



**MINIMIZING BROADCAST TRAFFIC IN SDN-BASED
SWITCHING NETWORKS**

BY
ABDULQADER ALI BAHAJ

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER NETWORKS

NOVEMBER 2018


KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA


DEANSHIP OF GRADUATE STUDIES

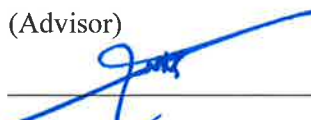
This thesis, written by **Abdulqader Ali Bahaj** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **Master of Science in Computer Networks**.



Dr. Ahmad Al-Mulhem
Department Chairman


Dr. Salam A. Zummo
Dean of Graduate Studies




Prof. Tarek R. Sheltami
(Advisor)


Dr. Talal M. Al-Kharoubi
(Member)


Dr. Anas A. Al-Roubaiey
(Member)

25/12/17
Date

© Abdulqader Ali Bahaj

2018

To my Family

ACKNOWLEDGMENTS

First and foremost, I am grateful to Allah for the good health and wellbeing that were necessary to complete this thesis. I would like to express my sincere thanks to my advisor Professor Tarek Rahil Sheltami for his continuous support and guidance during my master study. Also, I would like to thank him for his insightful comments, patience, and motivation. Besides my advisor, I would like to thank my thesis committee members: Dr. Talal Al-kharoubi and Dr. Anas Al-Roubaiey for their comments, suggestions, and encouragement, also for the thoughts which helped me to widen my research from various perspectives. I would like to thank KFUPM and Hadramout Establishment of Human Development for giving me the opportunity to pursue my M.S. degree. My sincere thanks are to my precious family for the support, and prayers during my academic journey.

TABLE OF CONTENTS

ACKNOWLEDGMENTS.....	IV
TABLE OF CONTENTS	V
LIST OF TABLES.....	IX
LIST OF FIGURES.....	X
LIST OF ABBREVIATIONS.....	XI
ABSTRACT	XIII
ملخص الرسالة.....	XIV
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement	2
1.2 Objectives	4
1.3 Methodology	5
1.4 Thesis Structure	8
CHAPTER 2 SOFTWARE DEFINED NETWORK (SDN).....	10
2.1 Introduction	10
2.2 History of SDN	10
2.2.1 Active Networks	11
2.2.2 Control-data Planes Separation	12
2.2.3 OpenFlow and Network OS	12
2.3 Traditional Networks	13
2.3.1 The OSI model	13

2.3.2	Networking devices	15
2.4	Limitations of Traditional Networking Technologies	17
2.5	SDN Architecture	19
2.5.1	Infrastructure Layer	21
2.5.2	Control Layer	21
2.5.3	Application Layer	22
2.6	Communication Interfaces	22
2.6.1	Southbound Interface	23
2.6.2	Northbound Interface	23
2.6.3	East-west Bridge	23
2.7	SDN Applications	24
2.7.1	Enterprise Networks	24
2.7.2	Data Centers and Cloud Computing	25
2.7.3	Wireless Networks	26
2.7.4	Security	27
2.7.5	Multimedia and QoS	27
2.7.6	Home Networks	28
2.7.7	Information-Centric Networking	28
2.8	Development Tools	29
2.8.1	Simulators and Frameworks	29
2.8.2	Controller	31
	CHAPTER 3 OPENFLOW PROTOCOL.....	35
3.1	Introduction.....	35
3.2	Architecture of OpenFlow	37
3.3	OpenFlow Channel Connection	38
3.3.1	Connection setup on TCP & TLS	38
3.3.2	Connection setup with multiple controllers.....	39
3.4	OpenFlow Message Types	40
3.4.1	Symmetric Messages	40
3.4.2	Asynchronous Messages	40
3.4.3	Controller-to-Switch messages	41
3.5	OpenFlow Tables	43
3.6	Pipeline Processing	45

3.7	Packet Format	47
3.8	OpenFlow Versions	48
3.9	OpenFlow Switch	49
	CHAPTER 4 RELATED WORK	50
	CHAPTER 5 TOOL USED FOR THIS PROJECT	53
5.1	Wireshark	53
5.2	Tcpdump	53
5.3	Oracle VM VirtualBox	53
5.4	Hping3	54
5.5	Arping	54
5.6	Scapy	54
5.7	Mininet	54
5.8	POX Controller	55
	CHAPTER 6 APPROACH DESIGN AND EVALUATION	58
6.1	L2 Learning Switch Method in SDN	58
6.2	Proposed Approach	58
6.2.1	ARP Handler Component	59
6.2.2	IP Handler Component	65
6.2.3	DHCP Handler Component	69
6.3	Complexity Analysis	73
6.4	Comparison between Different Network Approaches	74
6.5	Virtual Environment Setup	75
6.6	Network Topology	76
6.7	Results and Evaluation	77
6.7.1	Reduce ARP Traffic	77
6.7.2	Reduce Control Traffic	78

6.7.3	ARP Response Time	79
6.7.4	CPU Load	80
6.7.5	Ping Response Time	81
6.7.6	Blocking Timeout Periods.....	82
CHAPTER 7 CONCLUSION AND FUTURE WORK		86
REFERENCES		87
VITAE		93

LIST OF TABLES

Table 1 Cisco Recommendation for Broadcast Domain Size.....	2
Table 2 Summary of Common OpenFlow Controllers.....	34
Table 3 OpenFlow Table Example	43
Table 4 OpenFlow Entries Columns.....	43
Table 5 Matched Header Fields in OpenFlow	47
Table 6 Comparison between Different Approaches on Handling Broadcast Traffic	74
Table 7 Specifications for Virtual Environment.....	75

LIST OF FIGURES

Figure 1 Abstracted ARP Component	6
Figure 2 Abstracted IP Component.....	7
Figure 3 Abstracted DHCP Component	8
Figure 4 Data and Control Planes in Networking Hardware	10
Figure 5 OSI 7 Layers	15
Figure 6 Difference between SDN and Traditional Networks.....	20
Figure 7 SDN Architecture	21
Figure 8 SDN Interfaces	24
Figure 9 OpenFlow Protocol.....	37
Figure 10 OpenFlow Messages.....	42
Figure 11 Flow Table Entry Components.....	45
Figure 12 Pipeline Processing.....	46
Figure 13 OpenFlow Versions Timeline.....	48
Figure 14 ARP Handler Component Flowchart.....	64
Figure 15 IP Handler Component Flowchart.....	68
Figure 16 DHCP Handler Component Flowchart.....	72
Figure 17 Clos Topology with 2 Core Switches and Fanout = 2.....	76
Figure 18 ARP Traffic	78
Figure 19 Control Traffic.....	79
Figure 20 ARP Response Time.....	80
Figure 21 CPU Load	81
Figure 22 Ping Response Time	82
Figure 23 Average CPU Load.....	83
Figure 24 Accumulative Control Packets (24 hours).....	85

LIST OF ABBREVIATIONS

AP	Access Point
ACL	Access Control List
AN	Active Network
API	Application Programming Interface
ARP	Address Resolution Protocol
A-CPI	Application Control Plane Interface
ATM	Asynchronous Transfer Mode
BPDU	Bridge Protocol Data Unit
BGP	Border Gateway Protocol
CPU	Central Processing Unit
DHCP	Dynamic Host Control Protocol
DNS	Domain Name System
DoS	Denial of Service
DPID	DataPath Identifier
EIGRP	Enhanced Interior Gateway Routing Protocol
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
HDLC	High-level Data Link Control
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ICN	Information-Centric Networking
IDC	International Data Corporation
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
IS-IS	Intermediate System to Intermediate System
IPX	Internetwork Packet Exchange
LAN	Local Area Network
LLC	Logical Link Control
MAC	Medium Access Control
MLD	Multicast Listener Discovery
NAT	Network Address Translation
NetBIOS	Network Basic Input/Output System
NFS	Network File System
OF	OpenFlow Protocol
ONF	Open Networking Foundation

OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
QoS	Quality of Service
PN	Programmable Network
PPP	Point-to-Point Protocol
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time To Live
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
WAN	Wide Area Network

ABSTRACT

Full Name : [Abdulqader Ali Obaid Bahaj]

Thesis Title : [Minimizing Broadcast Traffic in SDN-based Switching Networks]

Major Field : [Computer Networks]

Date of Degree : [Nov 2018]

Ethernet gets its prevalent position in local area networks (LAN) due to the features that it provides such as high speed, low-cost, and self-configuration. However, Ethernet suffers from scalability issue since it is difficult to be extended to large networks. The main reason for the scalability drawback is the broadcast and multicast services that are used for discovery within the network. For instance, hosts use Dynamic Host Configuration Protocol (DHCP), which depends on the broadcast service, in order to dynamically obtain network configuration information (e.g. IP address, DNS server, and default gateway). This results in congestion and resource consumption as the network grows larger.

While Software-Defined Network (SDN) has solved many legacy network issues, scalability remains one of the main problems since broadcast and multicast have been inherited. In this thesis, an SDN-based approach is proposed to eliminate Ethernet's flooding and overcome the scalability issue. The proposed approach uses the SDN controller to handle DHCP and ARP broadcast messages without the need to be broadcasted across the entire network. The experiments showed that the proposed approach enhanced the scalability by eliminating broadcast traffic from the data plane. Results also reported a decrease in controller overhead as well as control traffic in the control plane, and better response time was achieved.

ملخص الرسالة

الاسم الكامل: عبدالقادر علي عبيد باحاج

عنوان الرسالة: تقليل إزدحام رسائل البث في الشبكات المعرفة بالبرمجيات (SDN)

التخصص: شبكات الحاسب الآلي

تاريخ الدرجة العلمية: نوفمبر ، ٢٠١٨

حازت تقنية الإيثرنت (Ethernet) على موقعها الهام في عالم الشبكات المحلية نظراً لخصائصها التي تميزها عن باقي التقنيات مثل تكلفتها المنخفضة، و دعمها لخاصية التهيئة الذاتية و غير ذلك. و لكن من جهة اخرى، ليس لدى تقنية الإيثرنت القدرة على العمل بكفاءة عالية في شبكات مراكز البيانات الكبيرة (Data Centers) حيث و انها صممت بشكل أساسي لدعم الشبكات الصغيرة و المحدودة. و يرجع السبب إلى خاصية البث المستخدم في بعض البروتوكولات لأجل استكشاف الاجهزة في الشبكة مثل بروتوكول التهيئة الآلية للمضيفين (DHCP).

و لهذا كان لابد من إيجاد تقنيات جديدة تتعامل مع هذه المشكلة، و من أبرز هذه التقنيات تقنية الشبكات المعرفة بالبرمجيات. في هذه الرسالة البحثية، و من أجل التقليل من المشكلة المرتبطة بخاصية البث، اقترحنا منهجية معتمدة على الشبكات المعرفة بالبرمجيات حيث صممت لتتعامل مع اكثر بروتوكولين مسببين لهذه المشكلة و هما بروتوكول إيجاد العناوين (ARP) و بروتوكول التهيئة الآلية للمضيفين (DHCP). فكرة عمل المنهجية هي أن الرسائل البثية توجه لجهاز مركزي يسمى المتحكم. وظيفة هذا المتحكم انه يقوم بالتعامل مع هذه الرسائل و الإجابة عليها بطريقة سريعة و مختصرة لا تستدعي البث على كامل نطاق الشبكة معتمداً في ذلك على قاعدة بيانات مركزية تحوي معلومات عن كل الاجهزة في الشبكة.

CHAPTER 1

INTRODUCTION

Ethernet is prevailing network technology in LAN networks, which has been vastly used in enterprise, campus, and data center networks due to its low-cost, simplicity, and self-configuration capability. Its ease of use and low-cost rely on broadcast service or resource discovery protocols, for example, Address Resolution Protocol, Dynamic Host Configuration Protocol, Network Basic Input/Output System, Network Time Protocol and etc. Although broadcast in Ethernet is cost-efficient and simple to use, it has the following problems: (1) The communication in the network is affected by broadcast packets which consume a large amount of network bandwidth. Cisco report [6] gives a recommendation for the maximum number of hosts that can participate in a broadcast domain rely on the type of protocol used in the Ethernet network as appeared in Table 1. (2) To get a loop-free in Ethernet network, Spanning Tree Protocol (STP) is applied. As a result, STP brings some drawbacks such as, first, the bottleneck that happens near the root switch which may lead to recalculate the Spanning Tree. Second, wasting bandwidth of redundant links caused by disabling several paths reduces the maximum network transmission bit rate. Third, large time is needed for recalculating spanning tree when a link or switch goes down. (3) Broadcast makes it easy for any host in the network to acquire information of the other hosts. Moreover, this gives a chance for the attackers to hack the network. For example, ARP request or ARP reply can be easily forged and distributed to the networks by the hackers which may lead to information security problems, confusion in communication, or paralysis of the network. (4) Processing every broadcast packet by receivers'

hosts will overload their CPUs and Memories. As a result of these problems, Ethernet becomes non-scalable and less efficient. This causes to limit the development of Ethernet, especially in large data centers.

Table 1 Cisco Recommendation for Broadcast Domain Size

Protocols Types					
	IP	Netware	AppleTalk	NetBIOS	Mixed
Max#hosts	500	300	200	200	200

To deal with these problems, a large network is partitioned into small broadcast domains (VLANs or LANs). These broadcast domains are physically connected together via layer-3 switches/routers. However, as forwarding in layer-3 switches/routers depends on IP address, this destroys the benefit of forwarding inside the broadcast domain. Moreover, configuring and managing all the broadcast domains in the whole network require huge manual effort e.g. effort for subnets creation and configuration, VLANs creation, Access Control Lists (ACL) and etc. In addition, in the data center with the virtual environment, it is hard to migrate Virtual Machines (VMs) from one physical host server to another without interrupting their running state as they need to change their IP from the previous subnet to the new one. As a result, we argue that it is appropriate to build large LANs that are exclusively based on layer 2 technology (e.g. Ethernet) rather than using layer 3 (e.g. IP routing).

1.1 Problem Statement

The most well-known technology in local area networks is Ethernet. Within small geographic areas, Ethernet can be used in homes, campuses, or in enterprise networks. Ethernet supports resources sharing with high performance, which allows client-server scheme as well as the

principles of virtualization to distribute the load between servers and facilitate network administration. In the Internet protocol suite, the protocol of Ethernet falls within data link layer, and it is used to provide services for data link layer protocols such as Address Resolution Protocol (ARP), as well as upper layer protocols like Dynamic Host Configuration Protocol (DHCP) which resides within the application layer. Furthermore, Ethernet provides multicast protocols services, such as Bridge Protocol Data Units (BPDUs), which is a multicast packet used by the STP in the link layer and the Multicast Listener Discovery (MLD) protocol in the internet layer. Multicast and broadcast protocols have advantage of supporting several services, such as obtaining new IPs and destination MAC addresses, discovering neighboring nodes, and loop-free networking. Despite this advantage, the mechanism of the broadcast is unavoidable which cause negative consequences which include the following:

- As a result of broadcast packets, broadcast storms can occur in network topologies with several connection levels, e.g. tree topology, causing further negative impacts: 1) congesting links, 2) overload of the CPU of the switches, 3) Generation of MAC address flaps. In legacy networks, STP protocol is necessary to overcome the loop storm. Nonetheless, it generates multicast traffic that exhausts the bandwidth, and only seven hops are supported as a maximum bridged LAN diameter, consequently restricting the network scalability.
- Broadcast increase network traffic which results in competition and collision within the same link. Consequently, this leads to congestion, negative impact on response time, and loss of packets, Therefore, as shown in Table 1 Cisco suggests, in an empirical study [1], to use at most 500 devices in one collision domain. However, this is supported only within Ethernet-based networks and, thus, does not meet the recent technology needs, e.g. Internet of Things (IoT).

- The mechanism of broadcast increases the consumption of CPU resource of the hosts via receiving numerous amounts of irrelevant broadcast packets.
- The broadcast mechanism suffers from security leaks, such as when the ARP protocol is used for various types of attacks, such as poisoning, flooding, broadcast attacks, and spoofing. These security issues can stop entire network completely or cause the resources to be unavailable, e.g. Denial of Service (DoS) attack. Furthermore, a sniffing attack can happen when all hosts receive broadcast packets even though no request is made. Consequently, unauthorized hosts may intercept data.

1.2 Objectives

The main goal of this research is to propose an approach that addresses the problem of ARP and DHCP broadcast traffic in SDN network and supports a mechanism for applying the multi-path.

For the approach to be beneficial, it must achieve the following objectives:

- **Save Network Bandwidth**

The approach should save the bandwidth resource of the network by minimizing broadcast, avoiding broadcast storm (loop-free), and provide effective use of available bandwidth.

- **Self-Configuration**

The network should configure itself with a manner of "Plug and Play" and therefore it does not need to be configured manually.

- **Support Multi-path Transmission**

The approach should utilize effective multi-path method to minimize the probability of congestion and end to end delay.

- **Support all IP Configuration Types**

The approach should support both static and dynamic host IP configuration.

- **Security**

The network must be secure against suspicious ARP requests storm.

- **Scalability**

The network should be scaled dynamically without violating previous objectives or decreasing network performance when the network grows larger.

1.3 Methodology

In this research, we propose an SDN-based approach to address the problem of the broadcast traffic and to provide the mechanism of multi-pathing in Local Area Networks. The approach is based on the SDN architecture and it is divided into three main components, one component to handle the ARP packets, another one to deal with DHCP packets, and the last one to handle the IP packet and provide a multi-pathing and shortest path mechanisms in the approach. These different components are coded in the SDN controller. Once any packet arrives at the first-hop switch, it sends the packet to the controller. According to the type packet's type, the controller passes it to the appropriate component.

Figure 1 gives an abstract view of how the ARP component works. Here, once the ARP component receives an ARP packet, it checks whether it is a request or a response. If it is an ARP request, the component will update the requested information in the main database. Then, another checking process takes place to identify whether the requested MAC address is available in the main database. In such a case where the required MAC address exists, the ARP component will construct an ARP reply with the required MAC address and sends it to the appropriate

requester. On the other hand, if the requested MAC address does not exist, the controller will make Edge-Broadcast asking all the edge switches (first-hop switches) to send an ARP request to all directly connected hosts.

If the ARP packet is an ARP replay, the controller will first update the responder's information in the main database and forward the ARP replay to all cached requesters/. However, if the packet is neither request nor response, the controller automatically drops the packet.

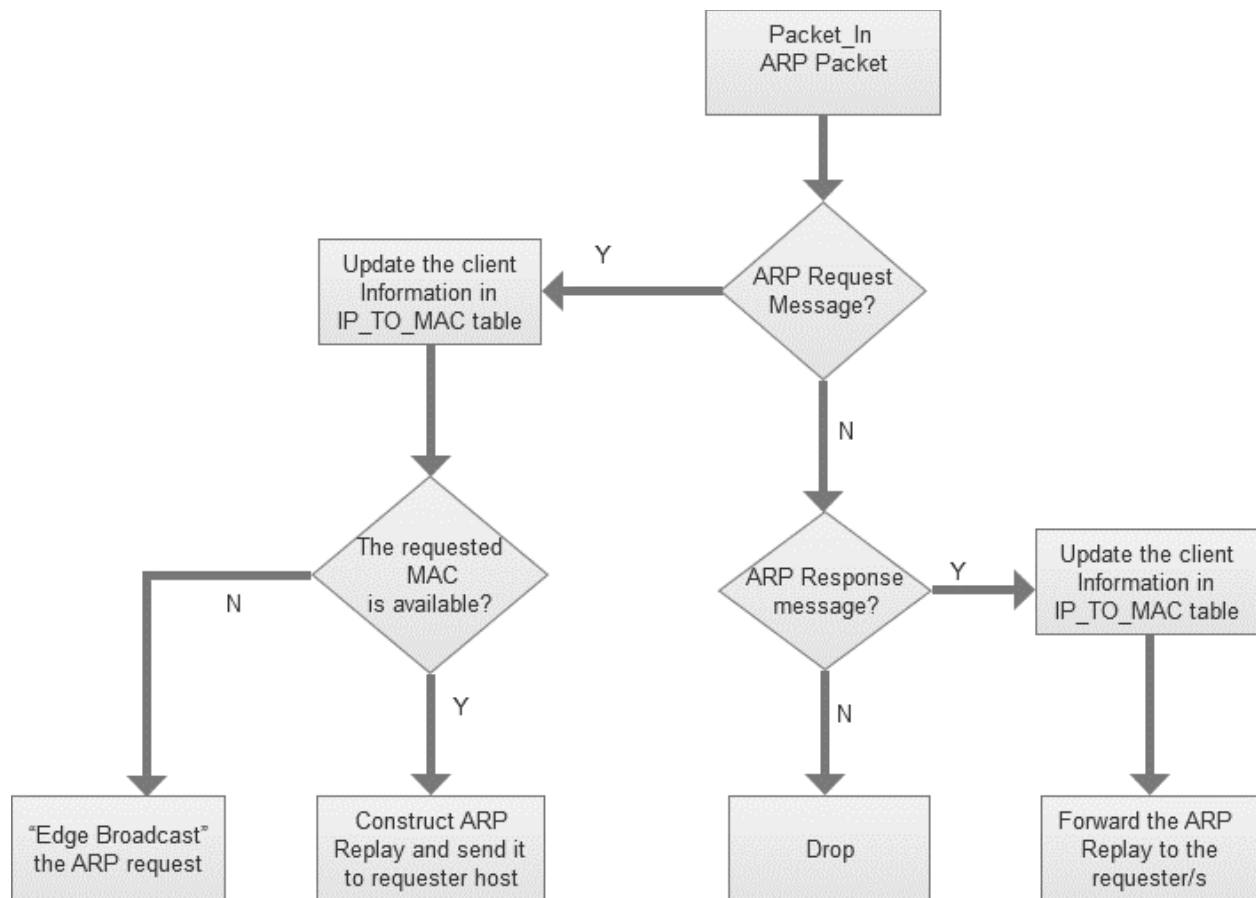


Figure 1 Abstracted ARP Component

Figure 2 illustrates how IP component works in the proposed approach. Once an IP packet pass to the IP component, the controller checks whether the destination IP is available in the main database. However, if the destination IP does not exist, it will ask access switches to flood that packet to all of their direct access hosts. On the other hand, if the destination IP address is actually available in the main database, the controller executes the following steps. First, it extracts the information of the destination location from the main database. Second, the controller calculates all shortest paths between the source and destination and chooses the best path that has less accumulative weight. Then, it will install flow entries along the chosen path to specify the track between the source and destination hosts for the ongoing communication. Finally, the links of the chosen path must be updated with the weight of the traffic type (Http, Ftp, or Other).

