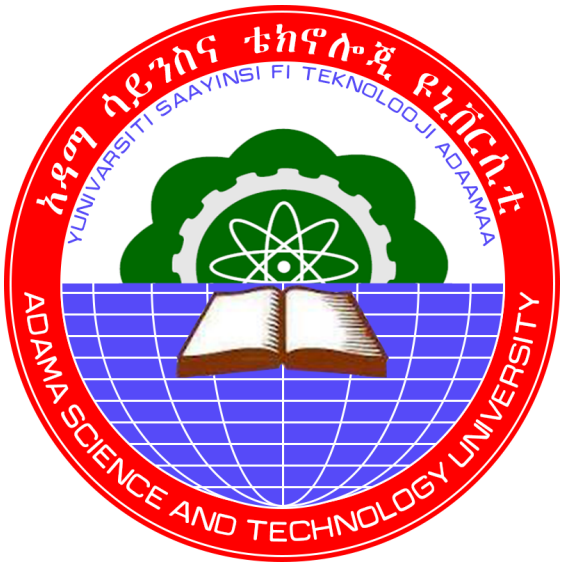
OpenFlow Switch Hybrid Flow Entry Mode in OpenDaylight Software-Defined Network Controller

by : Asnake Ayele



A Thesis submitted to the

Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement

for the Degree of

Masters of Science in Computer Science and

Engineering

Office of Graduate Studies

Adama Science and Technology University
(ASTU)

Jan. 2020

Adama, Ethiopia

OpenFlow Switch Hybrid Flow Entry Mode in OpenDaylight Software-Defined Network Controller

By: Asnake Ayele

Name of Advisor: Dr-Ing Frezewd Lemma

A Thesis submitted to the

Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement

for the Degree of

Masters of Science in Computer Science and

Engineering

Office of Graduate Studies

Adama Science and Technology University

(ASTU)

Jan. 2020

Adama, Ethiopia

DECLARATION

I hereby declare that this MSc Thesis is my original work and has not been presented for a degree in any other university and all sources of material used for this thesis have been duly acknowledged.

Name: . . . . . .

Signature: . . . . . .

This MSc Thesis has been Submitted for examination with my approval as a thesis advisor.

Name: . . . . . .

Signature: . . . . . .

Date of submission: . . . . . .

Approval of Board Members

We, the undersigned, members of the Board of Examiners of the final open defense by *OpenFlow Switch Hybrid Flow Entry Mode in OpenDaylight Software-Defined Network Controller* and examined the candidate. This is, therefore, to certify that thesis has been accepted in partial fulfillment of the requirement of the Degree of *Computer Science and Engineering*.

.....	.....	.....
Supervisor/Advisor	*Signature*	*Date*
.....	.....	.....
Chairperson	*Signature*	*Date*
.....	.....	.....
Internal Examiner	*Signature*	*Date*
.....	.....	.....
External Examiner	*Signature*	*Date*

ACKNOWLEDGMENTS

I am here today in the brink of eyes to declare their accomplishment, to be a nominee of Computer Science and Engineering graduate, for all of this hill climb achievement, almighty GOD that made this for me, absolute praise for GOD. Beyond this, I want to give a special reward to everybody, who sticks with me through thin and thick, day and night, low and height, for better and worst. Family?, wow it is amazed to declare absolute love to the family around there, I wish you "Etalem," long live to you and beloved families. And the rest of the family Emaye and enat, thank you for your support through a difficult time. For sure, this is a great time to be accompanied by the best advisor around in the whole carrier, Dr-Ing Frezewd Lemma. Since 1997 E.C, I know you are to sacrifice the whole have in your heart (RVUC). At last, I want to acknowledge all of you, my colleagues in CSE, thanks for this fantastic journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

In Software-Defined Network (SDN), OpenFlow enabled switch is currently lacking the flexibility in absence of adequate memory called Ternary Content Address Memory (TCAM). Particularly in proactive flow entry mode, meanwhile in reactive mode; responsiveness is main concern in mission critical operations. Several researches have shown that a flow writing approach is the primary performance factor in SDN infrastructure and its influence in applications/services that need sensitive response time to the request in latency hungry playing field. The aim of this study is to design the new hybridized flow writing approach, developed by integrating reactive and proactive approaches, and to recommend the best hybrid feature that can perform better than the existing. The study approach is to identify each segmentation based on sender's location where packet is coming, by looking header attribute for destination. Proactive approach is applied, if segment is identical, and a reactive approach is maintained where segmentation is different. Additionally, the system has some intelligent capability for removing proactive flows based on usage analyses over a certain period of time. The segmentation in the system is characterized by ip address prefix and service orientation. Upon comparing the performance of reactive and hybrid approaches using latency and throughput parameters, the later revealed that significant improvement in the controller responsiveness. The results show that the hybrid approach on the controller performance enhances positively, 18.4% latency and 23.5% in throughput improvements. Thus, as a key factor in applications/services, response time is reduced. Further research is needed to identify other factors that can improve the performance of SDN controller.

KEYWORDS

. . . . . .

