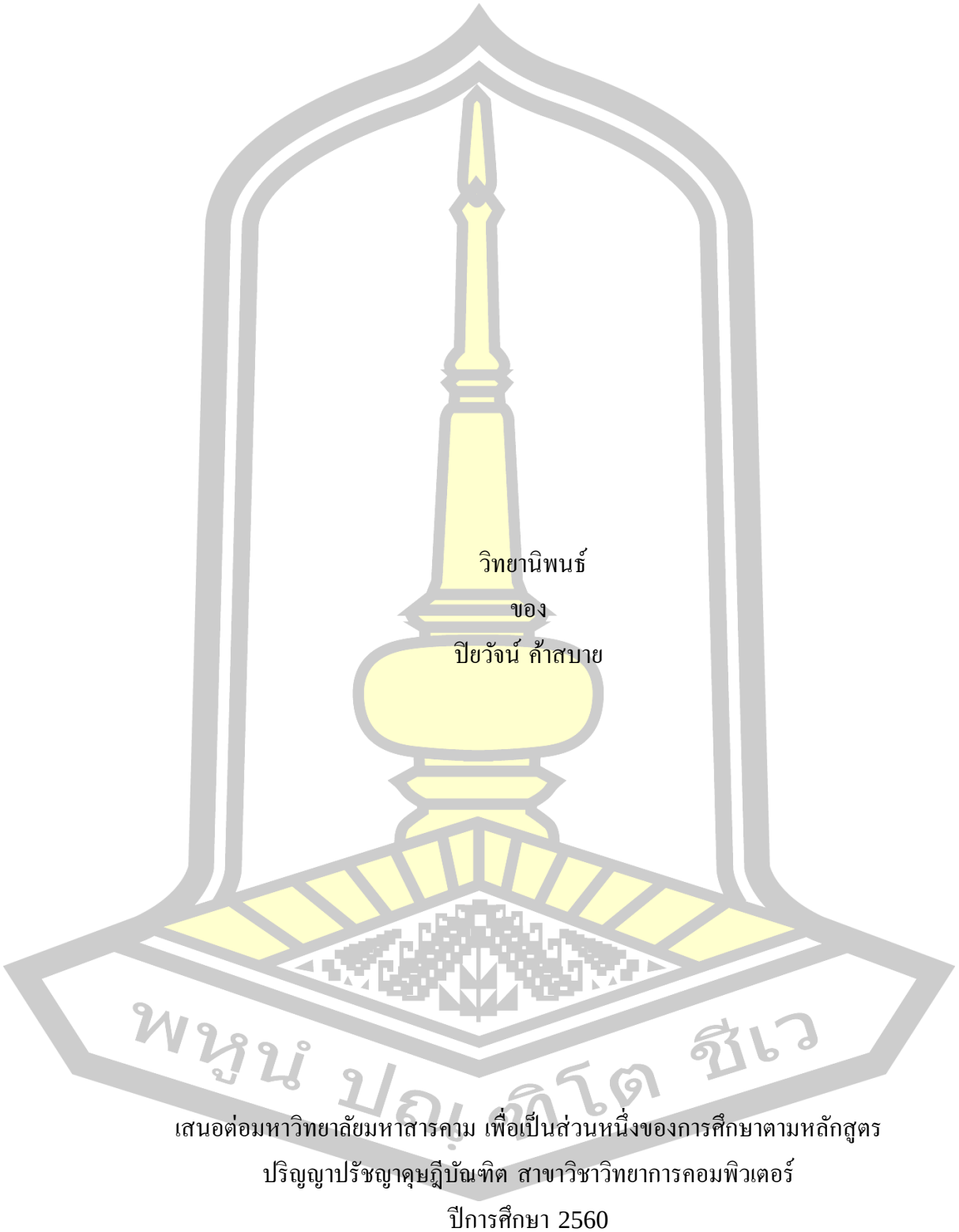
Priority-based Scheduling Policy for OpenFlow Control Plane

Piyawad Kasabai

A Thesis Submitted in Partial Fulfillment of Requirements for
degree of Doctor of Philosophy in Computer Science
Academic Year 2017

นโยบายการจัดกำหนดการตามลำดับความสำคัญสำหรับเพลนควบคุมโอเพนโฟว์



วิทยานิพนธ์

ของ

ปิยวัจน์ ค้าสบาย

เสนอต่อมหาวิทยาลัยมหาสารคาม เพื่อเป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาปรัชญาดุษฎีบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

ปีการศึกษา 2560

Priority-based Scheduling Policy for OpenFlow Control Plane

Piyawad Kasabai

A Thesis Submitted in Partial Fulfillment of Requirements

for Doctor of Philosophy (Computer Science)

Academic Year 2017

The examining committee has unanimously approved this Thesis, submitted by Mr. Piyawad Kasabai , as a partial fulfillment of the requirements for the Doctor of Philosophy Computer Science at Mahasarakham University

Examining Committee

..... Chairman
(Asst. Prof. Kornchawal Chaipah Ph.D.)

..... Advisor
(Somnuk Puangpronpitag , Ph.D.)

..... Committee
(Asst. Prof. Chatklaw Jareanpon , Ph.D.)

..... Committee
(Asst. Prof. Suchart Khummanee , Ph.D.)

Mahasarakham University has granted approval to accept this Thesis as a partial fulfillment of the requirements for the Doctor of Philosophy Computer Science

....

(Asst. Prof. Sujin Butdisuwan , Ph.D.) (Asst. Prof. Krit Chaimoon , Ph.D.)
Dean of the Faculty of The Faculty of Informatics Dean of Graduate School

Day..... Month... Year.....

TITLE Priority-based Scheduling Policy for OpenFlow Control Plane

AUTHOR Piyawad Kasabai

ADVISORS Somnuk Puangpronpitag , Ph.D.

DEGREE Doctor of Philosophy **MAJOR** Computer Science

UNIVERSITY Mahasarakham University **YEAR** 2017

ABSTRACT

Software Defined Networking (SDN) is a new network paradigm, allowing administrators to manage networks through central controllers by separating a control plane from a data plane. So, one or more controllers must locate outside switches. However, this separation may cause delay problems between controllers and switches. In this thesis, we therefore propose a Priority-based Scheduling policy for OpenFlow (PSO). The purpose of PSO is to give a higher priority for OpenFlow control messages in the in-band control network. Furthermore, PSO provides different priorities for OpenFlow control messages, based on contents/services (data traffic types) and/or customers for both in-band and out-of-band control network. The PSO is based on packet prioritization mechanisms in both OpenFlow switches and controllers. In addition, we have prototyped and experimented on PSO using a network simulator (ns-3). From the experimental results, the PSO can help the data flow with high priority acquire forwarding rules with lower delay under network congestion in control links (Normalized Load > 0.8), comparing to traditional OpenFlow.

Keyword : Software Defined Networking, OpenFlow, Switch-Controller Delay

ACKNOWLEDGEMENTS

The PhD thesis would not have been accomplished if without the help from several people. First of all, I would like to thank Dr. Somnuk Puangpronpitag for providing invaluable support throughout my Ph.D. program. I also thank to my colleagues in the Information Security and Advanced Network (ISAN) laboratory for a sharp discussion. I was very fortunate to have many friends both within and outside the Faculty of Informatics during my doctoral life. I thank them all for their being very supportive. I would also grateful to a few members of Distributed Systems & Services (DSS) research group (Leeds University, UK) for their comments and discussions. This thesis is partly supported by the Newton Mobility Grant (No: NI160138) from the UK's Official Development Assistance together with Office of Higher Education Commission (OHEC) Thailand, University of Leeds (UK), and Mahasarakham University (Thailand). I am also grateful to Prof. Karim Djemame for all of his supports, during six-month collaboration in Leeds (UK).

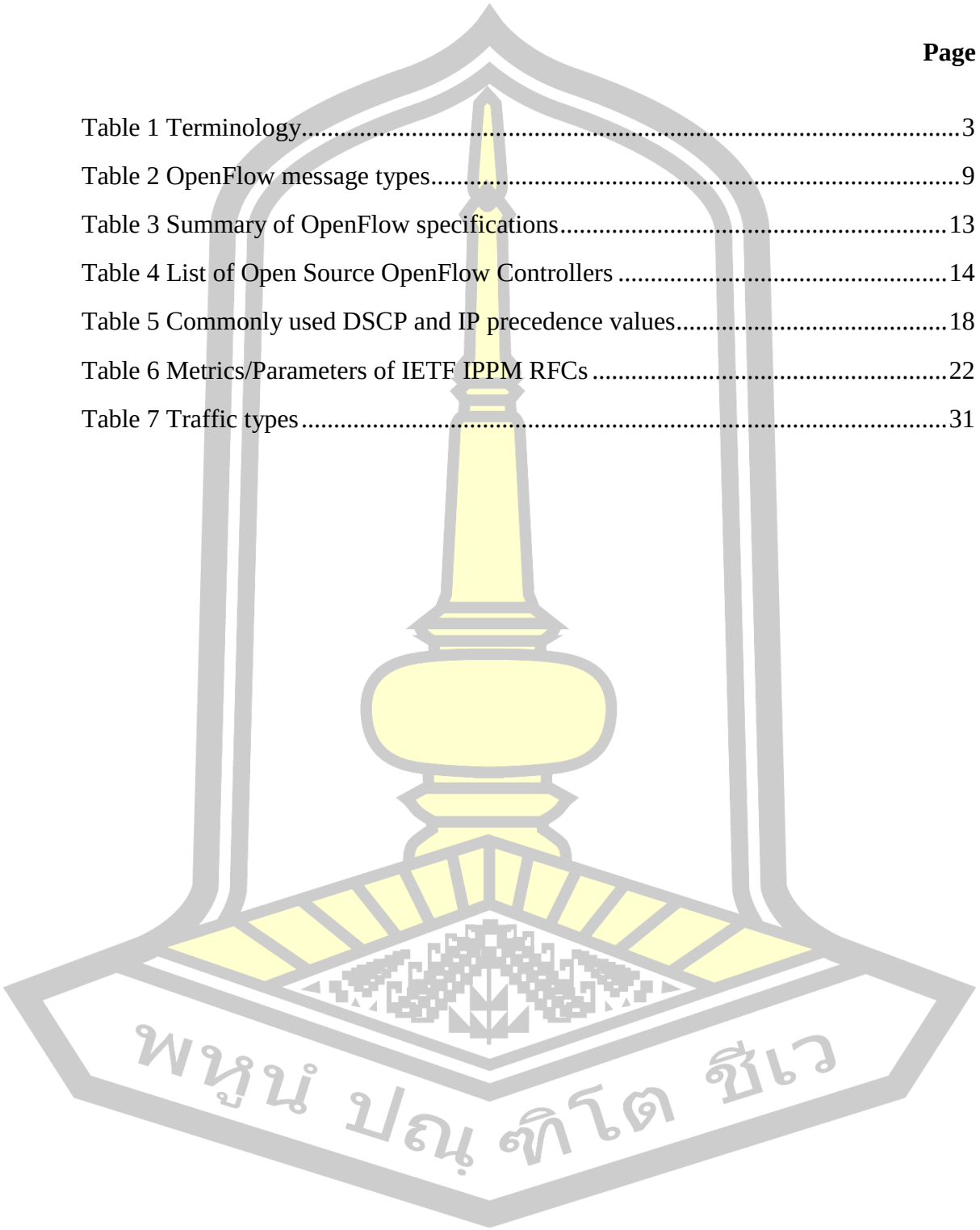
Piyawad Kasabai

TABLE OF CONTENTS

Page

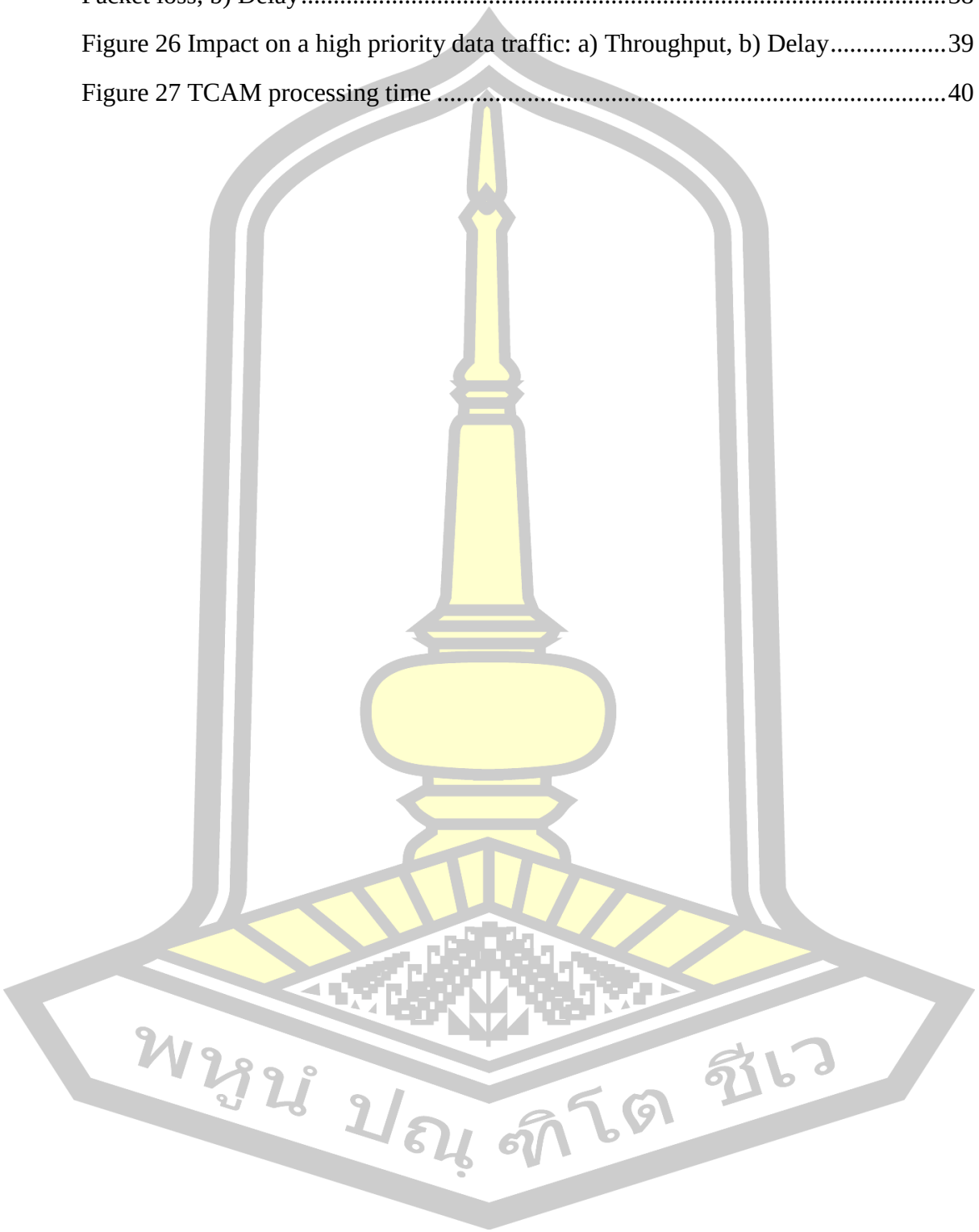
LIST OF TABLES

Page



LIST OF FIGURES

Page



CHAPTER 1

INTRODUCTION

1.1 Research Motivation

Traditional IP networks are complex and very hard to manage [1]. Classical switching and routing devices on the traditional IP networks are inflexible to optimize. These devices integrate both data plane and control plane on the same hardware device. So, the Software-Defined Networking architecture (SDN) [2] has been proposed to separate and operate between data plane and control plane. This architecture is designed to provide various perspectives in the programmable networks, such as manageability, scalability, and flexibility. So far, there have been several SDN-based solutions, such as SoftRouter [3], ForCES [4], and OpenFlow [5].