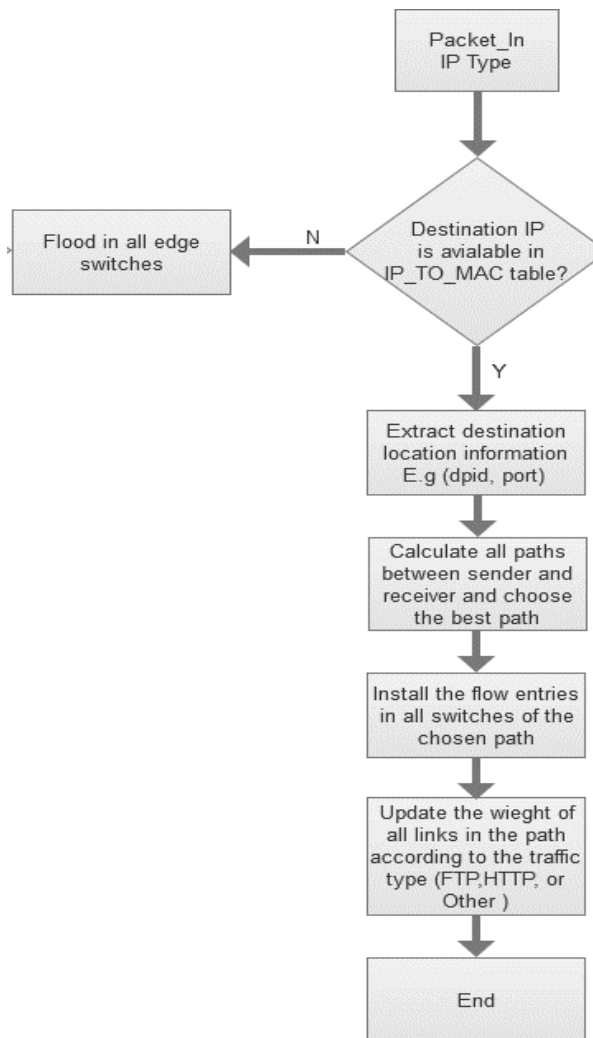


Figure 2 Abstracted IP Component

DHCP component is used to handle DHCP packets. Figure 3 clarifies how the component works. When the component receives a DHCP packet, it checks whether the packet comes from a DHCP server or from a client. If it comes from a DHCP server, the controller will forward it to the client. However, if the packet comes from the client, then the controller will forward it to the server.

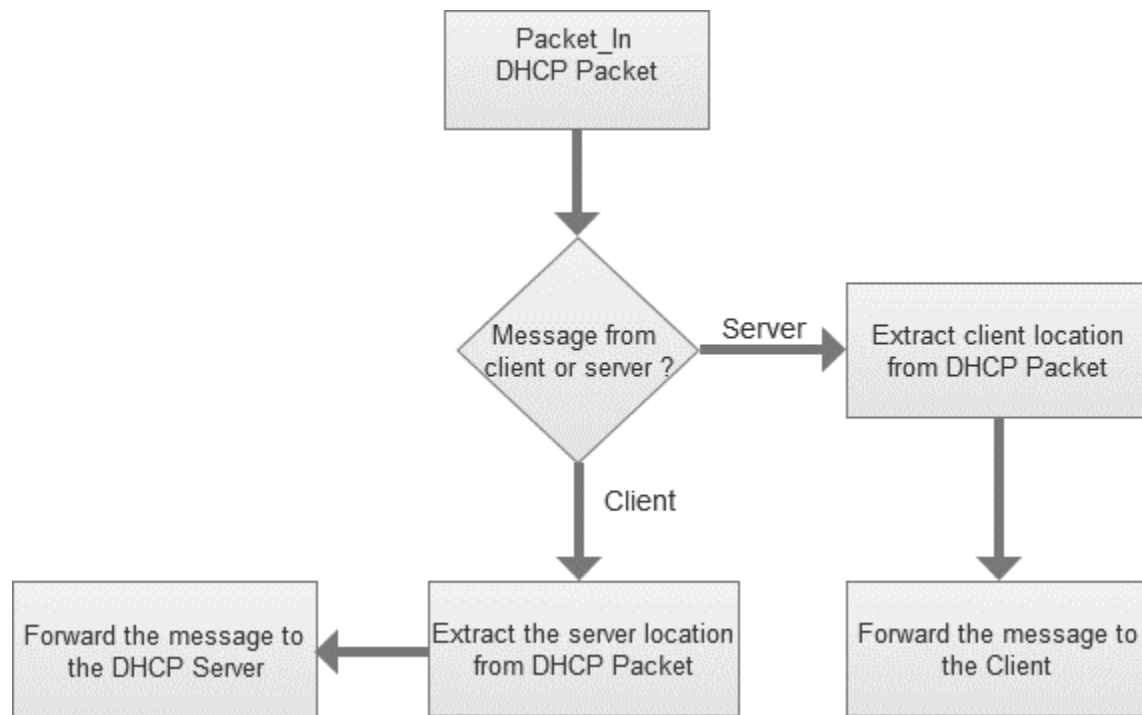


Figure 3 Abstracted DHCP Component

1.4 Thesis Structure

- Chapter 1: This is the introductory chapter that describes the problem statement, the proposed approach objectives, and the work methodology.
- Chapter 2: This serves as the background chapter, it contains information about SDN, traditional networks devices, related problems, and terms related to the understanding of these subjects.

- Chapter 3: This is the OpenFlow protocol chapter, it presents an explanation about the OpenFlow and its architecture.
- Chapter 4: This chapter presents a study of some existing approaches that are related to our approach.
- Chapter 5: This chapter explains the tools used for developing and coding this project.
- Chapter 6: This chapter presents the components of the proposed approach, analysis of the proposed approach, and discussion of the results and evaluation.
- Chapter 7: Concludes the thesis, and highlights ways to continue the research in the same field, which serves as future work.
- Appendix: Finally, the appendix chapter includes significant documents related to the thesis.

CHAPTER 2

Software Defined Network (SDN)

2.1 Introduction

According to the ONF (Open Network Foundation) (a non-profit consortium founded in 2011 to promote SDN and standardize its protocols) SDN is an architecture that Separates the control plane from the data plane, and unifies control plans in an external control software called "Controller" to manage multiple elements of the data plane via Application Programming Interface (API). It should be noted that the ONF includes more than 100 companies including the web giants, telecom operators, and manufacturers, which shows the immense interest of the manufacturers for SDN [2]. In this section, we present the architecture of the SDN and describe its main components.

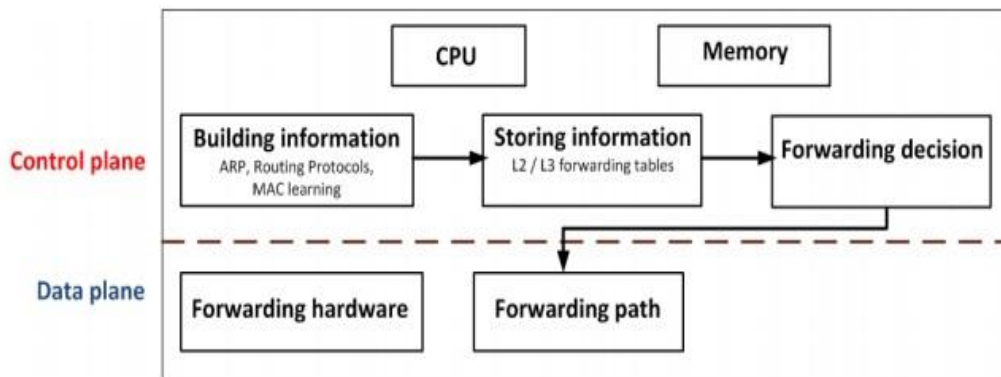


Figure 4 Data and Control Planes in Networking Hardware

2.2 History of SDN

The term of Software-Defined Networking (SDN) is not totally a new technology, but it is actually part of a long history of contributions, ideas, and developments to get networks that are more easily programmable. Many pieces of research and implementations that made what today

knew as SDN were applied since about 20 years. For example, to achieve simplicity of network management, SDN uses some ideas from early telephony networks, which separate control and data planes. According to [3], there are three historical phases in the development of SDN: 1) active networks (the mid-1990s to early 2000s), 2) control-data planes separation (around 2001 to 2007), 3) OpenFlow and network OS (2007 to around 2010). In this section, we give a brief overview of these phases, which can be considered as ancestors of SDN.

2.2.1 Active Networks

The network devices become required to be programmable due to two reasons. The first one is the difficulty that was facing the researchers in testing new ideas and protocols in a real infrastructure. Second, the large time and effort needed for standardizing new ideas and protocols in the Internet Engineering Task Force (IETF). Those issues make it necessary to develop new programmable network devices. Active Network (AN), created in the mid-1990s, is one of the first technologies that enabled network programming through applications. It offers network APIs for exposing the individual resources of the network node e.g. processing, storage, and packet queues. Also, it supports creating customized features to be applied on a certain set of packets that pass through the network node. Mainly, using different programming models in network nodes was the actual beginning of network virtualization research. Despite active networks were the first technologies use the programming in networks, they did not deploy widely. The reason besides that may relate to its limitations in performance and safety. Several studies have been carried out on ANs such as [3], [4].

Since packets of ANs can carry malicious programs, an alternative to ANs called Programmable Networks (PN) was proposed in 1999. The PNs inject programs inside the nodes of the network. These Nodes run programs only after a signaling and verification phase, to enhance security. The

ANs and PNs have sought to introduce programmability in networks through programmable packets and switches. These approaches have not reduced the complexity of the network infrastructure [2].

2.2.2 Control-data Planes Separation

The growth of traffic volume over the network beside the need for making the network more reliable led to improve new approaches that provide better management functions such as control the network traffic links (traffic engineering), reaction, traffic prediction, repair, and recovery network topology in case there is a problem, and so on [3]. These approaches were limited by the tight connection between the software (control plane) and hardware (data plane) of the network devices. Because of this coupling makes it difficult to configure and manage network device in flexible way. To deal with these challenges, researchers say that it is better to make separating between the data and control planes. This will simplify the network architecture and provide an abstraction on the physical infrastructure. Furthermore, because of the forwarding of the packet depends basically on a hardware, and control logic of software, the software will work better on a separate server that has higher resources (computing, memory) than on a single network node [4].

2.2.3 OpenFlow and Network OS

OpenFlow was designed to test the experimental protocols in universities and research institute. OpenFlow is driven by the SDN principle of decoupling the control and data forwarding planes. With OpenFlow, it is become possible to experiment with new ideas, algorithms, and protocols without interrupt or interfere with the production department or any other traffic. Open Networking Foundation (ONF) is an organization that is in charge of the OpenFlow and SDN protocols. We will talk about OpenFlow and its architecture deeply in the coming chapter.

2.3 Traditional Networks

The current and traditional network device uses a particular set of software and hardware. For example, Cisco vendor provides packages of both software and hardware that are integrated with a single package. These devices are physically static and needed to be configured one by one. This requires large efforts and makes configuration complex, therefore it becomes difficult to accomplish a good network.

2.3.1 The OSI model

The Open System Interconnection (OSI) is a conceptual and logical model which defines the networking framework to implement different network protocols in seven layers as it can be seen in Figure 5. Each layer consists of a set of conceptual services provided to the layers above and down it. Layers 1-4 are considered the lower layers, and mostly concern themselves with moving data around. Layers 5-7, the upper layers, contain application-level data. Each layer takes care of a very specific job, and then passes the data onto the next layer.

- Physical Layer (Layer 1)

This layer is responsible for conveying a stream of bits in different shapes (electrical impulse, radio or light signal) via the network at the mechanical and electrical level. It includes all the hardware means using for sending and receiving data on the various carrier such as cables, network cards, and physical equipment. Additionally, there are several protocols that belong to the physical layer such as Fast Ethernet, ATM, and RS232.

- Data Link Layer (Layer 2)

Data Link Layer is in charge of encoding and decoding data packets into bits. Also, it performs a lot of tasks such as physical layer errors handling, frame synchronization, flow control, and etc.

It mainly consists of two sublayers: Logical Link Control (LLC) layer and Media Access Control (MAC) layer. The LLC sublayer provides the tasks of frame synchronization, error checking, and flow control while The MAC sublayer controls all the permissions of how computer access and transmit data. Layer 2 includes several protocol and technologies such as PPP, ATM, FDDI, IEEE 802.3/802.2, IEEE 802.5/ 802.2, Frame Relay, HDLC.

- Network Layer (Layer 3)

Network Layer performs forwarding and routing technologies and transmits data from one network node to another. In addition, it is responsible for addressing, error handling, internetworking, packet sequencing, and congestion control. Examples for network layer protocols are IP, AppleTalk DDP, and IPX.

- Transport Layer (Layer 4)

Transport Layer transfers the data between hosts or end systems in a transparent manner. Moreover, it provides end-to-end communication, flow control, and error recovery. TCP and UDP are the most popular Transport Layer protocols.

- Session Layer (Layer 5)

Session Layer is responsible for setting up, managing, and terminating the connection between applications. Session layer examples include NFS, RPC, SQL, NetBIOS names.

- Presentation Layer (Layer 6)

Presentation Layer in charge of formatting and encrypting data in a way that data can be sent through a network. It provides a freedom from the issues of compatibility. The most used formats by Presentation Layer are ASCII, EBCDIC, TIFF, GIF, PICT, JPEG, MPEG, and MIDI.

- Application Layer (Layer 7)

Presentation Layer provides support for end-user processes and applications. All data constraints and syntax is identified in this layer. Application Layer examples include WWW browsers, HTTP, NFS, FTP, Telnet, and SNMP.

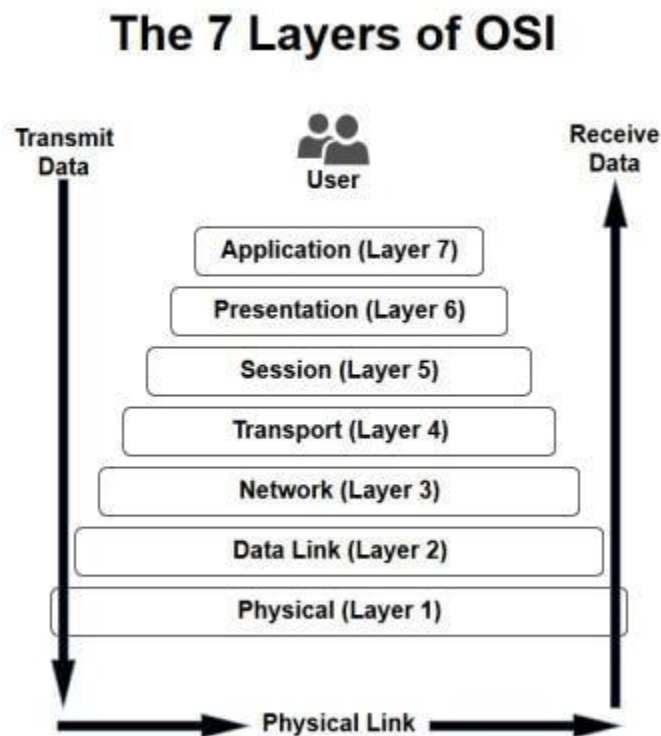


Figure 5 OSI 7 Layers

2.3.2 Networking devices

In the networking world, there are abundant kinds of networking devices that are used for communication. Some of those devices are simple and comes with a little logic while others are

complex with high capabilities. Further, there are other devices that are mixed. In this section, we will discuss the three most famous networking devices that are related to the routing.

2.3.2.1 Hub

A network hub is a connection point for the network devices which connects the LAN segments. It does not manage network traffic but for every received packet it forwards the packet to every port except the one the packet comes from [5]. According to the OSI model classification, the hub is a Layer 1 device. Because of the hub is a repeating device, large collisions would happen in the network and this affects the network capacity [6].

2.3.2.2 Switch

A network switch is a smart version of a network hub which can filter the packets between LAN segments. The switch is a layer 2 device (data link layer) and switches can work on layer 3 (network layer). There are two kinds of switches, managed and the unmanaged switch. The managed switch is a plug and play device while the unmanaged one is a configurable device. For forwarding packets in the correct port, the switch keeps mapping IP-to-MAC forwarding table of network hosts.

2.3.2.3 Router

The router has a different job of switch's job. Commonly, the router connects at least two Local Area Networks (LANs) or Wide Area Networks (WANs) and its ISP's networks. Routers typically are placed in the network as gateways, where it connects two different networks. To determine the best path for packets forwarding, routers use the IP packet header information and forwarding tables. Moreover, they use routing protocols such as (RIP, OSPF, IS-IS, etc.) for communicating and configuring best routes between hosts [7].

2.4 Limitations of Traditional Networking Technologies

In the traditional network, once the router receives a packet, it uses a specific algorithm to forward that packet to the suitable next hop. In this operation, the router depends on the embedded operating systems and the information in the routing table that is collected using the routing protocols such as RIP, EIGRP, OSPF, and GBP. As a result of processing a lot of incoming packets, the router could be overloaded. To forward the packets in a successful way, the routers have to perform two main tasks. First, the router must update their routing tables regularly to get the current status of the network which is considered as the main operation in the control plane. Second, the router forwards the incoming packet toward the destination according to its local routing tables. This operation is called forwarding or data plane operation[8]. Deciding which shortest path between the source and destination depends on the routing algorithm and the local routing table for every device. In an optimal routing algorithm, the algorithm must give us effective routing with minimum delay and maximum throughput. To keep updating the network status, routers need to exchange routing information periodically. This creates a lot of delays especially if the routing table is large. The computational burden on the router considerably increase if one of the nodes fails in the network, In order to find an alternate route; other routers carry out the same process [9].

According to [8], current market requirements such changing traffic patterns, the rise of cloud services, and growing demand of bandwidth cannot be achieved with the traditional network, rather, network designers are restricted with the limitations of traditional networks which include:

Complexity: In the computer networking field, there are several protocols which developed to offer a reliable connection between hosts. These hosts may operate on different distances, link

speeds, and topologies. So, to meet technical and business needs, protocol vendors have evolved networking protocols that provide high reliability, connectivity, and security. As most protocols designed to address specific problems in an isolated way, this leads to a level of complexity with configuring the networks. For example, any changes are made on the network need that all the network devices to be configured to work with those new changes. As a result of this level of complexity, traditional networks are considered as static nature networks.

Virtualization is a new technology that addresses the problem of static nature. It is working in a dynamic nature and no more physical location of hosts is needed. Nowadays virtualization can apply in many aspects such server virtualization, network virtualization, and storage virtualization. Therefore, virtualization brings additional problems to the traditional network such as addressing schemes, routing based design etc. Additionally, IP networks used in many enterprises provide different QoS levels according to the type of application e.g. voice, data, and video traffic. Therefore, configuring those different levels of QoS requires hard manual work. As all of those problems, traditional networks cannot dynamically adapt to changing traffic, application, and user demands.

Inconsistent Policies: Policies are a very important aspect in enterprise network because it defends the enterprise network against any negative consequences. So, to apply a specific network-wide policy, IT staff needs to go cross thousands of devices to configure that policy. Because of that complexity associated with traditional networks, it is very difficult for IT staff to apply consistent policies across the entire network.

Inability to Scale: A network must grow in line with the growing market demands to gain sustainable and competitive markets, users and profits. The network forecast analysis would be

helpful, but due to current dynamic market nature, it does not provide much help to plan scalability in advance. The complexity and inconsistent policies applied to traditional network limit the fast scalability of a network.

