong et al. [91] published an article describing an anonymous communication in SDN network. Their technique protects both IP and MAC addresses against host-to-host disclosure. They verified their method in cases involving both one-way and two-way anonymity. However, the source host has previous knowledge of the destination host's genuine IP and MAC addresses through their technique of anonymous communication. Additionally, their technique precludes the use of a collision prevention device. As a result of the risk of clashing several flow rules.

3.5 Context Aware communication in SDN

Context awareness is helpful in network communication because end users care only about the information they require but not about who or how they obtained them. IEEE defined Next Generation Service Overlay Networks (NGSON) for the benefit of network operators, service and content providers, and end users. For further information, see [100]. To offer the best possible user experience, NGSON partitions service-related operations from transport-related tasks and controls application services delivered by various networks via an overlay. SDNs, such as NGSON, can abstract away the distinction between application services (service discovery, registration, and composition) and transport capabilities. Numerous research, for example, reference

TABLE 3.5: LITERATURE RELATED TO SECURING ENTERPRISE NETWORKS USING ANONYMOUS COMMUNICATION

Literature	**Research Motivation**	**Proposed Solution**
NGSON [97]	Context-aware services in SDN using IEEE NGSON	Proposed the extension of NGSON that uses service oriented abstraction and programming at the control layer to optimize deliver process.
B. Martini et al. [98]	Context-aware delivery of application services in SDN	Proposed a service oriented SDN controller that allows service chaining of overlay networks to provide context aware delivery of application services and helps operators to monetize their network in different way.
S. Luo et al. [99]	Context-aware traffic forwarding scheme for SDN	Introduced a service layer between the application and control layers of SDN to provide context-aware traffic forwarding scheme.

[98] and [99], have proved the usage of SDN to offer context-aware services.

The authors of [99] presented context-aware traffic forwarding services in SDN to demonstrate how the SDN paradigm can be used to provide context. They used NGSON to create a service layer between SDN's traditional application and control layers. This service layer was developed to enable the integration of service-oriented features such as service discovery, service registration, and service orchestration inside an SDN environment. management.

3.6 Summary

In this chapter, we have highlighted the existing work related to deployment of SDN in enterprise networks. Moreover, as the proposed framework incorporates multiple technologies to provide secure communication, we have divided this chapter into multiple sections where each section highlights existing work concerning to each technology. we have highlighted the existing work related to deployment of SDN in enterprise networks. Moreover, as the proposed framework incorporates multiple technologies to provide secure communication, we have divided this chapter into multiple sections where each section highlights existing work concerning to each technology.

According to the literature review, the majority of the research is geared toward enterprise adoption of SDN, with an emphasis on utilising existing infrastructure. However, our study proposes a method for organisations to migrate to an SDN-based network based on Enterprise Integration Patterns (EIP) for increased efficiency, message consistency, built-in security, context-aware services, anonymous communication, and integration of existing business applications.

Chapter 4

Proposed Communication Framework

This chapter presents an overview of the design and implementation of the proposed communication framework. This communication framework incorporates Enterprise Integration Patterns (EIP) in an SDN environment to support communication between applications (services) with different backgrounds such as languages, platforms, data types, and interfaces. In addition, utilizing the benefits of SDN, this framework also provides built-in security, services that are context-aware, and host anonymity within the enterprise network.

The following sections introduce the architecture of the proposed communication framework in an SDN environment, followed by the technical details of the proposed modules of the framework that are required to provide necessary communication in an enterprise environment.

4.1 Framework Architecture

The communication framework's architecture is comprised of various SDN application modules developed for the RYU SDN controller [24], one of the most popular SDN controllers in the SDN world. Fig. 4.1 illustrates the general RYU controller architecture, as well as the modules depicted in yellow boxes, which are the result of this research. The framework includes the following modules: *Host Registration*: for the purpose of registering all hosts connected to the network; *Service Registry*: holds list of all the services available to hosts on the network; *IP Address Mapping*: compares hosts' genuine IP addresses to their hoax IP addresses; *Adaptive VLAN Management*: a module that facilitates reactive VLAN creation and deletion; *ARP Request Resolution*: a module that enables the resolution of gateway Ethernet addresses; and *Packet Forwarding*: a module that enables path identification between source and destination hosts. The following subsections detail each module.

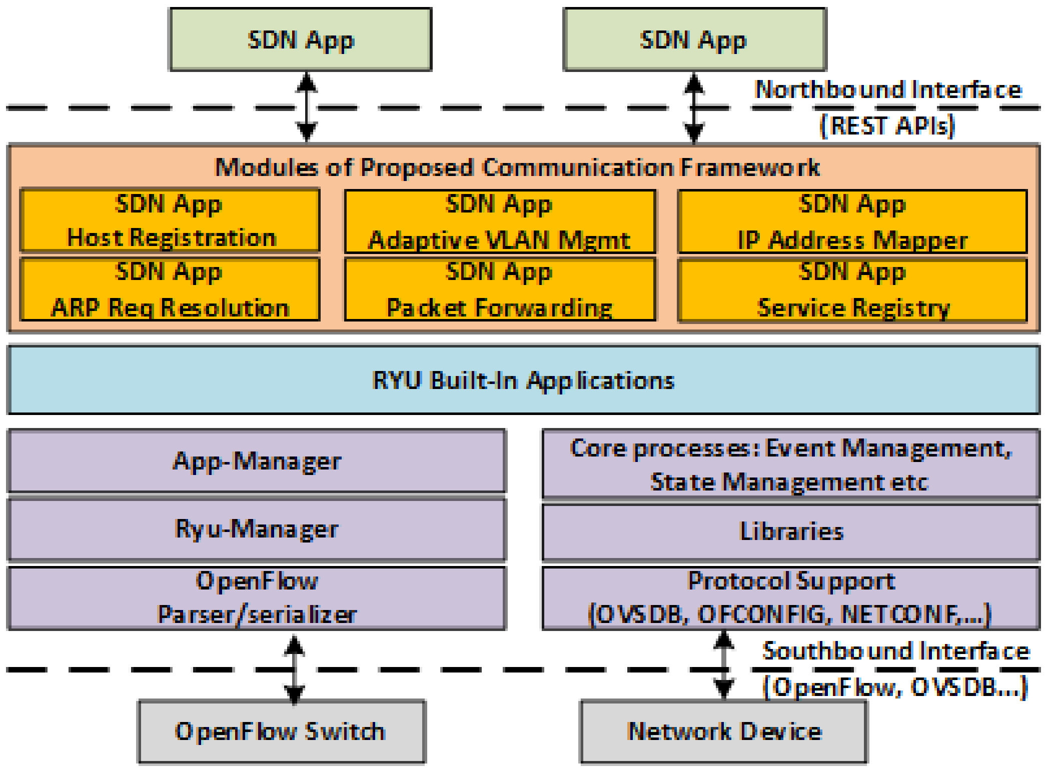

SDN App
SDN App
Northbound Interface (REST APIs)
Modules of Proposed Communication Framework
SDN App Host Registration
SDN App Adaptive VLAN Mgmt
SDN App IP Address Mapper
SDN App ARP Req Resolution
SDN App Packet Forwarding
SDN App Service Registry
RYU Built-In Applications
App-Manager
Core processes: Event Management, State Management etc
Ryu-Manager
Libraries
OpenFlow Parser/serializer
Protocol Support (OVSDB, OFCONFIG, NETCONF,...)
Southbound Interface (OpenFlow, OVSDB...)
OpenFlow Switch
Network Device

FIGURE 4.1: THE ARCHITECTURE OF RYU SDN CONTROLLER WITH ADDED FUNCTIONALITY (APPLICATIONS) DEPICTED IN YELLOW RECTANGLES FOR THE PROPOSED COMMUNICATION FRAMEWORK

4.1.1 Host Registration Module

The registration of hosts enables allowed communication in a private network environment. To enable legitimate communication, our framework demands all hosts to register before communicating in the network. During the host registration procedure, the SDN controller gathers data about each host such as MAC address, IP address, link-layer switch ID, and switch port number. The switch ID & port number inform the SDN controller about the physical location of the host within the network. Additionally, the registration module receives information about the host's application services. Fig. 4.2 summarises the information about each host acquired during the SDN controller's registration operation.

The information gathered during the registration phase benefits host communication in that when sender sends a message to recipient , the SDN controller only configures flow rules on the switches to facilitate the communication after verifying the registration of the both sender and receiver.

It is worth noting that for the sake of enterprise network security, all information gathered during the registration phase is marked as final; that is, once a host's registration is confirmed, the gathered information cannot be amended. Additionally, any host must be registered with the authorization of an authorized person.

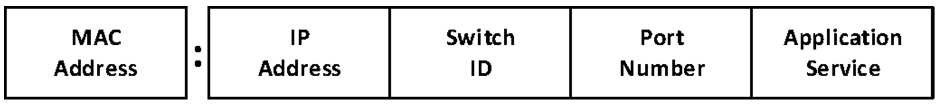


FIGURE 4.2: INFORMATION OF EVERY HOST COLLECTED BY THE REGISTRATION MODULE

4.1.2 Service Registry Module

The registration of services offered by different hosts are maintained using the service registry module. Any host supplying a service across the network have to enrol it using this module. During the process of hosts registration and requests regarding service discovery, this module is used. Fig. 4.3 illustrates the data acquired during the service registration procedure. The request regarding discovery of network services, this module is answerable for delivering a list of available services on the network. Additionally, the module provides clients with service-specific information while concealing the genuine IP address of the service provider. This module supports context-aware communication, which enables clients to find services without exposing the addresses of the actual application servers. Through the use of these context-aware services, network traffic is now routed according to application layer information.

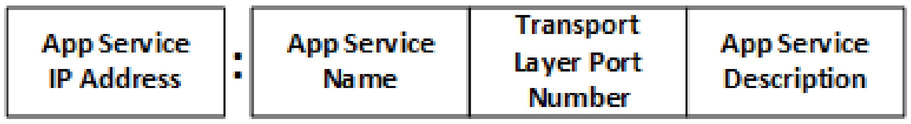


FIGURE 4.3: SERVICE REGISTRATION INFORMATION

4.1.3 IP Address Mapper Module

This framework provides a way for masking host IP address from other hosts in order to provide secure communication within a corporate network. This module generates a unique hoax IP address for each host dynamically using the given "HoaxIPAddressList" and maintains a mapping of actual IP addresses to the Hoax IP addresses. Along with the forged IP addresses, this module maintains the timestamp associated with each, as demonstrated in Fig. 4.4. When the incoming packet's source IP address is compared to any record, the timestamp of each record is updated. The fundamental approach for creating and mapping the fake IP address to the actual one, including the timestamp information, is summarised in Algorithm. 1.

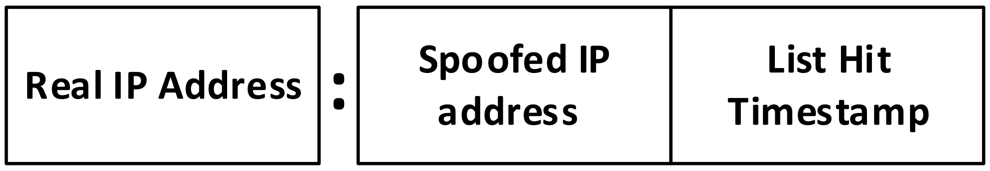


Figure 4.4: Hoax IP address and last hit timestamp information against actual IP address

This framework provides a way for masking host IP address from other hosts in order to provide secure communication within a corporate network. This module generates a unique hoax IP address for each host dynamically using the given "HoaxIPAddressList" and maintains a mapping of actual IP addresses to the Hoax IP addresses. Along with the forged IP addresses, this module maintains the timestamp associated with each, as
demonstrated in Fig. 4.4. When the incoming packet's source IP address is compared to any record, the timestamp of each record is updated. The fundamental approach for creating and mapping the fake IP address to the actual one, including the timestamp information, is summarised in Algorithm. 1.

Once each IP address has reached its maximum lifespan, the date information placed against each record is used to erase all inactive IP addresses. By deactivating inactive IP addresses, we ensure that each subsequent communication request from a host is routed through a new faked IP address. The algorithm 2 explains how to purge the "MappedIP" dictionary of inactive faked IP addresses. Additionally, when a host requests a service from the network, this module is activated to replace the requested host's IP address with the fake IP address and vice versa.

4.1.4 Adaptive VLAN Management Module

As discussed in Sec. 2.3, VLAN allows hosts on different networks to be grouped logically into a single broadcast domain or it may also allow a single network (single broadcast domain) to be partitioned into multiple smaller networks (multiple broadcast domains). The main benefit that can be obtained from VLAN management is the

Algorithm 1: Generation of hoax IP address against real IP address

```
Input  :
        Incoming IP address (IP);
        An IP range to create hoax IP addresses (HoaxIPList);
        A dictionary to map real IP against hoax IP (MappedIP)
Output:
        returns hoax IP address
// verifying if IP is not a Hoax IP address
if IP not in IPAddressMapper.MapppedIP.values() then
    if IPAddressMapper.isRegistered(IP) then
        return IPAddressMapper.Mapped[IP] // return hoax IP
    else
        // Generating random IP from the hoax IP List,
           mapping real IP against hoax IP, and removing
           newly assigned hoax IP from HoaxIPLtst
        hoaxIP=random.randint(0, len(IPAddressMapper.HoaxIPList));
        IPAddressMapper.MappedIP[IP]=IPAddressMapper.HoaxIPList[hoaxIP]
        IPAddressMapper.HoaxIPList.pop(hoaxIP)
        return hoaxIP
    end
else
    // Incoming packet has a hoax IP address
    for key, val in IPAddressMapper.MapppedIP.items() do
        if val == IP then
            return key // returning real IP address
        end
    end
end
```

Algorithm 2: Removal of inactive spoofed IP addresses

```
Input :
        A dictionary of Hoax IP addresses vs real IP addresses
        Python library time package
        HOAX_IP_MAX_LIFETIME = 30 seconds
Output:
        A dictionary with only active Hoax IP vs real IP addresses
for key, val in IPAddressMapper.MapppedIP.items() do
    // getting the current timestamp
    current_timestamp = time.time
    if current_timestamp - val['lastHitTimestamp'] >HOAX_IP_MAX_LIFETIME then
        // removing an entry form 'MappedIP' dictionay where
           its inactive time is greater than max lifetime
        IPAddressMapper.MapppedIP.pop(key)
    end
end
return IPAddressMapper.MapppedIP
```

security of hosts within an enterprise network, where hosts can access each other data if they are on the same VLAN ID. In the traditional network, VLAN management is a tough, tedious, and error-prone task since the network administrator has to manually configure VLANs on all the switches on the network. The job of configuring VLANs becomes even more difficult for larger enterprise networks.

VLAN benefits were also incorporated in SDN by the OpenFlow protocol [101]. Multiple fields and operations are specified by OpenFlow protocol for VLAN-based communication. These fields and operations are listed below.

1. *Push Tag*: Function through which VLAN tag is inserted on the packet header.
2. *Pop Tag*: Function through which VLAN tag is removed from the packet header.
3. *Action➔Set Field*; Field through which Tag Control Information is modified.

This framework incorporates VLAN management to overcome the challenges faced during the manual configuration of VLANs. By using the benefits of SDN, an adaptive VLAN management module is proposed. Our work is the extension of [81] to support dynamic and adaptive VLAN management. The dynamic feature of this module creates VLANs reactively between the two registered hosts willing to communicate in

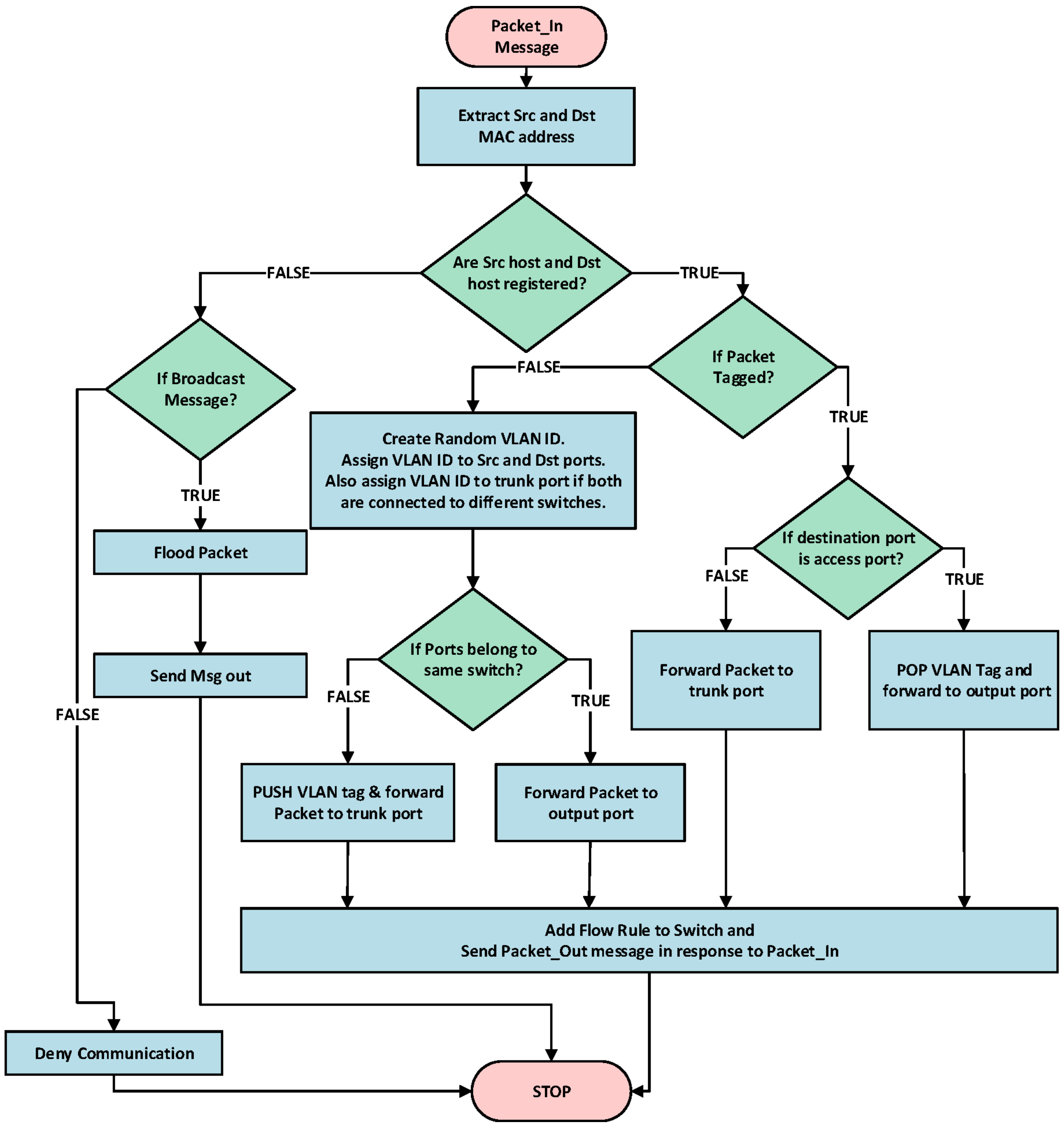

Packet_In Message
Extract Src and Dst MAC address
Are Src host and Dst host registered?
FALSE
TRUE
If Broadcast Message?
If Packet Tagged?
FALSE
TRUE
Create Random VLAN ID. Assign VLAN ID to Src and Dst ports. Also assign VLAN ID to trunk port if both are connected to different switches.
TRUE
Flood Packet
If destination port is access port?
FALSE
TRUE
If Ports belong to same switch?
Send Msg out
FALSE
TRUE
FALSE
Forward Packet to trunk port
POP VLAN Tag and forward to output port
PUSH VLAN tag & forward Packet to trunk port
Forward Packet to output port
Add Flow Rule to Switch and Send Packet_Out message in response to Packet_In
Deny Communication
STOP

FIGURE 4.5: FLOW CHART AND WORKING OF THE *Adaptive VLAN Management* MODULE

an enterprise network. Similarly, the adaptive feature of this module creates different VLANs for different communications. Moreover, the VLAN setup between the two hosts is only established for the duration of the communication. Once the hosts (sender and receiver) finish their communication, the switches automatically remove all the flow rule entries from the switches deployed to support VLAN communication between any communicating hosts. Thus for every new communication between two hosts, a new VLAN is created with a random VLAN ID till the completion of the communication.