SDN defines some abstraction layers in computer networking. These abstraction layers separate a control plane from a data plane. The data plane locates in switch hardware, whereas the control plane is software running on one or more servers, called 'controller'. The data plane provides the simplest function of switches, i.e., forwarding packet according to a set of rules. The rules in the switch are managed by software at the controller. In general, several switches are controlled by a controller using an SDN protocol, such as OpenFlow.

OpenFlow has been widely deployed in various network products, and attracts several network industries [6]. The OpenFlow protocol defines control messages to handle a switch. The control messages may be sent on a separated network from the data traffic (called out-of-band control network), or may be sent on a shared network infrastructure with the data traffic (called in-band control network). Most of the scalability in OpenFlow network relates to the decoupling of the control and data planes. In particular, the first packet of a new flow is sent by a switch to the controller to acquire a forwarding rule. This may increase network load, and make the control plane a potential bottleneck [7]. In addition, since the flow tables of switches are configured in real-time by an external device, there is also the extra delay, introduced by the flow rule request process. Since SDN networks grow in scale and complexity, the control traffic may suffer from a delay, resulting in inefficient network as studied in [8].

Several solutions [8-12] have also been proposed to reduce the delay of the control traffic. However, some solutions support only a specific communication between the control and data planes (in-band or out-of-band control networks). The solution in [8] has proposed an initial design to fix this problem, but with a rather high overhead for traffic tagging.

Hence, this thesis proposes a Priority-based Scheduling policy for OpenFlow (PSO) to fix the aforementioned problems. The purpose of PSO design is twofold: 1) to overcome the bandwidth competition between data traffic and control traffic for the

in-band control network, 2) to provide high-priority OpenFlow packet-in messages of a specific traffic (such as real-time services) or a specific user for both in-band and out-of-band control networks.

1.2 Objectives

1) To analyze delay of acquiring forwarding rules in Software-Defined Networking (SDN)
2) To design solutions to help an OpenFlow message with shorter delay in acquiring forwarding rules in SDN
3) To evaluate the proposed solutions in terms of throughput and delay of a specific traffic by using network simulator

1.3 Key Contributions

This thesis presents delay problems in SDN, and proposes an enhanced design of the OpenFlow switch. Our design offers the following properties:

1) in-band and out-of-band support,
2) low delay for targeted traffic even under unstable situations,
3) scalability and feasibility to implement.

1.4 Scope

1) Among several SDN protocols, this work is based on OpenFlow protocol, which is the most widely used SDN protocols.
2) For communication between control and data planes, out-of-band control network will be evaluated in this work.
3) Distributed and centralized controllers are two SDN architecture types. This work mainly focuses on centralized controller. This centralized controller allows several switches to connect a single controller.
4) The performance evaluation of this work uses simulation techniques. ns-3 [13] will be used in this work.
5) Differentiated Service Code Point (DSCP) of IP header is originally deployed for Quality of Service (QoS) issues. However, this work takes the DSCP into the design mainly to mitigate the delay of higher priority traffic only, not covering all QoS parameters.
6) For network deployment, this work will mainly focus on managed networks, not covering public networks.

1.5 Terminology

This study uses terminology described in [8, 14, 15]. Table 1 gives the description of each term.

Table 1 Terminology

Term	Definition
Asynchronous message	A message sent by a switch to a controller without request
Control plane	A plane related to software application that places in a controller to control a data plane
Control traffic	Traffic related to OpenFlow messages such as Packet-in messages
Data plane	A plane related to data traffic that locates in hardware and is responsible for forwarding packets
Data traffic	Traffic related to data packets, such as ARP packets, TCP packets, UDP packets and so on.
Flow-mod message	A message sent from a controller to a switch to modify flow entry
OpenFlow message	OpenFlow protocol unit or packet unit, sent over OpenFlow connection
Packer-in message	A message sent to a controller in the event of table-miss
Packet-out message	A messages sent from a controller for a switch to specific output port or action
Software-Defined Networking (SDN)	A modern network architecture that separates data and control planes to allow administrators to program network functionalities via a controller connected with network devices
Synchronous message	A message sent by a switch to a controller with request

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Limitation of Traditional Networks

The Internet has been expanded to serve millions of users. So, it has become more and more complex. Manageability of network devices is a major issue of this complexity. To serve several applications, these network devices must be managed in different levels of services. However, almost all current network devices are inflexible and closed systems. A traditional network device (e.g. router/switch) provides a unified stack of functions [16]. From Figure 1a, there are three main traditional router/switch functions as follows. First, a specialized packet-forwarding function is responsible to accept data packets, and then forward to the next hop according to the configurations of the router or switch. Second, an operating system, provided by the router/switch's vendor, is responsible to control the device as a whole. This operating system can be proprietarily optimized by each vendor for each underlying hardware platform. Third, network applications (such as network protocols) provide rules, used in operating system and the packet-forwarding hardware. There are several standards underlying router/switch (approximately 6000 RFCs) to provide layer 2&3 network functions. It is rather complicated and inflexible to optimize network behaviors on these traditional routers/switches due to vendor-dependency.

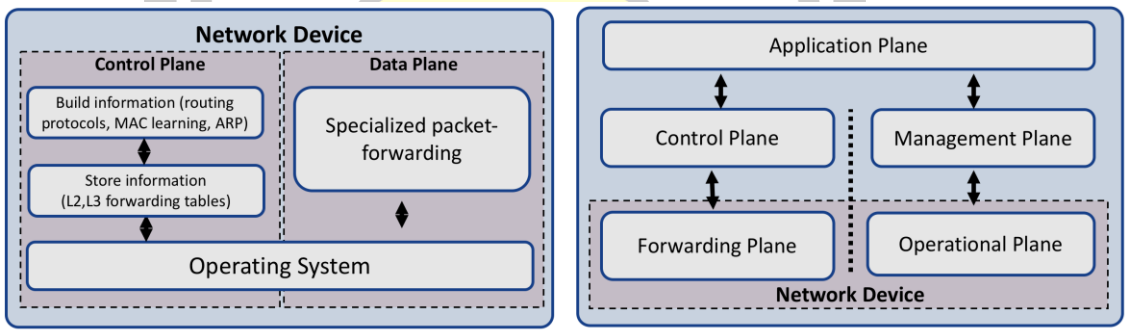


Figure 1 Traditional vs. SDN devices

A possible solution to this problem is the implementation of data handling rules as software rather than embedding them in hardware [17]. Routing Control Platform (RCP) [18] is one of the solutions, proposed by Caesar et al. RCP is a centralized platform, separating from data plane, to collect information about external destinations, and to select the BGP routes for each router. Recently, SDN is one of the most well-known solutions.

2.2 Software-Defined Networking (SDN)

Internet technologies have been developed to enable programmability. Active networks [19] were proposed to allow their users to inject customized programs into the network devices. The network devices extract and execute programs from injected data packets. In this solution, a new routing mechanisms and network services can be implemented without the modification of the network hardware. However, security and performance issues can be problems due to injection malicious programs into packets from attacker and executing injected-malicious programs [20]. So, this solution has not convinced.

Programmable networks [21] were developed to provide programmability in the network by allowing programs to execute network devices similar to active networks. However, the programs are not injected in data packets as with active networks. So, security issues on the programmable networks can be achieved.

Both active networks and programmable networks introduce a new network paradigm (named Software-Defined Networking: SDN). SDN has emerged from a research work, initially performed in 2004 as a part of researching a new network management paradigm [16]. This initial work was built in 2008 by two different groups. A company, named *Nicira Networks*, has created a network operating system, named NOX [22]. At the same time, an OpenFlow switch has also been created by the cooperation between *Nicira Networks* and a research team from Stanford University. OpenFlow is then widely supported by network industries [6]. Now, OpenFlow is managed by Open Networking Foundation (ONF) [6].

SDN is a new approach for network administrators to manage network functionality and provision. It provides software to program network devices dynamically. SDN focuses on roles of software in running networks through an abstraction of a data plane, and separating it from a control plane. This separation allows faster innovation cycles at both planes [14]. Several ideas and concepts are applicable to research and development in SDN standardization [23]. The Software-Defined Networking Research Group (SDNRG) [24] was chartered by Internet Research Task Force (IRTF) to investigate SDN from various perspectives with the goal to identify the approaches that can be defined, deployed and used in the near term as well as to identify future research challenges [14].

RFC 7426 [14] has described the layers and architecture of SDN. As shown in Figure 1b, SDN architecture consists of multiple planes, including Forwarding Plane (FP), Operational Plane (OP), Control Plane (CP), Management Plane (MP), and Application Plane (AP).

FP is widely referred to the "data plane" or the "data path". The FP is responsible for handling packets in the data path, based on the instructions received from the CP. The FP takes action from the instructions, such as forwarding, dropping, and changing packets. OP is responsible for managing operational states of the network device such as device active/inactive, port status, and port available. The OP relates to network

device resources (e.g. ports, memory), and it is usually the termination point for MP and AP. CP is responsible for making instructions sent to FP on how packets should be dropped or forwarded by one or more network devices. The CP may be related in OP information e.g., current port status or its capabilities. MP is responsible for monitoring, configuring, and maintaining network devices. The MP makes decision regarding the states of network devices, and may be used to configure the FP. For instance, the MP may set up all or parts of the forwarding rules at the first time. AP is a part of applications and services that define network behaviors.

The communication between FP and CP is provided by southbound APIs. OpenFlow protocol and OF-Config are used these APIs. Northbound APIs are used to establish CP and AP. These APIs enable innovative applications. REST are the most used northbound APIs and most of the controllers implement it.

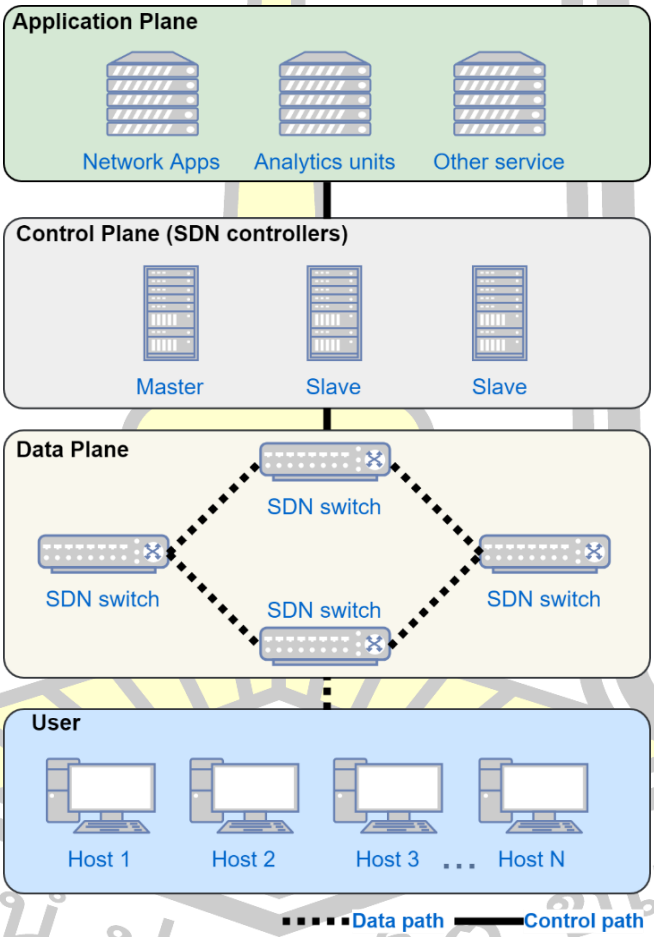


Figure 2 Overview of SDN

In general, SDN decouples control plane and data plane (as illustrated in Figure 2). The control plane is actually RFC 7426's CP together with MP, while the data plane is RFC 7426's FP together with OP. In addition, the controller may collect any information from analytic engine at the application layer or application plane (optional). The lower layer is the control plane, also called the network infrastructure layer. It

consists of forwarding network devices. The responsibilities of this plane are mainly data forwarding, monitoring and gathering statistics.

2.3 Communication between Control and Data Planes

To communicate between control and data planes, there are two alternatives, namely in-band control network and out-of-band control network. In the out-of-band control network, control traffic (an OpenFlow control message) is sent on a separate network from data traffic, whereas the control and data traffic share the same network link for the in-band control network. From the literature, the out-of-band control network has been focused by several studies [25]. It is also used by B4 (Google Software Defined WAN) [26]. Its advantages are as follows: 1) high security can be provided for control messages; 2) high availability can be provided even if there are failures in the data plane. However, this out-of-band control network is expensive to build due to the separation of network link. Sharma et al. [9, 10] have argued that the in-band control network would be more widely deployed since it is suitable for all types of topologies.

Centralized and distributed controllers are two alternatives for the SDN controller placement. For the distributed controllers, inter-connection links among controllers [27] must be built. Russ and Shawn [28] have suggested that the distributed controllers are rather complex and require heavy configuration to deploy, design, and manage. On the other hand, the centralized controller is much simpler. So, the centralized controller is more widely deployed comparing to the distributed controllers. However, SDN can grow in scale, and the number of the switches under the same centralized controller could be increased. This issue inevitably causes a network congestion problem [8].

In general, both centralized and distributed controllers can cause delay problems, namely inbound and outbound delays. The inbound delay (or inbound latency) happens when a switch generates packet-in messages and sends them to a controller. The outbound delay (or outbound latency) happens when a controller generates packet-out messages and sends them to a switch in order to install, modify, delete, and forward rules. He et al. [11] have found that both delays could result in the inefficiencies of the link between the switch and the controller.

For the above reasons, network inefficiencies may occur in SDN, and the delay of data traffic is then increased. For the in-band control network, this problem can be even more severe since the control traffic may be dropped, resulting in the failure of data traffic forwarding. In addition, there is still no explanation how the switch and the controller can prioritize different traffic types (data and control packets). Some previous studies [8-10, 12] are proposed to figure out and solve this problem, but all of them still have some drawbacks that will be further discussed in Section 2.10.