Vendor Dependence: In some cases, the enterprises are obliged to add new services or capabilities in response to changing in market or user demands. Some services and capabilities are vendor dependent and are not compatible to work with the different vendor. This causes the enterprises to response in slow or may uncompleted way.

2.5 SDN Architecture

Traditionally, a computer network is composed of interconnected network devices such as switches and routers devices. In each device, there is a data layer that includes the packet forwarding mechanism, and a control plan that incorporates the operating system and applications. In this model, the network devices are closed and there is no possibility to add a new application. The installation of new protocols depends on the production cycles that could be long. In order to open the network device and separate the applications from the forwarding devices, the architecture of "software-programmed networks", or SDN, was born. It consists of three main layers and communication interfaces [2]. Figure 6 shows difference between SDN and traditional networks and how the SDN architecture decouples control plane from the forwarding hardware. In this section we describe the SDN layers as well as the interfaces used to communication between these layers.

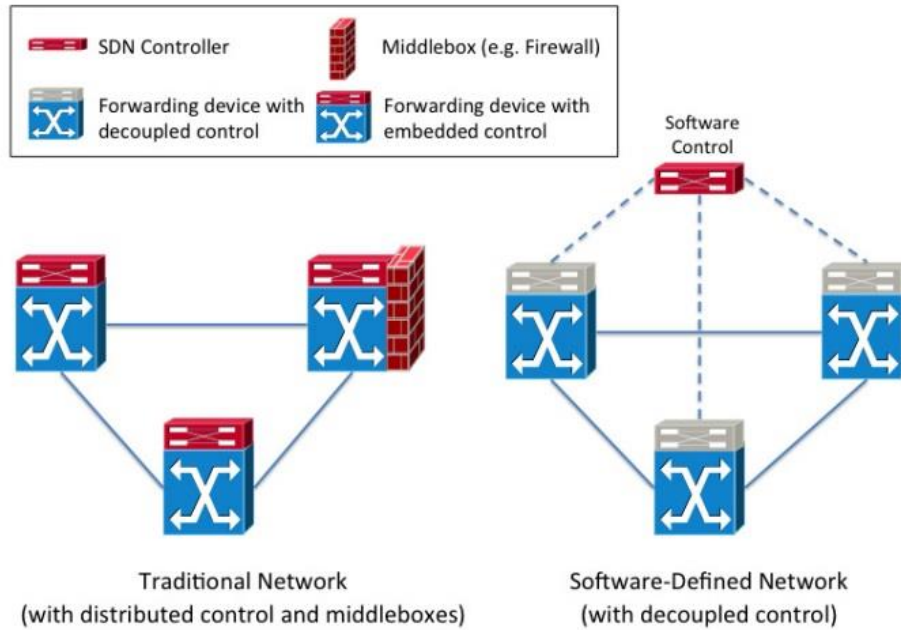


Figure 6 Difference between SDN and Traditional Networks

As it is obvious in Figure 7, the lowest layer is the infrastructure layer, also known as the "data plane". It contains forwarding devices such as physical and virtual switches. Its main role is to transmit data, monitor local information and collect statistics. The control layer, also called "control plane", consists of one or more control software (Controllers), these controllers use open South interfaces to control the behavior of the forwarding devices and communicate via North APIs with the layer superior to supervise and manage the network. The application layer (the highest layer) hosts applications that can introduce new network features, such as security, dynamic configuration, and management. The application layer exploits the global and remote view of the network provided by the control plane to provide appropriate guidance to the control layer. There are three types of interfaces that allow controllers to communicate with their environment: South/North and East/ West interfaces. We detail them later.

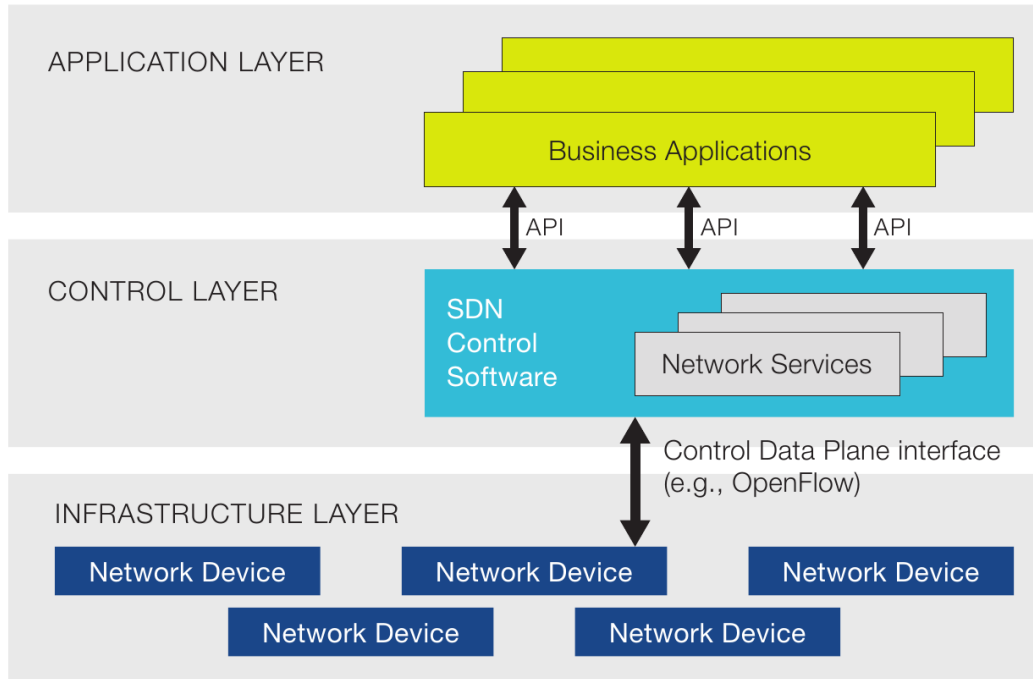


Figure 7 SDN Architecture

2.5.1 Infrastructure Layer

It is also known as data layer which is the bottom layer of SDN architecture. It consists of a set of network devices such as physical switches, routers, virtual switches, and wireless access point. The network devices are interconnected together to form a single network, and they communicate with each other using diverse transmission media like copper wires, optical fiber, and wireless radio. As the control logic and algorithms are offloaded to a controller, those network devices are accessible via an open interface at an abstraction layer. The main function of this layer is forwarding the packets according to the assigned rules/policies.

2.5.2 Control Layer

It is located at the middle of the of the SDN architecture and comprises of one or more controller servers. This layer works as a bridge between the infrastructure layer and application layer through its two interfaces (south-bound interface and the north-bound interface). Using the

south-bound interface, the control layer can interact with the infrastructure layer and can access the functions that are provided by network switches. Those functions include installing forwarding rules and reporting network status. On the other hand, the north-bound interface is used to interact with the application layer and works as a service access points to the user applications in different forms. The most famous form is the Application Programming Interface (API) which helps SDN Applications to access network devices and import all the information about the network status. This helps the system making its decision according to this information. Then, those decisions can be carried out by installing packet forwarding rules on the network switches via the API. Since multiple controllers will exist for a large administrative network domain, an "east-west" communication interface among the controllers will also be needed for the controllers to share network information and coordinate their decision-making processes [10].

2.5.3 Application Layer

It is the foremost layer in SDN architecture which hosts applications that can introduce new networking features, such as security, dynamic configuration, and management. The application layer uses the global and remote view of the network offered by the control plane to provide appropriate guidelines to the control layer. This layer communicates with the control layer using Application Control Plane Interface (A-CPI) also called as a northbound application interface.

2.6 Communication Interfaces

As it is obvious in Figure 8 the controller interacts with the other layers through the South and North interfaces and with the other controllers through an East-West interface. The following is discussion about these different interfaces.

2.6.1 Southbound Interface

They are communication interfaces that allow the controller to interact with the switches/routers of the infrastructure layer. In other words, it is a link between the control-plane and data-plane. This link actually is a channel between the controller and the underlying forwarding devices. OpenFlow is the most commonly used southbound interface, which is responsible for establishing a secure link between the controller and infrastructure devices. Another southbound interface that has similar goals like OpenFlow is ForCES. This protocol has its own rich features.

2.6.2 Northbound Interface

The north APIs work as northbound interfaces between the controller and the user applications of the application layer. The application developers use those interfaces to monitor and manage the network via customized programs. Since the traditional networks are static in nature, it is hard to achieve like this flexibility in network managing. There are a lot of programming languages that used to program network devices such as Flow-based management language (FML), pyretic, frenetic, and procera. Those types of languages help in designing high-level packet-forwarding policies for switches.

2.6.3 East-west Bridge

Large-scale networks are partitioned into smaller sub-network and each sub-network is controlled by its own controller. In this multi-controller architecture, each controller implements its own east/westbound APIs. The main function of those APIs is to synchronize network states and to ensure compatibility and interoperability between the different controllers.

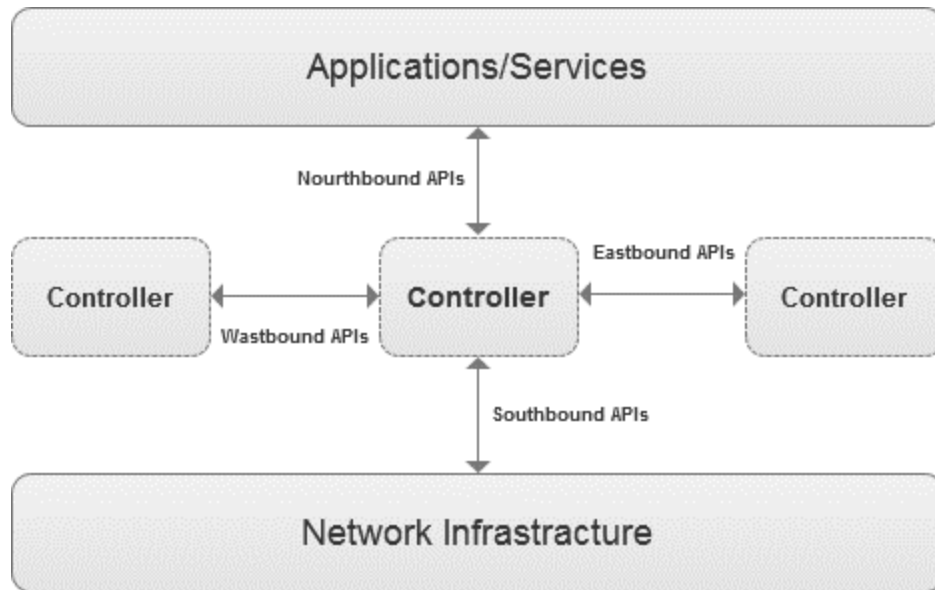


Figure 8 SDN Interfaces

2.7 SDN Applications

SDN networks have a wide variety of applications in network environments, providing customized control, an opportunity to eliminate intermediate equipment, and also simplify the development and deployment of new network services and protocols. According to IDC (International Data Corporation) [11], the SDN market consists of infrastructure equipment, control, and virtualization software, networking and security applications, to \$ 12.5 billion by 2020. This section presents some examples of SDN applications.

2.7.1 Enterprise Networks

In large enterprises networks, security and performance are the main requirements that are needed strictly. In addition, there is some other requirement and characteristics that must be satisfied depending on the type of the environment. For example, University networks environment is an exceptional case of enterprise networks. In such an environment, there are some temporary connecting devices that are not controlled by the university. Furthermore, the

universities must offer specific facilities and labs for researching and testing new ideas and protocols.

One of the main important factors in enterprise networks environments is network management. SDN is the appropriate management solution which uses programming to enforcing and setting network policies. Moreover, it provides special tools for monitoring network activity and tuning network performance. SDN can integrate the functions of middleboxes within the controller and this can simplify network management. Examples for middlebox functionalities that can be integrated are firewalls, NAT, load balancers, and network access control.

2.7.2 Data Centers and Cloud Computing

In recent years have been strongly marked by a new technology that is cloud computing or cloud computing. This technology consists of concentrating resources (storage, calculation, etc.) in data centers and offers them as a platform or service to the user. This trend has increased the amount of data handled and consequently the explosion of the number of data centers. This poses a challenge for the networks that transport traffic, especially for large operators with multiple connected data centers. Internet giant Google has been experimenting for 3 years with the benefits of the SDN in managing and controlling its data centers around the globe. The project consists of building an experimental core network called B4[12] in parallel with its operational core network that carries Internet traffic. B4 connects Google's distributed data centers by separating control and data plans. This allows rapid deployment of new services and centralized traffic engineering service in an SDN controller, increasing network throughput by 30% [13]. IT Resource Management is the most important challenge for Cloud networks. SDN is highly regarded as one of the most recent solutions, allowing easy configuration and management of cloud and data centers. Oracle SDN [14] is an example of a system that provides a virtualized

network to the data center. This system dynamically links virtual machines with network servers. Using the Oracle Fabric Manager interface, virtual network configuration and monitoring are possible anywhere, and the deployment of new services such as firewall, load balancing, and routing becomes on demand. According to Oracle, their proposal increases by 30 times the performance of applications.

2.7.3 Wireless Networks

Odin [15] exploits the programmability feature in SDN to be used in enterprise WLANs to get several functionalities such as mobility, authentication, and AP association decision. In Odin, WLAN services can be implemented by the network operator as network applications and no client-side modifications are needed. OpenRadio [16] suggests a wireless data plane with a programmable feature which depends on declarative and modular programming interfaces through the whole wireless stack. OpenRadio is designed to work with current wireless protocols such as LTE and WiFi which provide elasticity to adjust the Physical and Data Link layers. OpenRoads [17] and [18] is an open-source platform developed to examine several routing protocols, mobility solutions, and network controllers. OpenRoads control and configure the data plane and its devices by OpenFlow and SNMP. The centralized control capability can be exploited for making flow-based forwarding and routing in wireless mesh networks [19], for achieving complete programmability and virtualization in radio access networks [20], or for abstracting base stations in a local geographic area [21]. According to [22], developing new services and management of cellular data networks would be easier than before by using the SDN. In [23], they argue that using OpenFlow in SDN would optimize handovers in heterogeneous wireless environments.

2.7.4 Security

Due to its centralized control, SDN is very useful for applications that required security. OpenFlow Random Host Mutation (OFRHM) [24] is one of many techniques that used to protect network assets. In this technique, network assets are hidden from internal or external scanners that try to discover network targets. To avoid revealing host real IP used by the hackers, each host assigned a random virtual IP by SDN controller. [25] is another technique that deals with network security in office and home networks. This technique applies several algorithms for traffic anomaly detection. It supports OpenFlow switches and uses NOX as an SDN controller. According to experiments results, SDN-supported approach gives more precise and accurate identification of malicious traffic than the ISP-driven approach. In [26], enterprise networks are secured by using dynamic access control policies in which those policies are based on flow information. Furthermore, it gives the switches the ability to take actions such as packet dropping. There are some other researches that utilize SDN features to develop applications like security application development frameworks [27] and edge-based authentication gateways [28].

2.7.5 Multimedia and QoS

Today's Internet architecture relies on sending packets without considering the type of packets and the quality of the transmission. Multimedia applications such as streaming video, video on demand, video conferencing, WebTV, etc., require stable network resources and tolerate transmission errors and delays. Based on the centralized view of the network offered by SDN, it is possible to select, depending on the throughput, different paths for the various traffic flows. In [29] the authors provide the Video over software-defined Networking (VSDN), a network architecture that determines the optimal trajectory in using an overview of the network. The VSDN was implemented in NS3, and its behavior was analyzed using the complexity of the messages between the VSDN and the hosts/switches (adding a new flow table, requesting a new

service, deleting the service), the bandwidth and the transmission delay on the network. VSDN is a research project in with other improvements to be made.

2.7.6 Home Networks

Management of small network (e.g. home networks and café networks) becomes very difficult due to the market currently becomes filled with diverse types of low-cost network devices, which are familiar among users. For best simple management, SDN can be used for those types of networks. In [25], the authors discuss that the technology of SDN can offer a great method for detecting and addressing the problems of network security in home networks. The authors in [30] introduce a home router design dedicated for home networks which mainly used to monitor network operations, control network traffic flows and give the user the ability to manage network behavior. The authors of [31] propose home network data recorder for managing and troubleshooting home networks. To get flexible remote management, the authors of [32] exploit the programmable network feature in OpenFlow switches and propose a method to outsource the management of home networks to the third party with operations expertise and a broader view of network activity.

2.7.7 Information-Centric Networking

Information-Centric Networking (ICN) [33] is a novel approach which proposed to change the current Internet infrastructure from Point-to-Point Communication to content distribution. Due to ICN is a revolutionary technology, it requires a new network architecture. On this matter, SDN is a worthy platform to implement and test ICN [34] [35]. The authors in [36] discuss the implementation of ICN functionalities over a network-based OpenFlow and what are the modifications that are needed for OpenFlow to give better ICN functionalities. In [37], the authors clarify how SDN combined with ICN could and how can they deployed and tested. Furthermore, the authors suggest a new design for ICN solutions and testbed deployments for

future tests. On the other hand, the authors in [38] introduce an OpenFlow network architecture that provides an efficient caching method for ICN networks. C-flow [39] leverages recent OpenFlow functionalities to achieve content delivery in an efficient way and uses the byterange option of the HTTP header to deliver content to mobile hosts. Additionally, C-flow supports multicast and unicast by making the mapping between the files and their corresponding IP addresses [40].

2.8 Development Tools

As previously mentioned in chapter 2, SDN has been proposed to help network researchers to rapidly evaluate and innovate of new services and protocols. In the following section, an overview is provided for the current tools and platforms that are used for deploying SDN-based services and protocols.

2.8.1 Simulators and Frameworks

Mininet provides a virtual testbed and environment for development software-defined networks. It allows emulating and deploying an entire SDN network in a single machine. In addition, Mininet can seamlessly move SDN designs to the real hardware. For creating and experimenting a network, Mininet supports an extensible Python API and its code can include real Linux kernel, standard Unix/Linux network applications, and network stack. Although Mininet supports OpenFlow v1.0 by default; it can be modified to allow using a switch that implements a new version [41].

ns-3 is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. By default, ns-3 simulates entire networks, from applications down to channels. ns-3 supports OpenFlow v0.89 [42].

OMNeT++ is a C++-based discrete event simulation environment which developed in 1997 for simulating multiprocessors and distributed systems and modeling communication networks. OMNeT++ developed to be a powerful open-source network simulator that can be used by educational, research-oriented, and academic institutions. It is free for non-profit use and can be used under the Academic Public License. OMNeT++ supports OpenFlow v1.2 through a plugin. [43]

EstiNet is a simulator and emulator that used to test and evaluate the performances the application programs of OpenFlow controller. EstiNet uses an innovative simulation methodology called **kernel re-entering** which enable unmodified real applications to run on nodes in the simulated network. As a result, a real Linux operating system with any UNIX-based real application programs can be run by each simulated host. The *kernel re-entering* methodology also helps the EstiNet to work as a simulator and emulator at the same time. EstiNet can simulate thousands of OpenFlow v1.3.2 and v1.0.0 switches and run the real-world NOX, POX, Ryu, and Floodlight controllers [44].

Trema [45] is a full-stack and open source programming framework uses the Ruby and C programming languages for developing OpenFlow controllers for SDN networks.

Mirage [46] is an exokernel developed specially for constructing high-performance and secured network applications across a variety of platforms and it supports OpenFlow.

Maestro is a Java-based platform designed for achieving automatic and programmatic network control functions using modularized applications.

HyperFlow [47] is an event-based distributed control plane for the OpenFlow protocol, which supports the deployment of any number of controllers in the network. HyperFlow is physically

distributed but logically centralized: it achieves scalability while keeping the benefits of network control centralization. Through synchronization between controllers, HyperFlow shares the same state of the network-wide view between all the controllers. With HyperFlow, All the local requests can be served and processed through the local controllers without the need to contact with any remote node. As a result, the response time and the flow setup times with the data plane would be minimized. HyperFlow supports OpenFlow without any need to add any changes, only pretty modifications to current control applications. HyperFlow is flexible in dealing with network partitioning as well as component failures. Besides, it allows interconnecting independently managed OpenFlow areas.

Onix [48] is a Distributed Control Platform which implements the network control plane as a distributed system. It developed to be used by a large-scale network which uses a global view of the network. Additionally, Onix defines a useful and general API for network control implementations, which allow the control applications to read and write to any node in the network.

2.8.2 Controller

SDN controller is the core of the network which acts as the strategic control point in the SDN network. SDN controller can creates, updates, and removes flow entries in flow tables on a network device. Moreover, SDN controller typically runs on a central server which simplifies network management. It uses communication protocols such as OpenFlow to handle all communications between network applications and network devices. There are several SDN controllers which can be classified into two types, general controllers and special purpose controllers. The following is a list of some common SDN controllers.

General Controllers:

NOX [49] is the first OpenFlow controller which initially developed by Nicira and now is possessed by VMware. It provides a high-level programmatic interface for the development and management of the network control applications. NOX is written in C++ and Python.

POX [50] is a general open source Python-based OpenFlow controller for SDN control applications. POX enables rapid development and prototyping and becomes more commonly used than NOX. POX supports OpenFlow 1.0, Open vSwitch/Nicira extensions and has partial support for OpenFlow1.1.