The flow chart and working of this module are illustrated in Fig. 4.5. Moreover, a precise step-by-step procedure of this module is exhibited in the Algorithm. 3. This module is invoked by the *Packet_IN* message from the OF switch. Upon receiving the *Packet_IN*, this module performs the series of steps to create reactive VLAN. These steps are listed below:

Step 1: Extraction of MAC addresses from *Packet_IN* message is performed. Then registration of participating hosts is verified using *Host Registration* module.

Step 2(a): If the registration of both hosts is confirmed, then this module inspects the incoming message for the VLAN tag. If the VLAN tag is missing from the message, then the incoming message is treated as the first message of new communication. Upon identifying the new communication, this *Adaptive VLAN Management* module creates a random VLAN ID and allocates this VLAN ID to the switch ports (access ports) through which source and destination hosts are connected. It is to be noted that source and destination hosts switch port numbers and switch IDs are retrieved from the *Host Registration* module. Once the access ports of both communicating hosts are assigned VLAN ID, this module also assigns the VLAN ID to the trunk ports of switches if the two communicating hosts are connected on different switches.

Step 2(b): If the VLAN Tag information is present in the incoming message, then it is presumed that the VLAN between the communicating host is already configured. However, the *Packet_IN* message with tagged VLAN ID is received by the SDN controller because it is generated by that switch that has received this tagged message from its trunk port and has no information (flow rules) to forward this tagged message. The *Adaptive VLAN Management* module matches the destination host switch ID against the switch that has generated the textitPacket_IN message. If the switch IDs are matched, it means that destination hosts lie on the same switch that has generated the textitPacket_IN message. In the matched case, the VLAN Tag information is removed from the message and is forwarded to the access port through which the destination host is connected. For the mismatched case, the next trunk port is identified and the message is forwarded to that trunk port.

If the VLAN Tag information is present in the incoming message, then it is presumed that the VLAN between the communicating host is already configured. However, the *Packet_IN* message with tagged VLAN ID is received by the SDN controller because it is generated by that switch that has received this tagged message from its trunk port and has no information (flow rules) to forward this tagged message. The *Adaptive VLAN Management* module matches the destination host switch ID against the switch that has generated the textitPacket_IN message. If the switch IDs are matched, it means that destination hosts lie on the same switch that has generated the textitPacket_IN message. In the matched case, the VLAN Tag information is removed from the message and is forwarded to the access port through which the destination host is connected. For the mismatched case, the next trunk port is identified and the message is forwarded to that trunk port.

Step 3: The failure in the verification of both or either of the host registration, mentioned in step 1, requires further investigation i.e., the incoming message is further checked for an ARP broadcast message. If the message is an ARP broadcast then it is flooded in the network. Otherwise, failure in the registration of any single host (source and destination) results in denial of communication. The failure in the verification of both or either of the host registration, mentioned in step 1, requires further investigation i.e., the incoming message is further checked for an ARP broadcast message. If the message is an ARP broadcast then it is flooded in the network. Otherwise, failure in the registration of any single host (source and destination) results in denial of communication.

4.1.5 ARP Request Resolution Module

Address Resolution Protocol [102], is a protocol that resolves host permanent physical address (MAC address) against the host changeable IP address (usually IPv4 address) in a local area network (LAN). All hosts in an ethernet maintain a special kind of table known as *ARP Cache*. In this *ARP Cache*, a list of IP addresses and their associated MAC addresses are maintained. When any host wants to send a message to another

Algorithm 3: Adaptive VLAN Algorithm

```
Input :
        Hosts registration (Registration Class);
        Packet_IN message from any switch;
        List of access and trunk ports
Output:
        Add flow rules on the switches
Extract Packet_IN headers;
Retrieve src and dst MAC addresses;
if src and dst registered then
    if Packet tagged with VLAN_ID then
        if dst_port == access_port then
            match = in_port, dst_mac_address, VLAN_ID;
            actions = PopVlan( ) && actionOutput(output_port);
            add_flow(src_switch,1,match, actions);
        else if dst_port == trunk_port then
            match = in_port, dst_mac_address, VLAN_ID;
            actions = actionOutput(output_port);
            add_flow(src_switch,1,match, actions);
        end
    else
        create random VLAN_ID;
        if dst_switch ≠ src_switch then
            match = in_port, dst_mac_address;
            actions = PushVlan( ) && setField(vlan_vid=VLAN_ID) && actionOutput(output_port);
            add_flow(src_switch,1,match, actions);
        else
            match = in_port, dst_mac_address;
            actions = actionOutput(output_port);
            add_flow(src_switch,1,match, actions);
        end
    end
else
    if dst == 'ff:ff:ff:ff:ff:ff' then
        actions = actionOutput(flood_pkt);
        out= PacketOut(switch, in_port, actions, data);
        send_msg(out);
    else
        logger.error("Packet_IN", src_switch, src, dst_switch, dst, "registration verification returns
        false");
    end
end
```

host, it first checks the *ARP Cache* for the mapping of the intended receivers IP and MAC addresses. If the mapping is missing in the *ARP Cache*, then the host broadcasts a request message known as *ARP Request* for the resolution of the MAC address of the intended receiver. Upon receiving the *ARP Request* message which contains the receiver IP address, the receiver generates a unicast response message known as *ARP Response* revealing the MAC address of itself.

If hosts are attached to separate subnets in SDN environments, the SDN controller must resolve the MAC addresses of the gateways on every network. This gateway MAC address resolution is essential because OF switches operate as a router when two connecting hosts are on different subnets. The proposed framework includes a *ARP Request Resolution* module for managing the MAC addresses of all the enterprise network's gateways. This module contains the static mapping between each gateway's IP address and MAC address. When the OpenFlow enabled switch receives a *ARP Request*, it constructs a *Packet_IN* message and sends it to the SDN controller through the control path. This module answers to the *ARP Request* message with a *APR Reply* message by providing the gateway's MAC address via the *Packet_OUT* message.

4.1.6 Packet Forwarding Module

Packet Forwarding module is invoked on the reception of *Packet_IN* message in which destination host is on the different network as compared to the source host. To handle such situations where both communicating hosts belong to different subnets, the *Packet Forwarding* module along with *ARP Request Resolution* module provide the communication mechanism. For the security of hosts in the network, the configuration of each host on a different subnet is recommended as it is verified from the results of security analysis of this module in chapter 6, section 6.3.

The flow chart and working of this module are illustrated in Fig. 4.6. To forward the packet to the destination host, this module requires services of *Host Registration, ARP Request Resolution* and *Network Map DB* modules. The *Host Registration* module is required for the verification of both the source and destination hosts and it is also used to acquire both communicating hosts' switch IDs and port numbers based on the IP address mapping. The *ARP Address Resolution* module is needed to provide the

Algorithm 3: Packet forwarding algorithm

```
Input :
        Physical position of switches (NetworkMap DB);
        Hosts registration (Registration Class);
        Packet_IN message from any switch;
Output:
        Flow rule to forward the incomming packet
Extract Packet_IN headers;
if $Packet \neq ARP$ then
    get $src$ and $dst$ MAC addresses ;
    verify $src$ and $dst$ registration;
    if $src$ and $dst$ registered then
        dst_switch= get $dst$ switch_ID;
        dst_port = get $dst$ port#;
        if $src\_switch == dst\_switch$ then
            actions = setField(dst_mac_add) && actionOutput(dst_port);
            match = Match the ipv4 dst;
            add_flow(src_switch,1,match, actions) ;
        else
            uplink_port=NetworkMap.getUplinkPort(src_switch, dst_switch);
            actions = setField(dst_mac_add) && actionOutput(uplink_port);
            match = Match the ipv4 dst;
            add_flow(src_switch,1,match, actions) ;
        end
    else
        logger.error(”Packet_IN”, src_switch, src, dst_switch, dst, ”registration
        verification returns false”);
    end
end
```

mapping of the gateway MAC address against its IP address. Similarly, the *Network Map DB* module is essential to provide information regarding the uplink port of the switch to forward traffic to the next switch if the destination host is not present on the same switch. Details of the *Network Map DB* are presented in sec. 4.1.7.

The algorithm of this *Packet Forwarding* module is presented in Algo. 4 and is divided into steps mentioned below:

Step 1: Upon the arrival of the *Packet_IN* message, the packet type is checked whether the incoming packet is an *ARP Request* packet or not. If the packet is an *ARP Request* packet then it is handled by the *ARP Request Resolution* module.

Step 2: After ensuring that request is not an *ARP Request*, the source and destination hosts' IP and MAC addresses are extracted. First, it is ensured that the IP addresses mentioned in the *Packet_IN* message are not hoaxed IP addresses. If any of the IP addresses is the hoax address, then it retrieves the real IP address from the *IP address Mapper* module.

Step 3: As the destination host belongs to another subnet, the destination MAC address mentioned in the *Packet_IN* message would be the gateway MAC Address of the source host. Therefore, base on the destination IP address, this module using the services of *Host Registration* module acquires the destination MAC address. After obtaining the real IP and MAC addresses, the registration of both hosts is verified. As the destination host is of a different subnet as compared to the source host, the destination MAC address mentioned in the *Packet_IN* message would be the gateway MAC Address of the source host. Therefore, base on the destination IP address, this module using the services of *Host Registration* module acquires the destination MAC address. After obtaining the real IP and MAC addresses, the registration of both hosts is verified.

Step 4: After the verification of hosts, based on the destination IP Address, the information of the destination host is obtained from the *Host Registration* module such as its switch ID and port number through which the destination host is connected on the network.After the verification of hosts, based on the destination IP Address, the information of the destination host is obtained from the *Host Registration* module such as its switch ID and port number through which the destination host is connected on the network.

Step 5: After obtaining the information regarding the switch IDs of both the communicating hosts, the switch IDs are matched. If the switch IDs are matched, the action field in the flow rule is set to modify the destination MAC address of the incoming packet with the actual MAC address of the destination host. Furthermore, if anonymous communication is enabled then the action filed is also set to modify the source IP address with the hoax IP address. After setting the flow rule with the match and action fields, the flow rule is added on the switch to handle further packets of the same flow. After obtaining the information regarding the switch IDs of both the communicating hosts, the switch IDs are matched. If the switch IDs are matched, the action field in the flow rule is set to modify the destination MAC address of the incoming packet with the actual MAC address of the destination host. Furthermore, if anonymous communication is enabled then the action filed is also set to modify the source IP address with the hoax IP address. After setting the flow rule with the match and action fields, the flow rule is added on the switch to handle further packets of the same flow.

Step 6: Similarly, if the comparison of the switch IDs of both the communicating hosts is not matched, then it means that the source and destination hosts are not on the same switch and it can also be a situation in which the Packet_IN message is generated by that switch that has received a packet for whom it does not have an associated flow rule. In this situation, this module using the services of the *Network Map DB* module obtains the information of the next uplink port where the destination host is connected. Based on the information of the uplink port, the action field in the flow rule is adjusted with the output port is set as the uplink port. This forwarding of the packet to the next switch means that now it is the responsibility of the next switch to forward the packet to either the destination host or the further switch if the destination is not present in the next switch. Moreover, if anonymous communication is enabled then the action filed is also set to modify the source IP address with the hoax IP address. After setting the flow rule with the match and action fields, the flow rule is added on the switch to handle further packets of the same flow.

The working of this module is depicted in Fig. 4.6.

4.1.7 Network Map

The *Network Map DB* module contains information of the physical location of each switch along with its uplink ports. Through this module, the proposed framework maintains the global view of the network topology i.e, the location of each switch and their interconnection among themselves on the network. This database is used by the *Packet Forwarding* module to acquire uplink port information if it wants to forward

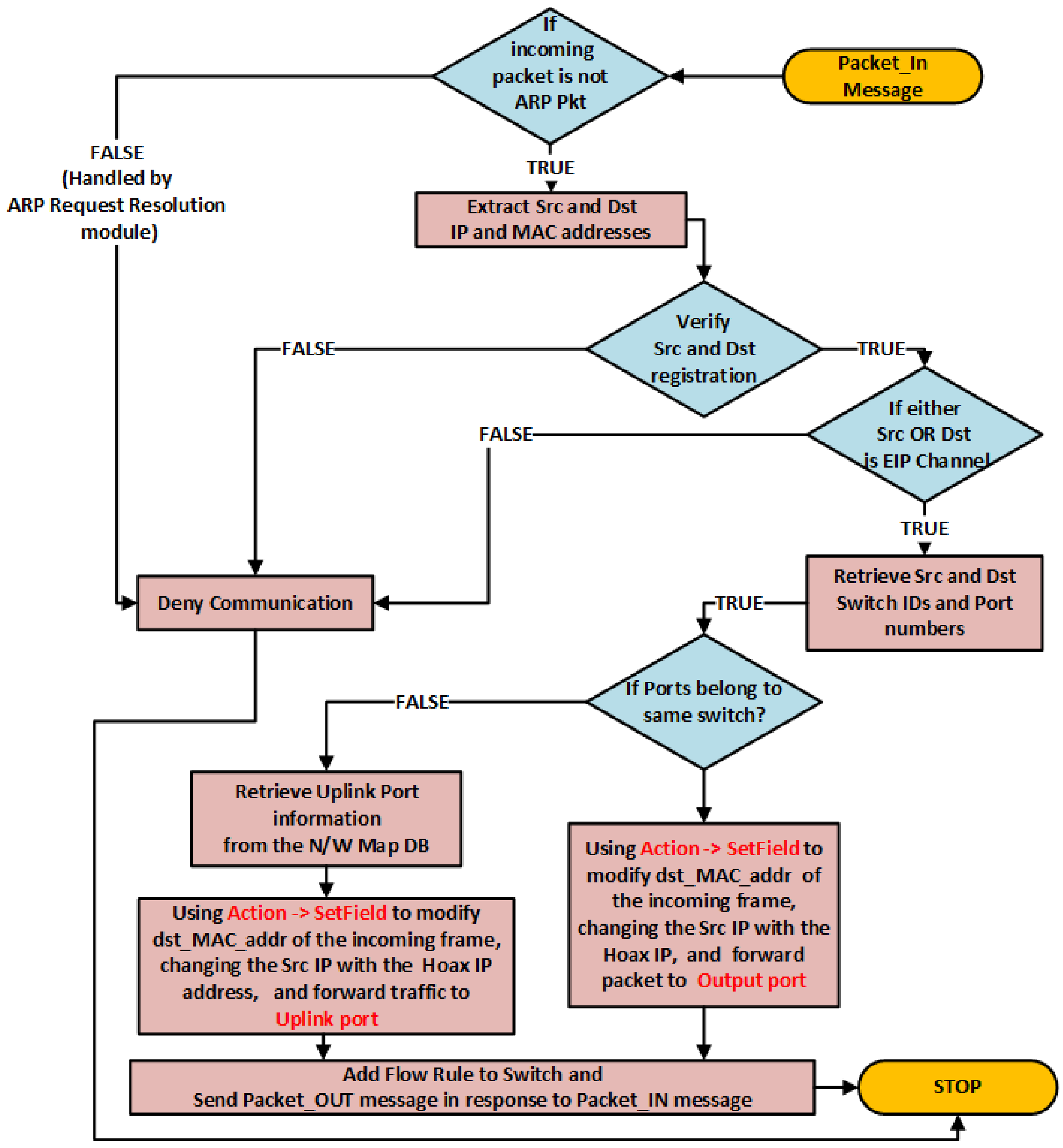

Packet_In Message
If incoming packet is not ARP Pkt
TRUE
FALSE (Handled by ARP Request Resolution module)
Extract Src and Dst IP and MAC addresses
Verify Src and Dst registration
FALSE
TRUE
If either Src OR Dst is EIP Channel
FALSE
TRUE
Retrieve Src and Dst Switch IDs and Port numbers
Deny Communication
TRUE
If Ports belong to same switch?
FALSE
Retrieve Uplink Port information from the N/W Map DB
Using Action -> SetField to modify dst_MAC_addr of the incoming frame, changing the Src IP with the Hoax IP address, and forward traffic to Uplink port
Using Action -> SetField to modify dst_MAC_addr of the incoming frame, changing the Src IP with the Hoax IP, and forward packet to Output port
Add Flow Rule to Switch and Send Packet_OUT message in response to Packet_IN message
STOP

FIGURE 4.6: FLOW CHART AND WORKING OF *Packet Forwarding* MODULE

the packet to another switch by invoking the *getUplinkPort( )* method.

4.2 Summary

In this chapter, we have presented detail discussion of the design and implementation of the proposed communication framework. This communication framework incorporates Enterprise Integration Patterns (EIP) in an SDN environment to support communication between applications (services) with different backgrounds such as languages, platforms, data types, and interfaces. This framework consists of multiple modules of RYU controller that provide different network services. In addition, utilizing the benefits of

SDN, this framework also provides built-in security, services that are context-aware, and host anonymity within the enterprise network.

Chapter 5

Integration of EIP in SDN

In this chapter, the integration of proposed communication framework with Enterprise Integration Patterns (EIP) is explained. Integration of our framework with EIP is done to provide fault-tolerant capability, message reliability and asynchronous communication. The following sections presents the phase wise integration of EIP in the proposed communication framework.

5.1 Integrating EIP in the Proposed Framework

In an organisational network, several programmes provide a variety of services to customers, and the majority of these applications operate independently of one another. Occasionally, the addition of a new service or the growth of an existing service may necessitate a complete rewrite of the information system. However, the simplest method of integrating heterogeneous modules is not to redesign the entire information system, but to enable them to communicate despite their disparate implementation platforms, languages, and data types. When one of these applications requests a service from another, a formal method for communication between them must exist. As detailed in Chapter 2 section 2.2, Enterprise Integration Patterns (EIP) [8] is one method for establishing formal communication between applications. The EIP messaging system is built on the concept of consistent message queuing. Between client applications and the messaging system, messages are queued asynchronously. The primary advantage of messaging systems is their loose coupling, which enables asynchronous message exchange and consistent communication. By utilising a message system, applications can focus exclusively on the data's substance, rather than becoming entangled in the data transmission process. EIP is integrated with the SDN environment in this research to enable asynchronous and reliable communication between applications while leveraging the benefits of the SDN paradigm, including programmability, global visibility, low operational costs, and the ability to adapt to changing network demands.

Integration of EIP in the proposed communication framework is performed phase wise. In the initial phase, EIP is incorporated in the preliminary framework design using only four modules of the proposed framework i.e., *Host Registration, Adaptive VLAN Management, Packet Forwarding, and ARP Request Resolution.* Later, two new modules were included in the proposed communication framework to support context-aware and anonymous communication. These modules are *Service Registry*, and *IP address Mapper*. Following subsections presents the phase wise integration of EIP.