2.4 OpenFlow Switch

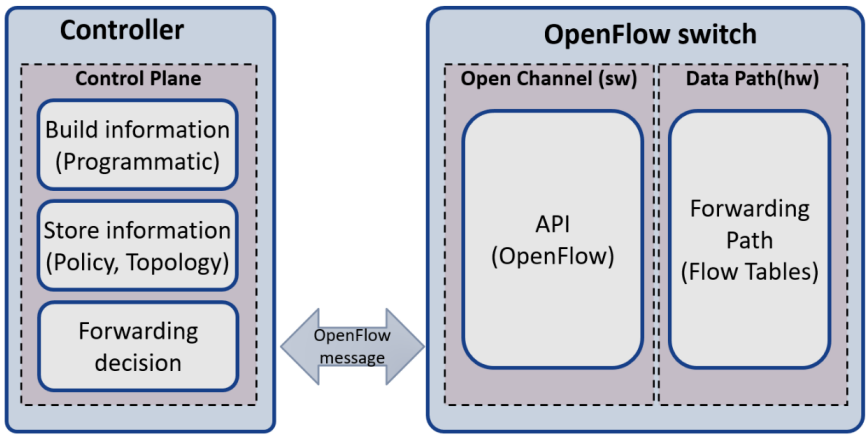


Figure 3 OpenFlow switch main components

An OpenFlow switch forwards data packets according to a set of rules in flow tables. These rules are managed by a software-based controller at the control plane outside the switch (as shown in Figure 3). The OpenFlow switch consists of *secure channel (open channel), flow table and OpenFlow protocol*. The *secure channel* is a software API to connect with the controller, allowing commands and packets to communicate between the controller and the switch. The *flow table* is built in the switch hardware using Ternary Content Addressable Memory (TCAM). It contains a list of flow entries, which define the rules for forwarding/dropping packets. Each flow entry consists of *match fields*, *counters* and *instructions*. For each incoming packet, the packet header is compared to the *match fields* of each entry. If matched, the packet is processed according to the *instructions*. The *counters* are used to collect statistics about the packets. The *OpenFlow protocol* provides a standard for communication between controllers and switches by defining control traffic between them. Each control traffic has the following header structure (shown in Figure 4)

version (8bits)	type (8bits)	length (16bits)
xid (32bits)		

Figure 4 OpenFlow header

buffer_id (32bits)		
total_len (16bits)	reason (8bits)	table_id (8bits)
cookie (64bits)		
match (TLVs)		
pad	data	

Figure 5 OpenFlow packet-in message format

The structure of the OpenFlow header is described in Figure 4, consisting of four fields, namely *version*, *type*, *length*, and *xid*. *version* specifies the version number of the OpenFlow protocol. There are several categories of OpenFlow messages (as shown in Table 2), such as symmetric messages, switch configuration messages, asynchronous messages, and controller command messages. Each category contains different types. For example, *OFPT_PACKET_IN* is a type of asynchronous messages, describing packet-in messages (as shown in Figure 5). The *length* field indicates the total length of the message. The *xid* indicates transaction ID, associated with the packet. The *types* can have the following values, as shown in Table 2.

Table 2 OpenFlow message types

Type	**Value**
Symmetric messages	
OFPT_HELLO	0
OFPT_ERROR	1
OFPT_ECHO_REQUEST	2
OFPT_ECHO_REPLY	3
OFPT_EXPERIMENTER	4
Switch configuration messages	
OFPT_FEATURES_REQUEST	5
OFPT_FEATURES_REPLY	6
OFPT_GET_CONFIG_REQUEST	7
OFPT_GET_CONFIG_REPLY	8
OFPT_SET_CONFIG	9
Asynchronous messages	
OFPT_PACKET_IN	10
OFPT_FLOW_REMOVED	11
OFPT_PORT_STATUS	12
Controller command messages	
OFPT_PACKET_OUT	13
OFPT_FLOW_MOD	14
OFPT_GROUP_MOD	15
OFPT_PORT_MOD	16
OFPT_TABLE_MOD	17
Multipart messages	
OFPT_MULTIPART_REQUEST	18
OFPT_MULTIPART_REPLY	19
Barrier messages	
OFPT_BARRIER_REQUEST	20
OFPT_BARRIER_REPLY	21
Controller role change request messages	
OFPT_ROLE_REQUEST	24
OFPT_ROLE_REPLY	25
Asynchronous message configuration	
OFPT_GET_ASYNC_REQUEST	26

Type (cont.)	Value (cont.)
OFPT_GET_ASYNC_REPLY	27
OFPT_SET_ASYNC	28
Meters and rate limiters configuration messages	
OFPT_METER_MOD	29
Controller role change event messages	
OFPT_ROLE_STATUS	30
Asynchronous messages	
OFPT_TABLE_STATUS	31
Request forwarding by the switch	
OFPT_REQUESTFORWARD	32
OFPT_BUNDLE_CONTROL	33
OFPT_BUNDLE_ADD_MESSAGE	34
Controller status asynchronous message	
OFPT_CONTROLLER_STATUS	35

When data packets are received at a switch, the switch uses an OFPT_PACKET_IN message (as shown in Figure 5) to manage these data packets. This message includes OpenFlow header, containing eight fields, namely *buffer_id*, *total_len*, *reason*, *table_id*, *cookie*, *match*, *pad*, and *data*. The *buffer_id* is used to identify a buffered packet, associated with the previous packet-in message. The *total_len* is the full length of the packet. The *reason* field indicates the context of the packet-in message. This filed can also contain other messages, such as output to a controller in apply-actions, invalid TTL. The table_id is ID of the flow table for looking up. The *cookie* field contains the cookie of the flow entry, caused the OpenFlow message to be sent to the controller. The *match* field is a set of OpenFlow Extensible Match (OXM) Type Length Values, containing data packet's header (pipeline) fields associated with the packet. The *pad* field is additional padding. This pad is set even if the data field is empty. The *data* field is a part of data packet that associates with a packet-in message such as Ethernet frame.

buffer_id (32bits)	
actions_len (16bits)	
pad (64bits)	
actions (TLVs)	

Figure 6 OpenFlow packet-out message format

The OpenFlow packet-out message (OFPT_PACKET_OUT) is illustrated in Figure 6. The *buffer_id* and *pad* fields are the same given in the packet-in message. The *actions_len* is the size of action array in bytes. The *actions* field contains an action list defining how the packet should be processed by the switch. The field may include an output port, packet modification, group processing.

The OFPT_FLOW_MOD message (as shown in Figure 7) is used to modify flow entry in the flow tables. The *command* field is used to specify the context of this message i.e., new flow, modify all matching flows, modify entry, delete all matching flows, delete entry. The *idle_timeout* and *hard_timeout* fields indicate how quickly flow

entries expire. The *priority* indicates priority within the specified flow table. The *out_port* and *out_group* are optional fields (more details can be found in OpenFlow specification [20]).

buffer_id (32bits)		
cookie (64bits)		
cookie_mask (64bits)		
table_id (8bits)	command (8bits)	idle_timeout (16bits)
hard_timeout (16bits)		priority (16bits)
buffer_id (32bits)		
out_port (32bits)		
out_group (32bits)		
flags (16bits)		importance (16bits)
match (TLVs)		

Figure 7 OpenFlow flow-mod message format

For the IP network, each OpenFlow message is encapsulated and sent over TCP at the default port no. 6653. It is usually encrypted using Transport Layer Security (TLS) (as shown in Figure 8).

OpenFlow
TLS
IP
Ethernet

Figure 8 OpenFlow over Internet Protocol

The *flow tables* of the OpenFlow switch are deployed to operate the incoming data packets. For example, when data packets arrive at the switch, the switch then inspects each packet's header and tries to match it with a flow entry in the flow tables. If matched, the switch then takes the action of *instructions* in that flow entry. If the header of the packet is not matched with any flow entry, this case is called *table-miss* event. The packet is then encapsulated into an OpenFlow packet-in message. After that, the switch sends the packet-in message to the controller to request an action or a new flow entry that will be stored in the flow tables. The controller responds by sending an OpenFlow packet-out message and maybe an OpenFlow flow-mod message back to the switch. Since the controller is software-based, so it can be dynamically programed to provide manageability.

2.5 Evolution of OpenFlow Specifications

Different versions of OpenFlow specifications are available. The first version was the OpenFlow version 0.2.0, released in 2008. The most widely deployed specification is the version 1.0 [20]. This version has used 12 header fields of the Ethernet frame and IP packets (as shown in Figure 9) coming into the switch. A packet can be matched to a flow entry in the flow tables by using one or more header fields of the packet. After

that, in the OpenFlow 1.1 specification, the OpenFlow switch has contained metadata and Multiprotocol Label Switching (as shown in Figure 10). This version has supported several flow tables and a group table. A packet can be changed by one or more flow tables (pipeline processing). OpenFlow specification 1.1 has also introduced a new action (called instructions). Previously, an action can be: 1) forwarding the packet, or 2) dropping the packet. However, the instructions, from version 1.1, include also modifying a packet, and updating an action set. In OpenFlow specification version 1.2, IPv6 addressing has been added. The OpenFlow specification version 1.3 has then been released since 2012. This version can control some QoS by adding meter tables. It is possible to handle the rate of packets through per-flow meters. The major extension for OpenFlow specification version 1.4 has supported a new set of port properties to add optical ports to OpenFlow switch. Finally, OpenFlow specification version 1.5 has introduced egress tables. This tables enable processing to be done in the context of the output port, instead of the input port as the previous version. Table 3 shows the summary of OpenFlow specifications.

Ingress port
Ethernet src
Ethernet dst
Ethernet type
VLAN id
VLAN priority
IP src
IP dst
IP proto
IP ToS bits
TCP/UDP src port
TCP/UDP dst port

Figure 9 Match fields of a flow table entry in OpenFlow 1.0

Ingress port
Metadata
Ethernet src
Ethernet dst
Ethernet type
VLAN id
VLAN priority
MPLS label, MPLS EXP traffic class
IP src
IP dst
IP proto
IP ToS bits
TCP/UDP src port
TCP/UDP dst port

Figure 10 Match fields of a flow table entry in OpenFlow 1.1

Table 3 Summary of OpenFlow specifications

Feature	**OF 1.0**	**OF 1.1**	**OF 1.2**	**OF 1.3**	**OF 1.4**	**OF 1.5**
Ethernet: src,dst,type	X	X	X	X	X	X
IPv4: src,dst,proto,ToS	X	X	X	X	X	X
TCP/UDP: src_port, dst_port	X	X	X	X	X	X
Per table, flow, port, queue statistics	X	X	X	X	X	X
MPLS: label, traffic class		X	X	X	X	X
OpenFlow Extensible Match (OXM)			X	X	X	X
IPv6: src,dst,flow label			X	X	X	X
Per-flow meter & meter band				X	X	X
Optical ports					X	X
Egress table						X

2.6 Open Source OpenFlow Controllers

As shown in Table 4, there are various open source OpenFlow controllers, such as NOX [22], POX [29], Beacon [30], Floodlight [31], OpenDaylight [32], ONOS [33], Ryu [34], and so on. NOX has been developed by researchers at Stanford University and Nicira Networks. NOX provides a programming platform for controlling one or more OpenFlow switches. POX has been extended from NOX, and rewritten in python to support various platforms. Beacon has been developed in Java by researchers at Stanford University. Floodlight is an extension from Beacon by Big Switch Networks. It has been developed in Java to deploy software applications by using REST APIs. OpenDaylight has been developed in Java by the Linux Foundation (sponsored by several network industries, such as Cisco, NEC, IBM and so on). It is an open platform for customizing and automating scalable networks by focusing on network programmability. ONOS has been developed as a next-generation SDN solution for service providers, with a focus on scalability and performance. ONOS has also hosted by Linux Foundation, and written in Java. The software of this controller has been implemented as an Apache Karaf OSGi container, allowing interaction through Java APIs and REST APIs. This controller provides several features and standard protocols, such as OpenFlow, NETCONF [35], YANG model [36]. Ryu is a component-based SDN framework, written in python. Ryu provides various standard protocols for managing network devices, such as OpenFlow, NETCONF, OF-config and so on.

Table 4 List of Open Source OpenFlow Controllers

Name	Programming Language	License	OpenFlow versions	GUI	Southbound APIs	Northbound APIs
NOX	C++	GPL	OF1.0	Python, QTP4	OF1.0	REST API
POX	Python	Apache	OF1.0	Python, QT4	OF1.0	REST API
Beacon	Java	BSD	OF1.0	Web based	OF1.0	REST API
Floodlight	Java	Apache	OF1.0, 1.3	Web based	OF1.0, 1.3	REST API
OpenDaylight	Java	EPL	OF1.0-1.4	Web based	OF1.0,1.4, NETCONF, YANG,OF-Config	REST API
ONOS	Java	Apache	OF1.0-1.5	Web based	OF1.0, OF1.3, NETCONF, YANG	REST API
Ryu	Python	Apache	OF 1.0-1.5	Ryu GUI	OF1.0-1.3, NETCONF, OF-config	REST API

2.7 Role of SDN Controllers

As previous mentioned, SDN is a new network paradigm, allowing manageability and flexibility that traditional networks suffer from. In SDN, intermediate network devices (switches/routers) is just a forwarding device or a dump-device. The brains of the network are controllers. Applications at controller act as a strategic control point in the network. They manage the flow control of several switches/routers in the network from the centralized controllers. So, SDN allows any business logics to be intelligently deployed in the network without depending on or limiting to the vendors of network devices.

According to the Internet-draft report [37], the goal properties of SDN controllers are scalability, reliability, programmability, intercommunity, security, and manageability. An SDN controller interacts with network devices through southbound interfaces. Several models are interacted, including topology management, route management, host management, flow-tables management, interface management, database management, and so on. Topology management is calculated by using the

information (e.g., Link Layer Discovery Protocol (LLDP), BGP Link State [38], and so on), which is reported from the network devices. At the controller, route management is calculated from the abilities of network devices, such as link cost, bandwidth, and network information. Host management is a management of all hosts in the network, which takes functions of MAC and ARP learning. Flow-tables management is responsible for the basic functions of forwarding/routing storage. Interface management is a configuration of all ports in the network devices, including dynamic and static interface configuration. Database management involves in a management of all tables in the network devices with data synchronization.

From Figure 11 and Figure 12, two hosts (host-1 and host-3) want to communicate with each other. The controller in the figures has been implemented with the "learning switch application" for the following tasks:

1) When a data packet (SYN port 80) arrives at the switch, the switch inspects the packet's header, and tries to match it with a flow entry in the flow tables.
2) If the header of the packet is not matched with any flow entry. The switch generates OpenFlow packet-in message, including buffer-id (Buffer ID=100) and other fields. The packet is then encapsulated into the OpenFlow packet-in message.
3) The switch sends the OpenFlow packet-in message to the controller to request an action or a new flow.
4) The controller checks OXM fields of the OpenFlow packet-in message (i.e. data packet SYN port 80), and takes a destination MAC address. It then looks up a stored information of a MAC-address-switch-port map to find the destination switch-port, where the destination (host-3) is connected. The controller then generates an OpenFlow packet-out message and set its action to deliver the packet to the destination switch-port.
5) If the destination MAC address cannot be found from the MAC-address-switch-port map, the controller sets the action of the OpenFlow packet-out message to flood all ports of the switch.
6) The OpenFlow packet-out message is then sent from the controller back to the switch.
7) The controller may also send an OpenFlow flow-mod message with the same action of the packet-out message if it wants to store the action into the rule table.