Ryu [51] is a Component-based framework that integrates with OpenStack and supports OpenFlow. It Provides software component with well-defined API for network management and control applications. It developed by NTT laboratories and supports various versions of OpenFlow, OF-Config, Nicira extensions. With Ryu, it can easily set up a multi-node OpenStack environment using pre-configured Ryu VM image file.

MUL [52] is a C-based OpenFlow controller platform which has the multi-threaded infrastructure at its core. It allows a multi-level northbound interface for hooking up applications. Additionally, it supports OpenFlow v1.0 and v1.3.

Beacon is an open source cross-platform modular OpenFlow controller implemented in Java. It Developed at Stanford University and supports both event-based and threaded operation.

Floodlight Java-based OpenFlow controller based on Beacon runs within a JVM. Floodlight is the core of Big Switch Controller from Big Switch Networks. Floodlight supports virtual and physical OpenFlow switches and can handle mixed OpenFlow and non-OpenFlow networks.

NodeFlow is a Node.JS-based OpenFlow controller which relies on a protocol interpreter called OFLIB-NODE.

Special Purpose Controllers:

RouteFlow is a platform for providing virtualized IP routing services on top of OpenFlow networks. RouteFlow utilizes routing software such as Quagga to build a virtual topology that is mirrored to a physical topology. It allows for the deployment of flexible, inexpensive and easy to administrate networks.

FlowVisor is an OpenFlow controller which works as a Transparent Proxy between OpenFlow switches and multiple OpenFlow Controllers. It creates slices of network resources that can be managed by different controllers.

SNAC [53] is Open-source NOX-based OpenFlow controller, with the web-based graphical user interface, for production enterprise networks. It supports admission control, showing network components and a policy definition language.

Resonance is a Network Access Control application built using NOX and OpenFlow.

OFLOPS [54] is a generic and open framework that enables tests of OpenFlow-enabled switches and implementations. It measures the capabilities and bottlenecks between the forwarding plane of the switch and the remote control application.

Table 2 below summarizes common SDN controllers with their specifications.

Table 2 Summary of Common OpenFlow Controllers

Name	Program Language	Company	OF. Version
NOX	C++/Python	Nicira	1.0
POX	Python	Nicira	1.0
Ryu	Python	NTT	1.3
MUL	C	Kucloud	1.3
Beacon	Java	Stanford	1.0
Floodlight	Java	BigSwitch	1.0
NodeFlow	JavaScript	-	1.0
RouteFlow	C++	CPQD	1.0
FlowVisor	C	ON.LAB	1.0
SNAC	C++	Nicira	1.0
Resonance	Python	-	1.0
OFLOPS	C	-	1.0

CHAPTER 3

OpenFlow Protocol

3.1 Introduction

OpenFlow is an open network protocol that design to control the network equipment programmatically. OpenFlow is the most popular SDN technology, and it originally proposed and implemented by Stanford University in late 2008, subsequently, it was standardized by the ONF [55]. The main goal under designing this protocol is to help researchers to work directly with real network devices to test their experimental protocols. Before emerging OpenFlow, it was very difficult to test any new protocol or ideas in the production network environment.

In the last two decades, a large improving has happened in network technology especially in speed, reliability, ubiquity and security features. At the level of physical layer, network devices have enhanced with regard to computational power and the tools that are used to inspect operations. On the other hand, since its early days, little changes have occurred in the network infrastructure. So, to add new services or features new component are added at the level of higher layers (4 - 7) whereas the lower physical layers (1 - 3) remains same without any changes. Network devices in the current infrastructure are responsible for processing network level decisions such as routing or network access. Most of the network vendors have designed their network devices to operate in a proprietary fashion. So, it is difficult for researchers to experiment with their new ideas such as routing protocols. Moreover, practicing experimental ideas in real production network may lead to network failure.

In the current infrastructure, network level decisions e.g. (routing and network access) are handled by the networking devices. These networking devices are manufactured by enormous commercial vendors that use diverse firmware. For this, to build a network that supports different vendors, Open fashion deployment is required rather than the proprietary fashion. The Open fashion is vendor independent deployment while proprietary fashion is vendor dependent deployment. Therefore, based on the vendor and its networking device, testing novel research ideas e.g. (routing protocols) has been difficult. Additionally, trying any experimental ideas in the production environment may lead to network failure at some point. As a result of this, the traditional network infrastructure is static and inflexible which does not help in coming up with new innovations in this field[56].

Because of Ethernet switches and routers have lookup tables; it is easy to implement firewalls, QoS, NAT or to gather performance statistics. OpenFlow takes into consideration the shared functions that are supported by most vendors. This will lead to accomplishing standard method to utilize flow tables for all network devices from a different vendor. Furthermore, it allows to partition network based on flow into 14 different flow classes. This will help in to organize network traffic and the flow classes can be set together or isolated to be routed, processed, or controlled in the desired way. Now OpenFlow can be used in critical areas where isolation of production and research is a crucial function.

With OpenFlow, Network administrators are offered a programmatic control to define the flows between sources and destination. Moreover, OpenFlow provides a method to get rid of packet processing for defining a path in the router, network management, and saving power consumptions while expanding a network. OpenFlow has attracted a lot of developers and manufacturers of network servers, routers, and switches.

In this section, we start by introducing the architecture of OpenFlow then we describe its different specifications, and we end with the different transmission elements that support OpenFlow.

3.2 Architecture of OpenFlow

As discussed earlier, network devices come with two planes control plane and data plane. Control plane is the software side which is responsible for all network intelligence and logic while the data plane is the hardware side where forwarding takes place. To achieve a centralized and programmable control over control plane, network devices supporting OpenFlow and a controller holding network logic are needed. OpenFlow is depended on switches with flow table and offers programmable, virtualized, open switching platform to control switch devices via software. The function of a switch, router or both can be implemented by OpenFlow. Also, the control path of networking device it can be facilitated to be controlled programmatically via OpenFlow protocol as demonstrated in Figure 9.

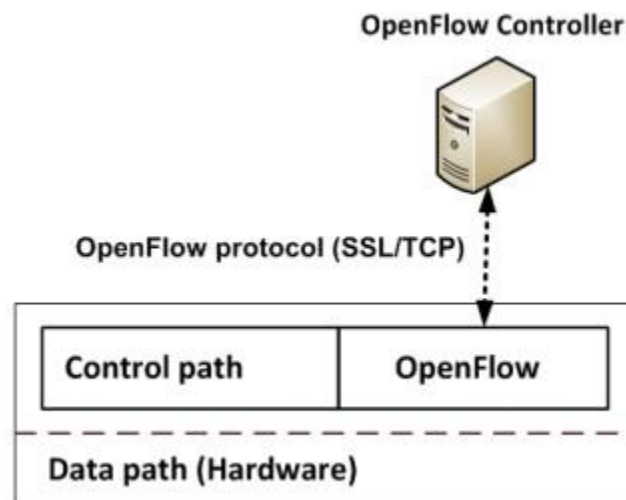


Figure 9 OpenFlow Protocol

OpenFlow controller communicates with the OpenFlow-based switch via channel secured by either SSL or TLS, But this does not mean that the connection is not prone to denial of service (DoS) attack. So, it is needed to implement a robust security measure to avoid such an attack. In OpenFlow architecture, data path flow forwarding still resides on the switch, but flow forwarding decisions are made in a separate OpenFlow controller or hierarchy of controllers, which is implemented in a server(s) that communicates with OpenFlow enabled switch(es) in the network through OpenFlow protocol.

3.3 OpenFlow Channel Connection

OpenFlow has a channel connection that acts as a communication medium between the switch and controller to send and receive OpenFlow messages. To create this channel, OpenFlow switch must initiate a connection to the controller. Multiple connections can be parallely established by one switch to one or more controllers. One of those multiple connections can be acts as a master and the other connections act as a salve or equal [57].

3.3.1 Connection setup on TCP & TLS

Once the switch runs, it starts establishing a communication with the controller using a configurable IP address and a transport port. The transport port can be the default one or the user can define a specific port. The default ports for all previous OpenFlow versions is either 6633 or 976. However, IANA has allocated port number 6653 to ONF for OpenFlow protocol. OpenFlow protocol uses either TCP or TLS as a communication protocol. The switch can create a TCP connection with the controller by using standard TCP/IP socket which is established by connecting to the controller's IP address and port number using the connect system call provided by the underlying operating system. On the other hand, to establish a TLS connection, a mutual authentication must take place between the switch and the controller by exchanging signed

certificates. Two certificates are required by the switch: one certificate to authenticate the switch for the controller and the other one to authenticate the controller for the switch. Additionally, TLS uses an open-source library called (OpenSSL) to secure the communication channel between the controller and the switch. OpenSSL is written in C programming language and has a number of cryptographic functions and utilities that provide communications security for the SSL and TLS protocols [57].

3.3.2 Connection setup with multiple controllers

Sometimes the switch is required to connect to more than one controller; in this case, the switch is configured to establish a communication channel with all the controllers. Connecting to multiple controllers is needed for the following reasons:

- To increase the reliability of the system
- To offer role-based load balancing between the controllers

When the switch is connected to multiple controllers, one of the three roles can be assigned to every controller; master role, slave role, or the equal role. The role of every controller is maintained by the switch with respect to the controller's connection. However, the controller's connection is identified by the switch using the channel ID which is a combination of the Auxiliary ID and the Datapath ID.

OpenFlow switch initiates a connection to all the configured controllers immediately after its initialization. Then it concurrently maintains all the connections to all the controllers. The process of establishing a communication channel to multiple controllers is the same as that of establishing a connection to a single controller. As the switch can establish connections and process requests from multiple controllers concurrently, it must support mechanisms to process

and read messages from those several controllers. So, OpenFlow Switch uses different threads or processes to handle the different requests from the multiple controllers [57].

3.4 OpenFlow Message Types

OpenFlow messages are sent and received between the controller and the OpenFlow device that it manages. This section describes the different types of OpenFlow messages.

3.4.1 Symmetric Messages

Symmetric messages are sent from both switch and controller with any solicitation from them.

The symmetric messages should be sent and processed by the OpenFlow switch except for the error message that will not. The following is a group of common symmetric messages:

- **Hello message:** messages exchanged between the switch and controller upon connection startup
- **Echo request and echo reply message:** to verify the liveness of a controller-switch connection, and as well can be used to measure its latency or bandwidth.
- **Error message:** to notify a problem to the other side of the connection
- **Experimenter message:** provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space.

3.4.2 Asynchronous Messages

Asynchronous messages are sent from both the switch and the controller when any state change happens in the system. Similar to symmetric messages, asynchronous messages also should be sent without any soliciting between the switch and the controller. The following is a set of asynchronous messages that can be sent by the switch to the controller:

- **Packet-in message:** To transfer a packet received from specific ports in the switch to the controller for further processing.
- **Flow-removed message:** sent from the switch to the controller when a flow entry is removed from the flow table.
- **Port-status message:** Sent by the switch to the controller when any changing happens in the status of the switch port such as port is added, modified, or removed.
- **Table-status message:** sent by the switch to the controller when any changing happens in the table status such as the number of entries in the table exceeds the threshold value.
- **Controller-role status message:** sent by a switch to the group of controllers when the role of a controller is changed.
- **Request-forward message:** sent by the switch to inform the controllers about the successful executing of a modify request message that sent previously by a specific controller.

3.4.3 Controller-to-Switch messages

Controller-to-switch messages are sent from the controller to switch which is used to inspect and manage the state of the switch. Some of the Controller-to-switch messages do not require the switch to replay back a response to the controller. The following is common messages of this type.

- **Features:** is a request sent by the controller to the switch asking the identity and the basic capabilities of a switch.
- **Read-state:** is used by the controller to collect different information related to the switch such as statistics, configuration, and capabilities.

- **Modify-state:** sent by the controller to manage the switch state such add flow/group entry in the OpenFlow tables, delete flow/group entry from the OpenFlow tables and set switch port properties.
- **Packet-out:** is a message that sent by the controller asking the switch to forward packets received via Packet-in messages out of a specified port on the switch.
- **Barrier:** the controller can use a barrier request to ensure that message dependencies are met, and the switch can use barrier reply to notify the controller about the completed operations.
- **Role-request:** controller uses this message to set or query the role of its OpenFlow channel. This is suitable for a switch that connects to multiple controllers.
- **Asynchronous-configuration:** the controller uses this message to set or query an additional filter on the asynchronous messages. This is suitable for a switch that connects to multiple controllers.

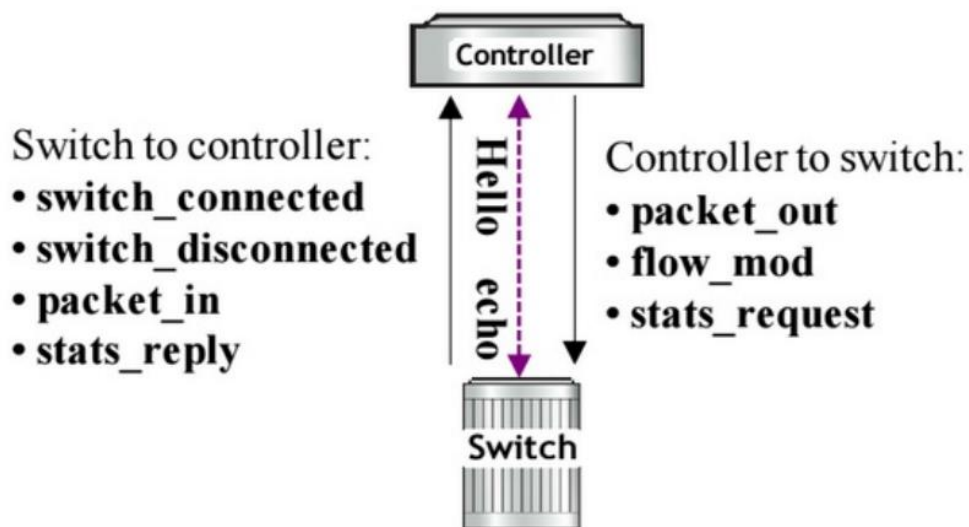


Figure 10 OpenFlow Messages

3.5 OpenFlow Tables

A switch may have more than one Flow Table such as Forwarding, QoS, firewall, and NAT tables. In this section, the discussion will clarify what the switch table and what are in it. Table 3 gives a simple and short example for the set flow entries of the flow table.

Table 3 OpenFlow Table Example

#	Header Fields	Actions	Priorities
1	If in_port = 1	output to port 2	100
2	If IP = 10.1.2.3	rewrite to 84.4.3.2, output port 6	200

The first flow entry in Table 3 states that if a packet comes to the switch from the port 1, it should be forwarded from port 2. However, the second flow entry says if a packet comes from port 1 and has a source IP 10.1.2.3, rewrites the source IP 84.4.3.2 and forwards it to port 6. As is obvious, there are two flows match a certain packet. To deal with such conflict situation the priority column is used. In this case, the second flow will be executed because it has a higher priority than the first flow.

Table 4 OpenFlow Entries Columns

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie
--------------	----------	----------	--------------	----------	--------

A flow table consists of multiple flow entries as exemplified in Table 4. According to the OpenFlow specification 1.4 [58], the fields as shown in Table 4 are the columns that make up an entry in the Flow table. The flow entries make up one flow table, and multiple flow tables make up the processing pipeline of OpenFlow which is explained in the next section.

- **Match Fields:** they contain the packet headers, the ingress port, and optional metadata included by the previous table. These fields are used to match against the incoming packet.
- **Priority:** this is the matching precedence of the flow entries when there is more than one entry for matching against the incoming packet.
- **Counter:** updated with every matched packet and used to track the number of packets that match against particular flow entry.
- **Instructions:** is a set of instructions to be executed against the matched packet.
- **Timeout:** This is the valid maximum amount of time for the flow entry before it expired by the switch.
- **Cookie:** is an opaque value chosen by the controller and can be used to filter flow modification, flow statistics, and flow deletion.

Figure 11 below describes in details the components of the Flow Table Entry.

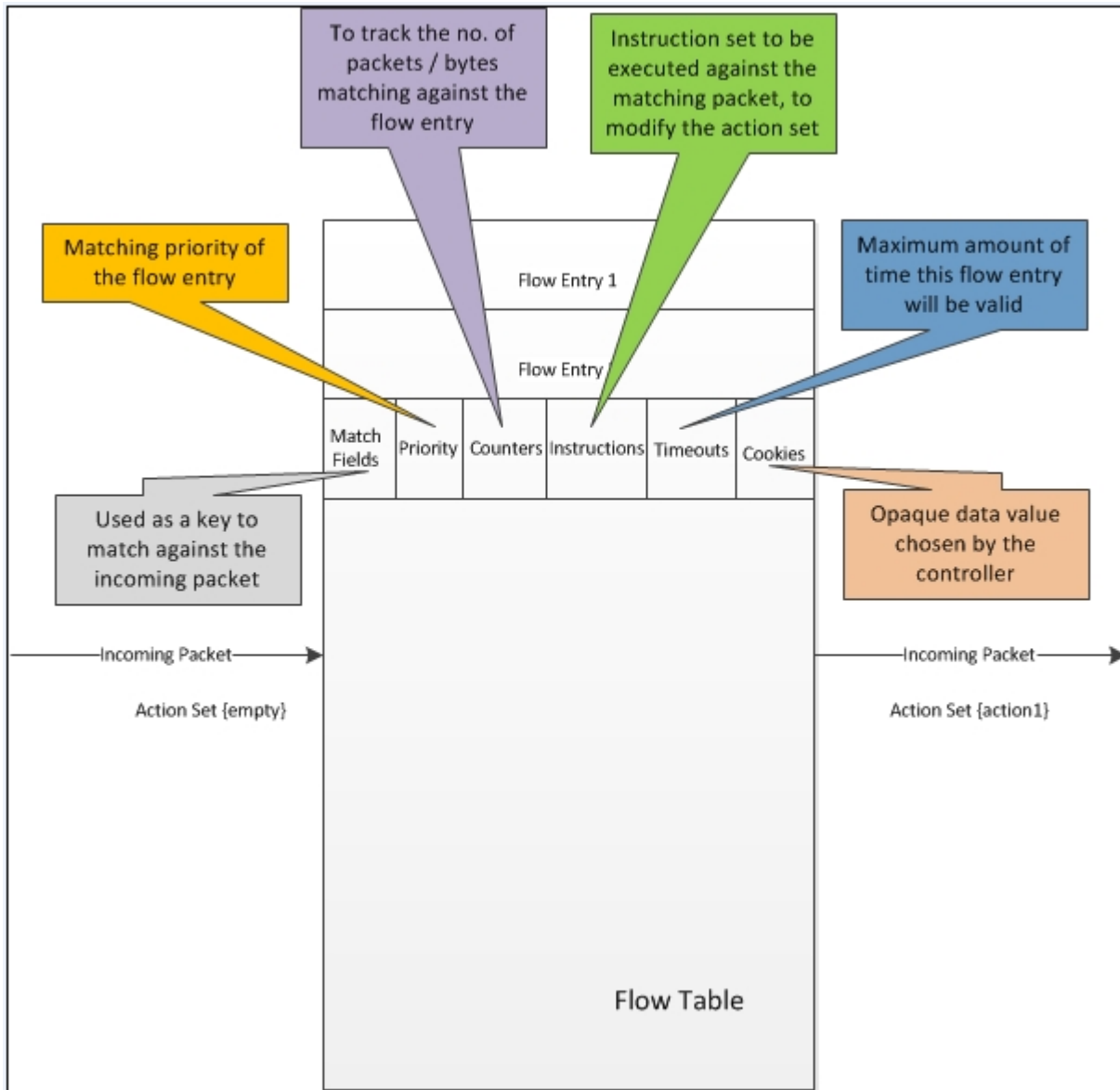


Figure 11 Flow Table Entry Components

3.6 Pipeline Processing

Every OpenFlow switch is required to have at least one flow table but they can contain more if needed. The pipeline of an OpenFlow switch defines how packets interact with the flow tables, as shown in Figure 12. The Figure assumes multiple flow tables as the pipeline with only a single flow table is greatly simplified.

In OpenFlow pipelining, the flow tables are sequentially numbered, starting at 0. The processing of packets always starts at the first flow table (0), where a packet is a match against the flow entries it contains. Depending on the outcome of the packet processing in the first table, the other tables may be used [58].