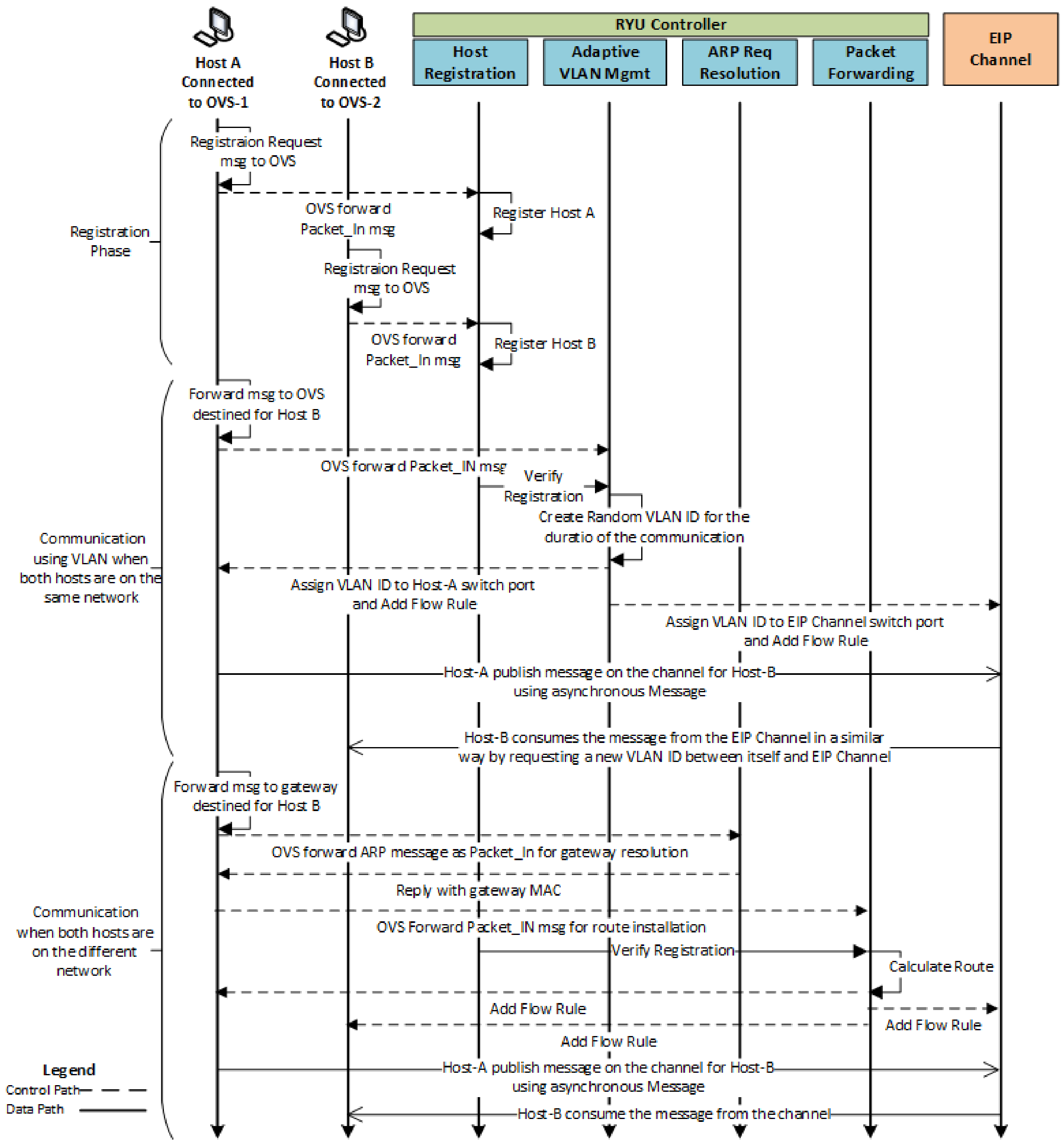

Host A Connected to OVS-1
Host B Connected to OVS-2
RYU Controller
Host Registration
Adaptive VLAN Mgmt
ARP Req Resolution
Packet Forwarding
EIP Channel
Registration Phase
Registraion Request msg to OVS
OVS forward Packet_In msg
Register Host A
Registraion Request msg to OVS
OVS forward Packet_In msg
Register Host B
Communication using VLAN when both hosts are on the same network
Forward msg to OVS destined for Host B
OVS forward Packet_IN msg
Verify Registration
Create Random VLAN ID for the duratio of the communication
Assign VLAN ID to Host-A switch port and Add Flow Rule
Assign VLAN ID to EIP Channel switch port and Add Flow Rule
Host-A publish message on the channel for Host-B using asynchronous Message
Host-B consumes the message from the EIP Channel in a similar way by requesting a new VLAN ID between itself and EIP Channel
Communication when both hosts are on the different network
Forward msg to gateway destined for Host B
OVS forward ARP message as Packet_In for gateway resolution
Reply with gateway MAC
OVS Forward Packet_IN msg for route installation
Verify Registration
Calculate Route
Add Flow Rule
Add Flow Rule
Add Flow Rule
Host-A publish message on the channel for Host-B using asynchronous Message
Host-B consume the message from the channel
Legend
Control Path
Data Path

FIGURE 5.1: PHASE I: A SEQUENCE OF MESSAGE FLOW BETWEEN HOST A AND HOST B USING MESSAGE SYSTEM (EIP CHANNEL).

5.1.1 Phase I

In this phase, EIP is incorporated in the preliminary framework design comprising of only four modules depicted in Fig. 5.1. With the integration of *Messaging System (EIP)*, any application on the network desires to communicate with the other application executing on another host must utilize the messaging system to deliver its messages. In this research, three core components of *Messaging Systems* have been used which are Message, Message Channel, and Message Endpoint. In this communication framework, each host has a special interface (Message Endpoint) that allows any application to transmit or receive messages to / from the messaging channel. When an application running on one host desires a service from the another application on the same or different network, it constructs a message (request message) and forwards that message to its Message Endpoint. It is the job of the Message Endpoint to deliver and receive messages into/from the Message Channel. Message Endpoint is also responsible for converting the message format if different messaging formats are being used by the communicating applications.

5.1.1.1 Registration Phase

Fig. 5.1 depicts the working of the proposed communication framework and the sequence of messages delivered between the communicating hosts after the integration of EIP in SDN. The first step towards the secure communication environment is always the registration and verification of hosts present in the network. As it can be observed from Fig. 5.1 that the first few messages are of the registration of two communicating hosts i.e., Host A and Host B. Here Host A and Host B are connected to OF switch 1 (OVS-1) and OF switch 2 (OVS-2) respectively. Both hosts through their own OF switch sends a registration request message that is forwarded to the SDN controller because of the table-miss flow entry that forwards any message to the SDN Controller. Upon receiving the registration request the *Host Registration* module registers hosts by recording their information such as IP and MAC addresses, OF switch ID, and switch port number.

5.1.1.2 VLAN based communication

Using this framework, hosts must configured using two different sets of configuration layouts. In the first layout, all hosts belong to the same subnet and portray the ethernet topology of a single subnet. If the hosts' configuration is of a single subnet topology then communication between hosts is provided by the *Adaptive VLAN Management* module of this communication framework. As it can be seen from Fig. 5.1 that

Host A wants to acquire the services of Host B, it sends a message for Host B to its Endpoint. Upon receiving the message, the Endpoint running inside Host A will send this message to OF switch destined for Message Channel (EIP Channel). Upon receiving the message from Host A, the OF switch first matches the message header information against the flow rules deployed on the OF switch. If message header information is not matched against any rule then OF switch will forward this message to the SDN controller using the *Packet_IN* message.In our proposed framework, hosts can be configured using two different sets of configuration layouts. In the first layout, all hosts belong to the same subnet and portray the ethernet topology of a single subnet. If the hosts' configuration is of a single subnet topology then communication between hosts is provided by the *Adaptive VLAN Management* module of this communication framework. As it can be seen from Fig. 5.1 that Host A wants to acquire the services of Host B, it sends a message for Host B to its Endpoint. Upon receiving the message, the Endpoint running inside Host A will send this message to OF switch destined for Message Channel (EIP Channel). Upon receiving the message from Host A, the OF switch first matches the message header information against the flow rules deployed on the OF switch. If message header information is not matched against any rule then OF switch will forward this message to the SDN controller using the *Packet_IN* message.

As both hosts (Host A and EIP Channel) belong to the same network, the *VLAN Management* module is invoked which first verifies that if the VLAN ID is already created between the communicating hosts or not. If it is a new communication then the VLAN module creates a random VLAN ID for the Host A and EIP Channel, then it deploys flow rules on the switches to support VLAN based communication between them. After the installation of flow rules to support communication between Host A and EIP Channel, now Host A can send messages to EIP Channel destined for Host B. The EIP Channel maintains a message queue for each host/service and places Host B messages on the Host B queue.

Likewise, flow rules are installed in the same manner between Host B and the EIP Channel using a new VLAN ID. Now the application at Host B can retrieve its messages buffered on the EIP Channel through its own Endpoint. It is worth mentioning that

Point-to-Point Channel is being used that ensures that only Host B can successfully receive its own messages. Moreover, the EIP Channel and the communication framework are fully capable of handling multiple communications simultaneously. Likewise, flow rules are installed in the same manner between Host B and the EIP Channel using a new VLAN ID. Now the application at Host B can retrieve its messages buffered on the EIP Channel through its own Endpoint. It is worth mentioning that Point-to-Point Channel is being used that ensures that only Host B can successfully receive its own messages. Moreover, the EIP Channel and the communication framework are fully capable of handling multiple communications simultaneously.

The results presented in Section 6.2 of Chapter 6 are based on this reactive installation of VLAN scenario.

5.1.1.3 Communication between Hosts on Different Networks

The second hosts' configuration layout focuses on the communication between hosts on different networks. It is recommended for the security purpose that each host be configured on an individual network having /30 CIDR prefix. When hosts are on different networks then communication is mainly managed by the *ARP Request Resolution*. The other supported modules are *ARP Request Resolution*, and *Network Map DB*. As each host is configured on an individual subnet then to support communication between different subnets, each host must be configured with its own gateway IP address. Address resolution of each gateway MAC address associated with its host is handled by the *ARP Request Resolution* module.The second hosts' configuration layout focuses on the communication between hosts on different networks. It is recommended for the security purpose that each host be configured on an individual network having /30 CIDR prefix. When hosts are on different networks then communication is mainly managed by the *ARP Request Resolution*. The other supported modules are *ARP Request Resolution*, and *Network Map DB*. As each host is configured on an individual subnet then to support communication between different subnets, each host must be configured with its own gateway IP address. Address resolution of each gateway MAC address associated with its host is handled by the *ARP Request Resolution* module.

As depicted in Fig. 5.1, all hosts (A, B, and EIP Channel) are connected to different subnets. When Host A desires a service from Host B, it first sends an ARP request message to acquire the MAC address of its gateway. To send a packet to Host B, Host A (Endpoint) must first send the packet to its gateway. Once the message reaches Host As OF switch (intended for Host B) having the EIP Channel as destination address, the switch forwards this message to the SDN controller using *Packet_IN* message. Upon identifying the different subnet of destination (EIP Channel) as compared to the sender (Host A), the *Packet Forwarding* module deploys flow rules to support communication between these two hosts.

Similarly, flow rules are installed in the same manner between Host B and the EIP Channel using the *Packet Forwarding* module for the consumption of messages buffered inside the EIP Channel for the application at Host B. The complete sequence of messages mentioned in this scenario is also illustrated in Fig. 5.1. Similar to the first layout scenario, the result presented in Section 6.3 of Chapter 6 are based on this scenario where two communicating hosts are connected to different subnets and are communicating with each other using the messaging system.

5.1.2 Phase II

This phase extends the previous one by including two more modules, IP Address Mapper and Service Registry, of our proposed framework to support context-aware communication and provide packet anonymity through asynchronous message exchange. The Service Registry module offers context-aware communication by allowing hosts to find services without disclosing the real addresses of the application servers. Our framework support network traffic using application layer information. It is vital to emphasis that we created a message-based communication system to ease application communication. The sequence diagram in Fig. 5.2 illustrates all of the events / message exchanges between two communicating hosts. Where hosts (A and B) are connected to the network via their own gateways and are on distinct subnets. The initial phase of this communication infrastructure is the registration of hosts. As seen in Fig. 5.2, host B provides a service and registered it during the registration step using the Service Registry module.

Host-A sends a Packet IN message seeking service discovery following the registration procedure. The *Host Registration* module transfers the service discovery request to the *Service Registry* module, which answers with all the services that network is providing. After receiving the list of services, host A may pick one and transmit the message to the OpenFlow enabled switch, which results in a Packet IN message to the SDN controller. The Packet Forwarding module verifies that both hosts are registered and then makes a request to the IP Address Mapper module to change the source host's IP address. When the Packet Forwarding module gets the bogus IP address, it determines the route between Host-A and the EIP Channel. The controller will use a Flow Mod message to apply flow rules to the OF-switch between the sender and the EIP channel. Subsequent communications from Host-A will be sent straight to the EIP channel, bypassing the SDN controller completely. When Host-A sends messages to Host-B through the EIP channel, the OF-switch replaces Host-real A's IP address with a bogus IP address and forwards the message to the EIP channel. This replacement of the source IP address might mask the identity of the transmitting host.

Likewise, when the receiver (Host-B) chooses to consume the messages contained inside the EIP channel, it sends a request message to the EIP channel, resulting in the installation of flow rules on the switches linking the EIP channel and the receiver. Additionally, when the EIP channel answers to a request, it passes the message to the bogus IP address. The SDN controller will set flow rules between the source EIP channel and the destination Host-B. In this situation, the flow rules will substitute the true IP address of Host-B for the fake IP address. Additionally, as illustrated in Fig. 6.8, the results discussed in Section 6.4 of Chapter 6 are based on this phase.

5.2 Summary

In this chapter, we presented the procedure of integrating our novel communication framework with Enterprise Integration Patterns (EIP). As this novel framework consists of six modules, the integration was done in phases. In the first phase, we integrated first four modules then two more modules were also integrated with EIP during the second phase. Integration of our framework with EIP is done to provide fault-tolerant capability, message reliability and asynchronous communication. Through time-line / sequence diagrams, we highlighted the messages sequences to support the network communication.In this chapter, we presented the procedure of integrating our novel communication framework with Enterprise Integration Patterns (EIP). As this novel framework consists of six modules, the integration was done in phases. In the first phase, we integrated first four modules then two more modules were also integrated with EIP during the second phase. Integration of our framework with EIP is done to provide fault-tolerant capability, message reliability and asynchronous communication. Through time-line / sequence diagrams, we highlighted the messages sequences to support the network communication.

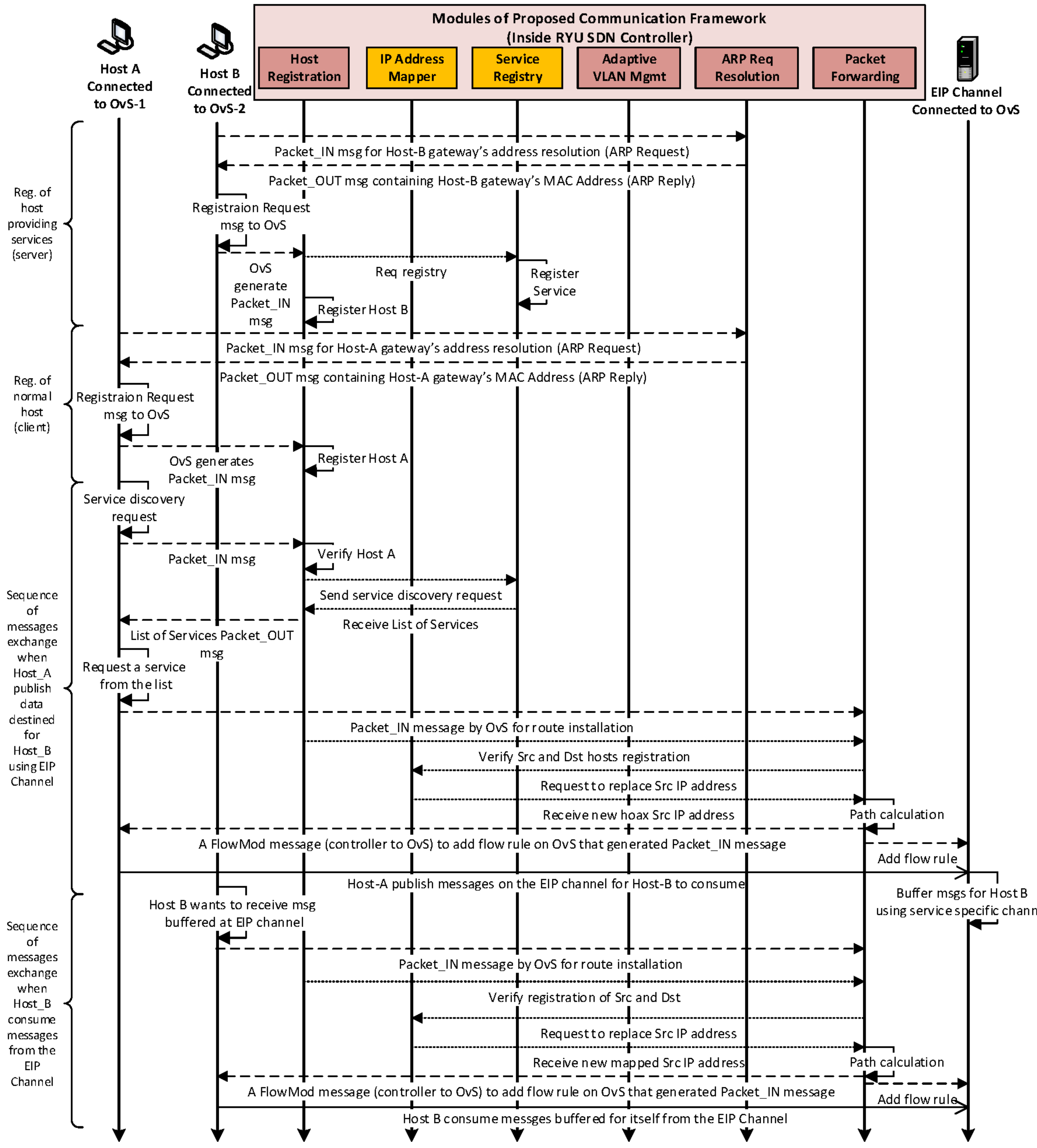

Modules of Proposed Communication Framework (Inside RYU SDN Controller)
Host Registration
IP Address Mapper
Service Registry
Adaptive VLAN Mgmt
ARP Req Resolution
Packet Forwarding
Host A Connected to OvS-1
Host B Connected to OvS-2
EIP Channel Connected to OvS
Reg. of host providing services (server)
Packet_IN msg for Host-B gateway's address resolution (ARP Request)
Packet_OUT msg containing Host-B gateway's MAC Address (ARP Reply)
Registraion Request msg to OvS
OvS generate Packet_IN msg
Req registry
Register Service
Register Host B
Reg. of normal host (client)
Packet_IN msg for Host-A gateway's address resolution (ARP Request)
Packet_OUT msg containing Host-A gateway's MAC Address (ARP Reply)
Registraion Request msg to OvS
OvS generates Packet_IN msg
Register Host A
Sequence of messages exchange when Host_A publish data destined for Host_B using EIP Channel
Service discovery request
Packet_IN msg
Verify Host A
Send service discovery request
Receive List of Services
List of Services Packet_OUT msg
Request a service from the list
Packet_IN message by OvS for route installation
Verify Src and Dst hosts registration
Request to replace Src IP address
Receive new hoax Src IP address
Path calculation
A FlowMod message (controller to OvS) to add flow rule on OvS that generated Packet_IN message
Add flow rule
Host-A publish messages on the EIP channel for Host-B to consume
Buffer msgs for Host B using service specific chann
Sequence of messages exchange when Host_B consume messages from the EIP Channel
Host B wants to receive msg buffered at EIP channel
Packet_IN message by OvS for route installation
Verify registration of Src and Dst
Request to replace Src IP address
Receive new mapped Src IP address
Path calculation
A FlowMod message (controller to OvS) to add flow rule on OvS that generated Packet_IN message
Add flow rule
Host B consume messges buffered for itself from the EIP Channel

FIGURE 5.2: PHASE-II INTEGRATION OF EIP IN PROPOSED FRAMEWORK.

Chapter 6

Evaluation

The primary objective of this research was to include Enterprise Integration Patterns into SDN by defining the communication mechanisms necessary between communicating hosts. As a result, this chapter evaluates the suggested communication architecture in terms of the correctness of the flow rules established and the efficiency of communication. Additionally, the our framework is assessed in terms of host anonymity and the overall security of the SDN-based corporate network.

Three sections comprise the assessment process. The first section's purpose is to verify communication between hosts utilising responsive VLANs once all participating hosts are connected to a certain network. It's worth noting that all conversations between hosts must take place through the EIP Channel. To accomplish the aforementioned goal, we examined this framework's accuracy in terms of installing flow rules between hosts and its efficiency in terms of the time necessary to build VLANs and install flow rules. The second section assesses communication between hosts that are members of distinct networks. The second portion will verify the implementation of proper flow rules to enable communication between hosts on various networks and to protect network hosts from various network assaults. The last section of the study evaluates the proposed communication architecture using context-aware services and packet anonymity between hosts.

In the following sections, we first introduce the prototype setup environment. Later, each evaluation phase is discussed in detail which includes the test topology, threat model, host configurations, and discussion on the generated results.

6.1 Environment Setup

Mininet [103]is being used as a simulator, and an SDN controller is deployed using a component-based RYU controller. Additionally, Containernet [104] is mounted on top of Mininet to facilitate the deployment of Linux containers in the Mininet as hosts. All application modules presented in this Chapter are implemented in Python as the RYU SDN Controller application. Whatever service registry could be deployed to register network hosts' application services. We created a new registry service and also performed an evaluation of PyService-Registry [105].

Similarly, any messaging system that is capable of simulating the EIP Channel may be employed. In our research, we use RabbitMQ [106] and Apache ActiveMQ [107]to evaluate this framework. Above-mentioned (RabbitMQ and ActiveMQ)systems use Advanced Message Queueing Mechanism (AMQP) [108].