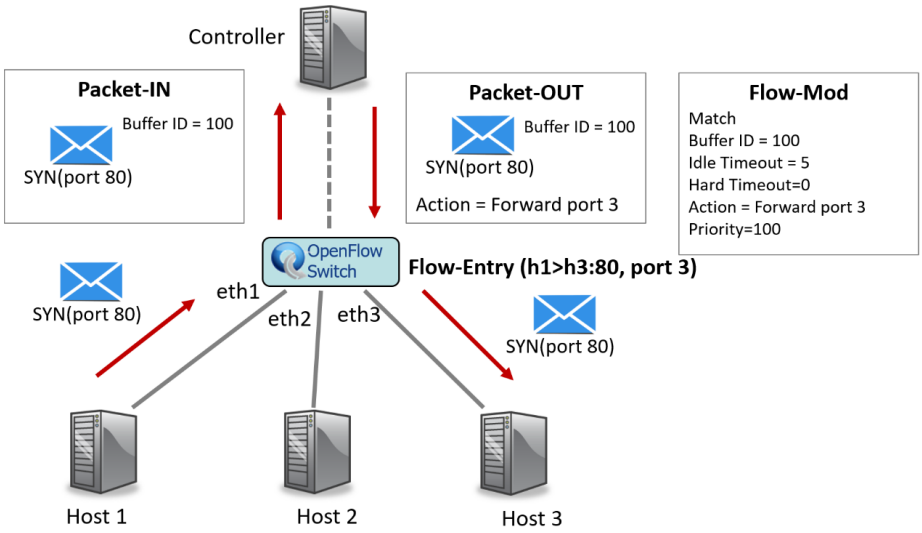


Figure 11 Example of OpenFlow protocol (Host 1 sends a message to host 3)

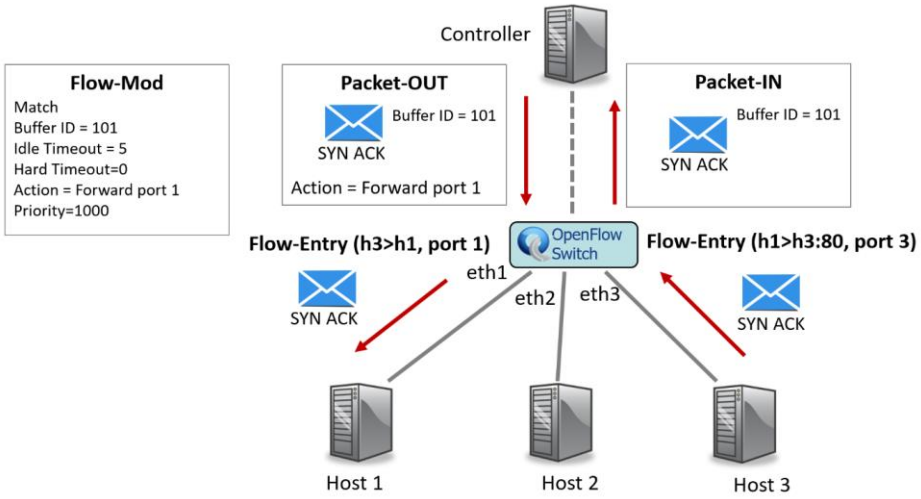


Figure 12 Example of OpenFlow protocol (Host 3 responses to host 1)

2.8 SDN in the Real World

The SDN revolution was initiated by the development of OpenFlow during 2008-2009. A lot of developments occur among a group of engineers at Stanford University. Now (as of July 2018 while writing this report), OpenFlow version 1.5.1 is the latest version, released in 2015 by Open Network Foundation (ONF)[6].

Many large networking companies have embedded OpenFlow technology into their products, such as Google and AT&T. In addition, these companies are parts of ONF and participate in designing OpenFlow. An OpenFlow has been deployed in many networking companies such as Google, Cornell University, REANZZ, and so on.

2.8.1 Cornell University

Cornell University has been running OpenFlow and OpenDaylight in production since 2014. Director of Computing and Information Science at Cornell's College of Engineering, says that [39] "we did not know exactly what the applications of the future would be, so we decided that we wanted to go with an OpenFlow network so we would have lots of flexibility.", and "SDN technology gives the network better performance and flexibility by allowing traffic flows to be redirected dynamically".

2.8.2 REANZZ New Zealand

REANNZ is the New Zealand's own National Research and Education Network (NREN), providing researchers and scientists with the ultra-fast network. This network allows researchers to store and share data and collaborate with other researchers in New Zealand and around the world in real-time. REANNZ's networks run on an OpenFlow-based switching, with the goal of providing an open networking environment [40].

2.8.3 Google B4

Google manages one of the largest enterprise networks and cloud deployments in the world. Engineers with direct experience have point out that OpenFlow is a key element of the Google architecture. OpenFlow has been used in both inside the Google data center, and to interconnect data centers, as a Wide Area Network (WAN) application [26, 41].

2.8.4 SDN Products

A lot of network industries have produced OpenFlow products to provide SDN deployment such as Cisco, ECI Telecom, Ericsson, Extreme Networks, Fujitsu, H3C, Hewlett Packard Enterprise, Huawei, Juniper Networks, NEC, Nokia (Alcatel-Lucent), and so on [6].

2.9 Differentiated Service Code Point (DSCP) of the DS field

In the Internet Protocol (IP), differentiated Services (DiffServ) [42] is a traffic control architecture, relying on the 8-bit DS field (in place of the outdated Type of Service (ToS) field [43]) in the IP header. DS field consists of the first six bits for the Differentiated Services Code Point (DSCP) and the other two bits for Explicit Congestion Notification (ECN). Due to its six-bit length, DSCP can support up to 64 different classes of traffic. DiffServ routers then decide on per-hop basis how to forward packets based on their class. First three bits of DSCP indicates IP precedence. These bits are called Class Selector (CS), prioritizing traffic types by class (CS0 – CS7, lowest to highest priorities respectively). For our design, DSCP value can be used to classify different traffic types (as shown in Table 5)

Table 5 Commonly used DSCP and IP precedence values

DSCP [41]		IP precedence [42]	
Value	**Name**	**Value**	**Description**
000 000 (0)	CS0 (default)	000 (0)	Routine, Best Effort
001 000 (8)	CS1	001 (1)	Priority
010 000 (16)	CS2	010 (2)	Immediate
011 000 (24)	CS3	011 (3)	Flash (voice or video signaling)
100 000 (32)	CS4	100 (4)	Flash Override
101 000 (40)	CS5	101 (5)	Critical (voice streams)
110 000 (48)	CS6	110 (6)	Internetwork Control
111 000 (56)	CS7	111 (7)	Network Control

2.10 Previous Solutions

According to the OpenFlow specification [15], an OpenFlow switch provides limited Quality of Service (QoS) through a simple queuing mechanism to manage data packets. A matched packet can be treated by output queue ID. Packet scheduling using queues is not defined by the latest specification (protocol version 0x06), and is switch dependent. First-In-First-Out (FIFO) queue is commonly used in the OpenFlow switch. In in-band control network, an OpenFlow message may compete with data packets. In out-of-band control network, the OpenFlow control messages of delay-sensitive traffic cannot gain low delay in competing with the OpenFlow control messages of delay-tolerant traffic. Without considering different prioritization, an OpenFlow messages may wait, or may be dropped in queue resulting in the increase of transmission delay of data traffic in acquiring forwarding rule.

Sköldström [44] have evaluated resource allocation in OpenFlow-based wide area networks in both in-band and out-of-band control networks. Their experimental results have shown that OpenFlow messages may compete with data traffic for network resources (e.g. bandwidth). This competition could finally cause the controller been disconnected after suffering from significant latency due to the increase of data traffic.

He et al. [12] have proposed SDN-based control suitably responsive for critical management applications, named Mazu: taming latency in software-defined networks. The first technique in Mazu is to avoid CPU processing events due to data plane packet arrivals by redirecting packets to a fast proxy. This process is tasked with generating the necessary messages for the controller. The first technique can overcome the inbound latency. Second, a technique to reduce the outbound latency is as follows: 1) flow engineering is used to compute paths such that the latency of installing forwarding state at any switch is minimized; 2) rule offloading is used to compute strategies for opportunistically offloading portions of forwarding state to be installed at a switch to other switches downstream. The Mazu techniques have been proposed to bypass the slow embedded switch CPU by redirecting unmatched packets to a proxy. However, the proxy in SDN may suffer network congestion itself. In case of a large number of switches are connected to the same proxy, this situation can cause latency to be

increased dramatically. This situation is not mentioned in the approach. In addition, Russ et al. [28] have pointed that a scale and speed are major problems of single point connection (like a proxy). Therefore, adding proxy to solve the latency problem may become unmanageable and unavailable.

In [9], the authors have proposed Quality of Service (QoS) framework using the SDN technologies and test the framework in failure-conditions. This study shows that an effectively high QoS can be achieved by prioritization different traffic. In [10], the authors have proposed queuing and failure recovery functionalities for OpenFlow in in-band control network. The results of the queuing functionality show that control traffic can be served with the highest priority and hence, data traffic cannot affect the communication between the controller and a switch. This study proposes to separate a queue for control and data traffic, serving the control traffic queue before the data traffic queue. The queues are provided by OF-Config (OpenFlow Configuration and Management Protocol) and OVSDB (Open vSwitch Database Management Protocol). The queues are controlled by Linux traffic control commands (e.g., Reference, Trafficlab1.1, and Trafficlab1.3 switches). Yet, the proposed solutions in [9, 10] have not designed to prioritize different types of control messages. The mechanism of this work also needs vendor specific options to handle queue priority. Moreover, this work is only designed for the in-band control network, and cannot be deployed for the out-of-band control network.

Traffic prioritization for OpenFlow has also been standardizing in [8]. This work is proposed to optimize OpenFlow protocol by appending a priority tag to the OpenFlow packet-in message and adding the Priority-based Flow Rule Request Message Processing Mechanism (PFRRMPM) at the switches and controllers. The PERMPM defines two modules, namely flow rule request sending module and flow rule request receiving module. Each module contains a service-type-based priority table to classify packet priorities. For instance, timely services (such as, the video streaming) possess a higher priority, compared to the background traffic. This solution can help the data flow with delay sensitivity to acquire the forwarding rule with shorter waiting delay, when there are excess flow rule request messages in the SDN. However, by adding priority tag to the OpenFlow packet-in messages, this work would significantly cause an overheard to the size of the control messages. In contrast, our work does not have such an overhead since it uses the existing DS field in the standard IP header of the control messages for priority marking. In comparison to ours, this work aims to support both in-band and out-of-band control networks as same as our work. Yet, this work has no detail of how to classify different traffic priorities in their design. It is only an initial design, with no prototype. There has been no experiment and performance evaluation to test their design. Our work has proposed more details for prioritizing different traffic, and different types of OpenFlow messages. We have also implemented the prototype of our design. Furthermore, experiments and performance evaluations have been done to demonstrate the success of our design in the out-of-band control network. We also expect positive results in the in-band control network.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Overview of Research Methodology

This thesis aims to propose and evaluate a new design (called a Priority-based Service Scheduling Policy for OpenFlow) to mitigate the delay problem for high-priority OpenFlow packet-in messages, based on packet contents or a specific user.

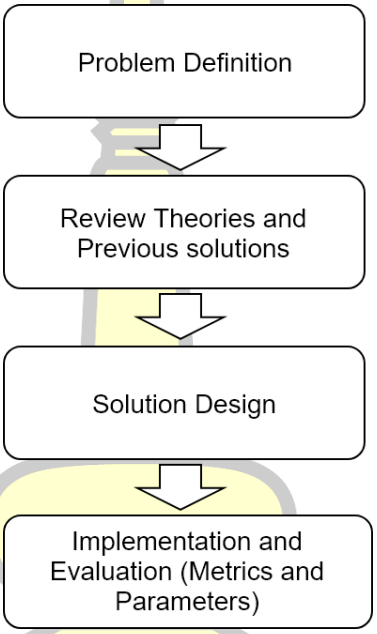


Figure 13 Overview of Research Methodology

Figure 13 shows the overview of research methods. For problem definition, delay in OpenFlow network is focused in this thesis. we will further describe delay problems and our solution in Chapter 4. Previous solutions are discussed in Chapter 2. In summary, some solutions support only in-band control network, whereas some solutions support only out-of-band control network. Although the solution in [8] has proposed an initial design to fix this problem in both in-band and out-of-band control networks, but with a rather high overhead for traffic tagging. For implementation and evaluation, there are three performance evaluation techniques (i.e., analytical model, network simulation and measurement testbed). We implement a prototype on network simulation and describe the reasons for choosing it in Section 3.2. The objective of this thesis is to design a solution for helping an OpenFlow message with shorter delay in acquiring forwarding rules in SDN. So, we describe delay in SDN and other parameters that will be used for performance metrics in Section 3.3.

3.2 Implementation and Performance Evaluation Techniques

For implementation, we implement a prototype on the ns-3 [13], based on the OpenFlow module version 1.3 for ns-3 [45]. For performance evaluation, Puangpronpitag [46] has investigated various of evaluation techniques including analytical model, network simulation and measurement testbed.

3.2.1 Analytical Model

An analytical model uses a mathematical modeling to evaluate proposed technique. An effectiveness of this model involves in the estimation and classification patterns of data. The results are commonly reported in terms of the estimated means and variances. This model may not be applied to evaluate our proposed method.

3.2.2 Network Simulation and Emulation

Simulation is a widely used tools to evaluate studies, both in academic research and industrial. It can provide the dynamic behavior of complex networks. There are several simulation tools (e.g., OMNeT [47], OPNET [48], ns-2 [49], ns-3 [13] and so on) to provide various of network environment. In general, these tools are based on an event-based stochastic technique. This technique is a set of full events and time to compute a network scenario. Each event (called sequence) contains specific time to process. Network emulation using virtualization technologies to provide realistic physical links, and analyze network behaviors in a discrete situation.

According to ns-3 website [12], ns-3 is a discrete-event network simulator for Internet systems, targeted primarily for research and educational usage. The ns-3 project is designed, following to the popular ns-2 simulator. ns-3 is also free software, licensed under the GNU GPLv2, and is publicly available for research, and development. The goal of the ns-3 project is to develop a preferred open simulation environment for networking research. ns-3 supports research on both IP and non-IP networks. In ns-2, simulation scripts are written in OTcl script, but simulation scripts in ns-3 are written in C++. ns-3 provides better support than in ns-2 for the following items:

- Modularity of components,
- Scalability of simulations,
- Integration of externally developed code and software utilities,
- Emulation,
- Tracing and statistics,
- Validation.

3.2.3 Measurement Testbed

A testbed network consists of specific hardware and software to evaluate an approach. To make a testbed, the combination of hardware and software is required to run the experiments. This testbed may be difficult to perform due to unmanageability and high cost. Puangpronpitag [46] has argued that several parameters in the testbed

may be disturbed by unpredictable parameters. So, the experimental results may be misled.

3.3 Performance Metrics & Parameters

In the standardization, IETF published several RFCs to provide performance metrics for IP networks (presented in Table 6).