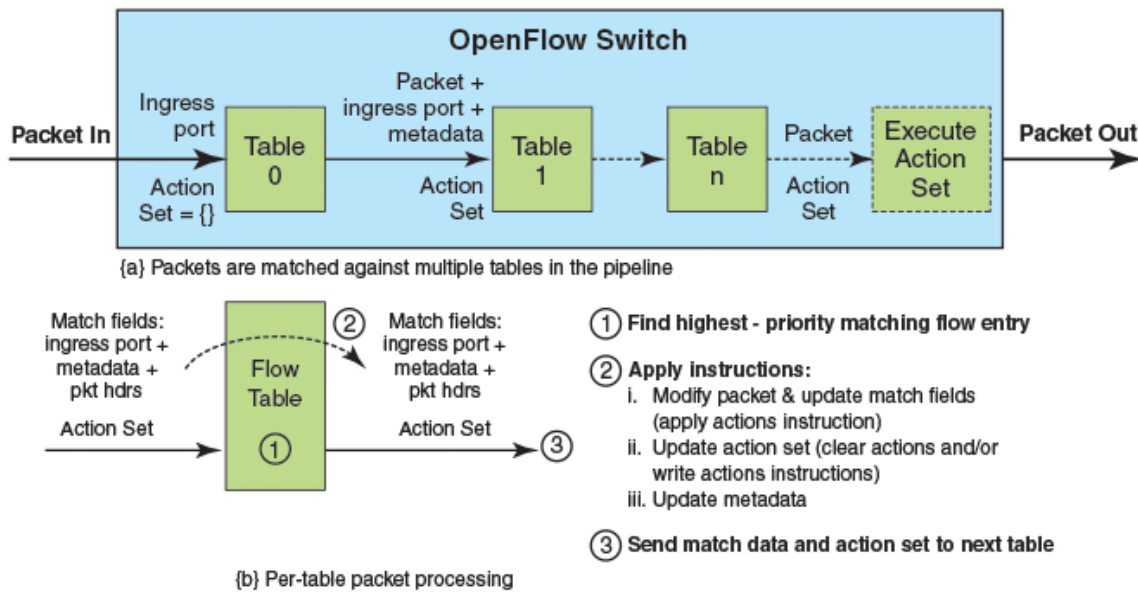


Figure 12 Pipeline Processing

Every packet in OpenFlow has an action set associated with it. This is by default empty and follows the packet through the whole pipeline process. The action set can be modified by flow entries, accommodating changes until it reaches the execute action at the end of the pipeline. When a flow table processes a packet, it is matched against the flow entries of that flow table. If a matching flow entry is found, the actions set (instructions) for that flow entry is executed. An instruction may contain the use of the GotoTable action where the packet is sent to another table where the same process happens again. Do note that the GotoTable action may only direct a packet to a flow table with a larger table number than itself. This means that the pipeline processing cannot go back only forward. Obviously, then for the last table, there cannot be any

GoToTable instructions. If there are not any GotoTable instructions in a flow table that matches for a specific packet, the pipeline processing stops and executes the action set it has acquired so far [58].

On the other hand, if a packet does not match any flow entries in a flow table it is a table miss. What a switch should do with missed packets depends on the configuration in the form of a table miss flow entry. These options for the switch are to drop the packet, pass them to another table or send them to the controller [58].

3.7 Packet Format

OpenFlow packet format mainly consists of three fields: headers counter, and action that is installed on a flow entry in the flow table.

- **Header fields:** these are the mating fields which identify the packet flow by matching the packet against a specific field as they are shown in Table 5

Table 5 Matched Header Fields in OpenFlow

Ingress port	VLAN ID	Ethernet			IP			TCP/UDP	
		Src	Dst	Type	Src	Dst	Protocol	Src	Dst

- **Counter:** updated with every matched packet and used to track the number of packets that match against particular flow entry.
- **Actions:** The action field responsible for specifying how the packets will be processed and can be one of three actions: 1) forwarding the packet to a certain port or ports

(sometimes rewetting over some header fields “optionally”), 2)forwarding the packet to the controller, 3)Dropping the packet.

3.8 OpenFlow Versions

The first version of OpenFlow was released in 2009. It would then be almost two years before version 1.1 came out and added support for Multi-table pipeline processing, MPLS, and QinQ. Followed by the release of version 1.2 10 months later, it added support for IPv6 and additional extensibility. In 2012 version 1.3 was released adding support of QOS alongside with other features, followed by the release of 1.4 in 2013. Version 1.4 introduced support for decision hierarchy and multiple controllers along with other features. At the end of last year, 2014, the specifications for 1.5 was released and approved by (ONF) board, but has not yet been approved by all parties and finalized. Although new releases of OpenFlow come out, there is a lack of vendors including support for the newest versions of OpenFlow in their products before the market demands it. Figure 13 below shows OpenFlow versions timeline.

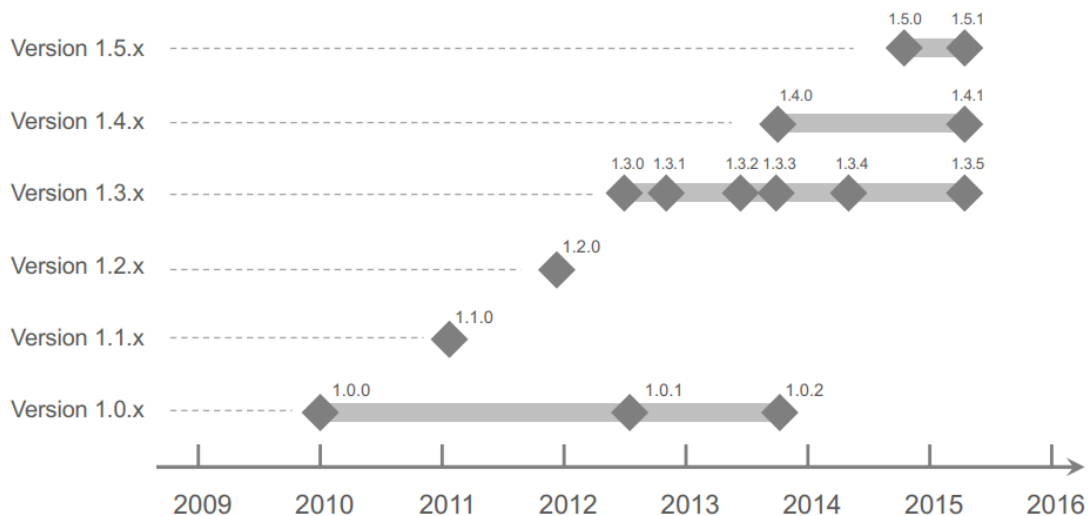


Figure 13 OpenFlow Versions Timeline

3.9 OpenFlow Switch

OpenFlow switch differentiates from the normal switch in a way that it does not make the decisions itself. Indeed, it needs a central controller that will be responsible to make these decisions [59]. Each OpenFlow must have support from at least one flow table which is edited by the controller through a set of commands (e.g. add, delete or update). Every flow table contains a set of flow entries which specify what the switch has to do with the packet if it matches any one of them. These flow entries can be installed in the switch either proactively before the packets come, or reactively as a response when the switch receives a packet. In case the packet does match any entry in the flow table, the switch will send the packet to the controller to make a decision how to do with the packet.

The biggest constraint with OpenFlow switches has been in the lower-end section as they come with less TCAM space. TCAM is a special lookup RAM for switches that can take three different inputs: 0, 1 and X. It is used to store flow rules, for fast lookups. But as OpenFlow supports very fine-grained control, there may be many rules required and running out of TCAM space to store the rules is a problem.

CHAPTER 4

Related Work

EtherProxy [60] provided a studying about ARP and DHCP broadcast traffic and presents a proposal for dealing with this types of traffic using a specific device that works as a proxy. EtherProxy is a backward compatibility in which no changes are made on the legacy LAN equipment or the network clients. EtherProxy is nothing just a new additional device that linking Ethernet nodes and as a result, more devices needed to install when the number of network nodes increased.

In [61], the authors study the common issues in current Ethernet networks and suggest an approach for eliminating the flooding in Ethernet. The approach works by using a distributed control plane in the Ethernet. Moreover, it changes the broadcast traffic protocols such as ARP and DHCP with directory-based switches. Therefore, the backward compatibility is not supported because its system depends on the new integrated directory service for requests responding. For all of that, the easiness of deployment and management of Ethernet is no longer exists in this approach.

SEATTLE [62] presents a distributed control plane Ethernet architecture which provides a scalable and floodless L2 Ethernet network. With SEATTLE, all ARP and DHCP message is sent in a unicast manner as the approach using the one-hop directory services. The approach uses Link Statement Protocol to transfer state information of hosts to make distributed ARP cache tables that are used for the directory services. The major issue with SEATTLE is the loss of interoperability with the legacy Ethernet switches and the other existing broadcast protocols.

MOOSE [63] studies the problem of scalability in Ethernet that uses a flat addressing, in which the size of MAC table increases in proportion to the number of hosts in the network. MOOSE comes with a schema that associates the switch identifier with a hierarchical MAC address. This replaces the MAC address of the hosts in the forwarding table and removes the need for switches to maintain a large forwarding table. This addressing schema consists of 24 bits of a host identifier and 24 bits of switch identifier. The switch identifier is used in the hierarchical MAC address to indicate the location of a particular host. In lieu of using spanning tree algorithm to determine the shortest path, MOOSE use a type of layer 2 inter-switch routing protocol to forward the frames. Also, MOOSE uses Enhances Lookup (ELK) dictionary service to replace ARP and DHCP protocol. In addition, it deals with the dynamic changes and mobility of the hosts inside the network. As MOOSE is designed to give a high-performance Ethernet architecture, it remains has problems related to the system complexity and the high management cost.

FSDM [64] provides an approach that can be considered as adapted or a complement for MOOSE. FSDM eliminates permanently the using of the flooding mechanism in the Ethernet network, depending for that on two hash tables IP-MAC (Mapping IP to MAC) and MACMap (Mapping from MAC to host location). It uses an SDN-based proxy controller to response the ARP and DHCP messages. The major drawback of FSDM is that it does not support handling hosts with static IPs.

To break the scalability bottleneck by utilizing overlay network, lots of architectures have been proposed [65] [66] [67]. The L2 network are been partition by the approaches into small segments of broadcast domains, network devices that are called edge bridges are been utilized to make a connection between the network core and segments. Unfortunately, the scalability

problem of the inter-segment address resolution is not solved by them. On the other hand, a central database approach is been utilized by the VL2 [68] and Portland [69]. In any case location-based addresses are required by Portland and VL2, which are a locator-specific address (LA) and Pseudo MAC (PMAC) respectively. This causes any migrated VM to reassign the its address and reconfigure the network.

The Distributed Registration based Address Resolution Protocol (DRARP) is proposed by the work [70] and Mac-in-MAC (EMiM) is enabled by the end-user. The amendment of existing hardware in end hosts is required by these approaches. Despite that NetLord [71] is been utilized by a push-based ARP proxy model, broadcast still happens on the boot of new VM. Additionally the requirement of its memory is proportional to the number servers and however may prompt to scalability problem as there is a growth in the number of servers. The improvement of Portland as a distributed alternative is been attempted by the Torii-HLMAC [72], but there is also dependent of the specific topography, fat tree by it. In addition, ToriiHLMAC utilizes another addressing scheme that depends on MAC address, Hierarchical Local MAC (HLMAC), and tree-based forwarding. This new scheme makes it incompatible with traditional Ethernet switches.

CHAPTER 5

Tool Used for This Project

5.1 Wireshark

Wireshark is an open-source program used for network troubleshooting and packet analysis. It is considered as a cross-platform program, and it runs under Unix as well as Microsoft Windows operating systems. It provides dissection of networking protocols. However, some protocols require a dissector plugin in order to attain optimum usage. Wireshark is capable of capturing unknown packets. However, openflow.lua plugin is required in order to decode particular information about packets of OpenFlow or to properly filter them. By supporting GUI (Graphical User Interface), Wireshark provides live view of traffic based on monitored the network card. Wireshark is used in this thesis to monitor at the loopback interface, since all the traffic of Mininet testbed comes through that virtual network card.

5.2 Tcpdump

Tcpdump is very similar to Wireshark, as this tool also captures packets passing an interface. There is, however, no GUI as it is a command line tool. It runs on most Linux systems and will be used on the individual virtual hosts in Mininet to analyze network behavior.

5.3 Oracle VM VirtualBox

VirtualBox is an x86 virtualization software developed by Sun Microsystems that is freely available as Open Source. It allows the use of fully functional operating systems to run virtually on a host system. For this thesis, VirtualBox will be used to run the Mininet image on an OSX host computer.

5.4 Hping3

Hping3 is a network tool able to send custom ICMP/UDP/TCP packets and to display target replies like ping does with ICMP replies. It handles fragmentation and arbitrary packet body and size, and can be used to transfer files under supported protocols.

5.5 Arping

Arping probes hosts on the attached network link by sending Link Layer frames using the Address Resolution Protocol (ARP) request method addressed to a host identified by its MAC address of the network interface.

5.6 Scapy

Scapy is a powerful program used for interactive manipulation of packets. It has the ability to decode or forge packets of different kinds of protocols, send and capture them, match replies and requests, and other services. It is able to handle several traditional tasks easily such as tracerouting , scanning, probing, testing of units, attacks or discovery of network (it enables the replacement of 85% of nmap, hping, arpspoof, arping, arp-sk, tethereal, tcpdump, p0f, etc.). In addition, it performs well when executing other particular tasks in which the majority of other tools are unable to handle, such as injecting user's 802.11 frames, sending of invalid frames, combining techniques (e.g. ARP cache poisoning ,VLAN hopping, decoding of VOIP on encrypted WEP channel), etc.

5.7 Mininet

Mininet is an emulator used to create a network of virtual controllers, switches hosts, and links in one machine. In order to emulate SDN in mininet, OpenFlow is supported by Spawned switches and the virtual hosts run under Linux operating system. Mininet depends on network namespaces, a Linux feature, and it requires kernels version above 2.2.26 in order to function

properly. Therefore, Mininet can run real code including network stack and real Linux kernel and as well as network-related applications. Mininet has the ability to emulate networks having different sorts of customs since it supports several customized topologies. Moreover, Python APIs are provided which enable network building and testing through python scripts. Mininet is generally used to show proof of concept, not performance, because of the overhead resulted when data flows are emulated. The reason is that packets first must be exchanged between the virtual switches in order to enable emulation of packet flows. Consequently, a Packet-In is sent by the switch to the controller, in which a kernel to user space context switch takes place and overhead is induced. This reduces traffic of control plane that is emulated by Mininet testbed.

The benefits of using Mininet for research purposes is that it requires less resources unlike using a full-deployed virtual network. It has better scalability, faster boot up, and easier be installed. Therefore, Mininet is selected in this thesis as the ideal choice to run OpenFlow controller againstOpen vSwitches. In Mininet, vSwitch is denoted as ovs, and it is an open source, multilayer, production quality, virtual switch.

5.8 POX Controller

POX framework is used to interact with switches that support OpenFlow and developed using Python programming language (NOXRepo.org, 2015). POX is considered a sibling of NOX , which is the original controller of SDN developed using in c++, where scripting language is the main difference between them. The reason that Murphy McCauley, the maintainer of NOX/POX, recommends to start with POX controller is that POX is being developed using Python, which means that most platforms (e.g. Windows and Linux/Unix) can support it. Several modules, resembling several types normal switch behavior, can be included when installing a POX. Moreover, custom modules, as well as other modules of routing, are supported by POX it. Some

components of POX support core functionalities that serve as either convenient features or examples. The list below shows some of these components:

- `forwarding.l2_learning`: This component of POX allows an OpenFlow switch to serve as learning switch and run in layer 2 (L2). This component is used to make the rules of flow as exact as possible. For example, it attempts to match on as many fields as possible. Therefore, distinct TCP connections will lead to distinct flow rules.

- `forwarding.l3_learning`: Since it is considered as an L3 device, this component is supposed to serve as a router. However, it is a L3-learning switch and, it is used to test the requests and responses of ARP.

- `forwarding.l2_multi`: This component is considered a kind of learning switches but with a further feature. Traditional learning switches identify connections on a switch-by switch basis, and decisions are made according to what is connected to these switches. `Forwarding.l2_multi` depends on `openflow.discovery` to recognize the network topology. Consequently, if one switch identifies the location of a MAC address is, the rest switches will also do, and will be able therefore to make decisions based on that.

- `openflow.discovery`: Link Layer Discovery Protocol is used by this component in order to discover the topology of network.

`openflow.spanning_tree`: information provided by `openflow.discovery` is used by this component in order to prevent loops in the network, similar to Spanning Tree protocol.

- `openflow.keepalive`: This component is used to refresh the connection, by periodically sending ECHO requests, in order to avoid disconnection of switches. The reason of this procedure is that

OpenFlow switches consider the idle case as a loss of connectivity, that is, they disconnect after a period of time the connection to the controller is idle.

- `proto.dhcpd`: It serves as a DHCP server, where DHCP addresses are leased out to clients.
- `misc.gephi_topo`: It is used to provide a visualization of detected hosts, links, and switches.

CHAPTER 6

Approach Design and Evaluation

6.1 L2 Learning Switch Method in SDN

In SDN, the POX controller can simulate the L2 learning switch by handling the broadcast packets in the same way of the legacy Ethernet networks. Initially, when a host needs to start a communication with another host, it broadcasts an ARP request asking for the MAC address of the destination host which is corresponded to its IP address. The first-hop switch receives that ARP request, encapsulate it inside **Packet-In** message, and send it to the controller using OpenFlow protocol. After that, the controller responds with a flow rule that will be encapsulated inside **Packet-Out** message to the switch forcing it to flood the broadcast packet to all ports except the receiving port. This process will be repeated with all switches in the network until it reaches the destination. If the spanning tree is enabled, the controller will instruct all the OpenFlow switches along the broadcast tree to forward the broadcast packets. In this case, there will be no redundant links and multi-path mechanism will not be supported in the network. In this research, the proposed approach will be compared and evaluated against L2 learning switch with spanning tree enabled in all experiments.

6.2 Proposed Approach

As mentioned earlier in Chapter 1, ARP and DHCP are the two major sources of broadcast traffic in Ethernet network. We propose a new approach to reduce such broadcast traffic with the use of SDN paradigm. A centralized SDN-based controller with a global view of the entire network is used to process all ARP and DHCP messages. Therefore, ARP and DHCP broadcasting to the entire network becomes unnecessary. In the proposed approach, we take into

account that the approach must meet all objectives mentioned in Chapter 1. The proposed approach is divided into three components, such that every component is responsible for handling a specific type of packet. These components are 1) ARP Handler Component that deals with ARP packets, 2) IP Handler Component that deals with IP packets, and 3) DHCP Handler Component that deals with DHCP packets. The following sections highlight how the aforementioned components work.

6.2.1 ARP Handler Component

In Ethernet network, communication between hosts absolutely depends on the hosts MAC addresses, for any host to establish communication with any other host, a host checks its local cache to confirm whether the destination MAC address is available, however a host broadcasts an ARP request to all hosts in the entire Ethernet network, if the destination MAC address is unavailable. In this SDN-based approach, the broadcasted ARP request will be intercepted by the first-hop switch and forwarded it to the controller. Once the controller receives ARP request, it asks the ARP Handler Component to deal with that packet. Once the ARP component receives the ARP packet, it checks the packet header to identify whether the packet is an ARP request or ARP replay. If the packet is identified as an ARP request, another checking process is required to identify whether the IP address of the source host (i. e requester host) is in the **Unavialable_Hosts_List**. The **Unavialable_Host_List** is a list that store the source host's first-hop switch along with the IP address of the requested unavailable host that is been requested repeatedly more than **N Threshold** times. **N Threshold** is the number of ARP requests sent per second where N is configurable variable. If the IP address of the source host is found in the **Unavialable_Hosts_List**, this means that its MAC address is requested prior to the current request but it was unavailable in the network. As a result of this, the controller deletes the source

host from the list and sends a command to the associated switches that request the MAC address of this source prior to the current request, to delete the flow, this will force the switch to allow any ARP request targeted for this particular host. Furthermore, the source host information in the **IP_To_MAC** table will need to be updated. If the host information is not already available and the source host is new, then a new entry will be created for that source host. The entry is the information about the source host's network configuration and location which is a pair of information key and a value. The information key is the IP address of the host while the value is a set of information which includes MAC address, Switch ID (dpid), Switch port, Last_Seen, and a Boolean variable called "checked" for confirming whether the host is already checked for its availability or not. Thus, in the case of the requester's information that already exists in the **IP_To_MAC** table then only the Last_Seen variable will be updated with the current time.

Another checking process is needed for detecting the suspicious ARP attack or any repeated ARP requests for an unavailable host. To achieve this, the proposed approach take into account how many ARP requests sent by a particular host per second. If the number of ARP requests exceeded the **N Threshold**, then the controller checks if the exceeded ARP requests are for one or multiple destinations. If these ARP requests are for multiple destinations, the controller considers the source as a suspicious host and adds it to the **Suspicious_List** which is a list that store information about the IP addresses and locations of all the suspicious hosts in the network. After that, the controller sends a command to the associated first-hop switch that is directly connected to that suspicious host to block it from sending further ARP requests by forcing the first-hop switch to drop all the ARP requests that come from this suspicious host. In the proposed approach, **DHCP Lease Time** is used as a blocking timeout period. **DHCP Lease Time** is the period of time that an IP address is allocated to the host and it is configurable by the network

administrator. The reason for choosing this period is that the suspicious host must be blocked as long as possible and setting any short blocking timeout period may cause the suspicious host to attack the network again immediately after the blocking period has expired. Likewise, **DHCP Lease Time** is the most suitable and the longest blocking period that can be set. Moreover, the IP of the blocked host may be assigned to a new host after DHCP Lease Time has expired. On the other side, if the ARP requests that exceeded the **N Threshold** are for one destination, the controller considers the destination host is unavailable and sends a probe message to the destination host to check its availability. The controller waits for a response from the destination host, if it does not replay within a specific period of waiting time, the controller sends a command to the first-hop switch that is directly connected to the source to block any ARP request between these two particular hosts (i.e source and unavailable destination host) by imposing the first-hop switch to drop all the ARP requests between these two hosts. This helps to avoid sending unnecessary further ARP requests that will lead to overhead in the network and the controller itself. Also, this blocking state will last for a period of **DHCP Lease Time**. After the **DHCP Lease Time** elapses for any blocked host, the controller deletes the host from the blocked list (**Suspicious_List** or **Unavailable_Host_list**) and asks the associated switch to unblock that particular host because its IP address could be reused by assigning it to a new host.