6.2 Evaluation of VLAN Communication

This section aims to evaluate the proposed communication framework concerning the creation and deletion of VLANs among communicating hosts. The objective is to verify that all communications via EIP Channel are handled efficiently and accurately using reactive VLANs when communicating hosts belong to the same network. The reactive VLAN communication among hosts is managed by the *Adaptive VLAN* module of the proposed communication framework, where it creates a new VLAN between the communicating hosts reactively and abolishes the VLAN when hosts have completed their communication. Thus, for each communication, a new VLAN with a different VLAN ID is created between the communicating hosts for the duration of that communication.

It is worth mentioning that communication between any two hosts is only possible through EIP Channel. All the traffic is directed to EIP Channel by the hosts using their respective Endpoint. Furthermore, it is made sure by the SDN controller that communication is only possible between any host and the EIP Channel. All the other direct communication requests between the hosts are blocked by the SDN Controller by denying the flow rules installation. Therefore, communication between two hosts requires the establishment of two VLANs: the first one between the sender and the EIP Channel, and the second one between the receiver and the EIP Channel.It is worth mentioning that communication between any two hosts is only possible through EIP Channel. All the traffic is directed to EIP Channel by the hosts using their respective Endpoint. Furthermore, it is made sure by the SDN controller that communication

is only possible between any host and the EIP Channel. All the other direct communication requests between the hosts are blocked by the SDN Controller by denying the flow rules installation. Therefore, communication between two hosts requires the establishment of two VLANs: the first one between the sender and the EIP Channel, and the second one between the receiver and the EIP Channel.

It is worth mentioning that communication between any two hosts is only possible through EIP Channel. All the traffic is directed to EIP Channel by the hosts using their respective Endpoint. Furthermore, it is made sure by the SDN controller that communication is only possible between any host and the EIP Channel. All the other direct communication requests between the hosts are blocked by the SDN Controller by denying the flow rules installation. Therefore, communication between two hosts requires the establishment of two VLANs: the first one between the sender and the EIP Channel, and the second one between the receiver and the EIP Channel.It is worth mentioning that communication between any two hosts is only possible through EIP Channel. All the traffic is directed to EIP Channel by the hosts using their respective Endpoint. Furthermore, it is made sure by the SDN controller that communication is only possible between any host and the EIP Channel. All the other direct communication requests between the hosts are blocked by the SDN Controller by denying the flow rules installation. Therefore, communication between two hosts requires the establishment of two VLANs: the first one between the sender and the EIP Channel, and the second one between the receiver and the EIP Channel.

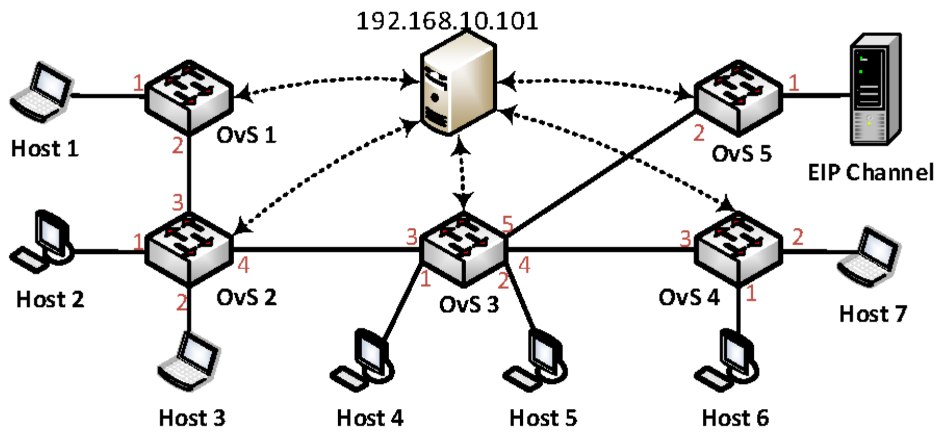


FIGURE 6.1: THE NETWORK TOPOLOGY USED IN THE EVALUATION PART 1 AND 2 MENTIONED IN SECTION 6.2 AND SECTION 6.3 RESPECTIVELY.

6.2.1 Test Topology and Hosts configuration

For the evaluation of the proposed framework, a network topology illustrated in Fig.6.1 is designed. In this scenario, all hosts belong to the same network i.e, address of every host id from the same IP pool and have the same network ID. Hosts' IP addresses along with their associated MAC address are listed in Table. 6.1. For the evaluation, a simple scenario is created where Host 1 wants to acquire the service that host Host 3 is offering via EIP Channel. Based on this scenario, the performance of the framework is tested with respect to the installation of accurate flow rules and efficiency of the framework related to the time needed to create reactive VLANs and installation of flow rules based on the newly created VLAN between the communicating entities. Following fields are used in flow rules to guide packet flow from source to destination switches.

1. **Match**: These fields are used to match the incoming packet header fields. In this scenario, following fields are used.
   (a) Ingress Port
   (b) VLAN: VLAN ID is matched.
   (c) Destination MAC Address
2. **Actions**: If the packet header fields are matched then actions are performed e.g., Pushing the VLAN tag or popping the VLAN tag etc. In this scenario, following actions are performed.
   (a) Out Port: Output port is identified by controller and Packet is forwarded to that port.
   (b) Set Field: VLAN ID is set.
   (c) Push VLAN: VLAN Tag is pushed.
   (d) Pop VLAN: VLAN Tag is Popped.
3. **Idle Timeout**: This filed is used to remove a flow entry from the switch after the given number of seconds, if no packet has been matched by the flow. In this scenario, Idle timeout is set to 50 second.

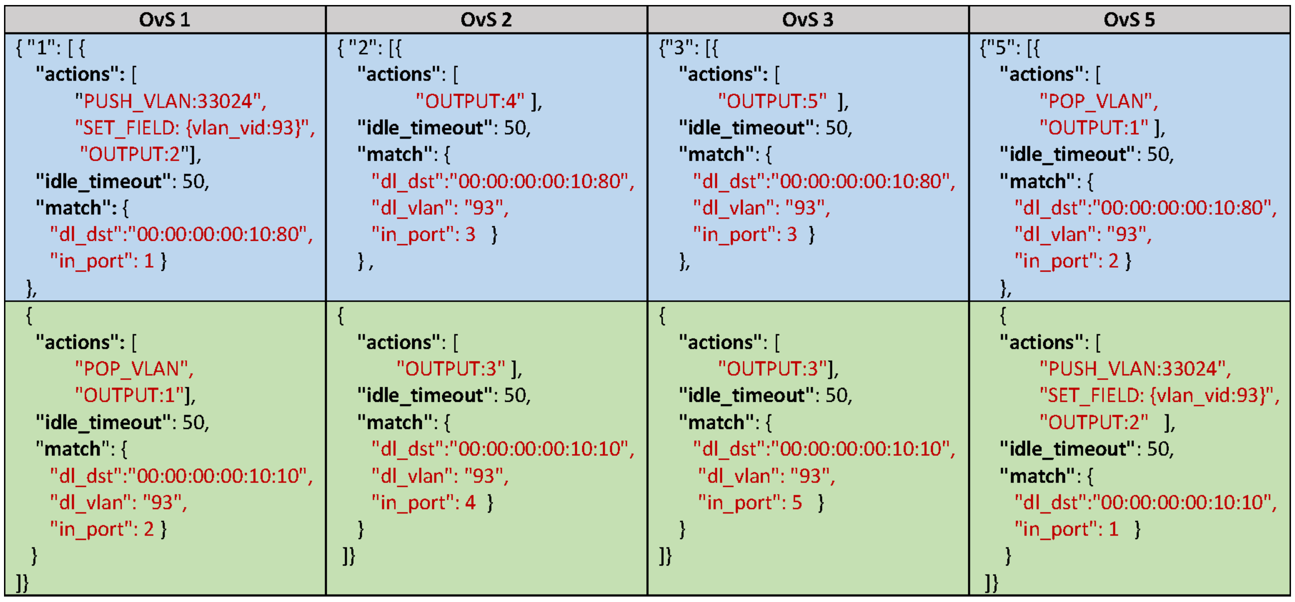
TABLE 6.1: EVALUATION PART 1: LIST OF ALL HOSTS AND THEIR NETWORK CONFIGURATIONS BASED ON 192.168.10.0 NETWORK ID AND /24 CIDR PREFIX.

Host Name	IP Address	MAC Address
Host 1	192.168.10.10	00:00:00:00:10:10
Host 2	192.168.10.20	00:00:00:00:10:20
Host 3	192.168.10.30	00:00:00:00:10:30
Host 4	192.168.10.40	00:00:00:00:10:40
Host 5	192.168.10.50	00:00:00:00:10:50
Host 6	192.168.10.60	00:00:00:00:10:60
Host 7	192.168.10.70	00:00:00:00:10:70
EIP Channel	192.168.10.80	00:00:00:00:10:80

6.2.2 Results Analysis and Discussions

This section summarises the results of the scenario mentioned above, which explains Host 1 wishes to obtain the service offered by Host-3. The application on Host-1 sends a request packet on the channel. As we can see in Fig. 6.1, there are four OF switches on the pathway connecting Host 1 and the EIP Channel. Our framework's VLAN module generates a VLAN ID to facilitate communication among Host-1 and channel,and configures OF-switch rules on all OF-switches along the channel. The flow rules implemented on OF switches connecting Host 1 and EIP Channel are depicted in Fig. 6.2. It should be noted that only essential fields are included here for the sake of readability of flow rules. As seen in Fig. 6.2, the VLAN module generates a random VLAN ID 93 to facilitate communication. The OF-rules are configured in such a manner that they match the EIP Channel's MAC address as the destination MAC address and Actions field is configured to push the VLAN ID 93 tag onto the frame. The frame is then sent to the output port connecting linking the other OF switch. The output port (trunk port) of OF switch 1 (OvS 1) is set to 2. The intermediate switches (OvS-2 and OvS-3) just match the destination MAC address and the VLAN ID 93 tag; they do not process the frame; they simply send it to their trunk ports. When the frame reaches the destination switch (the OF switch for the EIP Channel), the VLAN ID 93 Tag is removed and the frame is sent to the output port (in this case, the switch port connected to the EIP Channel). Similarly, on the reverse path between EIP Channel and Host 1, frames with the destination MAC address of Host

1 will be sent to the trunk port tagged with VLAN ID 93. The VLAN tag is popped upon arrival at the destination switch, and the remainder of the frame is sent to the switch port associated to Host 1. Additionally, the flow rule's idle timeout field is set to 50 secs, allowing the OF-rules to be erased from the OF-switches after 50 secs of inactivity. Due to the fact that messaging system is being used in our framework, all messages pertaining to Host-3 are stored on the channel that are send by host-1.



OvS 1

```
{ "1": [ {
  "actions": [
      "PUSH_VLAN:33024",
      "SET_FIELD: {vlan_vid:93}",
       "OUTPUT:2"],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:80",
    "in_port": 1 }
 },
 {
  "actions": [
      "POP_VLAN",
      "OUTPUT:1"],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:10",
    "dl_vlan": "93",
    "in_port": 2 }
  }
]}
```

OvS 2

```
{ "2": [{
  "actions": [
       "OUTPUT:4" ],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:80",
    "dl_vlan": "93",
    "in_port": 3  }
  } ,
{
  "actions": [
       "OUTPUT:3" ],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:10",
    "dl_vlan": "93",
    "in_port": 4  }
  }
]}
```

OvS 3

```
{"3": [{
  "actions": [
       "OUTPUT:5"  ],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:80",
    "dl_vlan": "93",
    "in_port": 3  }
  },
{
  "actions": [
       "OUTPUT:3"],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:10",
     "dl_vlan": "93",
     "in_port": 5   }
  }
]}
```

OvS 5

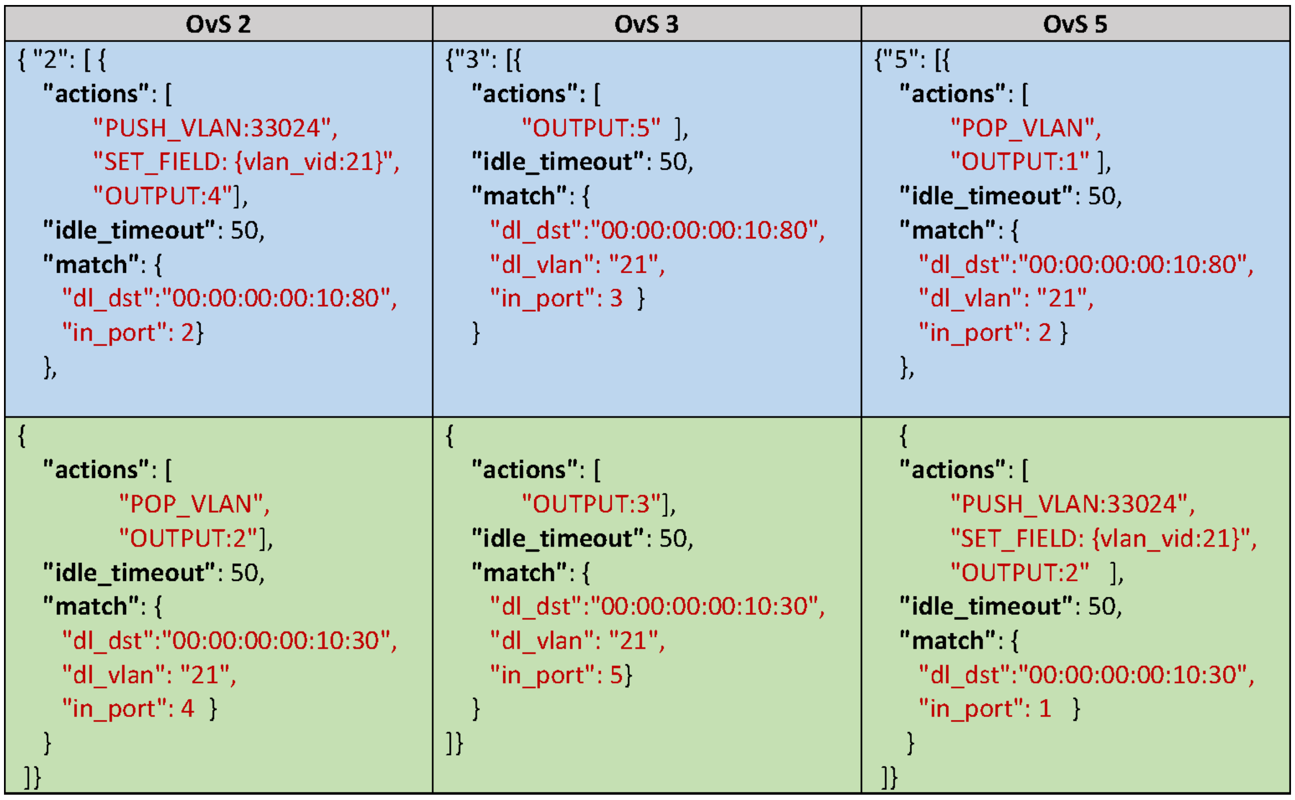
```
{"5": [{
  "actions": [
      "POP_VLAN",
      "OUTPUT:1" ],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:80",
    "dl_vlan": "93",
    "in_port": 2 }
  },
  {
  "actions": [
      "PUSH_VLAN:33024",
      "SET_FIELD: {vlan_vid:93}",
      "OUTPUT:2"   ],
  "idle_timeout": 50,
  "match": {
    "dl_dst":"00:00:00:00:10:10",
    "in_port": 1   }
  }
]}
```

FIGURE 6.2: FLOW RULES DEPLOYED ON OF SWITCHES TO SUPPORT VLAN-BASED COMMUNICATION (HOST-1 AND EIP).

Likewise, Host 3 (service running on Host 3) can receive its EIP Channel messages at any moment by contacting the SDN controller to deploy the necessary flow rules. Refer to Fig. 6.3 to see how the SDN controller, using the proposed VLAN module, produces a random VLAN ID 21 for connection among Host 3 and EIP and apply the necessary flow rules on all switches along the path.

6.2.2.1 Performance w.r.t Flow Rules Installation

Our framework was evaluated against a variety of situations involving a variety of hosts in order to measure the performance of the the VLAN Management module. The efficiency is measured by the precision of flow rules written on the switches employing distinct VLAN IDs to ensure exact communication between communicating hosts. Additionally, it is observed that an access port could be a member of numerous VLAN



OvS 2

```
{ "2": [ {
   "actions": [
        "PUSH_VLAN:33024",
        "SET_FIELD: {vlan_vid:21}",
        "OUTPUT:4"],
   "idle_timeout": 50,
   "match": {
     "dl_dst":"00:00:00:00:10:80",
     "in_port": 2}
   },

{
   "actions": [
          "POP_VLAN",
          "OUTPUT:2"],
   "idle_timeout": 50,
   "match": {
     "dl_dst":"00:00:00:00:10:30",
     "dl_vlan": "21",
     "in_port": 4  }
   }
 ]}
```

OvS 3

```
{"3": [{
   "actions": [
        "OUTPUT:5"  ],
   "idle_timeout": 50,
   "match": {
     "dl_dst":"00:00:00:00:10:80",
     "dl_vlan": "21",
     "in_port": 3  }
   }

{
   "actions": [
        "OUTPUT:3"],
   "idle_timeout": 50,
   "match": {
     "dl_dst":"00:00:00:00:10:30",
     "dl_vlan": "21",
     "in_port": 5}
   }
]}
```

OvS 5

```
{"5": [{
   "actions": [
        "POP_VLAN",
        "OUTPUT:1" ],
   "idle_timeout": 50,
   "match": {
     "dl_dst":"00:00:00:00:10:80",
     "dl_vlan": "21",
     "in_port": 2 }
   },

   {
   "actions": [
        "PUSH_VLAN:33024",
        "SET_FIELD: {vlan_vid:21}",
        "OUTPUT:2"   ],
   "idle_timeout": 50,
   "match": {
     "dl_dst":"00:00:00:00:10:30",
     "in_port": 1   }
   }
]}
```

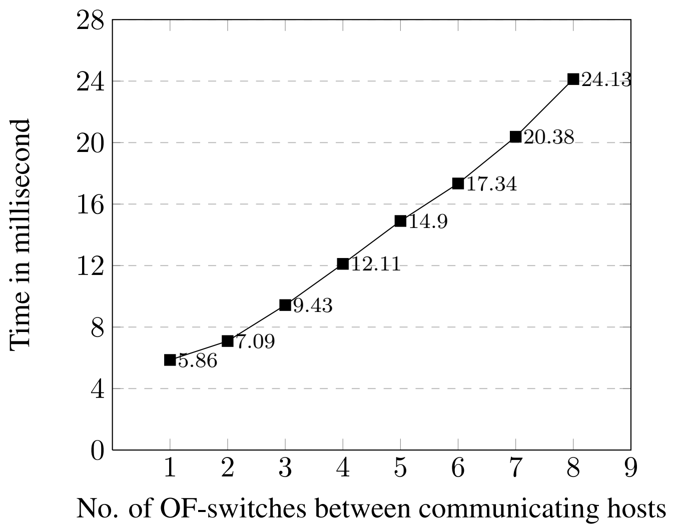
FIGURE 6.3: FLOW RULES DEPLOYED ON OF SWITCHES TO PROVIDE COMMUNICATION BETWEEN HOST-3 AND EIP.

IDs under SDN. This is because VLAN tagging is applied exclusively by switches to forward traffic and is deleted by OF switches upon arrival at the target port. Without the VLAN tag, the OF switch passes the packets to the host. This is in contrast to typical switches, which allocate each access port to a single VLAN ID. As seen in Figures 6.2and 6.3, the VLAN ID between Host 1 and the EIP Channel is 93, meanwhile the VLAN ID connecting Host 3 and the EIP Channel is 21. Through two distinct flow rules, the port through which EIP Channel is linked is allocated both VLAN IDs (VLAN ID 93 and 21). As a consequence, Host 1 and Host 3 can connect with EIP Channel concurrently using distinct VLAN IDs.