Table 6 Metrics/Parameters of IETF IPPM RFCs

Category	**IETF IPPM RFCs [50]**
Framework	2330
Sampling	2330 3432
Loss	2680
Delay	2679(One-way) 2681(Round-trip) 3391 (Delay Variation)
Availability	2678

According to Hanemann et al. [51], the authors have summarized from [50], including the set of elementary metrics to indicate network performance. There are four main elementary metrics, namely availability, loss & error, delay and throughput.

The availability is considered that how robust the network (i.e., percentage of time to run without any problem). The loss & error indicate the network congestion conditions or transmission error, such as radio signal problem. The delay also indicates the network congestion. The delay is measured either one-way delay (time to transmit from source until receiving at destination), or round-trip delay (one-way delay from source to destination plus one-way delay of destination sent acknowledgement back). There are several delays in computer networks (i.e., processing delay, queuing delay, transmission delay, propagation delay). In SDN, delay measurements are quite different from traditional networks. The throughput indicates amount of data that a user can transfer through network in time unit. Hence, we will use the following performance metrics to evaluate our approach:

3.3.1 Delay in SDN

Delay is a crucial index of the operation efficiency of SDN networks, especially for real-time applications (such as Voice over IP). For the event of table-miss in OpenFlow switches, a data packet will be encapsulated into an OpenFlow packet-in message. The message is then sent to a controller to acquire forwarding rule. In case of network congestions, this OpenFlow packet-in message can cause an increase of the delay. Moreover, in an in-band control network, OpenFlow control messages may wait in queue and may be dropped due to the competition with data packets. There have been several studies on the delay in SDN, such as Long et al.[8], Hsiao and Chang [52]. The

delay measurement of this work is based on these previous studies. The details are described as follows.

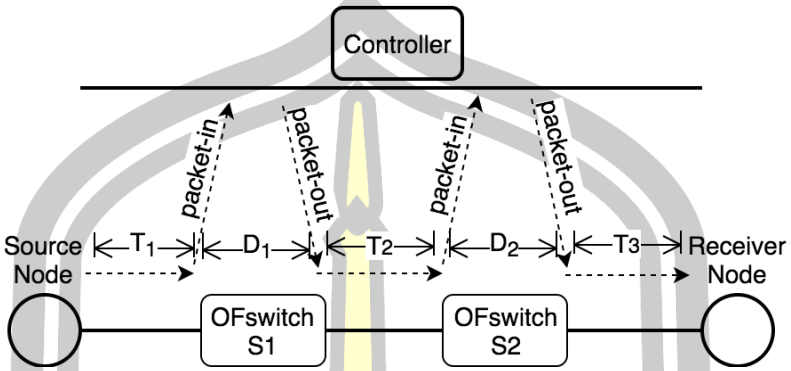


Figure 14 Delay measurement

As shown in Figure 14, delay of a data packet can be measured as the summation of transmission time (T_1, T_2, T_3), and delay in each switch or each hop delay (D_1, D_2). T_1 is a transmission time, counting the time from a source node to a switch S1. D_1 is the hop delay, counting the time from switch S1 sending a packet-in message to the controller until switch S1 receiving a packet-out message (i.e. acquiring forwarding rules). This time includes processing time at both switch S1 and the controller, queuing delay at both switch S1 and the controller, and transmission time of the packet-in and packet-out messages. For the next hop, T_2 is a transmission time from switch S1 to switch S2. The controller can look at the network end-to-end while making instruction for the switches because it has a full physical and logical view of the network topology. So, flow rules of switch S2 are known. These rules can be installed automatically [53]. So, the time to acquire forwarding rules will be excluded from D_2. For this reason, D_2 is obtained by the summation of processing delay in the flow-tables (TCAM packet matching delay) and queuing delay in switch S2. Experimental results of this work will be evaluated in term of this delay.

3.3.2 Packet Loss

Packet loss is defined as fraction of the total transmitted packets that have not been received at the receiver. In this work, packet loss is described as the percentage of packets lost with respect to packets sent. Packet loss is generally caused by network congestion. In SDN, packet loss in control links directly affects data traffic. Packet loss can be obtained as follows:

$$Packet\ loss = \frac{Number\ of\ packet\ lost}{Number\ of\ packet\ sent} \times 100$$

3.3.3 Throughput

Throughput is defined as the rate of successful packets delivered over a communication channel. Throughput is usually measured in bits per second (bps). In

SDN, network congestion in control links can reduce the throughput of both control and data traffic. Throughput can be obtained as follows:

$$Throughput = \frac{The\ number\ of\ delivered\ (bits)}{Time\ taken}$$

3.3.4 Overhead of PMT

According to TCAM operations, the OpenFlow module in ns-3 [45] considers the concept of virtual TCAM to estimate the average searching time of flow tables. To provide a more realistic delay, this module uses sophisticated search algorithms for packet matching such as binary search trees.

The following equation is used to estimate the processing time of flow tables:

$$\text{Processing time of flow tables} = K \times \log_2(n)$$

where *K* is the processing time for a single TCAM operation; *n* is the number of entries on pipeline flow tables. For our design of PMT, a binary search tree is used for packet matching in the PMT. So, the following equation is used to estimate the processing time of PMT:

$$\text{PMT processing time} = K \times \log_2(m)$$

where *K* is previously mentioned; *m* is the number of rules in PMT. Due to the same processing time of single TCAM operation between flow tables and PMT, the following equation can be used to estimate the processing time of flow tables after adding the delay of the PMT:

$$\text{The processing time flow tables and PMT} = K \times \log_2(n\ x\ m)$$

PMT adds some overhead in a switch. So, this overhead will be evaluated in Chapter 5.

3.4 Result Analysis

According to Wang and Xia [54], there are three main components (network topology, traffic model and performance metrics) to get the results from network simulators as shown in Figure 15.

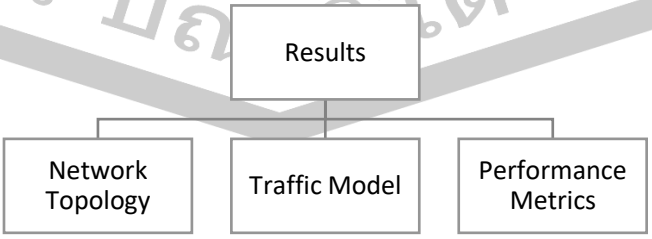


Figure 15 Three components of network simulation tool

For network topology, a parking-lot topology is used to evaluate our proposed design. This topology is suitable to evaluate the situation in network unsuitable, such as the competition among control traffic in an out-of-band control network (further described in Chapter 5). For traffic model, we define several traffic models to experiment (further described in Chapter 4). Finally, the performance metrics have previously been described in Section 3.3.

If an experiment is simulated/measured repeatedly, the result will be different each time. In Statistics, a confidence interval is a type of estimation, statistically computed several results. In this work, each simulation will be run 50 times using a different Random Number Generator (RNG) seeds to get the averaged results, quoted with error bars with respect to confidence intervals of 95%.

3.5 Testing and Validation

Ns-3 [13] provides tools to allow for both model validation and testing scripts. These scripts perform self-validation that contains a specific set of input with known outputs. The simulated results of these scripts can notify the user whether pass or fail. So, our proposed design will be validated by using these scripts.

CHAPTER 4

DESIGN

4.1 Problems and Solution Design

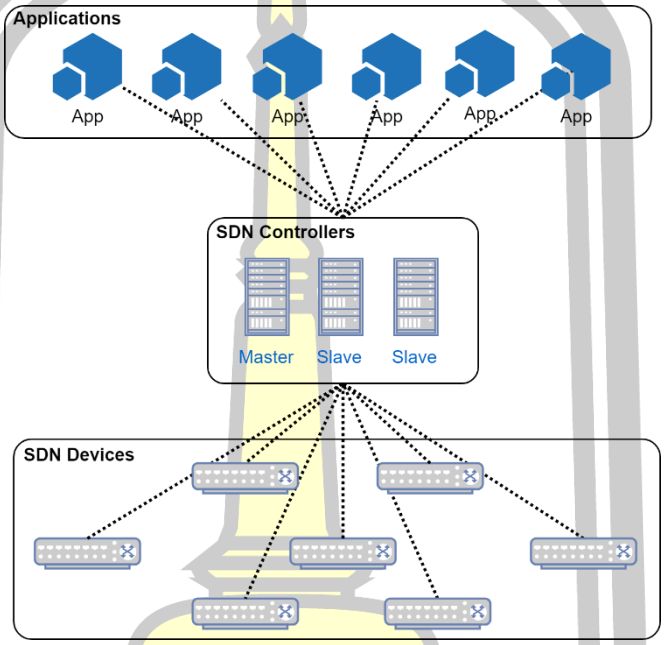


Figure 16 Overview of centralized control in SDN

In the centralized controller environment, there may be several switches under the same controller, resulting in the bandwidth competition among OpenFlow control messages in the out-of-band control network (as show in Figure 16). This competition could increase the transmission delay of data traffic. Without considering different data traffic types and prioritizing their control traffic properly, some delay sensitive services may finally fail. Network inefficiencies may then occur in SDN. Hsiao et al. [52] has previously pointed out that SDN suffers from the transmission delay, and this transmission delay is a significant issue of transmission quality for network operators.

Furthermore, the situation in the in-band control network would be even worse than the out-of-band control network. There is an extra competition between OpenFlow control messages and data packets. Without giving higher priority, the OpenFlow control messages may be dropped. This should cause the failure of forwarding data traffic at the end.

Moreover, different customers of a network provider may be under different Service Level Agreements (SLA). Up to the agreement between the provider and their customers, different customers may be treated differently in terms of transmission delay. In this regard, it is necessary to be able to prioritize the control messages of some specific customers differently according to their SLA.

Hence, this thesis proposes a Priority-based Service Scheduling Policy for OpenFlow (PSO). The purpose of PSO is to give a higher priority for control traffic in the in-band control network. Furthermore, PSO provides different priorities for OpenFlow control messages, based on contents/services (data traffic types) and/or customers (according to their SLA) for both in-band and out-of-band control networks.

4.2 Design of Priority-based Service Scheduling Policy for OpenFlow (PSO)

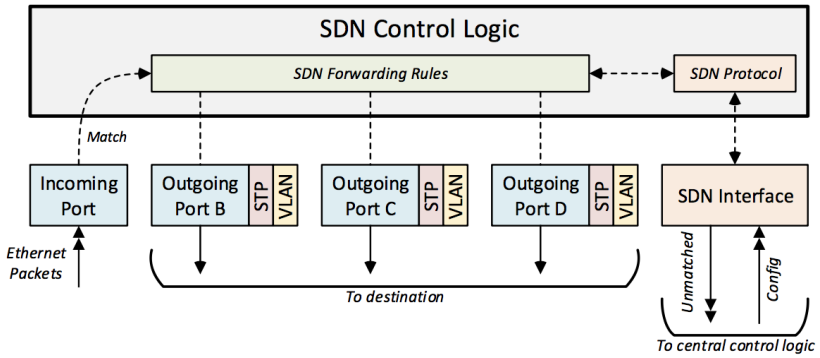


Figure 17 SDN-enabled Ethernet switch

Source: [55]

According to the SDN-enable Ethernet switch (as shown in Figure 17), Benjamin et al. [55] have proposed to separate between outgoing ports and controller ports (SDN interfaces). Unmatched packets are sent from the switch to the controller via the SDN interface, while matched packets are forwarded to the outgoing ports according to the action, specified in the flow tables. We propose a design for the in-band control network as follows. For any OpenFlow switches, which are not connected to the controller, they will be connected to the next switch via an SDN interface. So, the unmatched packets will be forwarded to the next hop's SDN interface until reaching the controller. For our design, this interface will be used to manage both OpenFlow messages and data packets.

In general, there is no queuing functionality, proposed in the OpenFlow protocol [15]. All messages are served equally. So, we propose Priority-based Scheduling Policy for OpenFlow (PSO) to provide a queuing mechanism. This queuing mechanism automatically prioritizes OpenFlow messages serving in queues before data traffic.

4.2.1 PSO Modules

Figure 18a) shows the architecture of traditional OpenFlow switches. Generally, an OpenFlow switch looks up all incoming packets from an IN_PORT queue to match with its flow tables. The incoming packets are then served at an OUT_PORT queue, by taking actions according to the rule in the matched flow entry. In general, queue management is FIFO (First In First Out) with a drop-tailed algorithm. All packets are treated equally. According to our PSO design (illustrated in Figure 18b), PSO modules are embedded into both OpenFlow switches (PSO switch module) and controllers (PSO

controller module) respectively. These modules provide a special queuing mechanism to automatically prioritize OpenFlow messages and data traffic.