Index terms: Hybrid flow, Data Center, ODL l2switch, SDN, OpenFlow switch, flow mode

. . . . . .

CHAPTER ONE

Introduction

1.1 Background

The development of the technology and exponential increasing network populations results in growing bandwidth and latency demands in a legacy network structure. Its difficult to managing ever-growing security issues. A recently emerged network paradigm (SDN framework) [2], The newly developing SDN provides a feasible solution for complicated management problems. The architecture of the SDN is organized in layers as the same as the legacy systems. However, the separation of control and data plane from the switch/router appears to be a critical difference from the legacy network. The control plane unit, intelligence, is taken away from the device, which leaves the switch/router as a dumb device. Data layer is nothing more than forwarding duties based on matching rules. High level of the above theory design shown in Figure 1.1, this separation of layers had the motive for deveOps.

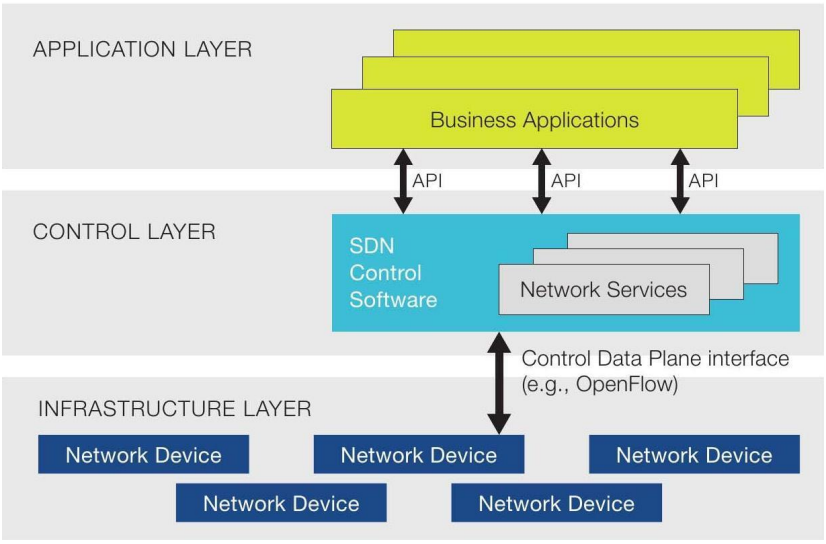


Figure 1.1: The Role of Controller in SDN Approach, source adopted from [3]

1.2 Motivation

Since then, in the computer networks, it is still bottle-necking around the technology. In today's computing power, memory technology, storage device, and programming language aspect are some of them that are around a million times fold speeds up roughly. However, nothing happens in the network industry, as it is so tightly controlled by monopoly and closed proprietary. Innovation is guaranteed by those who control the field; otherwise, it will not happen. Today's modern commodity computers' performance, at least in the networking aspect, did not get network speed to what is aimed; the achievement not guaranteed in terms of latency, bandwidth, and performance. It is the primary motivation to get involved in the network.

SDN is open for everyone. We can add whatever we want: a firewall, load balancing, Network Function Virtualization (NFV), controller optimization, and clustering. These are, by far, the obvious choice as an entry point in OpenFlow (OF) enabled switch architect of SDN. Those all mentioned cause leads to the problem definition.

1.3 Statement of the Problem

In the clear cut problem definition based on discussed SDN philosophy in section 1.1. the layers of SDN separated with different physical placement, when we look at the two methods of writing flow rules. In the switch/router, there are varieties of problems. First, let us look at the proactive approach of flow mode. The controller has a lot and lots of burdens. This mentioned problem resulted from handling when every event triggers for delivering flow rule for every forwarding device. Whether it is essential or not to that device, it keep memory management busy frequently. TCAM, a minimal amount of capacity and too much energy was consumed by nature, especially in searching local memory to match the rule when every packet arrives at the ingress port. This scenario is much worse for the controller processing capability due to adding/removing events occurring at any time. The worst case is that every rule has an unlimited lifetime in memory. Because the nature of static/proactive flow does not remove by switch, it makes memory starvation and adds non-valuable searching time.

In reactive mode, every forwarding device is responsible for packet-in to the controller node when the flow of the packet arrives and does not match the local rule. The controller sends

a packet-out message to the device to write the rule to take action based on that rule and even for future reference. In this case, the flow must wait until resolving the destination. The controller may also be overwhelmed with other requests. The request may drop and resent again. As a result, latency-sensitive applications/services may suffer a lot because of this problem. Moreover, another problem is the controller central intelligence and performance affected due to spending much time on handling packet-in/out duty.

This problem-focused on OpenDaylight (ODL) module and this module can not write flow based on destination IP Address match. The module only addresses using Media Access Control (MAC) which is not suitable for Traffic Engineering (TE). These are the issue in current trending in SDN needs addressing.