6.2.2.2 Efficiency

The framework's suggested VLAN Management module was further evaluated in terms of the time required to set flow rules on switches along the path to the destination from source in order to facilitate communication via reactive VLANs. Multiple tests (25 per location) were run using various locations of the communicating hosts, depending on

FIGURE 6.4: CREATION AND DEPLOYMENT OF REACTIVE VLAN WITH RESPECT TO TIME.



their actual positioning on the network, to determine the average time required to apply flow rules. The time required to configure a reactive VLAN and install necessary flow rules among source and destination hosts is depicted in Fig. 6.4. The y-axis in Fig. 6.4 represents the time in milliseconds required to install flow rules on the OF switches, while the x-axis represents the number of OF switches in between source and destination. It is obvious that the time required to set flow rules increases as the distance and number of OF switches between source and destination increases. Additionally, Fig. 6.4 demonstrates that the framework requires only 3 to 4 milliseconds to install flow rules on OF switches when the number of OF switches between the source and destination hosts is increased. The ability of SDN to create reactive VLANs automatically distinguishes it from conventional networks, which need an administrator to manually setup VLANs on all switches deployed in a business network.

Additionally, our proposed communication framework's efficiency claim is reinforced because redundant flow rule entries are deleted using the idle timeout field when the two communicating hosts terminate communication, thereby conserving Ternary Content Addressable Memories (TCAMs) on the OF switches.

6.3 Evaluation of Inter-Subnets Communication

This section evaluates the proposed architecture in terms of communication between hosts on various subnets. The second part's purpose is to validate the network hosts' communication accuracy, security, and anonymity. Notably, networking devices in the data plane, such as OpenFlow switch, serve as a router if each host in the network is issued an IP address from a separate subnet than the other hosts. Each host must be allocated an IP address and a gateway during this examination. By setting hosts with the /30 CIDR (Classless Inter-Domain Routing) prefix, each subnet comprises a maximum of one host. As with Part 1, all communications between couple of hosts must take place through an EIP Channel.

6.3.1 Test Topology and Hosts Configuration

To evaluate the proposed framework, we have used the same network topology mentioned in section 6.2 Fig. 6.1. In this scenario, all Hosts are configured on different subnets with /30 CIDR prefix and their configurations are mentioned in Table. 6.2 which includes IP address, gateway address and MAC address. To evaluate our proposed *Packet Forwarding, and ARP Request Resolution* modules, we created a scenario in which Host 2 is desirous to acquire a particular service form Host 4. Following fields are used in flow rules to guide packet flow from source to destination.

1. **Match**: These fields are used to match the incoming packet header fields. In this scenario, following fields are used.

   (a) Destination IP Address

2. **Actions**: If the packet header fields are matched then actions are performed e.g., forwarding the packet to output port, etc. In this scenario, following actions are performed.

   (a) Out Port: Output port is identified by controller and Packet is forwarded to that port

   (b) Set Field: Ethernet Destination Address is updated

3. **Idle Timeout**: This filed is used to remove a flow entry from the switch after the given number of seconds, if no packet has been matched by the flow. In this scenario, Idle timeout is set to 50 second.

TABLE 6.2: EVALUATION PART 2: LIST OF ALL HOSTS AND THEIR IP CONFIGURATIONS ALONG WITH DEFAULT GATEWAYS /30 CIDR PREFIX. HERE EACH HOST BELONGS TO AN INDIVIDUAL NETWORK HAVING ITS UNIQUE NETWORK ADDRESS.

Host Name	IP Address	MAC Address	Default Gateway
Host 1	192.168.10.5	00:00:00:00:10:05	192.168.10.6
Host 2	192.168.10.9	00:00:00:00:10:09	192.168.10.10
Host 3	192.168.10.13	00:00:00:00:10:13	192.168.10.14
Host 4	192.168.10.17	00:00:00:00:10:17	192.168.10.18
Host 5	192.168.10.21	00:00:00:00:10:21	192.168.10.22
Host 6	192.168.10.25	00:00:00:00:10:25	192.168.10.26
Host 7	192.168.10.29	00:00:00:00:10:29	192.168.10.30
EIP Channel	192.168.10.1	00:00:00:00:10:01	192.168.10.2

6.3.2 Results and Discussions

As discussed before in section 6.3.1, the configuration of hosts is critical for the safety of every host and computer /network, as hosts are configured on distinct subnets. The IP address of each communication host is accessible only to EIP Channel and is concealed from the other hosts in the aforementioned circumstance. During the host registration phase, hosts that provide applications service have to register their application services, and these services are available to all hosts enrolled in the network without exposing their IP addresses. When an application needs to obtain services from another application, it requests an SDN controller to apply flow rules on switches. The SDN controller builds a path between the requested application and the EIP by virtue of the Registration module's awareness of all hosts and their services. On the EIP Channel, a service-specific queue is maintained that contains all messages from various hosts that pertain to a given service. The requested application uses the same approach to get its messages and creates responses for each message over the EIP Channel. This method will assist in concealing the IP addresses of other hosts on the same corporate network, hence supporting/strengthening the communication's security feature

6.3.2.1 Performance w.r.t Flow Rules Installation

The communication correctness may be checked by examining the flow rules that support each communication. As previously described, a situation has been developed in which Host 2 wishes to obtain a service offered by Host 4, but Host 2 is uninformed of Host 4's IP address. In this case, Host 2 seeks the needed service from the SDN Controller. Using the Registration module to get global visibility, the SDN controller applies flow rules among Host 2 and the EIP channel through the Forwarding module. The flow rules established between Host 2 and the EIP Channel are depicted in Fig. 6.5. As seen in Fig. 6.5, the EIP Channel's IP address is compared to the network destination node in the Match field, and then the Action field is executed by assigning the EIP Channel's real MAC address, and the frame is sent to the switch's uplink port. Because the arriving message from any host carries the destination MAC address as the gateway MAC address, the MAC address is replaced. The SDN controller answers the gateway MAC address by substituting it with the real EIP Channel MAC address via the flow rule's "SET FIELD" action.

Similarly, messages cached for Host 4 can be consumed at any moment by requesting that the SDN controller apply flow rules between Host 4 and the EIP Channel. Fig. 6.6 illustrates all the pertinent flow rules implemented all along path connecting Host 4 to EIP Channel. The performance of this framework may be validated by the results shown in Figures 6.5 and 6.6, which demonstrate that the proposed framework accurately sets rules on SDN switches to enable communication between two hosts on separate subnets.

6.3.2.2 Security

The hosts' configuration mentioned in this evaluation i.e., each host must be configured on different subnets using

30 CIDR prefix, can eradicate the possibility of *Port or Network scanning attacks* [109], which are used for reconnaissance and can lead to further harmful attacks such as Denial of Service (DoS), Man-In-The-Middle (MITM), etc., attacks. For example, an adversary compromises any host in an enterprise network using our proposed communication framework, the information obtained from the compromised host, such as IP or MAC addresses, would make it difficult for the adversary to launch an attack or scan the entire network because IP address of each host is concealed from other hosts.

To verify the above security claim, NMAP network scanner is used to test the security

OVS-2	OVS-3	OVS-5

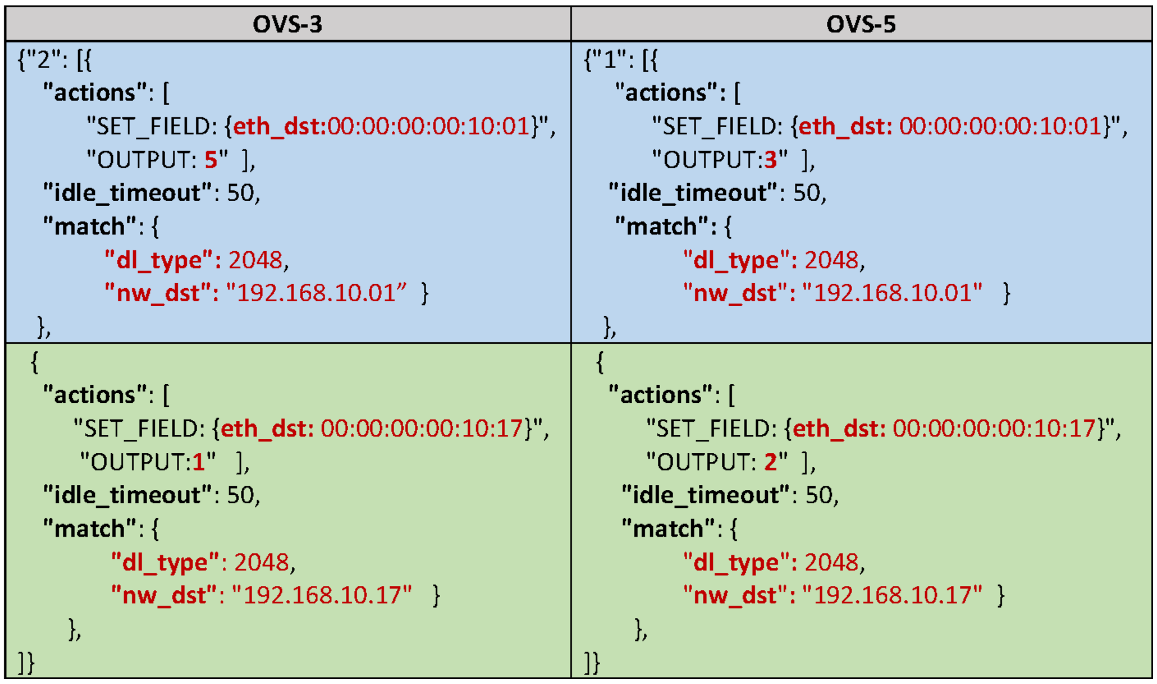
{"2": [{
"actions": [
"SET_FIELD: {eth_dst: 00:00:00:00:10:01}",
"OUTPUT: 4"],
"idle_timeout": 50,
"match": {
"dl_type": 2048,
"nw_dst": "192.168.10.01" }
},
	{"1": [{
	"actions": [
	"SET_FIELD: {eth_dst: 00:00:00:00:10:01}",
	"OUTPUT:5"],
	"idle_timeout": 50,
	"match": {
	"dl_type": 2048,
	"nw_dst": "192.168.10.01" }
	},
		{"4": [{
		"actions": [
		"SET_FIELD:{eth_dst: 00:00:00:00:10:01}",
		"OUTPUT: 1"],
		"idle_timeout": 50,
		"match": {
		"dl_type": 2048,
		"nw_dst": "192.168.10.01" }
		},

{
"actions": [
"SET_FIELD: {eth_dst: 00:00:00:00:10:09}",
"OUTPUT:1"],
"idle_timeout": 50,
"match": {
"dl_type": 2048,
"nw_dst": "192.168.10.9" }
},
]}
	{
	"actions": [
	"SET_FIELD: {eth_dst: 00:00:00:00:10:09}",
	"OUTPUT: 3"],
	"idle_timeout": 50,
	"match": {
	"dl_type": 2048,
	"nw_dst": "192.168.10.9" }
	},
	]}
		{
		"actions": [
		"SET_FIELD: {eth_dst: 00:00:00:00:10:09}",
		"OUTPUT:2"],
		"idle_timeout": 50,
		"match": {
		"dl_type": 2048,
		"nw_dst": "192.168.10.9" }
		},
		]}

FIGURE 6.5: FLOW RULES DEPLOYED ON OF SWITCHES BETWEEN THE SOURCE (HOST 2) AND DESTINATION(EIP CHANNEL). BOTH THE SOURCE AND DESTINATION HOSTS BELONG TO DIFFERENT NETWORKS. HERE HOST 2 IS PUBLISHING ITS MESSAGES FOR HOST 4 ON EIP CHANNEL.

of the proposed communication framework. The results obtained from the NMAP scans are listed in Table. 4 . Here it is assumed that an adversary has compromised Host 4 and obtained its IP address i.e., 192.168.10.17. Based on the obtained IP address of Host 4, it has launched a scan to gain information of other hosts inside the network. It can be verified from the Table. 4 that the first NMAP scan returns only two active hosts on the network i.e., host with IP address 192.168.10.18 and 192.168.10.17. These two hosts are Host-4 (itself) and its gateway. The remaining hosts that are mentioned in the topology Fig. 6.1 and having the configuration listed in the Table. 6.1 were alive but were not detected by the NMAP scanner. The reason behind this result is that this framework only allows hosts to communicate with each other through EIP Channel. Any direct communication request between hosts is blocked by the SDN controller by not deploying flow rules to support such communication.

Although the IP address of the SDN Controller (RYU Controller) is hidden from the hosts, to verify the security of the SDN controller a port scanning attack is also conducted for the RYU controller. RYU controller uses port # 6633 to communicate with the switches deployed on the data plane using OpenFlow to install flow rules. However, the NMAP network scanner was unable to detect this open port. Likewise, another NMAP scan i.e., TCP SYN port scan was also initiated on all the active hosts



OVS-3

```
{"2": [{
   "actions": [
        "SET_FIELD: {eth_dst:00:00:00:00:10:01}",
        "OUTPUT: 5"  ],
   "idle_timeout": 50,
   "match": {
          "dl_type": 2048,
          "nw_dst": "192.168.10.01"  }
  },
  {
   "actions": [
       "SET_FIELD: {eth_dst: 00:00:00:00:10:17}",
        "OUTPUT:1"   ],
   "idle_timeout": 50,
   "match": {
          "dl_type": 2048,
          "nw_dst": "192.168.10.17"   }
      },
]}
```

OVS-5

```
{"1": [{
   "actions": [
        "SET_FIELD: {eth_dst: 00:00:00:00:10:01}",
        "OUTPUT:3"  ],
  "idle_timeout": 50,
   "match": {
          "dl_type": 2048,
          "nw_dst": "192.168.10.01"   }
  },
  {
  "actions": [
       "SET_FIELD: {eth_dst: 00:00:00:00:10:17}",
       "OUTPUT: 2"  ],
   "idle_timeout": 50,
   "match": {
          "dl_type": 2048,
          "nw_dst": "192.168.10.17"  }
      },
]}
```

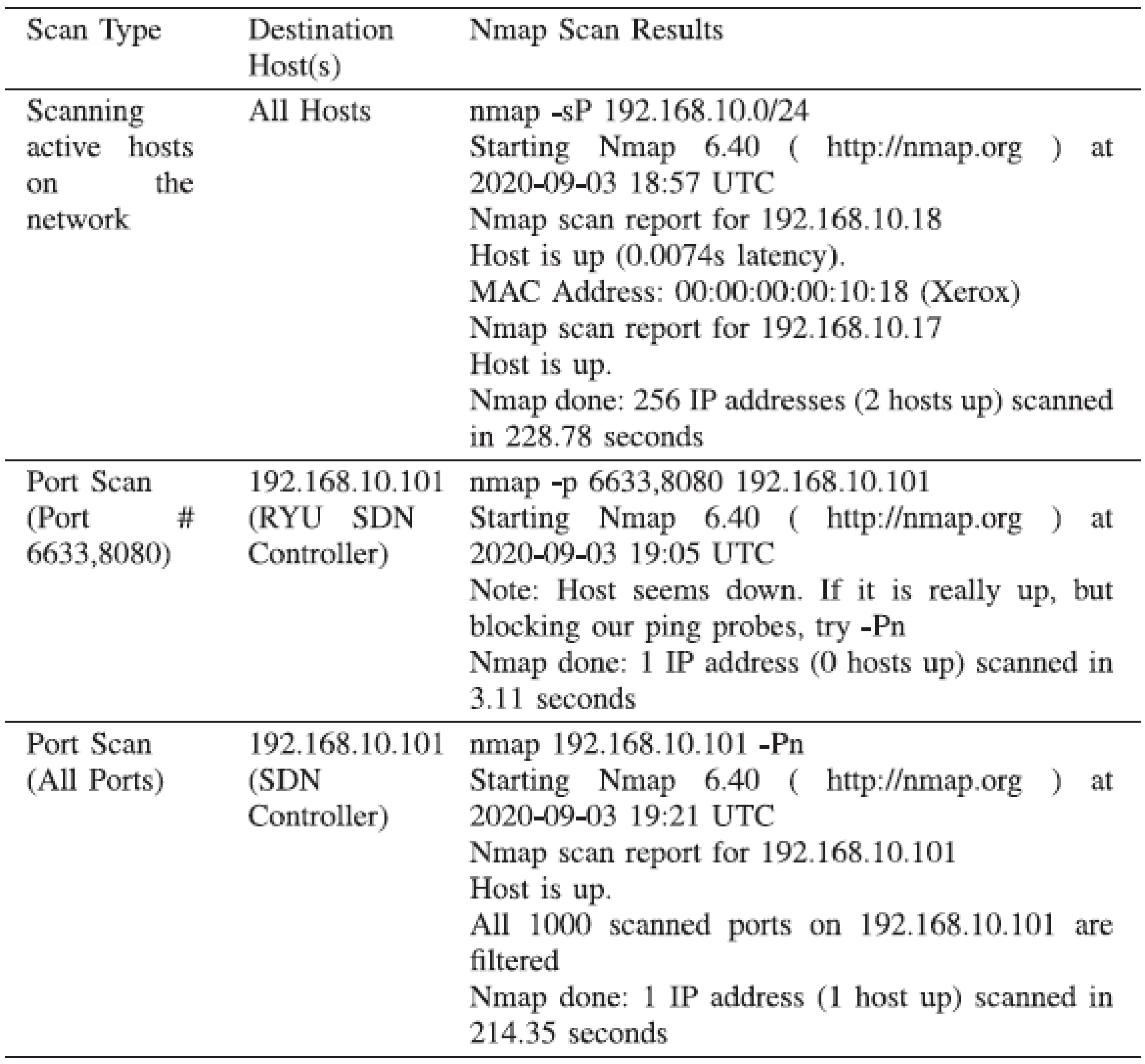
FIGURE 6.6: FLOW RULES DEPLOYED ON OF SWITCHES BETWEEN THE SOURCE (HOST 4) AND DESTINATION(EIP CHANNEL). HERE HOST 4 IS CONSUMING ITS MESSAGES, SENT BY HOST 2, STORED ON EIP CHANNEL.

mentioned in the Table. 6.1. As it can be noticed that apart from the source node (Host-4), none of the active hosts were detected. All the NMAP scans results listed in Table 6.2 further validates our stance of built-in security of enterprise networks using the proposed communication framework.

Furthermore, the information of each host such as IP address, MAC address, switch port number, and switch IDs, obtained during the registration phase by the proposed *Host Registration* module is not modifiable i.e., once the information is collected by the SDN controller then it is fixed. This feature of our proposed framework eradicates the possibility of *ARP Spoofing attack* [110]. Authors in [111] have also presented a similar approach to eliminate ARP spoofing attacks.

Based on the security analysis presented in this section the proposed communication framework offers built-in security from gaining reconnaissance about the network hosts, preventing ARP spoofing attacks, and eradicating port or network scanning attacks.