After checking the validity of ARP request against the **N Threshold**, next step is another checking process to confirm whether the destination information is available in the controller's database (**IP_To_MAC**) or not, if the destination information exists in the database, then checking the status of the entry is required. There are three states in checking the status of the entry which includes **Valid**, **Expired**, and **Waiting**. **Valid** state means the destination information is valid to be used, and in such a case, the ARP Handler Component constructs an

ARP reply with the destination information extracted from the database and forwards the ARP reply to the source host. **Expired** state means the destination is inactive for a long period of time (more than 5 Minutes) because ideally any host take 5 minutes to store mapping entry (IP, MAC) in its local cache, and its information needs to be updated in the controller database. A probe message will be sent to the destination to confirm whether the host still exists. The last state is the **Waiting** state which means the host has been in an **Expired** state and a probe message in, in this case, the controller will do nothing but to just wait for the ARP reply. For both **Expired** and **Waiting** states, if the controller does not receive a response within the waiting period time, it will delete the destination associated entry in its database and block all the ARP requests targeted for that destination as explained above. At the end of every state (**Valid**, **Expired**, and **Waiting**), request counter of the source is incremented by one, where the request counter controls how many times a source makes an ARP request. Moreover the request counter is being refreshed in every one second.

In such a case where the destination information does not exist in the controller's database, the controller will send a **Packet-Out** message that includes an ARP request, enforcing all the first-hop switches to broadcast the ARP request to all ports that are only connected to hosts (i.e edge broadcast). This type of partial flooding helps in reducing the broadcast traffic in the entire network and at the same time it prevents switching loops. To avoid broadcast repetition for the same destination from different sources, the controller creates a request queue for each requested destination with all of its associated requesters (source hosts that are waiting for an ARP Reply). Then, for every new source that requests the same destination that is already been requested by other hosts and had its own request queue, the controller only adds the new source to the request

queue without performing edge rebroadcast to all host. Once the controller gets the ARP reply, it will forward the ARP reply to every source in the request queue.

On the other hand, if the ARP packet is an ARP reply, the controller checks whether the host sending this ARP reply has a request queue that contains hosts that requested its MAC address, then the controller will first update the last_Seen variable for the ARP reply sender, and second forward the ARP reply to every source available in the request queue. However, the controller automatically drops that packet if the host that sent the ARP reply does not have request queue in the controller. Figure 14 shows the workflow for the ARP Handler Component.

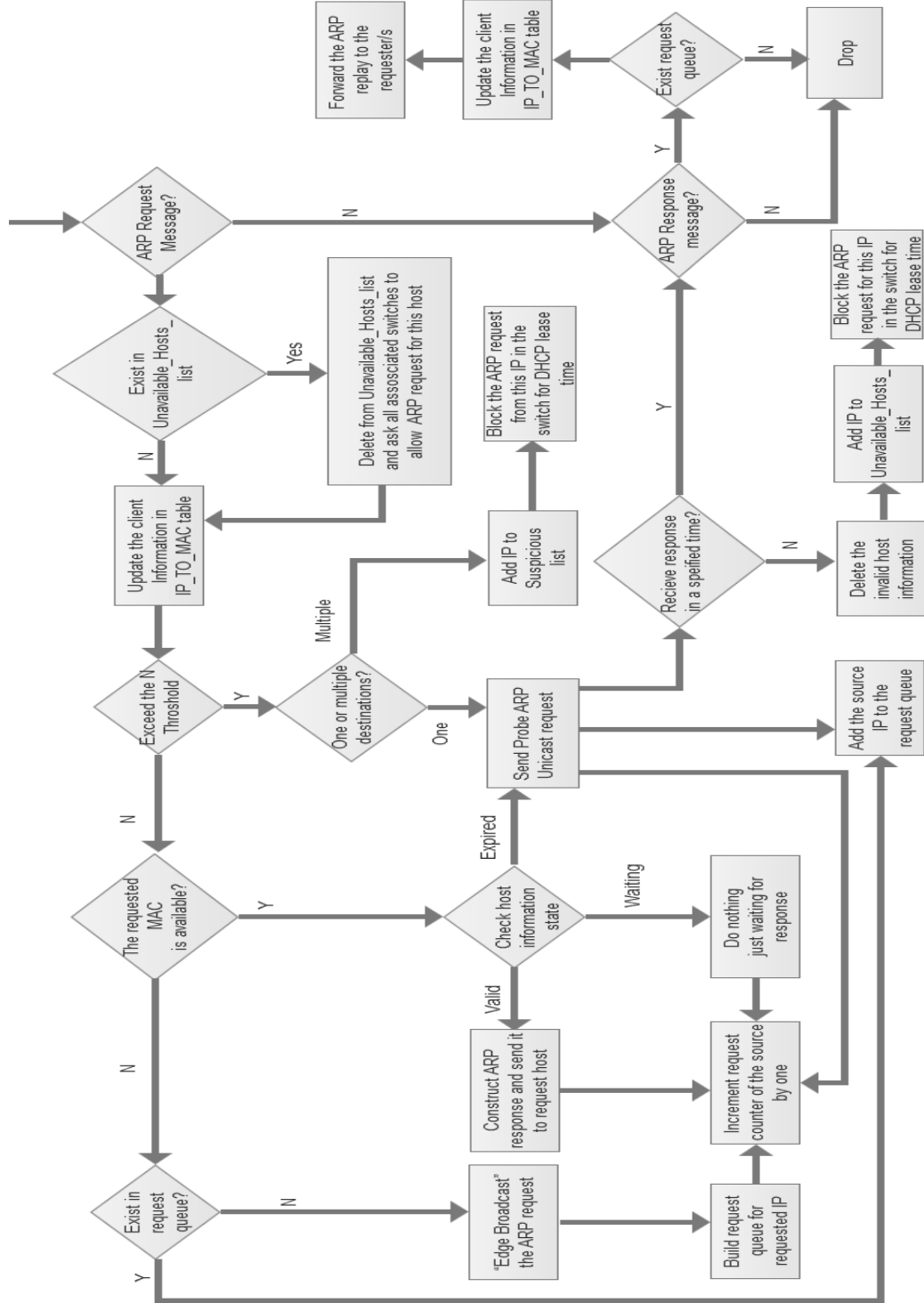


Figure 14 ARP Handler Component Flowchart

6.2.2 IP Handler Component

When an IP packet arrives at the first-hop switch, it will trigger a **Packet-In** event and send the IP packet to the controller. Once the controller receives the IP packet, it will direct that packet to IP Handler Component. To achieve multi-path load distribution among different routes between any two hosts, the proposed approach uses a procedure inspired by the method of Accumulative-Load Aware Routing (ALAR) [73]. ALAR is developed to work on the SDN-based architecture to enhance the utilizing of a network resource, minimizing the congestion, and reducing the end-to-end delay. ALAR depends on the knowledge of the entire network topology to sum the accumulative load of data flows on each link. With every new data flow installation, a certain amount of link's bandwidth is consumed. Therefore, the link with a lot of data flows will be ignored because it is more prone to be congested. In other words, the link with a small number of data flows should have a lower cost, thus it will be selected for installing the new data flow. To apply ALAR in the proposed approach, the controller needs to generate a topology graph which emulates the real network. It composes vertices and edges where vertices represent the switches and edges represent the links. This helps the controller to calculate the paths and its accumulative load easily from the topology graph without the need to test the real switches. The ALAR formulation for calculating the link cost is as follows:

$$C = \frac{B_{ref}}{B_L} \left(C_{init} + \sum_i C_i \cdot n_i \right)$$

Where i represent the different data flow types, C_i is the flow weight, n_i is the number of the flows of type i that passes through a particular link. Basically we have three types weights namely: **FTP** with weigh of 3, **HTTP** with weight ht of 2, and all other application protocols such as DNS, SNMP, POP and etc. are considered as **Other** with the weight of 1. For example:

let assume a link has 4 FTP and 5 HTTP installed flows and the weights $C_{ftp} = 3$ are $C_{http} = 2$ respectively, then the accumulative load on that link is $4*3 + 4*2 = 20$. C_{init} is a factor and configurable value that help in stabilizing the routing decisions over time, and it can be modified according to network condition. $\frac{B_{ref}}{B_L}$ is the fraction of link bandwidth which makes high bandwidth links to have lower cost compared with slow bandwidth links. Therefore, high bandwidth can transfer more data than low bandwidth links in the network.

When a **Packet-In** event is triggered by the first-hop switch, and the packet type is an IP packet, the controller will forward the IP packet to the IP Handler Component. First, the controller checks for the availability of destination IP in the **IP_To_MAC** table. If the destination IP does not exist, the controller will send **Packet-Out** to all first-hop switches imposing them to forward IP packet to all directly connected hosts. However, if the destination IP actually exists, the controller will extract the destination location information (dpid, port). In addition, the controller needs to know the application protocol for this communication, which can be (HTTP, FTP, or Other), to recognize the traffic weight cost. The controller figures out all the shortest paths between source and destination. Then, the best path with lower accumulative load is selected. The controller installs the corresponding flow rules to all the switches along the selected path. The installation of flow rules is performed in an inverse fashion, starting from the destination switch up to the source switch. This is useful for limiting the occurrence of **Packet-Ins** on switches that are close to the destination while a **Flow-Mod** is on the fly between the controller and these switches. Lastly, according to the type of the application protocol the controller updates all the links along the selected path with the corresponding weight. The dynamic flow expiration of idle timeout (period of inactivity) set to 10 seconds. Thus the installed flows

between any two communicated hosts are deleted after 10 seconds of the idle period. This helps to avoid re-installation of the flows within the period 10 seconds if there is a new communication from the same two hosts with the same application protocol. When a flow is deleted from a switch, the switch triggers **Flow-Removed** event to the controller telling it about the switch that removes the flow and the flow itself. This type of event helps the controller to update links in the selected path by subtracting this particular traffic weight from the total weights of all installed flows. Figure 15 shows the workflow for the IP Handler Component.

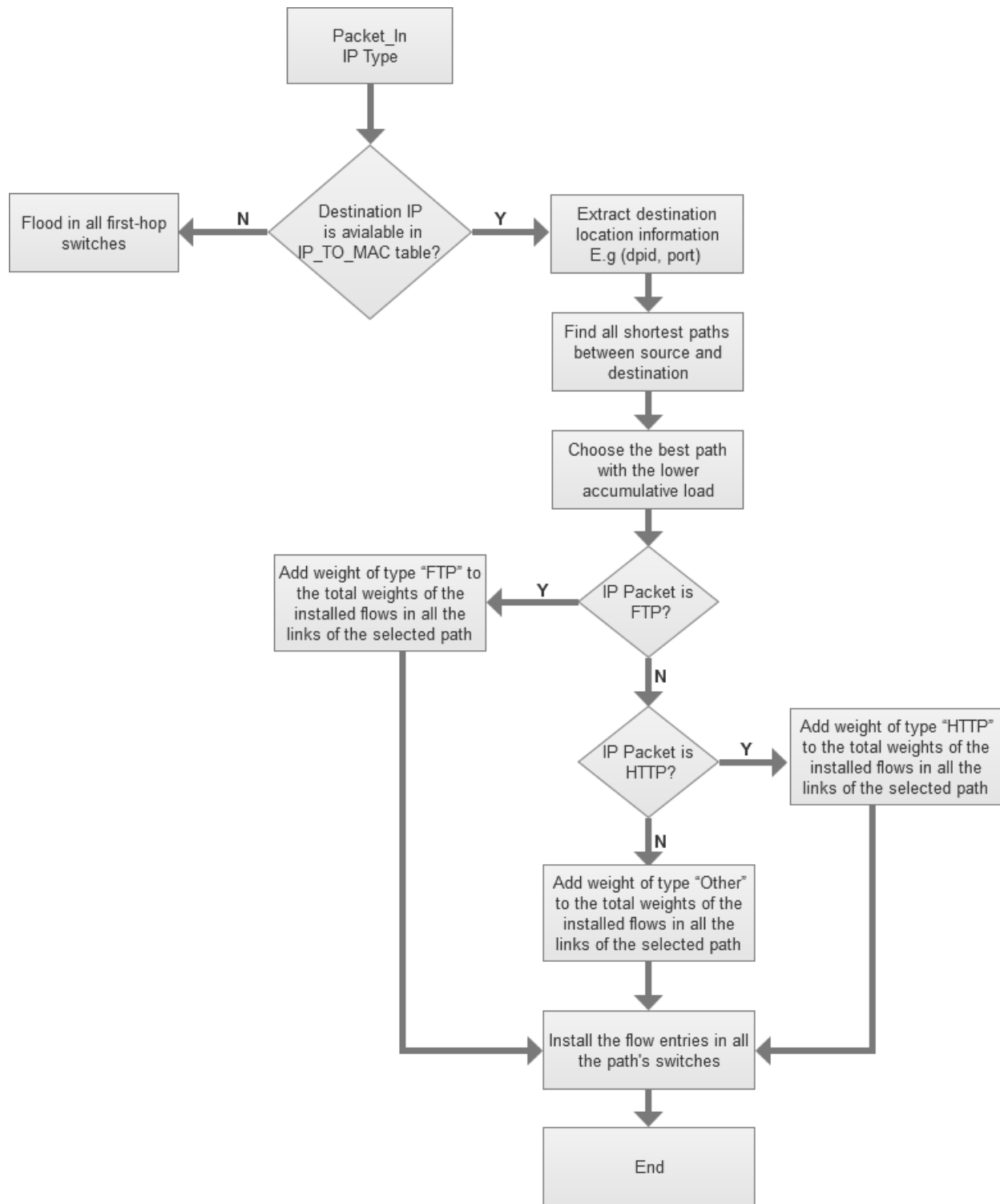


Figure 15 IP Handler Component Flowchart

6.2.3 DHCP Handler Component

DHCP is a protocol used to connect with a DHCP server to dynamically assign a host with an IP Address and other TCP/IP configuration information before connecting to the IP network. In DHCP protocol, the host initially broadcasts a **DHCP DISCOVERY** message to all the hosts in the network because the location of the DHCP server is not known to the host. Once the host that serves as DHCP server receives the **DHCP DISCOVERY** message, it replays with **DHCP OFFER** message containing an offered IP address for the host. After that, **DHCP REQUEST** and **DHCP ACK** messages come next and follow the same procedures.

In the proposed approach, when the host starts sending **DHCP DISCOVERY** message, asking for IP configuration, the first-hop switch receives that message and forward it to the controller through a secure channel encapsulated in a **Packet-In** message. The controller keeps the client location information in a temporary list, called **Requester_List**, in the form of (key, value) in which the key is the MAC address and the values is a pair of switch ID (dpid) and the switch port to which the host is connected. Then, the controller checks a predefined **DHCP_Server_List**, a list that stores all DHCP servers in the network, confirming whether there are existing DHCP servers, If not, the controller encapsulates the **DHCP DISCOVERY** message in **Packet-Out** message, and send it to all first-hop switches, asking them to send the **DHCP DISCOVERY** message to all directly connected hosts. In contrast, if the DHCP server list contains a number of servers, the controller will select one of them and forward the message to it. To achieve load balancing, the DHCP server selection will be in a round robin manner. Once the selected DHCP server receives the **DHCP DISCOVERY** message, it will reply with the configuration information in **DHCP Offer** message. The first-hop switch will get the message and forward it to the controller. Upon receiving **DHCP Offer** message, the controller will forward the message to

the client through **Packet-Out** message using the host location information that was already kept in the temporary list by hashing with the help of MAC address $H(\text{MAC}) \rightarrow (\text{dpid}, \text{port})$. Before forwarding the message, the controller has to check whether the DHCP server that sends the **DHCP Offer** message is available in the list. If the DHCP server is new, the controller will keep its IP and location information in **DHCP_Server_List** for further use. If there are more than one DHCP servers in the network, all will response **DHCP Offer**. However, the controller selects the first server reply, and saves the location information for all other servers in which they can be used in later time. Next the host MAC address is extracted from the **DHCP OFFER** message which is included in the **CHADDR** field of the message. The controller uses that MAC address to look up in temporary list for the location of the client $H(\text{MAC}) \rightarrow (\text{dpid}, \text{port})$, and then forward the **DHCP OFFER** to that location. Once the client receives the **DHCP OFFER** message, it will reply with **DHCP REQUEST** message telling the DHCP server that it accepts the offered IP address. Also, this message will be forwarded by the first-hop switch to the controller and the controller forward the message to the DHCP server after extracting the DHCP server location from **DHCP_Server_List** using hashing $H(\text{Server IP extracted from SIADDR field}) \rightarrow (\text{dpid}, \text{port})$. Lastly, the DHCP server sends a **DHCP ACK** message confirming that the IP assignment is successful. Before the controller forwards the **DHCP ACK** message to the client, it needs to extract the offered IP address from **YIADDR** field, and the MAC address from the **CHADDR** field and store them with the client location information (dpid, port) in the **(IP_TO_MAC)** table. After that, the controller deletes the information from the temporary list **(Requester_list)**, and forward **DHCP ACK** message to the client. In such a case where DHCP server sends **DHCP NACK** message, this means the IP assignment is unsuccessful; therefore, the controller deletes the information from the temporary **Requester_list**, and finally forward the

DHCP NACK message to the client. Figure 16 shows the workflow for the DHCP Handler Component.

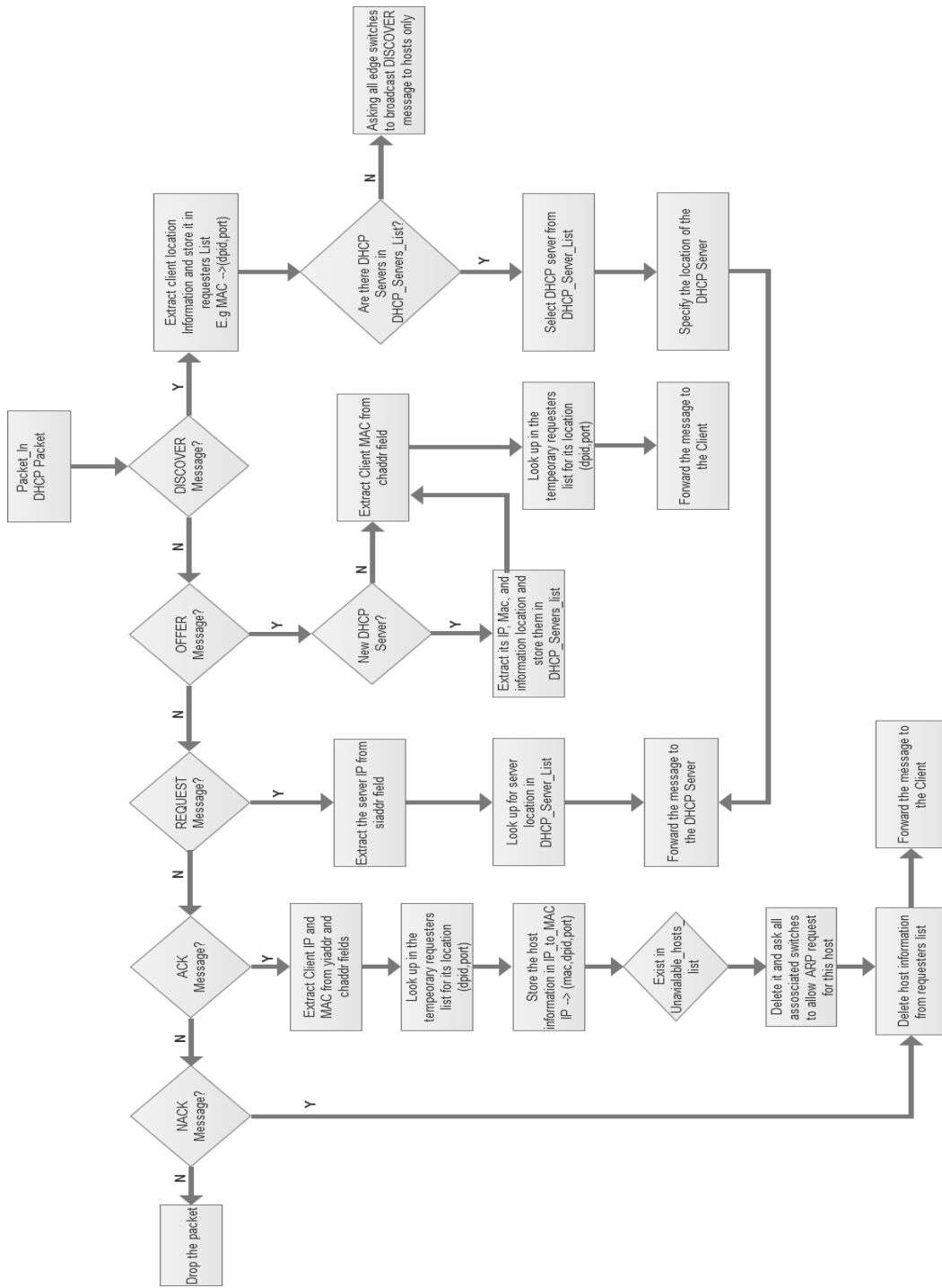


Figure 16 DHCP Handler Component Flowchart

6.3 Complexity Analysis

Let suppose we have a network with S OpenFlow switches, H hosts, and L_e Ethernet links. Then, to construct a spanning tree network, it will require $S - 1$ links. However, the number of redundant links will be $L_r = L_e - S + 1$.