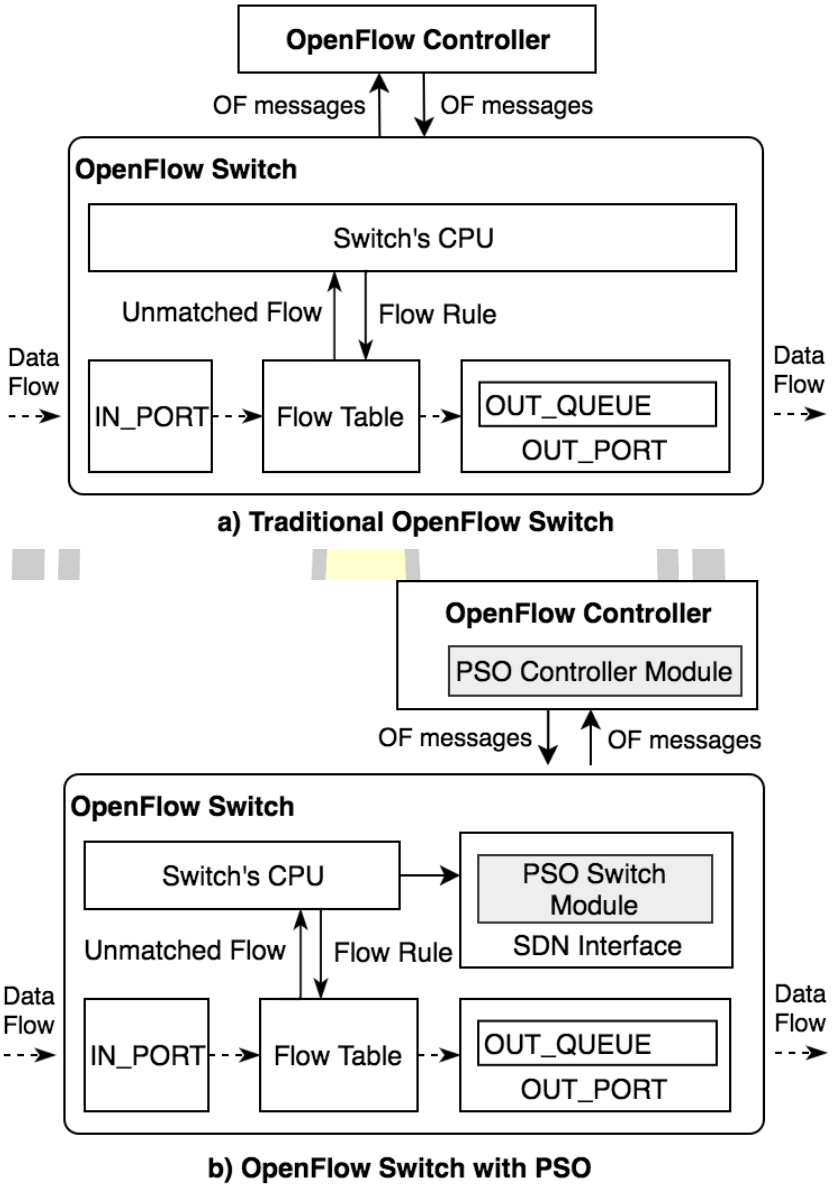


Figure 18 Traditional OpenFlow Switch vs. OpenFlow switch with PSO

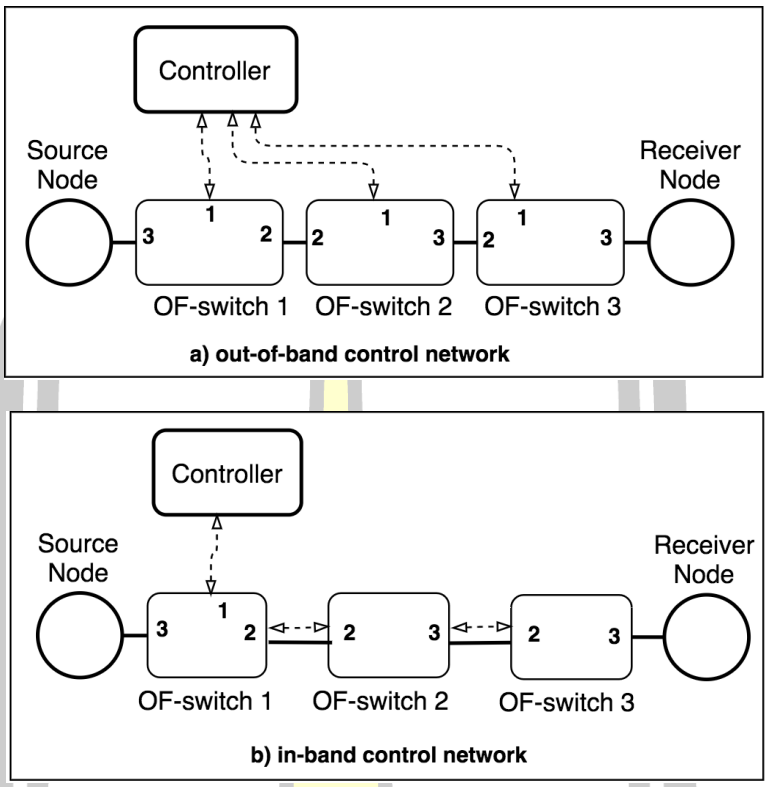


Figure 19 SDN interfaces

Instead of FIFO queue (as mentioned in Chapter 2), a PSO switch module provides special queue management on SDN interfaces of the OpenFlow switch. For the out-of-band control network, the SDN interface (or an SDN port) of an OpenFlow switch is a specific port of that switch, directly connecting to the OpenFlow controller [55]. As shown in Figure 19a, port-1 of the OF-switch 1, port-1 of the OF-switch 2, and port-1 of OF-switch 3 are SDN interfaces. For the in-band control network, some specific ports on each switch are deployed to pass OpenFlow control messages to the controller. We also call them SDN interfaces. Some of these SDN interfaces may directly connect to the controller, for example, port-1 of the OF-switch 1 (as shown in Figure 19b). Otherwise, some SDN interfaces on the in-band control network may indirectly connect to the controller via the other switches. For example, port-2 of the OF-switch 1, port-2 and 3 of the OF-switch 2, and port-2 of OF-switch 3 are SDN interfaces (as shown in Figure 19b). As previously mentioned, these SDN interfaces in the in-band network could suffer the delay due to the competition between OpenFlow control messages and data packets transmitting over the same interface. Our PSO switch module will mitigate this problem by providing special queue management on these interfaces to prioritize OpenFlow messages over data packets.

At the controller, a PSO controller module will provide special queue management for all interfaces to prioritize OpenFlow messages.

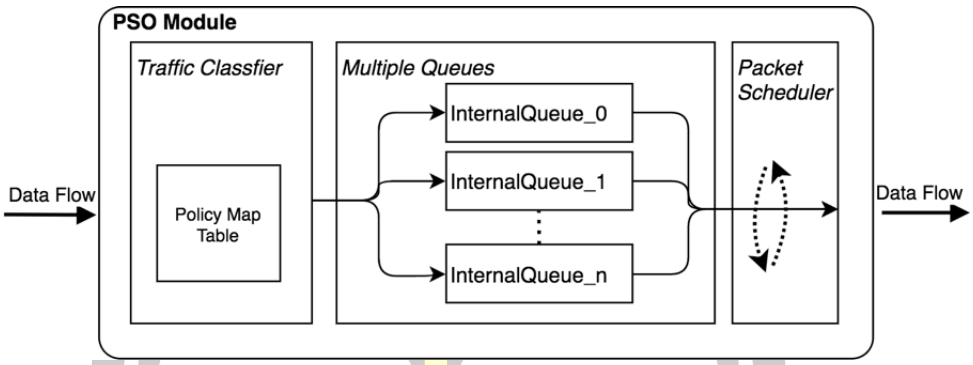


Figure 20 The components of PSO modules

Both PSO modules consist of a traffic classifier, multiple queues, and a packet scheduler (as illustrated in Figure 20). The traffic classifier differentiates packets by using a Policy Map Table (PMT). The multiple queues are internal queues for different priority traffic. The packet scheduler is a packet prioritization scheduling mechanism. The details of the prioritization will be further discussed in next section. The PSO controller module is designed to follow the prioritization set by the PSO switch module. It has a queuing mechanism corresponding to the PSO switch module.

4.2.2 Traffic Classifier

For any traffic arriving at an SDN interface, a traffic classifier will differentiate traffic according to a set of predefined rules in a Policy Map Table (PMT). The rules must be set by a network administrator at the controller. Otherwise, the traffic will be treated equally. The PMT will then be copied to all OpenFlow switches in the network using OFPT_SET_CONFIG, which is an OpenFlow control message for switch configuration. Each rule contains traffic type, match fields, and action (as shown in Figure 21. The traffic classifier will match the arriving traffic with traffic type and match fields, then follows the action of the matched record.

Traffic type	Match fields	Action

Figure 21 Policy Map Table (PMT)

Table 7 Traffic types

Traffic type no.	Traffic type description
1	OpenFlow configuration messages, and OpenFlow symmetric messages
2	OpenFlow packet-in message, OpenFlow packet-out messages, and other OpenFlow control command messages
3	Other OpenFlow messages
4	data packets

traffic type is a field to specify different types of traffic, as shown in

Table 7. In this work, we predefine four *traffic types* that should be treated with different priorities accordingly. The first type includes *OpenFlow configuration messages* (e.g., OFPT_SET_CONFIG) and OpenFlow symmetric messages (e.g., OFPT_ECHO_REQUEST, OFPT_ECHO_REPLY). The second type includes OFPT_PACKET_IN, OFPT_PACKET_OUT messages *and other control command* messages. The third type includes other OpenFlow control messages, such as OFPT_TABLE_STATUS. Finally, the forth type includes data packets. In the in-band control network, the data packets may share the same link with OpenFlow control messages. So, we give OpenFlow control messages higher priorities than data packets. Among the OpenFlow control messages, we give three different priorities as shown in the

Table 7. OpenFlow configuration messages are given the highest priority to ensure that any configurations by network administrators work out on time. *Packet-in* messages (OFPT_PACKET_IN), *packet-out* messages (OFPT_PACKET_OUT) *and other control command* messages are given a higher priority than other OpenFlow control messages since they carry important instructions between the controller and the OpenFlow switches. For the details of the types of OpenFlow control messages, they can be found from [15]. These predefined traffic types and priorities are also flexible, and may be specified differently by network administrators for different organizations.

match fields is exactly the same as *match fields* of flow tables, which details are given in the OpenFlow specification [20]. They contain several header fields to match against the header of data packets. The *match fields* can help specify application services (for example, protocol=TCP port=80 is specified “http” service). These *match fields* may also help specify the customers (such as, by looking at a specific source or destination IP addresses).

action contains an action defining how the traffic should be treated by the *packet scheduler*. *action* may be “setting DSCP values”, or “setting output queue ID”.

Example-1: In an out-of-band control network, a PMT is defined as follows:

```
Rule #1: traffic_type=1, action=dscp:CS7
Rule #2: traffic_type=2, ip_proto=17, udp_dst=20000, action=dscp:CS6
Rule #3: traffic_type=2, action=dscp:CS5
Rule #4: traffic_type=3, action=dscp:CS4
```

From example-1, a PMT contains four rules. *Rule #1* sets DSCP header of packets to CS7 for all OpenFlow configuration and OpenFlow symmetric messages. This is to ensure that any configuration commands by network administrators should get the highest priority. *Rule #2* gives the second priority to OpenFlow *packet-in*/*packet-out* messages of real-time services (UDP port 20000) by setting their DSCP header to CS6. *Rule #3* gives the third priority to OpenFlow *packet-in*/*packet-out* message of other services by setting their DSCP header to CS5. The last rule (*Rule #4*) gives the lowest priority to other OpenFlow messages. In summary, example-1 differentiates traffic by looking at its traffic types, and its services (using match-fields), and then marks its priority by setting DSCP header. These DSCP values will be later considered by a packet scheduler to handle the traffic according to its priority (such as using Weighted Fair Queue (WFQ)).

Example-2: In an out-of-band control network, a PMT is defined as follows:

```
Rule #1: traffic_type=1, action=queue_id:0
Rule #2: traffic_type=2, ipv4_dst=202.28.34.1/26, action=queue_id:1
Rule #3: traffic_type=2, ipv4_src=202.28.34.1/26, action=queue_id:1
Rule #4: traffic_type=2, action=queue_id:2
Rule #5: traffic_type=3, action=queue_id:3
```

From example-2, a PMT contains five rules as follows. *Rule #1* is to set queue_id=0 (the highest priority queue) for all OpenFlow configuration and symmetric messages. This is to give the highest priority to configuration commands from network administrators. *Rule #2* and *Rule #3* are to set queue_id=1 (the second highest priority queue) for OpenFlow *packet-in*/*packet-out* messages of a specific customer (source or destination IP address=202.28.34.1) since this customer may have a special agreement with the network provider. *Rule #4* is to set queue_id=2 (the third highest priority queue) for OpenFlow *packet-in*/*packet-out* messages of all other customers. *Rule #5* is to put all other OpenFlow messages into queue_id=3 (the lowest priority queue). The packet scheduler of this case manages output queue using Priority Queue.

Example-3: In an in-band control network, a PMT is defined as follows.

```
Rule #1: traffic_type=1, action=dscp:CS7
Rule #2: traffic_type=2, action=dscp:CS6
Rule #3: traffic_type=3, action=dscp:CS5
Rule #4: traffic_type=4, action=dscp:copy
```

In this example, there are both OpenFlow control messages and data packets, competing on the same connection due to an in-band control network. *Rule #1* will set DSCP header of packets to CS7 for all OpenFlow configuration and symmetric messages. This is to give the highest priority to configuration commands by network

administrators. *Rule #2* gives the second priority to OpenFlow *packet-in/packet-out* messages, and other OpenFlow control command messages, by setting their DSCP headers to CS6. *Rule#3* gives the third priority to other OpenFlow messages, by setting their DSCP headers to CS5. Finally, *Rule#4* gives the lowest priority to data packets, and set their DSCP headers to be equal to the DSCP value inside the data packets. In this case, DSCP of the data packets may be previously set to give different priorities. These DSCP values of data packets should be defined less than CS5. In the other case, DSCP of the data packets may not be set; thus, all data packets are treated equally. These DSCP values of OpenFlow control messages and data packets will be then considered by a packet scheduler to schedule the traffic according to their priorities (such as using WFQ).

Example-4: In an in-band control network, a PMT is defined as follows.

Rule #1: traffic_type=1, action=dscp:CS7
Rule #2: traffic_type=2, ip_proto=17, udp_dst=20000, action=dscp:CS6
Rule #3: traffic_type=2, action=dscp:CS5
Rule #4: traffic_type=3, action=dscp:CS4
Rule #5: traffic_type=4, action=dscp:copy

From example-4, a PMT contains five rules in an in-band control network. *Rule #1* is the same as the one given in the example-3. *Rule #2* gives the second priority to OpenFlow *packet-in/packet-out* messages of real-time services (UDP port 20000) by setting their DSCP header to CS6. *Rule #3* gives the third priority to OpenFlow *packet-in/packet-out* message of other services by setting their DSCP header to CS5. *Rule #4* gives a lower priority (DSCP=CS4) than *Rule #3* to other OpenFlow messages. The last rule (*Rule #5*) gives the lowest priority to data packets, and set their DSCP headers to be equal to the DSCP value inside the data packets. In this case, DSCP of the data packets may be previously set to give different priorities. These DSCP values of data packets should be defined less than CS4. In the other case, DSCP of the data packets may not be set; thus, all data packets are treated equally. These DSCP values of OpenFlow control messages and data packets will be then considered by a packet scheduler to schedule the traffic according to their priorities (such as using WFQ).

Example-5: In an in-band control network, a PMT is defined as follows.

Rule #1: traffic_type=1, action=queue_id:0
Rule #2: traffic_type=2, ip_proto=6, ipv4_dst=202.28.34.1/26, action=queue_id:1
Rule #3: traffic_type=2, ip_proto=6, ipv4_src=202.28.34.1/26, action=queue_id:1
Rule #4: traffic_type=2, action=queue_id:2
Rule #5: traffic_type=3, action=queue_id:3
Rule #6: traffic_type=4, action=queue_id:4

From example-5, a PMT contains six rules in an in-band control network. *Rule #1* is to set queue_id=0 (the highest priority queue) for all OpenFlow configuration and symmetric messages. *Rule #2* and *Rule #3* give the second priority to OpenFlow *packet-in/packet-out* messages to a specific customer (source or destination IP address=202.28.34.1). This rule allows a specific customer to have a special agreement with the network provider. *Rule #4* is to set queue_id=2 (the third priority queue) for OpenFlow *packet-in/packet-out* messages of all other customers. *Rule #5* is to set

queue_id=3 (the fourth priority queue) to all other OpenFlow messages. The last rule (Rule #6) is to put all data packets into queue_id=4 (the lowest priority queue). The packet scheduler of this case should manage output queue using Priority Queue.

4.2.3 Queue and Packet Scheduler

Multiple queues and a *packet scheduler* are last two components of PSO modules. Instead of FIFO drop-tail queuing, PSO modules provide a queuing mechanism that can prioritize different traffic. Weighted Fair Queue (WFQ) or Priority Queues (PQ) or min rate [15] or other suitable queues can be deployed for this purpose. The multiple queues are one or more internal queues, attached to a specific port (an SDN interface). These internal queues are used to schedule out packets from the SDN interface.