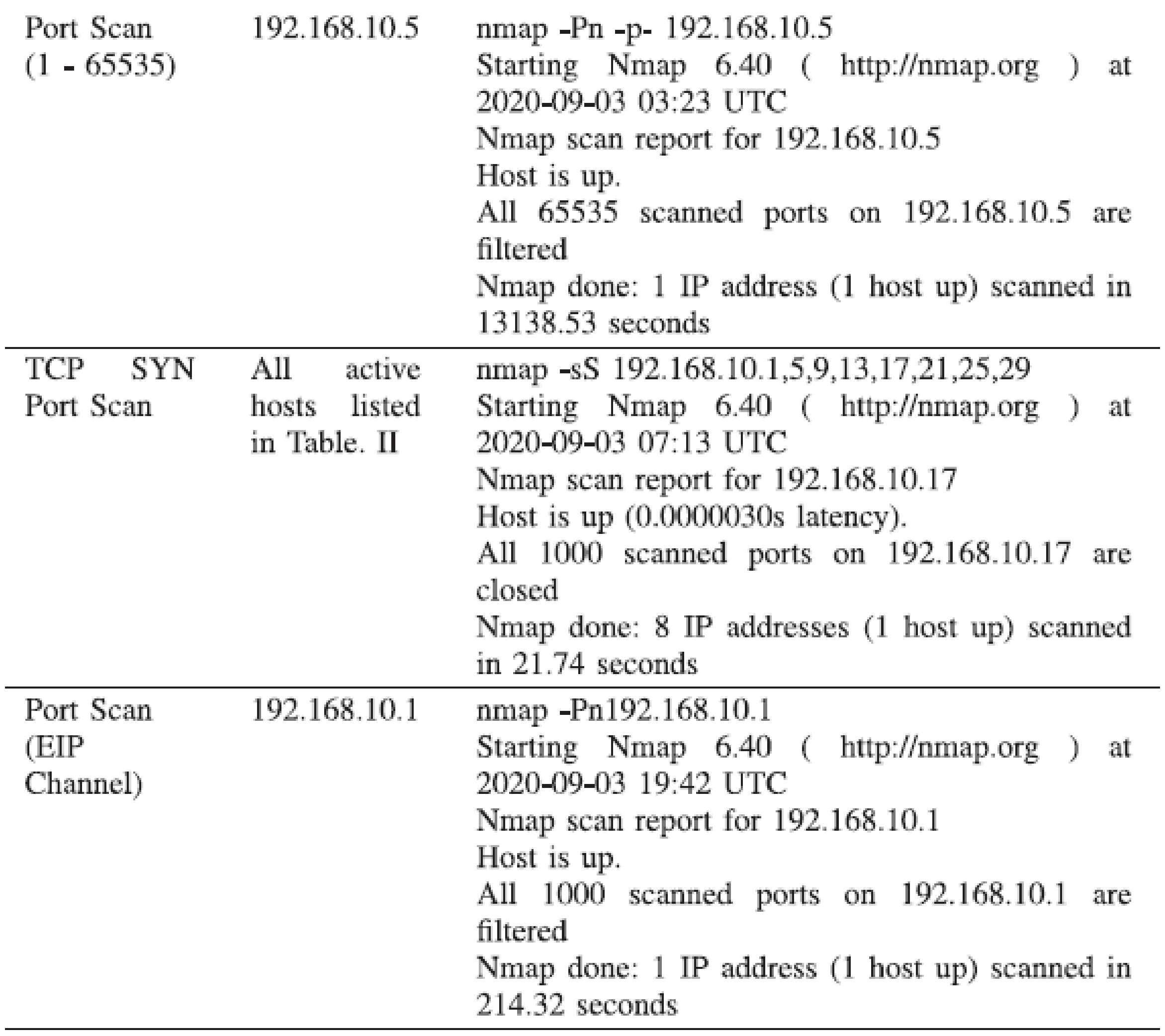
TABLE III
SECURITY EVALUATION OF SCENARIO 2 USING "NMAP NETWORK SCANNER." ALL SCANNING TESTS ARE DONE FROM HOST 4 WITH IP ADDRESS 192.168.10.17/30



Scan Type	Destination Host(s)	Nmap Scan Results
Scanning active hosts on the network	All Hosts	nmap -sP 192.168.10.0/24 Starting Nmap 6.40 (http://nmap.org) at 2020-09-03 18:57 UTC Nmap scan report for 192.168.10.18 Host is up (0.0074s latency). MAC Address: 00:00:00:00:10:18 (Xerox) Nmap scan report for 192.168.10.17 Host is up. Nmap done: 256 IP addresses (2 hosts up) scanned in 228.78 seconds
Port Scan (Port # 6633,8080)	192.168.10.101 (RYU SDN Controller)	nmap -p 6633,8080 192.168.10.101 Starting Nmap 6.40 (http://nmap.org) at 2020-09-03 19:05 UTC Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn Nmap done: 1 IP address (0 hosts up) scanned in 3.11 seconds
Port Scan (All Ports)	192.168.10.101 (SDN Controller)	nmap 192.168.10.101 -Pn Starting Nmap 6.40 (http://nmap.org) at 2020-09-03 19:21 UTC Nmap scan report for 192.168.10.101 Host is up. All 1000 scanned ports on 192.168.10.101 are filtered Nmap done: 1 IP address (1 host up) scanned in 214.35 seconds

6.4 Evaluation of Anonymous Communication

This section evaluates our proposed communication system in terms of hosts anonymity and the overall security of an SDN-based enterprise network. To do this, we designed a scenario where Host-2 requests a service from Host-6 via an asynchronous message exchange over the EIP Channel. The threat model for this article is that any registered host in an enterprise network can get compromised. Therefore, an adversary may use the compromised host to reveal the identity of the other hosts on the network by observing the traffic. Moreover, this framework does not protect against the adversary



Port Scan (1 - 65535)	192.168.10.5	nmap -Pn -p- 192.168.10.5 Starting Nmap 6.40 (http://nmap.org) at 2020-09-03 03:23 UTC Nmap scan report for 192.168.10.5 Host is up. All 65535 scanned ports on 192.168.10.5 are filtered Nmap done: 1 IP address (1 host up) scanned in 13138.53 seconds
TCP SYN Port Scan	All active hosts listed in Table. II	nmap -sS 192.168.10.1,5,9,13,17,21,25,29 Starting Nmap 6.40 (http://nmap.org) at 2020-09-03 07:13 UTC Nmap scan report for 192.168.10.17 Host is up (0.0000030s latency). All 1000 scanned ports on 192.168.10.17 are closed Nmap done: 8 IP addresses (1 host up) scanned in 21.74 seconds
Port Scan (EIP Channel)	192.168.10.1	nmap -Pn192.168.10.1 Starting Nmap 6.40 (http://nmap.org) at 2020-09-03 19:42 UTC Nmap scan report for 192.168.10.1 Host is up. All 1000 scanned ports on 192.168.10.1 are filtered Nmap done: 1 IP address (1 host up) scanned in 214.32 seconds

who can compromise the switches on the network. We believe that in an enterprise network, switches are physically secured and it is difficult for an adversary to attach traffic analysis device on the switch physically.

6.4.1 Test Topology and Hosts Configuration

The framework is evaluated in terms of anonymous communication between various hosts or application services. To analyze the framework, a tree topology with depth = 2 and fanout = 3 is utilized, as represented in Fig. 6.7. It's worth noting that the proposed architecture facilitates communication between hosts on the same subnet as well as between hosts on separate subnets. Each host in the architecture depicted in Fig. 6.7 is configured on a separate subnet for the purpose of evaluating the proposed framework's anonymity. The IP configuration of each host, as well as its MAC address and gateway address, are shown in Table. 6.3.

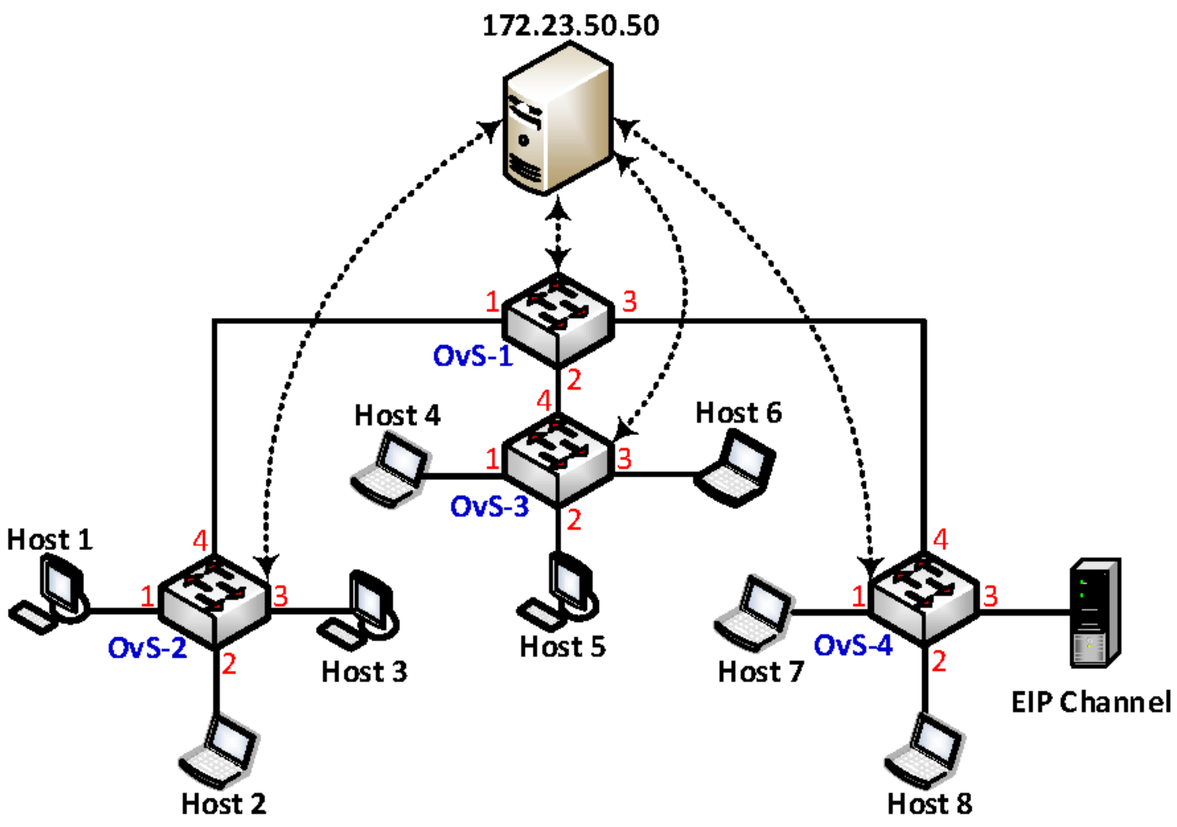


Figure 6.7: A test topology to evaluate anonymous communication of proposed framework.

1. **Match**: These fields are used to match the incoming packet header fields. In this scenario, following fields are used.

   (a) Destination IP Address

   (b) Source IP Address

2. **Actions**: If the packet header fields are matched then actions are performed e.g., forwarding the packet to output port, etc. In this scenario, following actions are performed.If the packet header fields are matched then actions are performed e.g., forwarding the packet to output port, etc. In this scenario, following actions are performed.

   (a) Out Port: Output port is identified by controller and Packet is forwarded to that port

   (b) Set Field: Ethernet Destination Address is changed

   (c) Set Field: Source IP Address is changed to its corresponding hoax IP address

3. **Idle Timeout**: This filed is used to remove a flow entry from the switch after the given number of seconds, if no packet has been matched by the flow. In this scenario, Idle timeout is set to 100 second.This filed is used to remove a flow entry from the switch after the given number of seconds, if no packet has been matched by the flow. In this scenario, Idle timeout is set to 100 second.

TABLE 6.3: CONFIGURATION OF EACH HOST ON TOPOLOGY DEPICTED IN FIG. 6.7

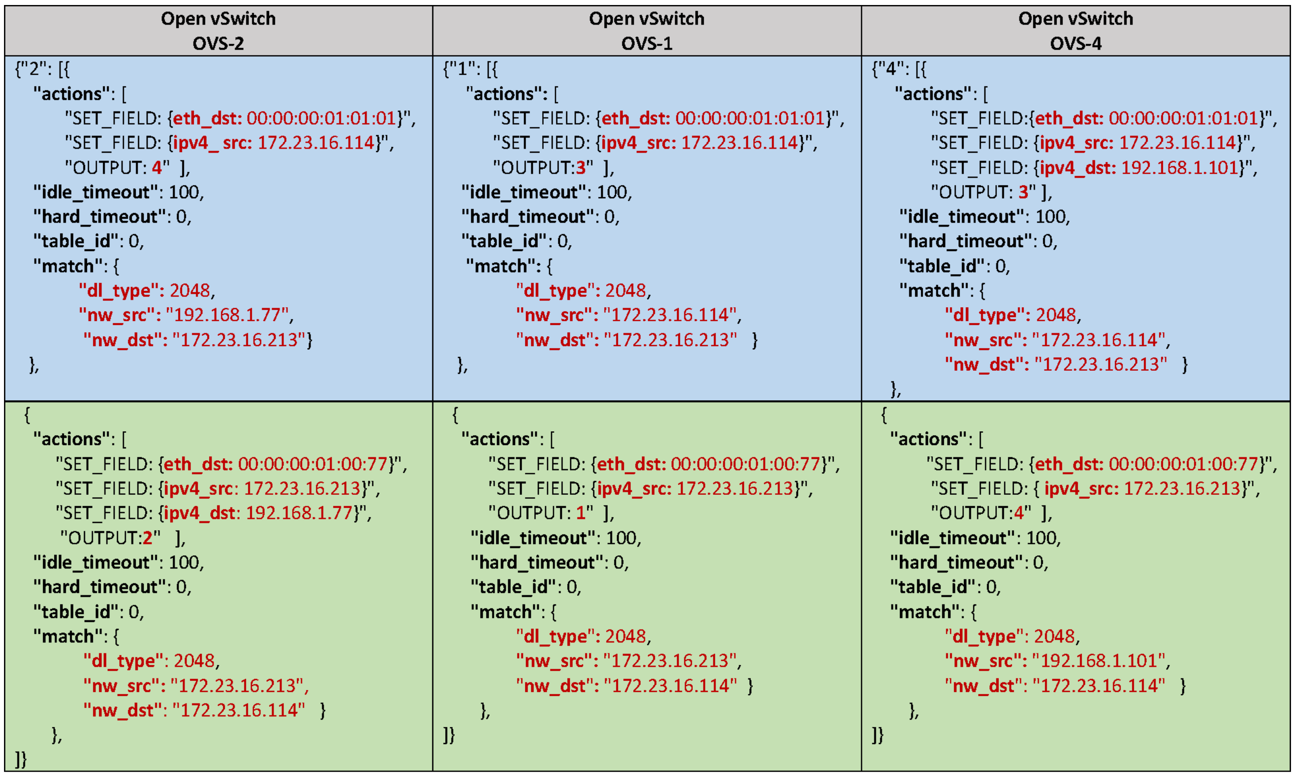
Host_Name	IP_Address	MAC_Address	Default_Gateway
EIP Channel	192.168.1.101	00:00:00:01:01:01	192.168.1.102
Host-1	192.168.1.57	00:00:00:01:00:57	192.168.1.58
Host-2	192.168.1.77	00:00:00:01:00:77	192.168.1.78
Host-3	192.168.1.129	00:00:00:01:01:29	192.168.1.130
Host-4	192.168.1.165	00:00:00:01:01:65	192.168.1.166
Host-5	192.168.1.181	00:00:00:01:01:81	192.168.1.182
Host-6	192.168.1.193	00:00:00:01:01:93	192.168.1.194
Host-7	192.168.1.213	00:00:00:01:02:13	192.168.1.214
Host-8	192.168.1.245	00:00:00:01:02:45	192.168.1.246

TABLE 6.4: HOAX VS ACTUAL IP ADDRESSES. THESE ADDRESSES ARE GENERATED DURING THE SIMULATION AND TESTING OF FRAMEWORK.

Host Name	Real IP Address	Hoax IP Address
EIP Channel	192.168.1.101	172.23.16.213
Host-1	192.168.1.57	172.23.16.205
Host-2	192.168.1.77	172.23.16.114
Host-3	192.168.1.129	172.23.16.67
Host-4	192.168.1.165	172.23.16.181
Host-5	192.168.1.181	172.23.16.165
Host-6	192.168.1.193	172.23.16.27
Host-7	192.168.1.213	172.23.16.147
Host-8	192.168.1.245	172.23.16.137

6.4.2 Performance and Results Analysis

As explained in Sec. 4.1, the framework's objective is to facilitate secure communication across several hosts in an enterprise setting. As a result, the correctness of anonymous communication between hosts is critical. Additionally, Messaging System



Open vSwitch
OVS-2

```
{"2": [{
   "actions": [
       "SET_FIELD: {eth_dst: 00:00:00:01:01:01}",
       "SET_FIELD: {ipv4_ src: 172.23.16.114}",
       "OUTPUT: 4"  ],
   "idle_timeout": 100,
   "hard_timeout": 0,
   "table_id": 0,
   "match": {
         "dl_type": 2048,
         "nw_src": "192.168.1.77",
          "nw_dst": "172.23.16.213"}
  },
 {
   "actions": [
       "SET_FIELD: {eth_dst: 00:00:00:01:00:77}",
       "SET_FIELD: {ipv4_src: 172.23.16.213}",
       "SET_FIELD: {ipv4_dst: 192.168.1.77}",
        "OUTPUT:2"   ],
   "idle_timeout": 100,
   "hard_timeout": 0,
   "table_id": 0,
   "match": {
          "dl_type": 2048,
          "nw_src": "172.23.16.213",
          "nw_dst": "172.23.16.114"   }
      },
]}
```

Open vSwitch
OVS-1

```
{"1": [{
   "actions": [
       "SET_FIELD: {eth_dst: 00:00:00:01:01:01}",
       "SET_FIELD: {ipv4_src: 172.23.16.114}",
       "OUTPUT:3"  ],
  "idle_timeout": 100,
  "hard_timeout": 0,
  "table_id": 0,
   "match": {
         "dl_type": 2048,
         "nw_src": "172.23.16.114",
         "nw_dst": "172.23.16.213"   }
  },
 {
   "actions": [
       "SET_FIELD: {eth_dst: 00:00:00:01:00:77}",
       "SET_FIELD: {ipv4_src: 172.23.16.213}",
       "OUTPUT: 1"  ],
   "idle_timeout": 100,
   "hard_timeout": 0,
   "table_id": 0,
   "match": {
         "dl_type": 2048,
         "nw_src": "172.23.16.213",
         "nw_dst": "172.23.16.114"  }
     },
]}
```

Open vSwitch
OVS-4

```
{"4": [{
    "actions": [
        "SET_FIELD:{eth_dst: 00:00:00:01:01:01}",
        "SET_FIELD: {ipv4_src: 172.23.16.114}",
        "SET_FIELD: {ipv4_dst: 192.168.1.101}",
        "OUTPUT: 3" ],
    "idle_timeout": 100,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
          "dl_type": 2048,
          "nw_src": "172.23.16.114",
          "nw_dst": "172.23.16.213"   }
   },
 {
   "actions": [
        "SET_FIELD: {eth_dst: 00:00:00:01:00:77}",
        "SET_FIELD: { ipv4_src: 172.23.16.213}",
        "OUTPUT:4"  ],
   "idle_timeout": 100,
   "hard_timeout": 0,
   "table_id": 0,
   "match": {
          "dl_type": 2048,
          "nw_src": "192.168.1.101",
          "nw_dst": "172.23.16.114"   }
      },
]}
```

FIGURE 6.8: OF ENABLED SWITCHES HAVE BEEN CONFIGURED WITH RULES TO FACILITATE ANONYMOUS COMMUNICATION BETWEEN HOST-2 AND THE EIP CHANNEL. HOST-2'S ACTUAL IP ADDRESS IS 192.168.1.77 AND EIP'S IS 192.168.1.101.

(EIP Channel) has the property of providing message dependability and fault-tolerance behaviour in the event of a hardware failure on any host.

To enable anonymous communication, this framework uses the *IP Address Mapper* module to produce bogus IP addresses and compares them to genuine IP addresses. These bogus IP addresses are produced at random from the system administrator's specified IP pool list. The table 6.4 compares bogus IP addresses generated during the simulation to genuine IP addresses. The administrator inserts the 172.23.16.0/24 network pool here to produce bogus IP addresses. Once the bogus IP address is generated against the host, every further communication will use the bogus IP address. We discuss the computational complexity, verify the accuracy of anonymous communication, and assess the framework's security in the following subsections.