Broadcast based on the traditional learning switches SDN makes control overhead over the controller by $O(S)$, because every switch forwards the control message to the controller. However, in the proposed approach, only the first-hop switches send the control messages to the controller which are only four types of messages: ARP request encapsulated in **Packet-In** message from the switch to controller, ARP request encapsulated in **Packet-Out** message from the controller to switch, ARP reply encapsulated in **Packet-In** message from switch to controller, and ARP reply encapsulated in **Packet-Out** message from controller to switch. Thus, the overhead over the controller will be minimized to $O(1)$.

Regarding the network traffic, every switch receives a new ARP request packet will send it to all ports that are part of the spanning tree network. Hence, there will be $S - 1$ copies for the switches and $H - 1$ copies for the hosts. However, the total amount of the network traffic will be $NT = H + S$ ARP requests. For DHCP broadcast messages, if we have N hosts in the network, then with L2 learning switch there will be $O(N)$ DHCP broadcast messages while in the proposed approach the DHCP broadcast messages will be suppressed to $O(1)$. Same with ARP messages, the rate of suppression will be reduced from $O(N)$ to $O(1)$.

The worst case with the proposed approach is when the controller does not have ARP information of the destination host. It needs to make edge broadcast in which the controller will ask all the access switches to broadcast the ARP request for hosts only. Therefore, the control

overhead will be $O(S_{Access})$, where S_{Access} is the number of access switches in the topology and the network traffic will become $NT = H + S_{Access}$.

6.4 Comparison between Different Network Approaches

According to the aforementioned research objectives, discussed in chapter 1, developing an approach that accomplishes all the objectives is difficult due to some of the objectives conflict with others. For instance, the proposed approach in FSDN [64] has a trade-off between eliminating the broadcast traffic totally and supporting hosts with static IP, since it achieves the former goal while it fails to achieve the later one. Another example, some approaches do not benefit from the property of redundant link because of the problem of a broadcast storm. Table 6 shows the comparison between some of the common existing SDN-based approaches that handle broadcast traffic. As it is noticed from Table 6, our proposed approach is the only one that deals with the broadcast traffic problem while satisfying all the objectives.

Table 6 Comparison between Different Approaches on Handling Broadcast Traffic

Approach	Save BW.	Self Conf.	Scalability	Security	All IP Conf.	Multi-path
Learning Switch with STP	No	Yes	No	No	Yes	No
Etherproxy	No	Yes	Yes	No	Yes	No
STATTLE	Yes	Yes	Yes	No	Yes	No
VL2	Yes	No	Yes	No	Yes	No
Portland	Yes	Yes	Yes	No	Yes	No
FSDM	Yes	Yes	Yes	No	No	No

PAST	Yes	Yes	Yes	No	Yes	No
Proposed Approach	Yes	Yes	Yes	Yes	Yes	Yes

6.5 Virtual Environment Setup

For our experiments, we used a Dell desktop with Intel(R) Xeon(R) 3.20 GHz, 8 cores CPU, and 24GB of RAM which uses Window 7 as host operating system. The host operating system runs two virtual machines (VMs). One VM works as a SDN controller which uses the Open-source POX controller. This VM deployed with specifications of (4 core processors, 2.5GHz, 12G RAM memory, and Linux operating system). The second VM used to build the testbed for the experiments, and the Mininet is used as emulator environment to create the virtual network. Mininet emulator runs a collection of end-hosts, links and switches on a single Linux kernel using lightweight virtualization. This VM deployed with specifications of (4 core processors, 2.5GHz, 12G RAM memory, and Ubuntu 14.04 operating system). The two VMs use NAT mode to connect to the guest host's Vmnet. Wireshark is installed to view OpenFlow messaging. Furthermore, Scrap library and Hping3 tool are used to generate the network traffic that is needed in some experiments. Table 7 summarizes the virtual environment specifications.

Table 7 Specifications for Virtual Environment

Type	Specification
VirtualBox version	4.3.20
Mininet version	2.2.0
Virtual OS	Ubuntu 14.04
Virtual RAM	12G
Virtual HDD	8G

Virtual NIC	NAT mode
No. of CPU's	4
Host OS	Windows 7
Switch Type	OpenVswitch

6.6 Network Topology

Data centers are usually structured in a Clos-like fashion consisting of three layers of switches. Starting from the root, we have the core, aggregation and access layers. The end hosts are connected to the access switches. Likewise, in this research we used Clos Topology as illustrated in Figure 17. The topology contains two core switches, three aggregate switches, and a number of access switches ranged from 10 to 25 according to the experiments needs. In addition, every access switch is connected to two hosts.

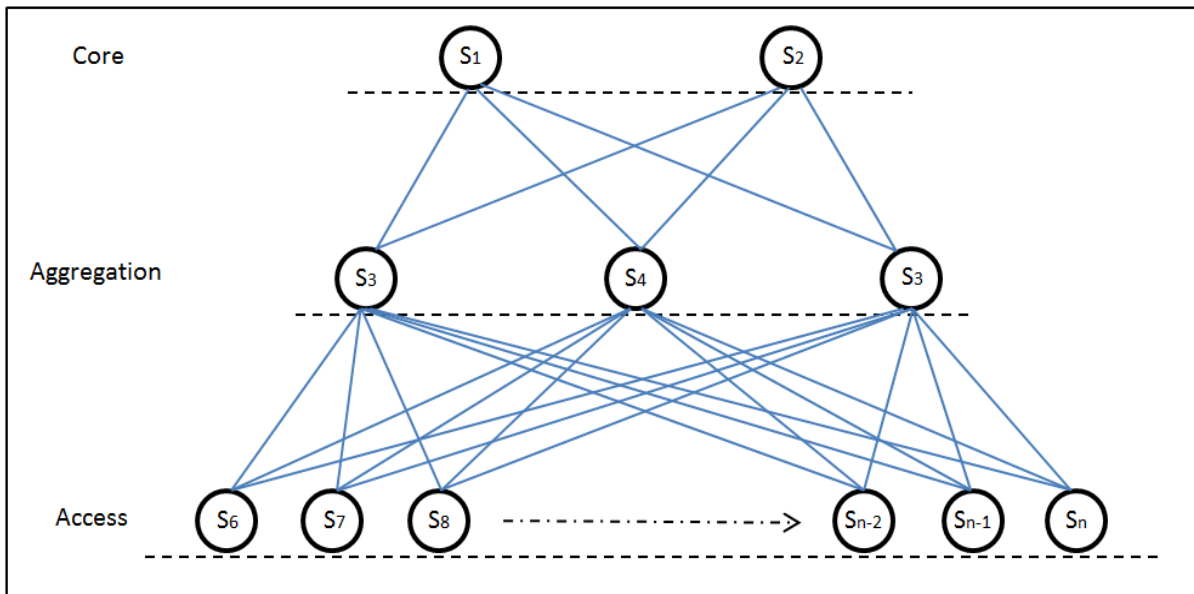


Figure 17 Clos Topology with 2 Core Switches and Fanout = 2

6.7 Results and Evaluation

In this section, a number of experiments and results are provided to demonstrate the performance of the proposed approach in comparison with L2 learning switch with spanning tree enabled approach. Furthermore, we show that our approach out-performs the L2 learning switch with spanning tree enabled approach in the following experiments.

6.7.1 Reduce ARP Traffic

For this experiment, we construct a tree topology with a depth of 3. The topology is divided into three levels of switches. The first level is the core which includes two switches, the second level is the aggregate which includes three switches, and the third level is the access switches which includes a range of switches (5, 10, 15, 20, and 25). The access level uses a different number of switches to see how the traffic is affected by the increase in the number of switches. Wireshark tool is used to capture the number of ARP packet copies generated in the network for every one original ARP request. Figure 18 shows that in our approach only two packets are generated per one ARP request which is the ARP request itself that sent by the host and the ARP replay which is constructed and sent by the controller. This is because the MAC address of the destination host that can be used to answer the ARP request is cached in the controller. Therefore, the controller does not need to send the ARP request to the destination host, but it uses its cached information to answer the request. However, in the traditional learning switch, the ARP request must be flooded to all the ports that are part of the spanning tree topology in which every host and switch must get a copy of the ARP request. Thus, there will be $S + H$ copies of the ARP request and the ratio of generated packet increases linearly in proportion to network scalability.

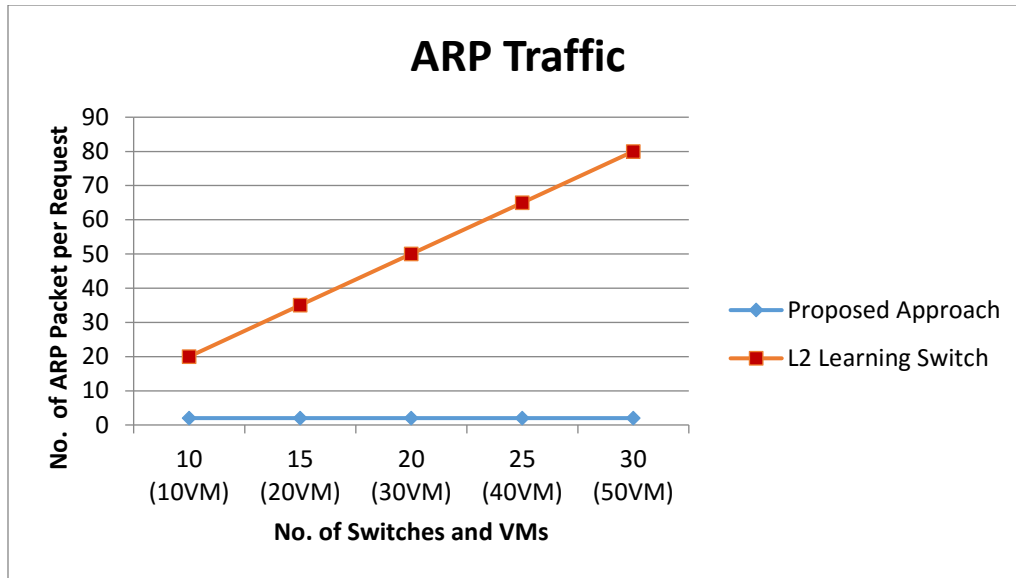


Figure 18 ARP Traffic

6.7.2 Reduce Control Traffic

Similar to ARP traffic experiment, the experiment here is applied on the topology of a depth 3 with three levels: core, aggregate, and access, and a range of switches from 10 to 30. One ARP request is sent, from one host asking for the MAC address of another host, to see the amount of control traffic that will be generated. The control traffic includes different OpenFlow messages such as **Packet-In**, **Packet-Out**, and **Flow-Mod** messages. Wireshark tool is used to monitor and count the quantity of those control traffic message. In Figure 19, it is clear that the proposed approach generates only two control messages which are one **Packet-In** message for sending the ARP request from the first-hop switch to the controller, and the second message is a **Packet-Out** includes the ARP reply that is encapsulated and sent by the controller. However, in the L2 learning switch with spanning tree enabled, every switch receives ARP request, it forwards it to the controller. The controller as a response, send a **Packet-Out** message telling the switch to flood the ARP request to all port that belongs to the spanning tree. This continues with all switches until the ARP packet reaches the destination host. Moreover, the controller sends

Packet-Mod to install rule flows in specific switches that are a part of a selected path between the source and destination. These rule flows inform the switches how to forward the ARP replay to reach the requester host. Hence, the amount of the controller traffic messages in the network is $O(S)$.

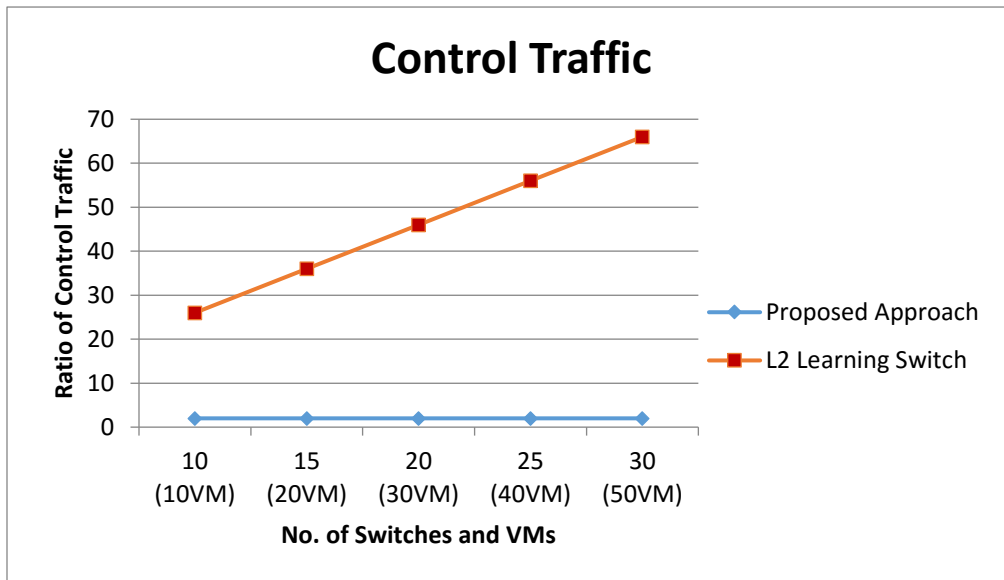


Figure 19 Control Traffic

6.7.3 ARP Response Time

In this experiment, the response time T is measured for a particular host by taking the difference $T = t_2 - t_1$ between the time t_1 when the ARP request is sent and the time t_2 when the corresponding ARP reply is received by the host. Figure 20 shows the average results over 10 individual measurements for the response time in both the proposed approach and the spanning tree-enabled L2 learning switch. There is a significant reduction of ARP response time achieved by the proposed approach, approximately from 57ms to 17ms. This happens because of the fact that, in the proposed approach, the controller immediately answers the ARP request with its cached information and no need to flood the ARP request in the entire network.

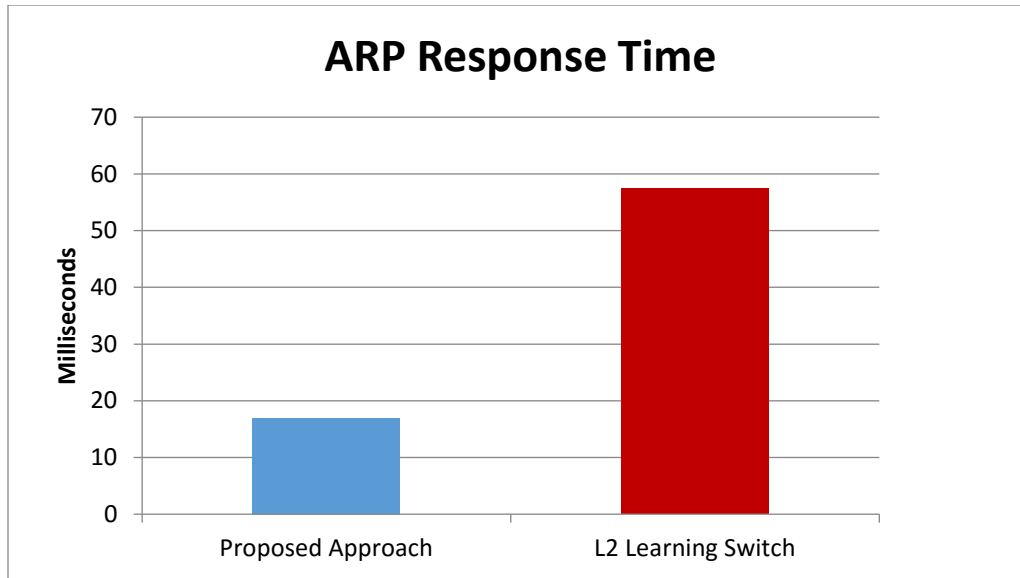


Figure 20 ARP Response Time

6.7.4 CPU Load

For this experiment, the topology with 30 switches is used. The measurement repeated 10 times with 10 different hosts and then the average of all the results is taken and plotted as it is obvious in Figure 21. In this experiment, the Scapy tool is used to generate background traffic in the network with different ARP request rates ranging from 50/s to 600/s. With the proposed approach, there is a slight increase in CPU load with all traffic rates where the load is between 2% and 8%. However, in L2 learning switch with spanning tree enabled, the CPU load is larger, and it increases almost linearly in proportion with the increase in the network traffic. This happens because for every ARP request, every switch in the network receives a copy of the ARP request and sends it inside **Packet-In** message to the controller, and receives **Packet-Out** message from the controller that tells the switch how to broadcast (how many and which ports) the ARP request. In addition, there is an overhead with calculating and installing the path for ARP reply.

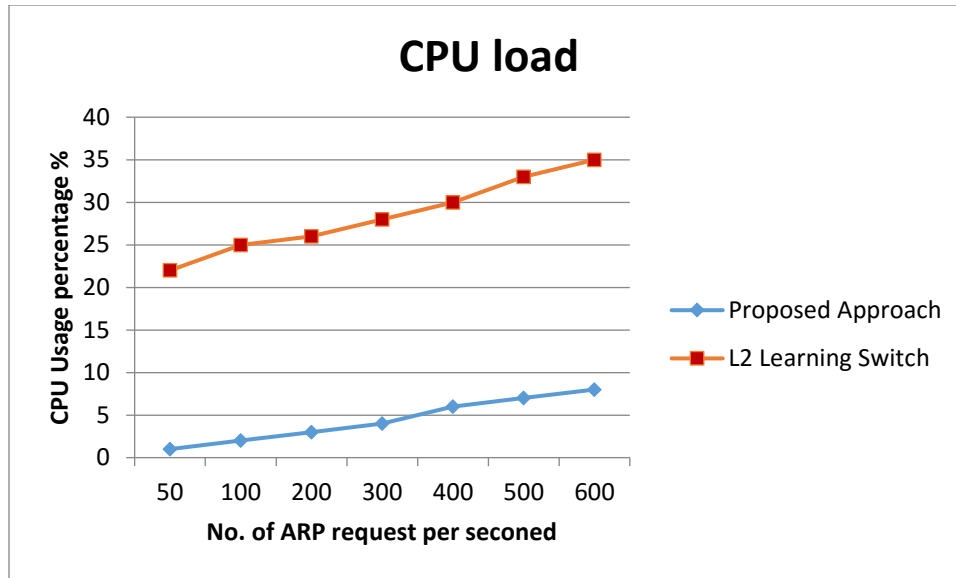


Figure 21 CPU Load

6.7.5 Ping Response Time

In this experiment, the Round Trip Time (RTT) is measured between 2 hosts and background traffic is injected between these two hosts. This action is repeated 10 times till the background traffic between the two hosts reaches 100Mbps. Hping3 is the tools that used to generate the background traffic and Ping program is used to measure the RTT between two hosts. Figure 22 illustrates the RTT delay from one host to another between the proposed approach and L2 learning switch approach. As shown in Figure 22, the delay for both approaches is almost the same at the first half of simulation time, but at the second half, the delay in L2 learning switch approach increases significantly. However, the proposed approach is kept at desirable values during all the measurement times. The main reason for this superiority of the proposed approach is that it uses multiple paths while the L2 learning switch approach uses a fixed path. In the last measurement time, the RTT of the proposed approach is 3ms while it reaches about 255ms in L2 learning switch approach.