After passing through the *traffic classifier*, packets will be differentiated according to the rules in PMT. After that, the DSCP values of the packets may be set (marked), or a queue ID may be specified. For the first case, the *packet scheduler* will schedule the packets according to the DSCP values and scheduling mechanisms (defined by the network administrator). For the second case, the packet scheduler will map the specified queue ID directly to a specific internal queue.

For example, the *traffic classifier* may specify traffic priorities by marking DSCP values of the IP header. These DSCP values can provide up to 64 traffic categories without an extra-overhead. The traffic scheduler can then use WFQ to handle different traffic priorities. In the other way, the traffic classifier may specify queue ID, and the packet scheduler then uses PQ or min rate for different traffic types.

4.2.4 Configuration of Policy Map Table

In general, a controller can set or query configuration parameters in an OpenFlow switch using the OpenFlow configuration messages. In this work, our PMT is defined by a network administrator at the controller, and distributed to OpenFlow switches using configuration messages or suitable configuration protocol (i.e., OF-Config), during the connection setup. The controller and switches then have the same PMT. The configuration steps are as follows:

1) The administrator configures the controller paths for all switches.
2) The administrator creates rules in PMT according to the organization policy at the controller.
3) The controller sends and updates PMT to all switches by using set configuration messages.
4) The administrator can check the PMT of any switch by using get configuration messages.

CHAPTER 5

PERFORMANCE EVALUATION

5.1 Network Simulation Scenarios

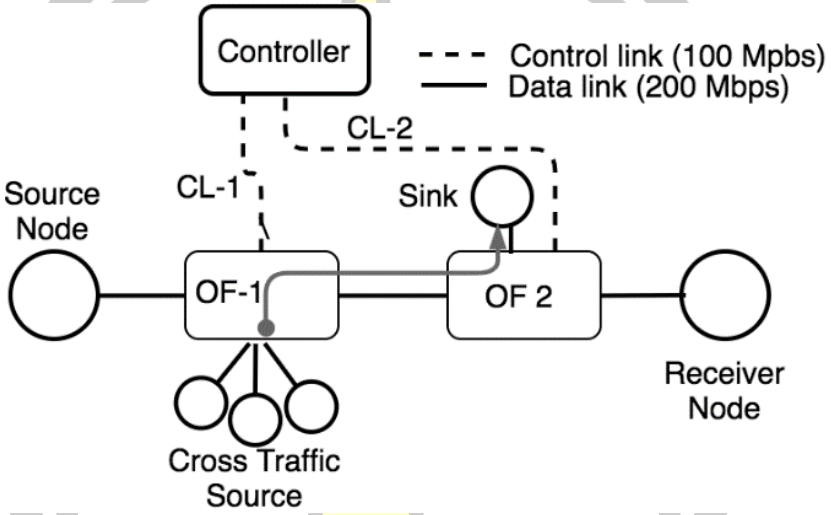


Figure 22 Network Scenario

The performance study will be conducted using the ns-3 [12]. The OpenFlow module version 1.3 for ns-3 is based on [45]. All nodes (source and receiver nodes, cross traffic nodes) implement first-in-first-out scheduling and drop-tail queuing. Network simulation scenario is shown in Figure 22. Out-of-band network will be evaluated. Each link of data traffic has a capacity of 200 Mbps. Each link of control traffic has a capacity of 100 Mbps. A specific data traffic is set to 1000 Kbps, sent from a source node to a receiver node. To make the competition among OpenFlow messages, switch OF-1 has cross traffic. Cross-traffic nodes generate several data flows and send them via switch OF-1 to the sink node. In this case, switch OF-1 will generate OpenFlow packet-in messages (associated with the data flows), which increase a load on CL-1.

Since the cross-traffic has increased, several OpenFlow packet-in messages are sent to the controller. In this case, a load on a control link (CL-1) (as shown in Figure 22) is then increased. So, we define this load on CL-1 as *Normalized Load (NL)*, and *NL* can be obtained as follows:

$$NL = \frac{Data\ rate\ (bps)}{Link\ capacity\ (bps)}$$

5.1.1 Experiment 1: a PMT for an out-of-band control network to give a priority for a specific service

For experiment 1, a PMT is defined as shown in the Example-1 (as described in Chapter 4). This PMT is to provide highest priority to OpenFlow messages of a specific data traffic. In this experiment, a PMT contains four rules. This is to give that any configuration commands by network administrators and OpenFlow *packet-in*/*packet-out* messages of real-time services (UDP port 20000) should get the higher priority than other traffic. The objective of this experiment is to test how the increase in load on a control link (CL-1) impacts to data traffic, and to test the PSO in terms of throughput and delay of a specific data traffic (high priority traffic)

5.1.2 Experiment 2: a PMT for an out-of-band control network to give a priority for a specific user/customer

For experiment 2, a PMT is defined as shown in the Example-2 (as described in Chapter 4). This PMT is to provide highest priority to OpenFlow messages of a specific user/customer. The objective of this experiment is to test how the increase in load on a control link (CL-1) impacts to data traffic, and to test the PSO in terms of throughput and delay of a specific user/customer traffic.

5.2 Simulation Results

To report the results, each simulation will be run 50 times to get the average results, quoted with error bars with respect to confidence intervals of 95%.

5.2.1 Experiment-1: a PMT for an out-of-band control network to give a priority for a specific service

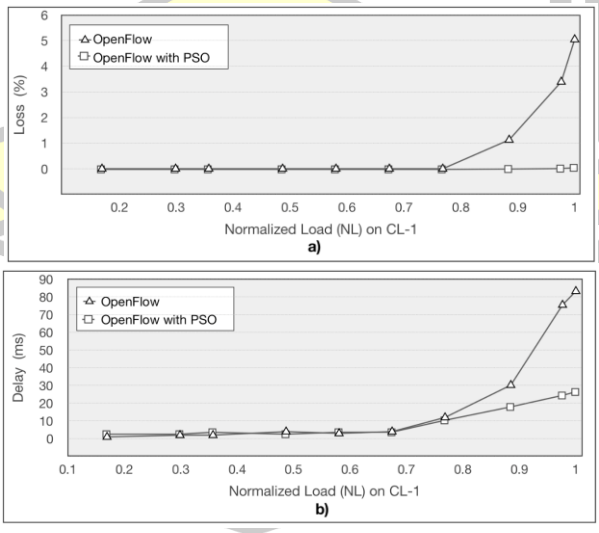


Figure 23 OpenFlow packet of a high priority data traffic on a control link (CL-1): a) Packet loss, b) Delay

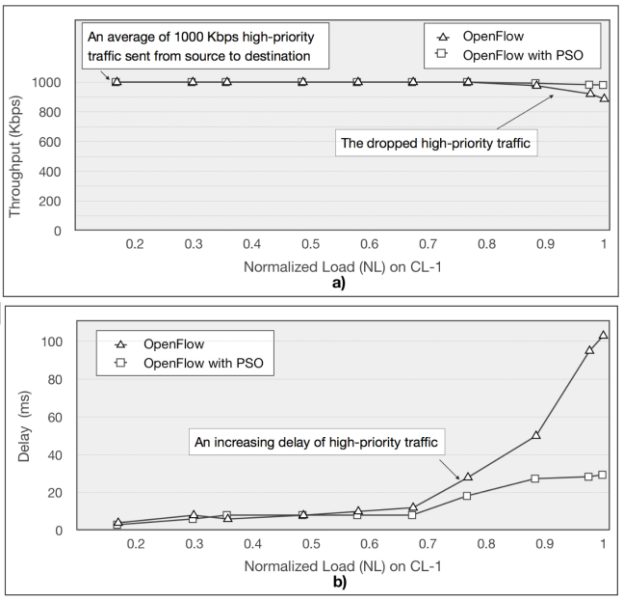


Figure 24 Impact on a high priority data traffic: (a) Throughput, (b) Delay

Figure 23 a) shows packet loss of OpenFlow packet-in messages and OpenFlow packet-out messages of high priority data traffic, comparing between OpenFlow with PSO and traditional OpenFlow. Figure 23 b) shows the hop delay in switch OF-1 of high priority data traffic, comparing between OpenFlow with PSO and traditional OpenFlow.

Under a low and medium NL (NL $\leq$ 0.8) over CL-1, the results have shown low OpenFlow packet loss (0%) and low delay (2.2 $\pm$ 0.6 ms) of both OpenFlow with PSO and traditional OpenFlow. However, at a high load (NL > 0.8), the congestion cause a significantly high packet loss (5 $\pm$ 1.4 %) in traditional OpenFlow. In this case, as the load increases, a switch drops more OpenFlow packet-in messages. After dropping, a switch has to retransmit these messages after their timeouts. This finally increases hop delay in switch OF-1 (83 $\pm$ 2.7 ms). Yet, even with a high load (NL > 0.8), OpenFlow with PSO provides a lower OpenFlow packet loss, and a lower hop delay (22 $\pm$ 2 ms) of switch OF-1, as shown in Figure 23 b).

Figure 24 shows throughput and delay of a high priority data traffic by comparing between OpenFlow with PSO and traditional OpenFlow. In traditional OpenFlow, at a high load (NL $>$ 0.8), some buffered high priority packets are then dropped after their timeouts because the OpenFlow control messages at CL-1 are dropped. So, the throughput is reduced to 887 $\pm$ 52 Kbps, and the delay of a high priority data traffic is increased to 103 $\pm$ 5 ms. However, in the OpenFlow with PSO even at a high load (NL $>$ 0.8), a higher throughput (992 $\pm$ 6 Kbps) of the specific data traffic can be provided. The PSO can also provide a low delay (27.6 $\pm$ 2.8 ms) in comparison to the traditional OpenFlow.

So, our PSO can help the data flow with high priority to acquire forwarding rules with lower delay under network congestion at the control link. In case of the congestion

at the control link, traditional OpenFlow would cause a severe problem to the delay sensitive services such as for Voice over IP.

5.2.2 Experiment-2: a PMT for an out-of-band control network to give a priority for a specific user/customer

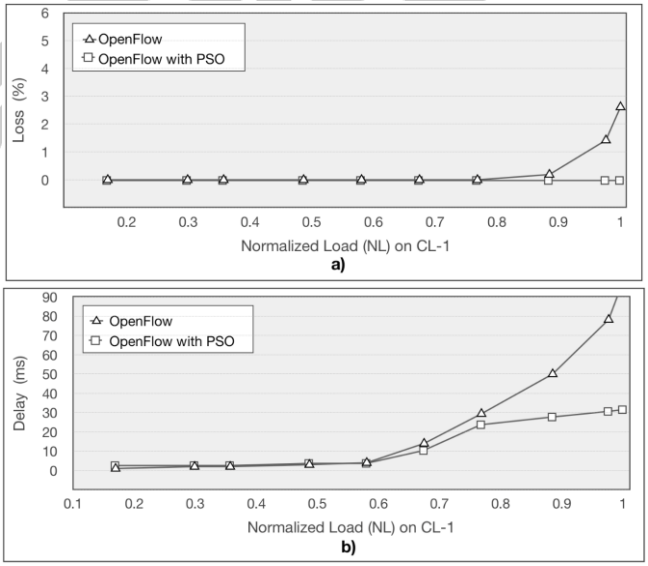


Figure 25 OpenFlow packet of a high priority data traffic on a control link (CL-1): a) Packet loss, b) Delay

Figure 25 a) shows packet loss of OpenFlow packet-in messages and OpenFlow packet-out messages of high priority data traffic, comparing between our design OpenFlow and traditional OpenFlow. Figure 25 b) shows the hop delay in switch OF-1 of high priority data traffic, comparing between our design OpenFlow and traditional OpenFlow.

Under a low and medium load (NL $\leq$ 0.8) over CL-1, the results have shown low OpenFlow packet loss (0%) and low delay (4.2 $\pm$ 0.8 ms) of both our design and traditional OpenFlow. However, at a high load (NL > 0.8), the congestion causes a significantly high packet loss rate (2.6 $\pm$ 1 %) in traditional OpenFlow. In this case, as the load increases, a switch drops more OpenFlow packet-in messages. After dropping, a switch has to retransmit these messages after their timeouts. This finally increases hop delay in switch OF-1 (98 $\pm$ 3.2 ms), as shown in Figure 25 a). Yet, even with a high load (NL > 0.8), our design provides a lower OpenFlow packet loss, and a lower hop delay (27 $\pm$ 2.3 ms) of switch OF-1, as shown in Figure 25 b).

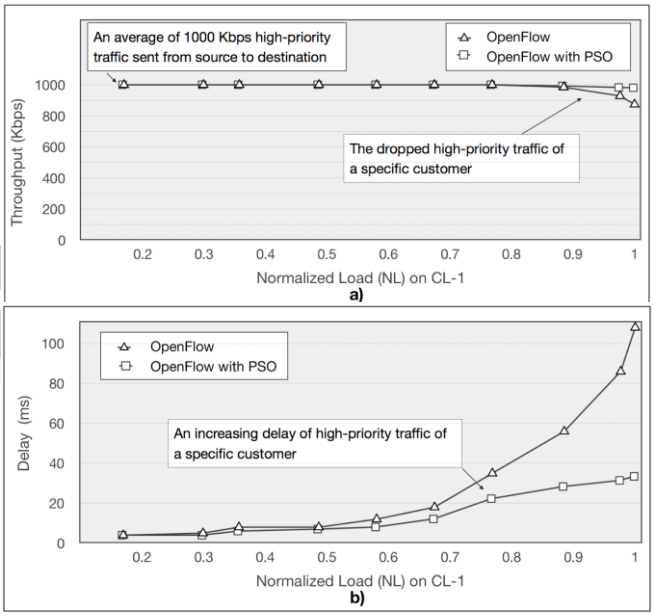


Figure 26 Impact on a high priority data traffic: a) Throughput, b) Delay

Figure 26 a) shows throughput and delay of data traffic of a specific customer by comparing between our design and traditional OpenFlow. In traditional OpenFlow, at a high load (NL > 0.8), some buffered high priority packets are then dropped after their timeouts because the OpenFlow control messages at CL-1 are dropped. So, the throughput is reduced to 874 ± 47 Kbps, and the delay of a high priority data traffic is increased to 108 ± 4 ms. However, in our design even at a high load (NL > 0.8), a higher throughput (1000 Kbps) of the specific customer traffic can be provided. Our design can also provide a low delay (33 ± 2 ms) in comparison to the traditional OpenFlow (as shown in Figure 26 b).

So, our design can help the data flow with high priority to acquire forwarding rules with lower delay under network congestion at the control link. In case of the congestion at the control link, traditional OpenFlow would cause a severe problem to some privilege user/customer.

5.3 Discussion of In-band Control Network

For the in-band control network, we have designed PSO to prioritize OpenFlow messages over data traffic. So, this mechanism would give positive results. However, OpenFlow mechanisms for in-band control network are still at their early stage. There are quite a few open-research issues to complete the design, such as bootstrapping mechanisms (to establish a communication path between switches and a controller), topology discovery mechanisms (to find the most suitable path from a switch to the controller), control path recovery mechanisms (to recover from the control path failure). Some studies (such as [56-58]) have initially investigated on the issues but still unsolved. The previous experiments over the in-band control networks, are only specific to each design. Therefore, we have not yet experimented on our PSO over the

in-band control network. The future work would be proposing several mechanisms, required for in-band networks. However, it is not in the scope of this thesis.