6.4.2.1 Computational Complexity

As explained earlier in Sec. 5.1.2, this architecture depends significantly on the Mapper and Forwarding modules to provide packet anonymity. To validate the efficiency of

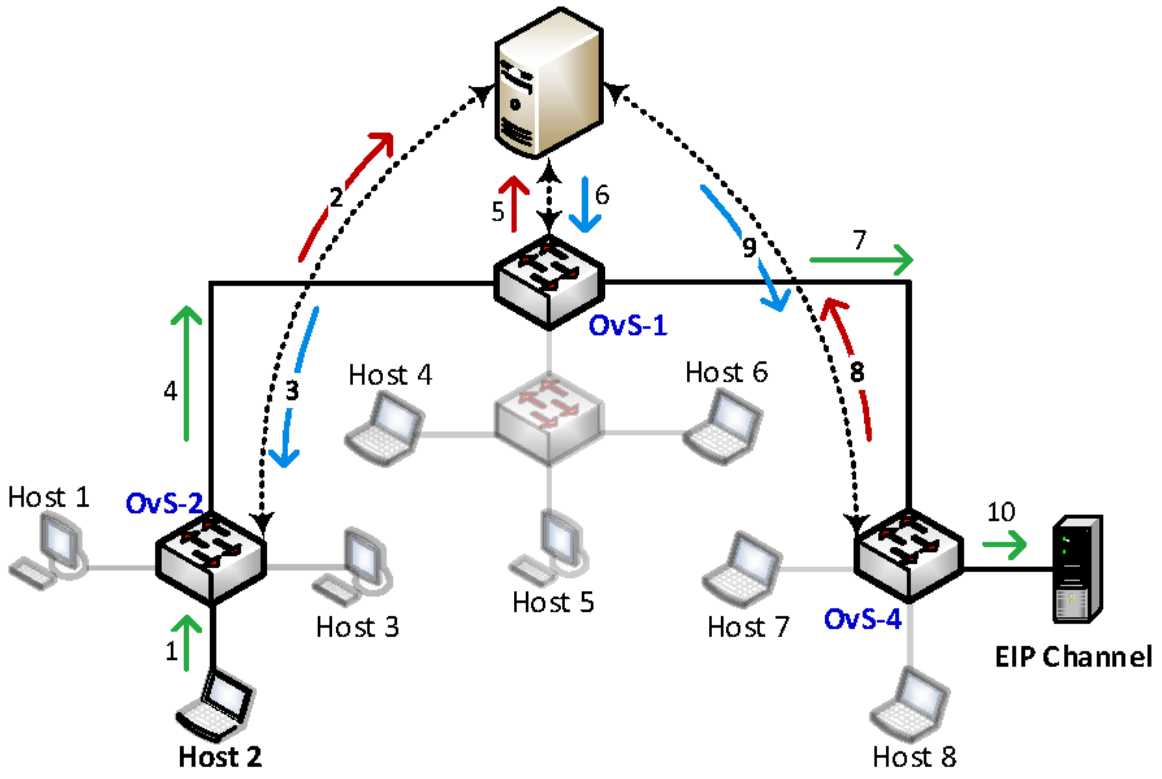


FIGURE 6.9: SERIES OF ACTIONS REQUIRED WHEN IMPLEMENTING FLOW RULES ON SWITCHES.

our framework, we employ Asymptotic Notation [112] to evaluate the time complexity of our Mapper and Forwarding modules. The phrase "time complexity" represents the time needed for an algorithm to finish its operation. The most well-known is asymptotic analysis, which employs Big-O notation to characterise an algorithm's worst-case situation as the input size rises. Following computations, the time complexity of the Mapper module, as given in Algo. 1, to generate a new false IP address or to obtain the genuine IP address using Big-O notation is mentioned below. Here, N is the total number of hosts in the IP address mapper's list.

$$IP\ Address\ Mapper = O(N)$$

Additionally, a loop is used to search for inactive spoofed IP addresses in the *MappedIP* dictionary. As a result, the time required to delete the stale faked IP addresses is as follow.

$$Removal\ of\ inactive\ spoofed\ IP\ addresses = O(N)$$

The communication is supported by our Forwarding module; moreover, it also requires the verification of each communicating host, as well as the creation of the hoax IP address or retrieval of the real IP address using the Mapper module; thereby, the overall time complexity of this anonymous communication to install flow rules for each Packet IN request is quite high.

$$Packet\ Forwarding = O(3N) + O(N)$$

Applying asymptotic notation rules by ignoring the coefficient of the highest order term. The entire complexity of time will be as follows.

$$System\ time\ complexity = O(N)$$

The aforementioned asymptotic analysis considerably enhances the efficiency of our proposed system. Similarly, by setting a timeout value in the idle timeout field of the flow rule, this framework discards the flow entry from the switch after a given period of inactivity (in Fig. 6.8, the idle timeout value is set to 100 seconds). This enables us to conserve TCAMs, which also supports our proposed framework's memory footprint efficiency claim.

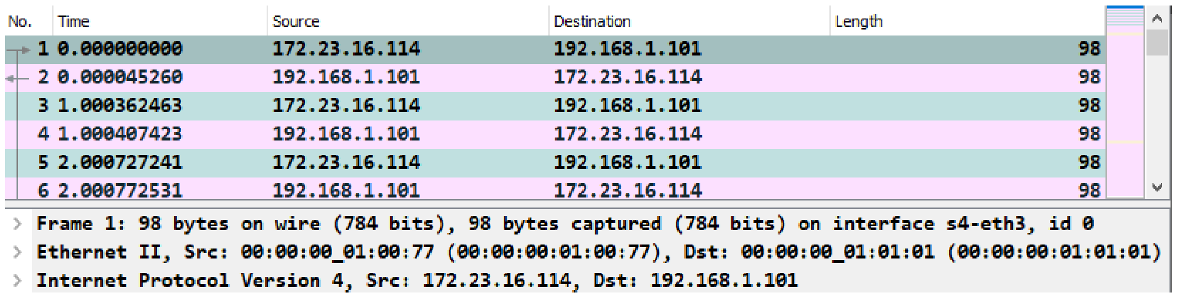
6.4.2.2 Accuracy w.r.t Anonymous Communication

To show the proposed framework's ability to support anonymous communication, we created a scenario in which Host-2 seeks to request a service from Host-6 through an asynchronous message exchange over the EIP Channel. Host-2 is assigned the bogus IP address 172.23.16.114 upon registration. It requires a list of the accessible application services on the network. When a service request is received, the Service Registry module reacts by producing a list of all enlisted application services. Host-2 picks an application service from a list and sends it a message (Host-6). Because communication between hosts is authorised only via EIP Channel, the SDN controller must first compute a route connecting Host-2 and EIP Channel and then configure flow rules to enable communication.

The SDN controller applies flow rules that route traffic through OF-switch-1 to OF-switch-4. when the packet reaches the EIP Channel (OF-switch-4), the destination fake IP address is replaced with the true EIP Channel IP address. The flow rules that have been established between Host-2 and the EIP Channel are depicted in Fig. 6.8. Likewise, to disguise the source IP address, the SDN controller guarantees that the flow rule substitutes the transmitting host's fake IP address for the genuine source IP address.

When a message between Host-2 and Host-6 reaches the EIP Channel, it is routed to the EIP Channel's service-specific queue, i.e., the service offered by Host-6. The application operating on Host-6 can consume its messages at any point in time by beginning communication with the EIP Channel, wherein the SDN controller will configure separate flow rules.

Additionally, we use the Wireshark [113] packet analyzer to prove our claim about anonymous communication. Both Fig. 6.10 and Fig. 6.11 reveal that Host-2 and EIP Channel are unable to divulge the genuine IP address of the other. It's worth remembering that the SDN controller will deny any attempt to send a message directly to another server on the network. Additionally, the SDN controller only supports communication between EIP Channels that are either senders or receivers. As a result of this communication topology, the requirement to conceal MAC addresses is eliminated.



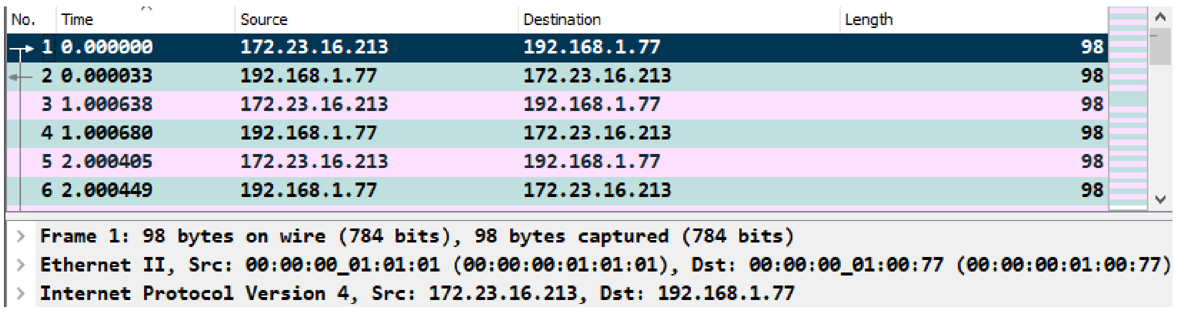
```
No. Time          Source          Destination     Length
  1 0.000000000   172.23.16.114   192.168.1.101   98
  2 0.000045260   192.168.1.101   172.23.16.114   98
  3 1.000362463   172.23.16.114   192.168.1.101   98
  4 1.000407423   192.168.1.101   172.23.16.114   98
  5 2.000727241   172.23.16.114   192.168.1.101   98
  6 2.000772531   192.168.1.101   172.23.16.114   98

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface s4-eth3, id 0
> Ethernet II, Src: 00:00:00_01:00:77 (00:00:00:01:00:77), Dst: 00:00:00_01:01:01 (00:00:00:01:01:01)
> Internet Protocol Version 4, Src: 172.23.16.114, Dst: 192.168.1.101
```

FIGURE 6.10: SNAPSHOT OF WIRESHARK OF HOST WITH IP ADDRESS 192.168.1.101 (EIP CHANNEL).

6.4.2.3 Security

As explained in Sec. 6.4, our threat model presupposes that each host in an organization's network is vulnerable to compromise. Once a host has been infiltrated, an adversary can monitor its inbound and outbound traffic by employing any packet analyzer, such as Wireshark, Metasploit, or others.



```
No. Time       Source          Destination      Length
1   0.000000   172.23.16.213   192.168.1.77     98
2   0.000033   192.168.1.77    172.23.16.213    98
3   1.000638   172.23.16.213   192.168.1.77     98
4   1.000680   192.168.1.77    172.23.16.213    98
5   2.000405   172.23.16.213   192.168.1.77     98
6   2.000449   192.168.1.77    172.23.16.213    98

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: 00:00:00_01:01:01 (00:00:00:01:01:01), Dst: 00:00:00_01:00:77 (00:00:00:01:00:77)
> Internet Protocol Version 4, Src: 172.23.16.213, Dst: 192.168.1.77
```

FIGURE 6.11: SNAPSHOT OF WIRESHARK OF HOST WITH IP ADDRESS 192.168.1.77 (HOST - 2).

We utilised the identical scenario of Host-2 sending messages to Host-6 through the EIP Channel to verify the proposed framework's security. Assume that Host-2 has been compromised and that an adversary is able to study the details of the incoming packets utilizing Wireshark, as demonstrated in Fig. 6.11. An attacker can use the source IP address to scan the whole network in order to find the accessible hosts. In the above scenario, the Nmap network scanner [115]is used to assess ours framework's security via a series of host discovery scans. The below-mentioned table summarises the findings from a series of Nmap scans: 6.5. By checking the source address of an incoming packet in Wireshark, for example, 172.23.16.213, the attacker can launch a scan to find all potential hosts on the network that use the network address 172.23.16.0/24. As indicated in the Table's top row. 6.5, the Nmap scan returns only the EIP Channel as a result. While all hosts are connected to the network and utilise Hoax IP addresses from the 172.23.16.0/24 network, the Nmap scanning programme did not detect them. This is because the proposed framework allows communication between two hosts only through EIP Channels; any explicit inquiry from each host is denied by the SDN controller.

Similarly, an attacker can do another scan by examining the IP address of the compromised host. In this case, Host-2's true IP address is 192.168.1.77, and a scan is generated to determine the location of other hosts on the network using that IP address. As seen in the second row of Table 6.5, the Nmap scanning programme detects just two hosts: Host-2 and its gateway with the IP address 192.168.1.78. Additionally, Nmap was unable to discover the EIP Channel that Host-2 uses to communicate with other hosts on the network.

TABLE 6.5: SECURITY VERIFICATION USING NMAP SCANNING TOOL. ASSUMING THAT HOST-2 IS COMPROMISED BY AN ADVERSARY WITH REAL IP ADDRESS 192.168.1.77/30.

Scan Description	**Scanned Host(s)**	**Scan Results**
Scanning active hosts using hoax IP address range extracted from Wireshark scan	All Hosts	**nmap -sP 172.23.16.0/24** Starting Nmap 6.40 (http://nmap.org) at 2020-11-22 17:43 UTC Nmap scan report for 172.23.16.213 Host is up (0.50s latency). Nmap done: 256 IP addresses (1 host up) scanned in 83.61 second
Scanning active hosts using real IP address range	All Hosts	**nmap -sP 192.168.1.0/24** Starting Nmap 6.40 (http://nmap.org) at 2020-11-22 17:47 UTC Nmap scan report for 192.168.1.78 Host is up (0.014s latency). MAC Address: 00:00:00:01:00:78 (Xerox) Nmap scan report for 192.168.1.77 Host is up. Nmap done: 256 IP addresses (2 hosts up) scanned in 228.74 seconds
Port Scanning	EIP Channel	**nmap -Pn 172.23.16.213** Starting Nmap 6.40 (http://nmap.org) at 2020-11-22 18:04 UTC Nmap scan report for 172.23.16.213 Host is up. All 1000 scanned ports on 172.23.16.213 are filtered Nmap done: 1 IP address (1 host up) scanned in 165.77 seconds

The last Nmap scan examines the EIP Channel's ports. Nmap recognised the EIP Channel's existence but was unable to access any of the open ports. These observations, summarised in Table 6.5, support our assertion that a secure communication system is necessary. Additionally, the proposed solution eliminates the possibility of ARP spoofing attacks, as the registration stage permanently stores each host's information, including its IP address, MAC address, OF-Switch number, and related switch port number. As a result, our platform is designed to be more secure than a normal SDN or traditional network in terms of host anonymity, ARP spoofing protection, and network or port scanning risks.The last Nmap scan examines the EIP Channel's ports. Nmap recognised the EIP Channel's existence but was unable to access any of the open ports. These observations, summarised in Table 6.5, support our assertion that a secure communication system is necessary. Additionally, the proposed solution eliminates the possibility of ARP spoofing attacks, as the registration stage permanently stores each host's information, including its IP address, MAC address, OF-Switch number, and related switch port number. As a result, our platform is designed to be more secure than a normal SDN or traditional network in terms of host anonymity, ARP spoofing protection, and network or port scanning risks.

6.5 Summary

In this chapter, we have evaluated the proposed framework with respect to the creation of dynamic VLANs between hosts residing on the same network. In addition, we also evaluated the communication between hosts residing on different subnets. Similarly, we also evaluated the anonymity of hosts during the communication using our framework. In all the evaluation tests, the proposed communication framework has performed well as depicted by the results mentioned in this chapter.we have evaluated the proposed framework with respect to the creation of dynamic VLANs between hosts residing on the same network. In addition, we also evaluated the communication between hosts residing on different subnets. Similarly, we also evaluated the anonymity of hosts during the communication using our framework. In all the evaluation tests, the proposed communication framework has performed well as depicted by the results mentioned in this chapter.

Chapter 7

Conclusion and Further Research

This chapter presents the concluding remarks together with some of the potential future research directions. Section 7.1 presents the conclusions drawn from the research work and Section 7.2 discusses the future research areas and improvements.

7.1 Conclusion

The advent of SDN has changed corporate network administration. SDN-based architecture enables network administration to be flexible across many applications, such as routing, switching, forwarding, and controlling. Additionally, by implementing the SDN paradigm, network managers may eliminate labor-intensive hardware setups and dependency on vendors for security solutions/devices.

Similarly, EIP is a set of patterns for communicating across applications using message-oriented communication. Enterprise networks have integration challenges due to the fact that the majority of applications are created separately, using disparate development platforms, languages, and data formats. EIP is one of the options available for establishing communication between applications operating on separate hosts. Due to the messaging system's asynchronous structure, it enables loose coupling and reliable communication between applications.

In this research, we merged both of the aforementioned technologies, namely EIP and SDN, and provided a novel communication framework that enables corporate networks to communicate in an accurate, efficient, secure, and fault-tolerant manner. We feel that incorporating EIP into SDN benefits network managers and software developers alike. To do this, we developed various SDN modules to support our message communication framework.

This research also evaluates different host configuration layouts using EIP. In the first hosts' configuration layout, all communicating hosts belong to the same network and communicate with each other by using this proposed framework's *Adaptive VLAN Management* module. It is noteworthy that within the same network, hosts can only communicate using VLAN tagging and communication between hosts is only possible through EIP Channel. This *Adaptive VLAN Management* module provides efficient deployment of VLANs between two communicating hosts (any host and the EIP Channel) for the duration of the communication. Moreover, one of the major findings of this research is that in SDN an access port (OF switch port) can be assigned different VLAN IDs to communicate with different hosts simultaneously, which is not allowed in traditional network switches.This research also evaluates different host configuration layouts using EIP. In the first hosts' configuration layout, all communicating hosts belong to the same network and communicate with each other by using this proposed framework's *Adaptive VLAN Management* module. It is noteworthy that within the same network, hosts can only communicate using VLAN tagging and communication between hosts is only possible through EIP Channel. This *Adaptive VLAN Management* module provides efficient deployment of VLANs between two communicating hosts (any host and the EIP Channel) for the duration of the communication. Moreover, one of the major findings of this research is that in SDN an access port (OF switch port) can be assigned different VLAN IDs to communicate with different hosts simultaneously, which is not allowed in traditional network switches.

Similarly, the second hosts' configuration layout is more suitable for enterprises that are mainly concern about the security of their data and hosts. In this configuration, each host belongs to an individual network. It is the job of the SDN controller through our proposed framework's modules such as *Packet Forwarding, ARP Request Resolution* and Network Map DB to provide communication between any host and EIP Channel residing on different networks. The security of this hosts' configuration layout is tested using the NMAP network scanner. The scan results validate that using this hosts' configuration layout, it is difficult for an adversary to gain information about the other hosts inside the network.

This communication framework is the initial step towards the integration of EIP in SDN and can be further enhanced and deployed in various enterprise networks where an organization receives multiple services from external sources such as Banking System, Immigration System, Tax Management System, etc.

7.2 Further Research

In this research, only a handful of modules are proposed to provide basic communication in an enterprise network. However, for this proposed communication framework to be deployed on the actual enterprise network and to completely utilize the benefits of SDN and EIP, few more modules are required. The following list specifies further research directions to enhance the overall communication and security of this framework so that the SDN-based networks can be deployed in enterprise networks without any network reliability concerns.

1. *Spanning Tree Protocol Application* - A loop free logical topology is required by developing an application using Spanning Tree Protocol (STP). Currently this framework does not provide loop free logical topology.

2. *Routing Protocol Application* - Although we have proposed a *Packet Forwarding* module to forward packets from one network to another. However, this *Packet Forwarding* module requires static entries of the physical locations (physical topology) of OF switches deployed on the enterprise network. A standalone routing application can overcome this limitation and can decided the shortest path between communicating hosts dynamically.

3. *Security Applications* - This communication framework provides built-in security and the evaluation results vindicate our claim of secure and anonymous communication. However, to be deployed on actual network, this communication framework further requires following security applications.

   (a) *Access Control List (ACL) application* - ACL application is required for blocking or permitting hosts from certain services in the network.

   (b) *Firewall application* - A centralized firewall application can provide overall network security.

   (c) *Deep Packet Inspection application* - adding this application in this framework can give multiple benefits and through random checking of packet contents helps making real time decisions.

   (d) *IPS / IDS application* - Including IPS / IDS application in this framework can help in identifying malicious or suspicious activity and IPS can proactively take measures to neutralize the threat.

4. *Load Balancer application* - This application can help in balancing the network traffic load between multiple paths. Moreover, it can also be used balance the load between multiple servers offering same services.

5. *Topology Mapping Application* - In the framework, we utilizes a *Network Map DB* module that holds information, which is statically saved, regarding the physical location of each switch. An application is required that automatically build a topology map of all the switches, their access and uplink ports.

6. *Testing the synchronous communicating application using Messaging System* - Although this framework support asynchronous communication. However, few services in enterprise network still requires synchronous mode of communication. For those services / applications, this framework may be tested with few modifications for synchronous communication using Messaging System (EIP).

APPENDIX A

Understanding Simple SDN Topology

To understand how SDN network communicates, we explain it through a simple SDN topology illustrated in Fig. 1. Here Host-1 having IP address 192.168.10.1/24 initiates communication with Host-2 with IP address 192.168.10.2/24. Both hosts are connected to separate switches which support OpenFlow communication. Fig. 1 illustrate a simple example to highlight the communication between two hosts in SDN environment. The whole communication is divided into following steps.