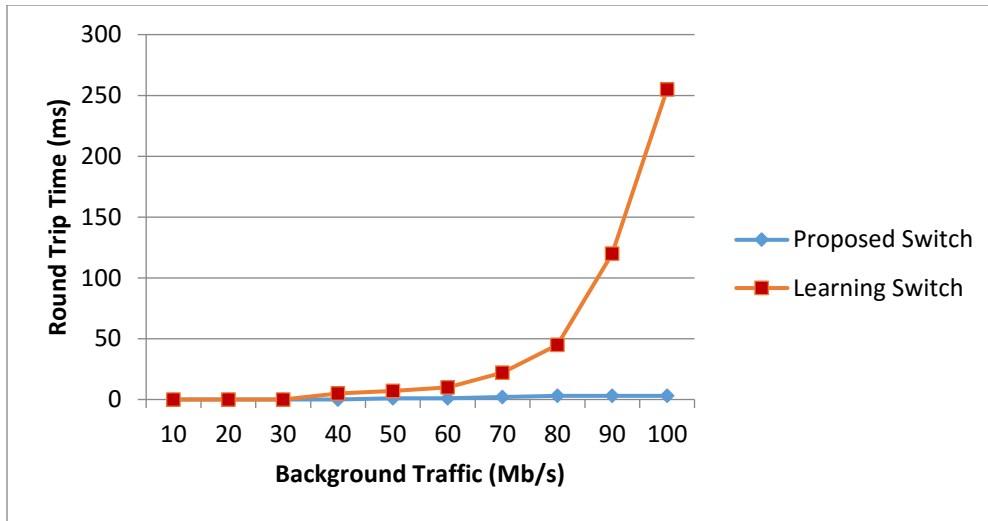


Figure 22 Ping Response Time

6.7.6 Blocking Timeout Periods

The proposed approach uses a mechanism to detect suspicious hosts, which attempt to cause network congestion with fake ARP broadcast packets, by counting how many ARP requests sent by a host per second. If any host exceeds the **N Threshold**, the number of requests sent per second where **N** is configurable, the controller considers it as a suspicious host and blocks it from sending further ARP requests. The blocking period can be either short or long. In the proposed approach, the two types are compared in order to choose the most ideal one. Short blocking timeout period is set to 5 minutes whereas the long blocking timeout period is set to **DHCP Lease Time**, that is, 8 hours which is the most used one.

Figure 23 shows results of the experiment conducted to measure the CPU load with two blocking timeout periods, 5 minutes and the **DHCP Lease Time** which is 8 hours. The experiment is conducted for each blocking type to measure the CPU load of 8 hours. The average CPU load for each hour of each blocking type is reported and plotted. In the experiment, 10 arbitrary hosts are chosen to work as suspicious hosts to send fake ARP requests with rate of 100 ARP requests per second and the **N Threshold** is set to 100 ARP packets per second. Scapy tool is used to

generate ARP requests and cause congestion in the network. Moreover, SAR Linux tool is used to measure the average CPU load. As shown in Figure 23, the average CPU load is slightly larger when using the 5 minutes blocking timeout period since the suspicious host flooded the network with fake ARP requests each time the 5 minutes expires before it is blocked again. However, Using **DHCP Lease Time** as blocking timeout period reported better results showing less CPU load since the fake ARP requests are sent only once before the suspicious hosts were detected and blocked (for 8 hours). Consequently, the network remained protected from fake requests flooding and traffic congestion is avoided. The average CPU load (for 8 hours) when applying 5 minutes blocking timeout period is 3.27, while it is 2.86 in case of **DHCP Lease Time** period.

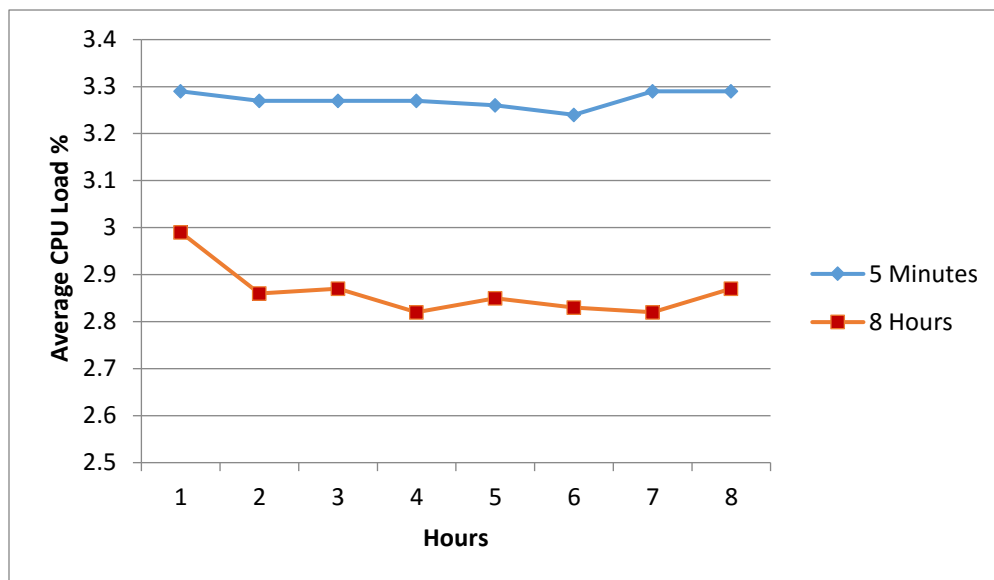


Figure 23 Average CPU Load

Figure 24 below shows the difference between the two blocking timeout periods, 5 minutes and **DHCP Lease Time** (8 hours) and the impact of each period on network traffic. The experiment measures the accumulative control traffic in the network for each hour during for the entire day (24 hours). In this experiment, a specific host is prepared to work as a suspicious host that sends

100 ARP request per second and the **N Threshold** is set to 100 ARP packets per second. According to the proposed approach, for every ARP request there will be two control packets. The first one is the **Packet-In** which is generated by the first-hop switch to send the ARP request to the controller once it receives the ARP packet from the hosts. The second packet is the **Packet-Out** which is used by the controller to answer the ARP request by encapsulating the suitable ARP response and send it to the first-hop switch, asking it to forward that ARP response to the requester host. As it shown in the Figure 24, the proposed approach reports better results when using **DHCP Lease Time** as a blocking time period since the network remains protected from fake requests flooding and traffic congestion is avoided. However, the network suffers from congestion when using 5 minutes blocking period since the suspicious host flooded the network with fake requests each time the 5minutes expires before it is blocked again. For example, after 8 hours the number of fake requests when using 5 minutes period is 19200 control packets while it is 200 when using **DHCP Lease Time** period. After 16 hours, the cumulative fake requests reach 38400 in case of 5 minutes period and 400 for **DHCP Lease Time** period. Eventually, the number of fake requests reaches 57600 after 24 hours for 5 minutes period and 600 in case of **DHCP Lease Time** period.

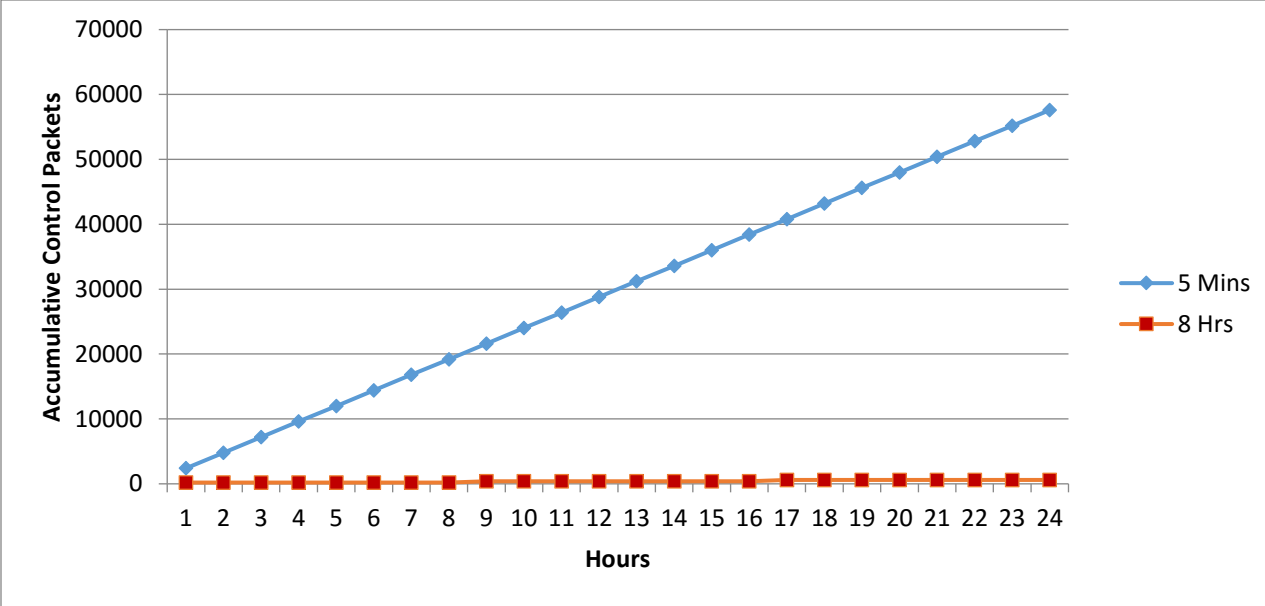


Figure 24 Accumulative Control Packets (24 hours)

CHAPTER 7

Conclusion and Future Work

Current Ethernet provides data transmission with high rates which gives an opportunity to expand the Ethernet from a small Local Area Network scope into large-scale networks. However, this is obstructed by the limitation in Ethernet scalability caused mainly by the broadcast traffic issue in some network protocols such as ARP and DHCP.

In this thesis, we studied the limitation of the traditional and currently proposed architectures that address the broadcast traffic issue. After that, SDN-based approach has been proposed to deal with this issue. A set of objectives has been suggested to be available when developing any beneficial approach that handles the broadcast traffic issue. Proposed approach satisfies all the objectives and simulation experiments shows that the proposed approach gives better results in minimizing broadcast with low data plane traffic and control plane overhead.

In future work, we plan to enhance the proposed approach to handle all well-known broadcast services and protocols. As this approach is designed mainly to work with OpenFlow-based switches, we will need to improve it to be able to work in environments that are composed of legacy switches and OpenFlow switches. Moreover, the proposed approach should be extended to deal with the IPv6 counterpart of ARP and multicast groups.

References

- [1] J. Menga, *CCNP practical studies: switching*. Cisco Press, 2003.
- [2] F. Benamrane, “Étude des Performances des Architectures du Plan de Contrôle des Réseaux ‘Software-Defined Networks,’” 2017.
- [3] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.
- [4] Á. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and L. J. García Villalba, “SDN: Evolution and opportunities in the development IoT applications,” *Int. J. Distrib. Sens. Netw.*, vol. 10, no. 5, p. 735142, 2014.
- [5] “Router vs Switch vs Hub: What’s the Difference? Webopedia.” [Online]. Available: https://www.webopedia.com/DidYouKnow/Hardware_Software/router_switch_hub.asp. [Accessed: 23-Feb-2018].
- [6] “Hubs Versus Switches—Understand the Tradeoffs,” p. 4.
- [7] “Extv3n3.pdf.” .
- [8] “Software-Defined Networking: The New Norm for Networks.” .
- [9] K. S. Sahoo, S. Mohanty, M. Tiwary, B. K. Mishra, and B. Sahoo, “A Comprehensive Tutorial on Software Defined Network: The Driving Force for the Future Internet Technology,” in *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, 2016, p. 114.
- [10] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, “Software-defined networking (SDN): a survey,” *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5803–5833, 2016.
- [11] “SDN Market to Experience Strong Growth Over Next Several Years, According to IDC,” 03-Feb-2016. [Online]. Available: <https://www.businesswire.com/news/home/20160203005954/en/SDN-Market-Experience-Strong-Growth-Years-IDC>. [Accessed: 22-Dec-2018].
- [12] S. Jain *et al.*, “B4: Experience with a globally-deployed software defined WAN,” in *ACM SIGCOMM Computer Communication Review*, 2013, vol. 43, pp. 3–14.
- [13] “Networking @Scale, May 2016 — Recap | Engineering Blog | Facebook Code.” [Online]. Available: <https://code.facebook.com/posts/1036362693099725/networking-scale-may-2016-recap/>. [Accessed: 15-Apr-2018].
- [14] “Oracle SDN Virtual Network Services Overview,” p. 18.

- [15] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 115–120.
- [16] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "Openradio: a programmable wireless dataplane," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 109–114.
- [17] K.-K. Yap *et al.*, "OpenRoads: Empowering research in mobile networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, 2010.
- [18] K.-K. Yap *et al.*, "Blueprint for introducing innovation into wireless mobile networks," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, 2010, pp. 25–32.
- [19] P. Dely, A. Kessler, and N. Bayer, "Openflow for wireless mesh networks," in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, 2011, pp. 1–6.
- [20] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, and L. Zeng, "OpenRAN: a software-defined ran architecture via virtualization," in *ACM SIGCOMM computer communication review*, 2013, vol. 43, pp. 549–550.
- [21] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 25–30.
- [22] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 7–12.
- [23] C. Guimaraes, D. Corujo, R. L. Aguiar, F. Silva, and P. Frosi, "Empowering software defined wireless networks through media independent handover management," in *Global Communications Conference (GLOBECOM), 2013 IEEE*, 2013, pp. 2204–2209.
- [24] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 127–132.
- [25] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International workshop on recent advances in intrusion detection*, 2011, pp. 161–180.
- [26] "Kinetic: Verifiable Dynamic Control." [Online]. Available: <http://resonance.noise.gatech.edu/>. [Accessed: 15-Apr-2018].

- [27] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, “FRESCO: Modular Composable Security Services for Software-Defined Networks.,” in *NDSS*, 2013.
- [28] M. Suenaga, M. Otani, H. Tanaka, and K. Watanabe, “Opengate on OpenFlow: system outline,” in *Intelligent Networking and Collaborative Systems (INCoS), 2012 4th International Conference on*, 2012, pp. 491–492.
- [29] H. Owens II and A. Durresi, “Video over software-defined networking (vsdn),” *Comput. Netw.*, vol. 92, pp. 341–356, 2015.
- [30] R. Mortier *et al.*, “Control and understanding: Owning your home network,” in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, 2012, pp. 1–10.
- [31] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou, “Instrumenting home networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 84–89, 2011.
- [32] N. Feamster, “Outsourcing home network security,” in *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, 2010, pp. 37–42.
- [33] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, “A survey of information-centric networking,” *IEEE Commun. Mag.*, vol. 50, no. 7, 2012.
- [34] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, “An openflow-based testbed for information centric networking,” in *Future Network & Mobile Summit (FutureNetw), 2012*, 2012, pp. 1–9.
- [35] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, “Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed,” *Comput. Netw.*, vol. 57, no. 16, pp. 3207–3221, 2013.
- [36] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, “Supporting information-centric functionality in software defined networks,” in *Communications (ICC), 2012 IEEE International Conference on*, 2012, pp. 6645–6650.
- [37] D. Syrivelis *et al.*, “Pursuing a software defined information-centric network,” in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 103–108.
- [38] X. N. Nguyen, D. Saucez, and T. Turletti, “Efficient caching in content-centric networks using OpenFlow,” in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, 2013, pp. 67–68.

- [39] J. Suh, H. Jung, T. Kwon, and Y. Choi, “C-flow: Content-oriented networking over openflow,” *Open Netw. Summit*, 2012.
- [40] H. Farhady, H. Lee, and A. Nakao, “Software-defined networking: A survey,” *Comput. Netw.*, vol. 81, pp. 79–95, 2015.
- [41] “MININET,” *Open Networking Foundation*. [Online]. Available: <https://www.opennetworking.org/mininet/>. [Accessed: 22-Dec-2018].
- [42] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, “Network simulations with the ns-3 simulator,” *SIGCOMM Demonstr.*, vol. 14, no. 14, p. 527, 2008.
- [43] A. Varga and R. Hornig, “An Overview of the OMNeT++ Simulation Environment,” in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ICST, Brussels, Belgium, Belgium, 2008, pp. 60:1–60:10.
- [44] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, “EstiNet openflow network simulator and emulator,” *IEEE Commun. Mag.*, vol. 51, no. 9, pp. 110–117, 2013.
- [45] “Trema.” [Online]. Available: <http://trema.github.io/trema/>. [Accessed: 14-Jan-2018].
- [46] C. Rotsos, R. Mortier, A. Madhavapeddy, B. Singh, and A. W. Moore, “Cost, performance & flexibility in openflow: Pick three,” in *Communications (ICC), 2012 IEEE International Conference on*, 2012, pp. 6601–6605.
- [47] A. Tootoonchian and Y. Ganjali, “HyperFlow: A distributed control plane for OpenFlow,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, pp. 3–3.
- [48] T. Koponen *et al.*, “Onix: A distributed control platform for large-scale production networks,” in *OSDI*, 2010, vol. 10, pp. 1–6.
- [49] N. Gude *et al.*, “NOX: towards an operating system for networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.
- [50] “The POX Controller.” [Online]. Available: <https://github.com/noxrepo/pox>. [Accessed: 20-Jan-2018].
- [51] “Ryu SDN Framework.” [Online]. Available: <https://osrg.github.io/ryu/>. [Accessed: 22-Dec-2018].
- [52] “Open MUL Foundation Home,” *Open MUL Foundation Home*. [Online]. Available: <http://www.openmul.org/>. [Accessed: 22-Dec-2018].

- [53] *The repository for the SNAC OpenFlow Controller. Contribute to bigswitch/snac development by creating an account on GitHub.* Big Switch Networks, 2018.
- [54] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, “OFLOPS: An Open Framework for OpenFlow Switch Evaluation,” in *PAM*, 2012, vol. 7192, pp. 85–95.
- [55] “Open Networking Foundation is an operator led consortium leveraging SDN, NFV and Cloud technologies to transform operator networks and business models,” *Open Networking Foundation*. [Online]. Available: <https://www.opennetworking.org/>. [Accessed: 20-Jan-2018].
- [56] K. Zarifis and G. Kontesidou, *Openflow Virtual Networking: A FlowBased Network Virtualization Architecture*. .
- [57] Kingston Smiler. S, *OpenFlow Cookbook*. .
- [58] O. TS-, “OpenFlow Switch Specification,” p. 206.
- [59] T. Limoncelli, “OpenFlow: A Radical New Idea in Networking - ACM Queue.” [Online]. Available: <https://queue.acm.org/detail.cfm?id=2305856>. [Accessed: 28-Sep-2018].
- [60] K. Elmeleegy and A. L. Cox, “Etherproxy: Scaling ethernet by suppressing broadcast traffic,” in *INFOCOM 2009, IEEE*, 2009, pp. 1584–1592.
- [61] A. Myers, E. Ng, and H. Zhang, “Rethinking the service model: Scaling Ethernet to a million nodes,” in *Proc. HotNets*, 2004.
- [62] C. Kim, M. Caesar, and J. Rexford, “Floodless in seattle: a scalable ethernet architecture for large enterprises,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 3–14, 2008.
- [63] M. Scott, A. Moore, and J. Crowcroft, “Addressing the Scalability of Ethernet with MOOSE,” in *Proc. DC CAVES Workshop*, 2009.
- [64] W. Jian, H. Tao, L. Jiang, and L. Yunjie, “A novel floodless service discovery mechanism designed for Software-Defined Networking,” *China Commun.*, vol. 11, no. 2, pp. 12–25, 2014.
- [65] M. Mahalingam *et al.*, “Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks,” 2014.
- [66] A. Edwards, A. Fischer, and A. Lain, “Diverter: A new approach to networking within virtualized infrastructures,” in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009, pp. 103–110.

- [67] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "CloudNaaS: a cloud networking platform for enterprise applications," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, p. 8.
- [68] A. Greenberg *et al.*, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, 2009, vol. 39, pp. 51–62.
- [69] R. Niranjana Mysore *et al.*, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Computer Communication Review*, 2009, vol. 39, pp. 39–50.
- [70] Sun, Xiaocui, and Zhijun Wang. "An efficient and scalable metro-Ethernet architecture." *Journal of Signal Processing, Image Processing and Pattern Recognition* 3.4 (2008).
- [71] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: a scalable multi-tenant network architecture for virtualized datacenters," in *ACM SIGCOMM Computer Communication Review*, 2011, vol. 41, pp. 62–73.
- [72] E. Rojas and I. G. Torii-HLMAC, "A distributed, fault-tolerant, zero configuration fat tree data center architecture with multiple tree-based addressing and forwarding," in *IEEE GLOBECOM*, 2012.
- [73] T.-T. Nguyen and D.-S. Kim, "Accumulative-load aware routing in software-defined networks," in *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, 2015, pp. 516–520.

Vitae

Name: |Abdulqader Ali Obaid Bahaj |

Nationality: |Yemeni |

Date of Birth: |2/11/1987|

Email: |abdulqader.bahaj@gmail.com|

Address: |Dhahran, Saudi Arabi |

Academic Background:|

- M.S in Computer Networks, Nov 2018 King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- B.S in Computer Information Systems, July 2010 Al-ahgaff University, Hadramout, Yemen.|

Publications:

- Bahaj, A. A., & Abouollo, A. M. (2017). Achieving Scalability with Data Owner Anonymity in Cloud Access Control. Transactions on Networks and Communications, 5(2), 01.
- Shaheen, A., Gaamel, A., & Bahaj, A. (2016). Comparison and Analysis Study between AODV and DSR Routing Protocols in VANET with IEEE 802.11 b. Journal of Ubiquitous Systems & Pervasive Networks, 7(1), 07-12.

Research Interests: Computer Networks, Virtualization Technology, and Cloud Computing.