5.4 Analysis of PMT Overhead

For our PSO, the PMT adds some overhead in a switch. However, this overhead is applicable for all situations (in examples 1-5 Chapter 4). For example, the maximum rule of these examples is six rules (m=6). According to Chavas [45], K is to set 20 µs. If flow tables have the minimum rule (n=1), the overhead of the flow tables plus PMT could be obtained: $20 \times \log_2(1 \times 6) \approx 52$ µs. Figure 27 shows the values of TCAM delay for more rules in PMT (including 64 rules), and more flow entries in the flow tables. Summarily, the overall TCAM-delay overhead is acceptable (less than 0.3 ms).

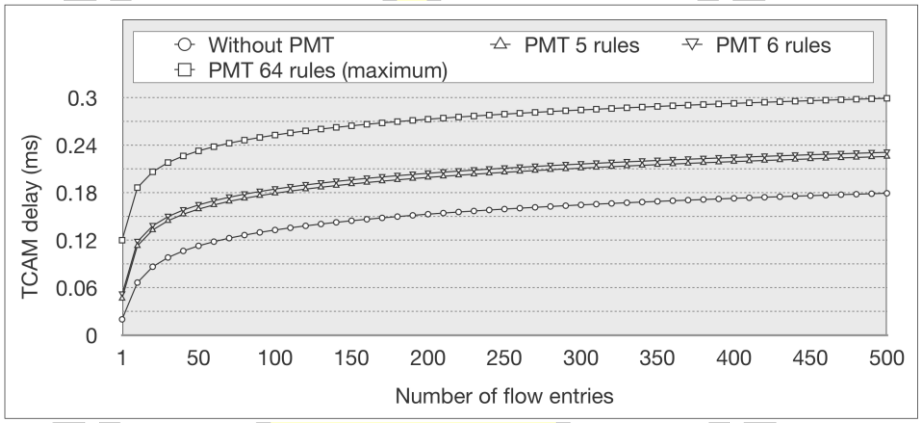


Figure 27 TCAM processing time

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Summary and Discussion

Internet technologies have been deployed to enable programmability. Therefore, active networks and programmable networks were proposed to allow their users to program the intermediate network devices. Recently, SDN is a new network paradigm that has been introduced for both active and programmable networks. SDN has been designed to separate control plane from data plane. Controllers are located in the control plane, whereas network devices (such as switch/router) are located in the data plane. OpenFlow has been deployed as a SDN protocol to communicate between both planes. The decoupling of the control and data planes is related to the manageability in the network. In particular, when the first data packet of a new flow arrives at a switch, the switch then inspects each packet's header and tries to match it with a flow entry in the flow tables. If the header of the packet is not matched with any flow entry, the switch generates an OpenFlow packet-in message and sent it to the controller to acquire a forwarding rule. This may increase network load, and make the control plane a potential bottleneck. In addition, since the flow tables of switches are configured in real time by an external device, there is the extra delay introduced by its flow setup process.

In this thesis, we have investigated into the aforementioned delay problem and proposed a design of Priority-based Scheduling policy for OpenFlow (PSO) as the solution. PSO modules are designed to be embedded in both OpenFlow switches and controllers. The purposes of PSO are: 1) to give the priority to control traffic in competing with data traffic for the in-band control network, 2) to provide high-priority OpenFlow packet-in messages of a specific traffic (such as real-time delay-sensitive audio/video) or specific users (such as users with special Service Level Agreement) for both in-band and out-of-band control networks. According to our design, PSO can be controlled by network administrators by putting the priority policies into the Policy Mapped Table (PMT). Our design has also proposed several PMT samples to prioritize different traffic, users, control messages. To mark different traffic priority, our traffic classifier module uses DS field [39] without adding any extra overhead to the traffic.

To experiment and evaluate our solution, this thesis has selected network simulation as a research method. The prototype of PSO modules have been implemented into the simulator by extending OpenFlow module version 1.3 for ns-3 [53]. Our experimental results in out-of-band control networks have shown that our PSO can help the delay-sensitive get forward in time even under network congestion in control links (with normalized load > 0.8), while the traditional OpenFlow switch fails in the same situation. In comparison with other previous solutions, our solution have less overhead, and support both in-band and out-of-band networks.

For in-band control networks, we also expect the same results. Yet, the design of several mechanisms in-band control network are still an open-research issue. So, it is put as a future.

6.2 Thesis Achievement

This work has evaluated the OpenFlow protocol, and proposed a new design of OpenFlow protocol successfully. The major contribution of this work can describe as follows:

1) This work has established a set of key performance evaluation criteria and performance metrics/parameters to evaluate the proposed solution.
2) A performance study by network simulation (ns-3) of traditional OpenFlow have been successfully completed. At a high load on control links, traditional OpenFlow causes delay problems to delay sensitive services and privilege user/customer.
3) The ideas learnt from the performance study of traditional OpenFlow have been deployed to propose a novel design of an innovative OpenFlow to differentiate traffic priorities.
4) A performance comparison between our new design and traditional OpenFlow has been done, and demonstrated a few advantages of our design. Under network congestion on control link, high priority traffic can be served in time by Priority Queue together with our mechanisms.

6.3 Future Work

Although several achievements have been claimed in this thesis, there would be also some weaknesses. Several ideas have occurred during work on this thesis. The following aspects discuss some restrictions of this thesis and the issues that would be investigated as future work.

6.3.1 Implementation and Complex Simulation Scenarios

In this work, network simulation scenarios are rather simple. However, these simple scenarios are useful to evaluate the situation in network congestion in SDN. There is no current simulation technology that can simulate networks of real size. Even if the model could be scaled, suitable tools to reach effectively the results are still difficult to find. So, the issues of simulation scale are remaining one of the simulation issues.

Module of OpenFlow version 1.3 for ns-3 has been implemented by Chaven et al. [45]. This module allows ns-3 to simulate OpenFlow networks, considering main features of this version. Some features are not yet support:

1) OpenFlow channel encryption: Switches and controllers may communicate through TLS connections. Since there is no TLS support on ns-3, the OpenFlow channel is implemented over TCP connection.

2) In-band control network: For in-band control network, controllers can manage switches remotely over a shared network link. However, due to the limitation of ns-3 modules and the uncompleted design of in-band mechanisms, this thesis has not yet experimented to evaluate in the in-band control environment. So, one of the future direction could be proposing the mechanisms for the in-band networks on the open-research issues, and experimenting on them.

6.3.2 Prototyping and Measurement on Testbed

The performance evaluation in this thesis has relied only on the network simulation. The network simulation is accepted by the research community and industries. The network simulation should be correctly taken as the real world. Building and prototyping a testbed are complex and expensive. However, after simulation, prototyping and testing on the real test-bed would be a good idea. So, the next step could be implementing and evaluating PSO on the real test-bed using a tool, such as GENI testbed [59].

REFERENCES

REFERENCES

[1] B. Theophilus and A. Aditya, "Unraveling the Complexity of Network Management," in *USENIX Symposium on Networked Systems Design and Implementation*, pp. 335–348.

[2] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 493-512, First Quarter 2014.

[3] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The SoftRouter Architecture," in *ACM SIGCOMM Workshop on HOTNETS*, 2004.

[4] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, *et al.*, "Forwarding and Control Element Separation (ForCES) Protocol Specification," IETF RFC-5810, March 2010.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Compututer Communication Review*, vol. 38, pp. 69–74, March 2008.

[6] ONF. (July 2017). *ONF: Open Networking Foundation*. Available: https://www.opennetworking.org

[7] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, January 2015.

[8] X. Long, W. Wang, X. Gong, X. Que, and Q. Qi, "Priority based Flow Rule Request Message Processing Mechanism in the OpenFlow Switch," IETF Internet-Draft, October 2016.

[9] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Gonçalves, R. Figueiredo, *et al.*, "Implementing Quality of Service for the Software Defined Networking Enabled Future Internet," in *Third European Workshop on Software Defined Networks*, Budapest, 2014, pp. 49–54.

[10] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band control, queuing, and failure recovery functionalities for openflow," *IEEE Network*, vol. 30, pp. 106-112, February 2016.

[11] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, *et al.*, "Latency in Software Defined Networks: Measurements and Mitigation Techniques," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, pp. 435–436, June 2015.

[12] K. He, J. Khalid, S. Das, A. Akella, L. Li, and M. Thottan, "Mazu: Taming Latency in Software Defined Networks," University of Wisconsin-Madison Technical Report, 2014.

[13] nsnam.org. (July 2017). *ns-3*. Available: https://www.nsnam.org

[14] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layer and Architecture Terminology," IETF RFC-7426, January 2015.

[15] Open Networking Foundation. (March 2015). *OpenFlow Switch Specification version 1.5.1 (Protocol version 0x06)*. Available:

https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[16] A. Morreale and M. Anderson, *Software Defined Networking: Design and Deployment*: CRC Press, 2014.

[17] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 2181-2206, Fourth Quarter 2014.

[18] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Merwe, "Design and implementation of a routing control platform," in *Conference on Networked Systems Design & Implementation*, 2005, pp. 15–28.

[19] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *DARPA Active Networks Conference and Exposition*, New York, NY, USA, 2002, pp. 2–15.

[20] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet*, vol. 6, pp. 302–336, 2014.

[21] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela, "A Survey of Programmable Networks," *SIGCOMM Computer Communication Review*, vol. 29, pp. 7–23, April 1999.

[22] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, *et al.*, "NOX: towards an operating system for networks," *SIGCOMM Computer Communication Review*, vol. 38, pp. 105-110, 2008.

[23] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *SIGCOMM Computer Communication Review*, vol. 44, pp. 87-98, April 2014.

[24] IRTF. (March 2017). *Software-Defined Networking Research Group (SDNRG)*. Available: https://datatracker.ietf.org/rg/sdnrg

[25] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Communications Magazine*, vol. 52, pp. 116-123, July 2014.

[26] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, *et al.*, "B4: Experience with a Globally-deployed Software Defined Wan," in *ACM SIGCOMM*, 2013, pp. 3–14.

[27] L. Schiff, S. Schmid, and P. Kuznetsov, "In-Band Synchronization for Distributed SDN Control Planes," *SIGCOMM Computer Communication Review*, vol. 46, pp. 37-43, 2016.

[28] W. Russ and Z. Shawn, "Cloudy-Eyed: Complexity and Reality with Software-Defined Networks," *The Internet Protocol Journal*, vol. 19, pp. 33-44, November 2016.

[29] POX project team. (July 2017). *POX*. Available: https://github.com/noxrepo/pox

[30] D. Erickson, "The Beacon Openflow Controller," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, New York, NY, USA, 2013, pp. 13–18.

[31] R. Izard. (July 2017). *Project Floodlight*. Available: http://www.projectfloodlight.org

[32] Linux Foundation. (July 2017). *OpenDaylight*. Available: https://www.opendaylight.org
[33] ONF. (March 2017). *ONOS*. Available: https://onosproject.org
[34] RYU project team. (July 2017). *Ryu SDN Framework*. Available: https://osrg.github.io/ryu
[35] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," IETF RFC-6241, June 2011.
[36] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)," IETF RFC-6020, June 2011.
[37] R. Gu and J. Wang, "SDN Controller Requirement," IETF Internet-Draft, July 2016.
[38] H. Gredler, J. Medved, S. Previdi, A. Farrel, and S. Ray, "North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP," IETF RFC-7752, March 2016.
[39] Open Networking Foundation. (July 2017). *Special Report: OpenFlow and SDN State of the Union*. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/special-reports/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf
[40] S. Cotter. (July 2017). *Reannz Deploys New Zealand's First Qrganisation-Wide SDN Switch*. Available: https://reannz.co.nz/news/reannz-deploys-new-zealands-first-organisation-wide-sdn-switch
[41] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Communications Magazine*, vol. 52, pp. 116–123, July 2014.
[42] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF RFC-2475, December 1998.
[43] P. Almquist, "Type of Service in the Internet Protocol Suite," IETF RFC-1349, 1992-07.
[44] P. Sköldström and K. Yedavalli, "Network virtualization and resource allocation in OpenFlow-based wide area networks," in *IEEE International Conference on Communications (ICC)*, Ottawa, 2012, pp. 6622–6626.
[45] J. Chaves, C. Garcia, and R. Madeira, "OFSwitch13: Enhancing Ns-3 with OpenFlow 1.3 Support," in *The workshop on Ns-3*, New York, USA, 2016, pp. 33–40.
[46] S. Puangpronpitag, "Design and Performance Evaluation of Multicast Congestion Control for the Internet," PhD thesis, University of Leeds, School of Computing, 2003.
[47] OpenSim Ltd. (July 2017). *OMNeT++*. Available: https://www.omnetpp.org
[48] OPNET Technologies. (July 2017). *OPNET*. Available: http://opnetprojects.com
[49] ISI.edu. (July 2017). *The Network Simulator - ns-2*. Available: https://www.isi.edu/nsnam/ns
[50] IETF. (March 2017). *IP Performance Metrics (ippm)*. Available: https://datatracker.ietf.org/wg/ippm

[51] A. Hanemann, A. Liakopoulos, M. Molina, and D. Swany, "A study on network performance metrics and their composition," *Campus-Wide Information Systems*, vol. 23, pp. 268–282, 2006.

[52] T. Hsiao and W. Chang, "Network controller for delay measurement in SDN and related delay measurement system and delay measurement method," US Patent US9419878 B2, August 2016.

[53] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE Way to Test OpenFlow Applications," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA, 2012, pp. 127–140.

[54] G. Wang and Y. Xia, "An NS2 TCP Evaluation Tool," Internet-Draft, April 2007.

[55] J. Benjamin, L. Niels, and A. Kuipers, "Scalability and Resilience of Software-Defined Networking: An Overview," *CoRR*, vol. abs/1408.6760, 2014.

[56] D. Kotani and Y. Okabe, "A packet-in message filtering mechanism for protection of control plane in openflow networks," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Los Angeles, California, USA, 2014, pp. 29–40.

[57] S.-C. Lin, P. Wang, and M. Luo, "Control traffic balancing in software defined networks," *Computer Networks*, vol. 106, pp. 260 - 271, 2016.

[58] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "A Self-Organizing Distributed and In-Band SDN Control Plane," in *International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2656-2657.

[59] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, *et al.*, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5-23, March 2014.

BIOGRAPHY

NAME	Mr. Piyawad Kasabai
DATE OF BIRTH	October 4, 1986
PLACE OF BIRTH	Nakhon Ratchasima Province
ADDRESS	85/1 M. 2 Rangka Yai, Phimai sub-distric, Nakhon Ratchasima Province 30110, Thailand
POSITION	Lecturer
PLACE OF WORK	Udon-Thani Rajabhat University, Udon-Thani Province, Thailand
EDUCATION	2009 Bachelor of Science in Computer Science, Mahasarakham University 2011 Master of Science in Information Technology, Mahasarakham University 2018 Doctor of Philosophy in Computer Science, Mahasarakham University