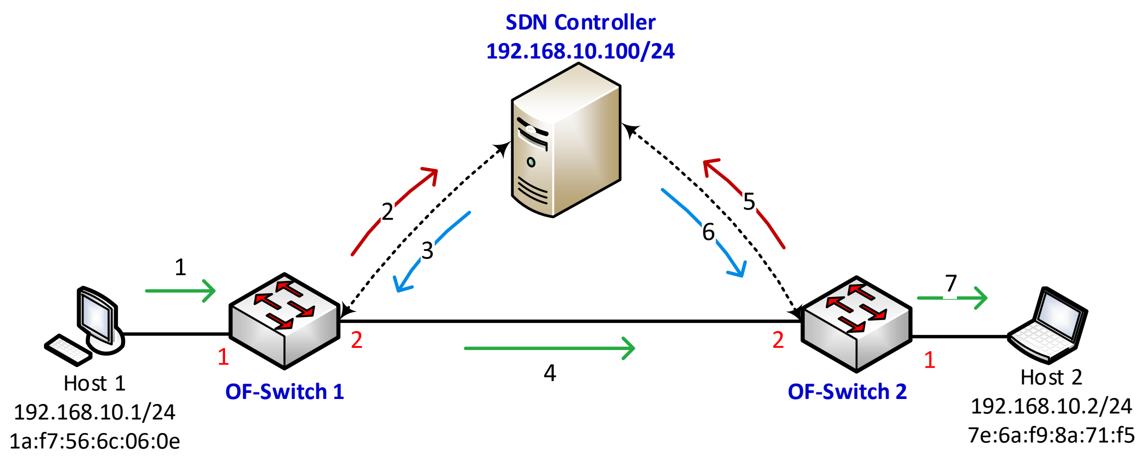


FIGURE 1: A SIMPLE EXAMPLE TO HIGHLIGHT THE COMMUNICATION BETWEEN TWO HOSTS IN SDN ENVIRONMENT.

Step 1: When Host-1 send data packets having Host-2 as destination, these packets are received at OF-Switch 1 through port 1.

Step 2: As OF-Switch 1 has no prior knowledge of Host-2, it generates a control messages known as *Packet_IN message* to SDN controller that contains the data packet sent by Host-1.

Step 3: Upon receiving the Packet_IN message, the SDN controller through application, for example *switching application* can decide that on which port it has to forward this packet. After calculating the path, SDN controller installs a flow rule on OF-Switch 1 using another control message known as *Flow Mod message*. Rule listed

Open vSwitch
OF-Switch 1

```
{"1": [{
    "actions": [
           "OUTPUT: 2"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst": "7e:6a:f9:8a:71:f5",
      "dl_src": "1a:f7:56:6c:06:0e",
      "in_port": 1”}
   },
  {
    "actions": [
           "OUTPUT: 1"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst":"1a:f7:56:6c:06:0e",
      "dl_src":"7e:6a:f9:8a:71:f5",
      "in_port": 2"}
   },
]}
```

Open vSwitch
OF-Switch 2

```
{"2": [{
     "actions": [
           "OUTPUT: 1"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst":"1a:f7:56:6c:06:0e",
      "dl_src":"7e:6a:f9:8a:71:f5",
      "in_port": 2"},
}
  {
    "actions": [
           "OUTPUT: 2"],
    "idle_timeout": 0,
    "packet_count": 5,
    "hard_timeout": 0,
    "table_id": 0,
    "match": {
      "dl_dst":"7e:6a:f9:8a:71:f5",
      "dl_src":"1a:f7:56:6c:06:0e",
      "in_port": 1”}
   },
]}
```

FIGURE 2: FLOW RULES INSTALLED DURING THE COMMUNICATION OF HOST-1 AND HOST-2 ILLUSTRATED IN FIG. 1. HERE RULES MENTIONED IN THE BLUE BOXES REPRESENTS THE FORWARDING PATH AND RULES FOR REVERSE PATH IS LISTED IN GREEN BOXES.

in Blue Box under OF-Switch 1 of Fig. 2.

Step 4: Once the flow rule is installed on OF-Switch 1 regarding the destination Host-2, all traffic is forwarded to port 2.

Step 5: Upon receiving the data packets from OF-Switch 1, the OF-Switch 2 also generates Packet_IN message containing the data packet sent by Host-1 for Host-2.

Step 6: Once again, SDN controller through its application decides the destination port number and installs a flow rule through Flow Mod message on OF-Switch 2. Rule listed in Blue Box under OF-Switch 2 of Fig. 2.

Step 7: After the installation of flow rule, the OF-Switch 2 now forwards the traffic of Host-2 coming from port 2 to port 1.

Step 8: When Host-2 receives data packets from Host-1, it then generates a response packet. This response packet will also follow the same procedure to reach Host-1 using flow rule installed by SDN controller. Flow Rules installed on both switches for reverse path between Host-2 and Host-1 are listed in Fig. 2 inside green boxes.

BIBLIOGRAPHY

[1] A. Nygren, B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. Malek, D. Talayco, D. Erickson *et al.*, “OpenFlow switch specification 1.5.0,” *Open Netw. Found., Tech. Rep. ONF TS-020*, 2014.

[2] S. Ryu, “Framework community: Ryu sdn framework,” 2015. [Online]. Available: http://osrg.github.io/ryu

[3] X. Fei, F. Liu, Q. Zhang, H. Jin, and H. Hu, “Paving the way for nfv acceleration: A taxonomy, survey and future directions,” *ACM Comput. Surv.*, vol. 53, no. 4, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3397022

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] N. Omnes, M. Bouillon, G. Fromentoux, and O. Le Grand, “A programmable and virtualized network & it infrastructure for the internet of things: How can nfv & sdn help for facing the upcoming challenges,” in *2015 18th International Conference on Intelligence in Next Generation Networks.* IEEE, 2015, pp. 64–69.

[6] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *arXiv preprint arXiv:1406.0440*, 2014.

[7] Open Networking Foundation (ONF), “Software-Defined Networking: The New Norm for Networks, Technical Reports,” 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[8] G. Hohpe and B. Woolf, *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Addison-Wesley Professional, 2004.

[9] B. Rauf, H. Abbas, A. M. Sheri, W. Iqbal, and A. W. Khan, "Enterprise integration patterns in sdn: A reliable, fault-tolerant communication framework," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6359–6371, 2021.

[10] B. Rauf, H. Abbas, A. Muqeem, W. Iqbal, Y. Abbas, M. Daneshmand, and F. Amjad, "Cacs: A context-aware and anonymous communication framework for an enterprise network using sdn," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[11] B. Rauf, H. Abbas, M. Usman, T. A. Zia, W. Iqbal, Y. Abbas, and H. Afzal, "Application threats to exploit northbound interface vulnerabilities in software defined networks," *ACM Comput. Surv.*, vol. 54, no. 6, jul 2021. [Online]. Available: https://doi.org/10.1145/3453648

[12] F. Van den Abeele, J. Hoebeke, I. Moerman, and P. Demeester, "Integration of heterogeneous devices and communication models via the cloud in the constrained internet of things," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 683425, 2015.

[13] A. Kamilaris, F. Gao, F. X. Prenafeta-Boldu, and M. I. Ali, "Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT).* IEEE, 2016, pp. 442–447.

[14] S. Wang, J. Wan, D. Li, and C. Zhang, "Implementing smart factory of industrie 4.0: an outlook," *International journal of distributed sensor networks*, vol. 12, no. 1, p. 3159805, 2016.

[15] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[16] L. Catarinucci, D. De Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone, “An IoT-aware architecture for smart healthcare systems,” *IEEE internet of things journal*, vol. 2, no. 6, pp. 515–526, 2015.

[17] O. Flauzac, C. González, A. Hachani, and F. Nolot, “SDN based architecture for IoT and improvement of the security,” in *2015 ieee 29th international conference on advanced information networking and applications workshops*. IEEE, 2015, pp. 688–693.

[18] S. Chakrabarty and D. W. Engels, “A secure IoT architecture for Smart Cities,” in *2016 13th IEEE annual consumer communications & networking conference (CCNC)*. IEEE, 2016, pp. 812–813.

[19] W. Stallings, *Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional, 2015.

[20] K. Benton, L. J. Camp, and C. Small, “OpenFlow vulnerability assessment,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 151–152.

[21] ONOS PROJECT, “Open networking operating system,” Accessed June 2020. [Online]. Available: http://onosproject.org/

[22] OPENDAYLIGHT PROJECT, “OpenDaylight Controller,” Accessed August 2020. [Online]. Available: https://www.opendaylight.org/

[23] Floodlight Project, “Floodlight Controller,” Accessed May 2020. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[24] F. Tomonori, “Introduction to ryu sdn framework,” *Open Networking Summit*, pp. 1–14, 2013.

[25] M. McCauley, “POX,” 2012. [Online]. Available: http://www.noxrepo.org/

[26] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: Towards an operating system for networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[27] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 13–18.

[28] Trema, "Trema project," Accessed Sep 2019. [Online]. Available: http://trema.github.io/trema

[29] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An API for application control of SDNs," in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 327–338.

[30] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks," in *OSDI*, vol. 10, 2010, pp. 1–6.

[31] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks.* ACM, 2012, pp. 19–24.

[32] Z. Cai, A. L. Cox, and T. Ng, "Maestro: A system for scalable openflow control," Tech. Rep., 2010.

[33] Naous, Jad and Erickson, David and Covington, G Adam and Appenzeller, Guido and McKeown, Nick, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems.* ACM, 2008, pp. 1–9.

[34] Open Networking Foundation (ONF) , Accessed August 2020. [Online]. Available: https://www.opennetworking.org/

[35] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN Controllers: A Comparative Study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON).* IEEE, 2016, pp. 1–6.

[36] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-Based Comparison and Selection of Software Defined Networking (SDN) Controllers," in *2014*

World Congress on Computer Applications and Information Systems (WCCAIS). IEEE, 2014, pp. 1–7.

[37] M. Paliwal, D. Shrimankar, and O. Tembhurne, "Controllers in SDN: A review report," *IEEE Access*, vol. 6, pp. 36 256–36 270, 2018.

[38] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," in *2018 Wireless Days (WD).* IEEE, 2018, pp. 54–59.

[39] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[40] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *computer networks*, vol. 121, pp. 100–111, 2017.

[41] S. Schaller and D. Hood, "Software Defined Networking Architecture Standardization," *Computer Standards & Interfaces*, vol. 54, pp. 197–202, 2017.

[42] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 127–132.

[43] B. Pfaff and B. Davie, "The Open vSwitch database management protocol," 2013, Internet Engineering Task Force, RFC 7047 (Informational), Dec. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7047.txt

[44] Smith, Mea and Dvorkin, M and Laribi, Y and Pandey, V and Garg, P and Weidenbacher, N, "OpFlex control protocol," *IETF, Apr*, 2014.

[45] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.

[46] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (netconf)," 2011.

[47] ONF, *Open Networking Foundation (ONF), OpenFlow management and configuration protocol (OF-CONFIG) v1.2*, 2014, tech. Rep. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-confi

[48] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[49] W. Stallings, "Software-defined networks and OpenFlow," *The internet protocol Journal*, vol. 16, no. 1, pp. 2–14, 2013.

[50] Y. Tseng, F. Naït-Abdesselam, and A. Khokhar, "A comprehensive 3-dimensional Security Analysis of a Controller in Software-Defined Networking," *Security and Privacy*, vol. 1, no. 2, p. 21, 2018.

[51] Tijare, PV and Vasudevan, D, "The Northbound APIs of Software Defined Networks," *International journal of engineering sciences and research technology*, 2016.

[52] M. Fakoorrad, "Application Layer of Software Defined Networking: pros and cons in terms of security," Master's thesis, Tallinn University of Technology, 2016.

[53] R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Netowork-based Software Architecture*, pp. 76–85, 2000.

[54] W. Zhou, L. Li, M. Luo, and W. Chou, "Rest api design patterns for sdn northbound api," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014, pp. 358–365.

[55] A. Bierman, M. Bjorklund, K. Watsen, and R. Fernando, "Restconf protocol," in *IETF RFC 8040*, 2017.

[56] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: Toward Software-Defined Mobile Networks," *IEEE Communications magazine*, vol. 51, no. 7, pp. 44–53, 2013.

[57] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking.* ACM, 2009, pp. 1–10.

[58] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, "Managing the network with Merlin," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks.* ACM, 2013, p. 24.

[59] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM Sigplan Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[60] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: Declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 109–114.

[61] T. Nelson, A. D. Ferguson, M. J. Scheer, and S. Krishnamurthi, "Tierless programming and reasoning for software-defined networks," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 519–531.

[62] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 217–230, 2012.

[63] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic foundations for networks," *Acm sigplan notices*, vol. 49, no. 1, pp. 113–126, 2014.

[64] J. R. N. P. Katta and D. Walker, “Logic programming for software-defined networks,” Accessed Sep 2019. [Online]. Available: http://frenetic-lang.org/publications/logic-programming-xldi12-slides.pdf

[65] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular sdn programming with pyretic,” *Technical Reprot of USENIX*, 2013.

[66] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers, “Imperative versus declarative process modeling languages: An empirical investigation,” in *International Conference on Business Process Management.* Springer, 2011, pp. 383–394.

[67] J. W. Lloyd, *Foundations of logic programming.* Springer Science & Business Media, 2012.

[68] N. Intent, “Definition and principles,” *Tech. Rec. The Open Networking Foundation (ONF)*, 2016.

[69] Z. Latif, K. Sharif, F. Li, M. M. Karim, and Y. Wang, “A Comprehensive Survey of Interface Protocols for Software Defined Networks,” *arXiv preprint arXiv:1902.07913*, 2019.

[70] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, “SDNi: A message exchange protocol for software defined networks (sdns) across multiple domains,” *IETF draft, work in progress*, 2012.

[71] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, and J. Bi, “Seamless Interworking of SDN and IP,” in *ACM SIGCOMM computer communication review*, vol. 43, no. 4. ACM, 2013, pp. 475–476.

[72] P. Lin, J. Bi, and H. Hu, “Btsdn: Bgp-based transition for the existing networks to sdn,” *Wireless Personal Communications*, vol. 86, no. 4, pp. 1829–1843, 2016.

[73] S. M. David Hucaby, *Cisco Field Manual: Catalyst Switch Configuration.* Cisco Press, 2002, ch. VLANs and Trunking.

[74] *IEEE Standard for Local and Metropolitan Area Networks (802.1Q)*, IEEE Computer Society Std., 2014.

[75] O. L. Barakat, T. Emadinia, D. Koll, and X. Fu, "Paving the Way towards Enterprise SDN Adoption: New Selection Strategies for Hybrid Networks," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, feb 2019.

[76] D. Levin, M. Canini, S. Schmid, and A. Feldmann, "Incremental SDN deployment in enterprise networks," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 473–474, 2013.

[77] S. Huang, J. Zhao, and X. Wang, "HybridFlow: A lightweight control plane for hybrid SDN in enterprise networks," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, jun 2016.

[78] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental Deployment of SDN in Hybrid Enterprise and ISP Networks," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2890955.2890959

[79] Y. Yamasaki, Y. Miyamoto, J. Yamato, H. Goto, and H. Sone, "Flexible access management system for campus VLAN based on OpenFlow," in *2011 IEEE/IPSJ International Symposium on Applications and the Internet*. IEEE, 2011, pp. 347–351.

[80] V.-G. Nguyen and Y.-H. Kim, "SDN-Based Enterprise and Campus Networks: A Case of VLAN Management," *Journal of Information Processing Systems*, vol. 12, no. 3, 2016.

[81] V. Nair, "Implementation of IEEE 802.1Q VLAN Tagging using RYU OpenFlow Controller." [Online]. Available: https://github.com/VarunDelft/SDN-RYU-VLAN

[82] J. Chen, J. Chen, J. Ling, and W. Zhang, "Failure recovery using vlan-tag in SDN: High speed with low memory requirement," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016, pp. 1–9.

[83] J. Chen, J. Chen, J. Ling, J. Zhou, and W. Zhang, "Link failure recovery in sdn: High efficiency, strong scalability and wide applicability," *Journal of Circuits, Systems and Computers*, vol. 27, no. 06, p. 1850087, 2018.

[84] M. B. Lehocine and M. Batouche, "Flexibility of managing VLAN filtering and segmentation in SDN networks," in *2017 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, 2017, pp. 1–6.

[85] P.-W. Tsai, P.-w. Cheng, C.-S. Yang, M.-Y. Luo, and J. Chen, "Supporting extensions of VLAN-tagged traffic across OpenFlow networks," in *2013 Second GENI Research and Educational Experiment Workshop*. IEEE, 2013, pp. 61–65.

[86] R. di Lallo, G. Lospoto, M. Rimondini, and G. Di Battista, "How to handle ARP in a software-defined network," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 2016, pp. 63–67.

[87] T. Alharbi and M. Portmann, "SProxy ARP-efficient ARP handling in SDN," in *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2016, pp. 179–184.

[88] T. Zhu, D. Feng, Y. Hua, F. Wang, Q. Shi, and J. Liu, "Mic: An efficient anonymous communication system in data center networks," in *2016 45th International Conference on Parallel Processing (ICPP)*. IEEE, 2016, pp. 11–20.

[89] T. Zhu, D. Feng, F. Wang, Y. Hua, Q. Shi, J. Liu, Y. Cheng, and Y. Wan, "Efficient anonymous communication in SDN-based data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3767–3780, 2017.

[90] Y. Wang, J. Yi, J. Guo, Y. Qiao, M. Qi, and Q. Chen, "A Semistructured Random Identifier Protocol for Anonymous Communication in SDN Network," *Security and Communication Networks*, vol. 2018, 2018.

[91] T. Wong, H. Cui, Y. Shen, W. Lin, and T. Yu, "Anonymous network communication based on SDN," in *2018 4th International Conference on Universal Village (UV)*. IEEE, 2018, pp. 1–5.

[92] T. Zeng, M. Shen, M. Wang, L. Zhu, and F. Li, "Self-adaptive anonymous communication scheme under sdn architecture," in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2015, pp. 1–8.

[93] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Tech. Rep., 2004.

[94] B. Zantout, R. Haraty *et al.*, "I2P data communication system," in *Proceedings of ICN*. Citeseer, 2011, pp. 401–409.

[95] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM transactions on information and system security (TISSEC)*, vol. 1, no. 1, pp. 66–92, 1998.

[96] R. Kang, L. Dabbish, and K. Sutton, "Strangers on your phone: Why people use anonymous communication applications," in *Proceedings of the 19th ACM conference on computer-supported cooperative work & social computing*, 2016, pp. 359–370.

[97] F. Paganelli, M. Ulema, and B. Martini, "Context-aware service composition and delivery in NGSONs over SDN," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 97–105, 2014.

[98] B. Martini, F. Paganelli, A. Mohammed, M. Gharbaoui, A. Sgambelluri, and P. Castoldi, "SDN controller for context-aware data delivery in dynamic service chaining," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[99] S. Luo, J. Wu, J. Li, L. Guo, and B. Pei, "Context-aware traffic forwarding service for applications in SDN," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. IEEE, 2015, pp. 557–561.

[100] *IEEE 1903.2-2017 - IEEE Standard for Service Composition Protocols of Next Generation Service Overlay Network*, IEEE Std., May 2018. [Online]. Available: https://standards.ieee.org/standard/1903_2-2017.html

[101] *OpenFlow Switch Specification (Version 1.3)*, Open Networking Foundation Std. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications

[102] D. C. Plummer *et al.*, "Ethernet address resolution protocol: Or converting network protocol addresses to 48. bit ethernet address for transmission on ethernet hardware." *RFC*, vol. 826, pp. 1–10, 1982.

[103] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[104] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[105] "PyService-Registry," https://github.com/cr0hn/pyservice-registry, accessed: Oct 2020. [Online]. Available: https://github.com/cr0hn/pyservice-registry

[106] *RabbitMQ*, accessed March 2020. [Online]. Available: https://www.rabbitmq.com/documentation.html

[107] B. Snyder, D. Bosnanac, and R. Davies, *ActiveMQ in action.* Manning Greenwich Conn., 2011, vol. 47.

[108] J. Kramer, "Advanced message queuing protocol (AMQP)," *Linux Journal*, vol. 2009, no. 187, p. 3, 2009.

[109] Bhuyan, Monowar H and Bhattacharyya, Dhruba Kr and Kalita, Jugal K, "Surveying port scans and their detection methodologies," *The Computer Journal*, vol. 54, no. 10, pp. 1565–1581, 2011.

[110] T. Kiravuo, M. Sarela, and J. Manner, “A survey of Ethernet LAN security,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1477–1491, 2013.

[111] M. Z. Masoud, Y. Jaradat, and I. Jannoud, “On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm,” in *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT).* IEEE, 2015, pp. 1–5.

[112] M. Sipser, “Introduction to the Theory of Computation,” *ACM Sigact News*, vol. 27, no. 1, pp. 27–29, 1996.

[113] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit.* Elsevier, 2006.

[114] D. Kennedy, J. O’gorman, D. Kearns, and M. Aharoni, *Metasploit: the penetration tester’s guide.* No Starch Press, 2011.

[115] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning.* Insecure, 2009.