1.4 Research Question

Based on the above problems, this study answered the following questions:

- *how to come up the better method for flow writing mechanism?*
- *How to improve latency and throughput in SDN using proposed method?*
- *What are the methods to integrate the method into the existing system and performance test?*

1.5 Objectives of the Study

1.5.1 General Objectives

The main objective of the study is to introduce OpenFlow Switch Hybrid Flow Entry Mode in OpenDaylight SDN Controller.

1.5.2 Specific Objectives

- To examine the current status of reactive and proactive flow writing mechanism in OpenFlow enabled switch.
- To deeply study the OpenFlow switch flow writing mechanism in SDN.
- To research the hybrid flow writing mechanism specifically as used within the SDN.
- To identify suitable case studies concerning hybrid flow writing evaluation.

- To analyze and compare reactive and hybrid approach as demonstrated by the case study.
- To critically evaluate the use of a hybrid flow writing mechanism at the SDN OF switch.
- To recommend improvements to the SDN, in their use of hybrid flow writing.

1.6 Methodology

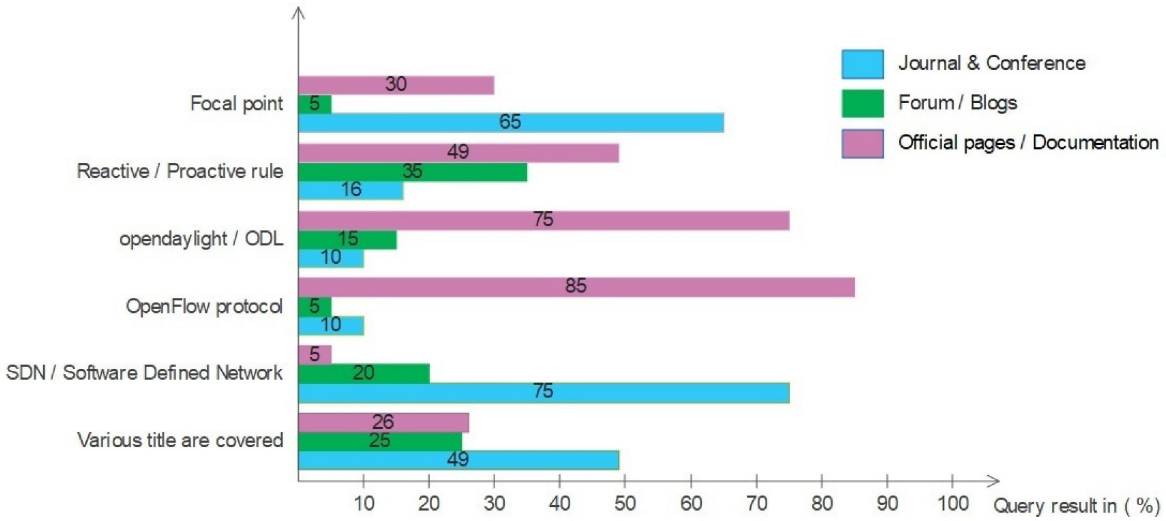


Figure 1.2: Methodology we used to find the gap

In this research, we obtained research paper, journal, and conferences primarily used from publication sites: IEEE Xplore Digital Library [4], ScienceDirect [5], SpringerLink [6], ACM Digital Library [7], Google Scholar [8]. he method we used in this research shown in Figure 1.2

1.7 Significance of the Study

The problem as stated in section 1.3, the choice is bandwidth or latency; we will have to sacrifice one to get another. It implies every task have their requirement, for example, in video streaming environment, Voice Over Ip (VOIP) service, chat services, and teleconferences needs cautious latency, on other hands file-sharing site needs reasonable bandwidth requirement. To fulfill requests of every service's claim must be filtered out to serve as it is. Collectively, the study gives in data center and cloud environment an essential behavior. In cloud networking, every tenant has their demands based on what service to use. Therefore, it should give what they want. Due to this, OpenStack, as a cloud OS architecture; Infrastructure As-A-Service

(Iaas) will benefit from this work. Their rules and regulations offer to serve based on user demand to fulfill, prepared Service License Agreement (SLA) accordingly with respect to standard ethical conduct.

1.8 Concerns for Ethical Conduct

This work is unreservedly thanks to fellow researchers who dedicated their researches to the open-source community and makes widely available as open technology. Moreover, it promotes the work done by the other researchers by giving relevant links and reference to the site where it resides. The writer tries to work to amend based on the previous researchers to promote. Further enhancement and contribute the community diligently by demonstrated valid findings without falsification, abundantly, and mischief. Also, to open future ideas and development by conduct extensively. In this research, we try to be honest and in real-life problem solve by conducting real problems in a real environment apart from a simulation.

1.9 Delimitation of the Study

The proposed solution could be used as a testbed to show the integration of ODL and OpenStack, and evaluate the result of the performance after extensive testing on ODL. This solution is assumed to work over a wide range of areas like data center, public or private cloud computing, middle-scale or large scale organizations, universities, and research institutes. The research is done under an SDN environment, not under an OpenStack. There are some boundaries put to the research, a single controller instead of multiple controllers, and specific only to the ODL controller. There are dozens of SDN controllers in the market, but our research is conducted based on the selected controller after carefully analyze in Table 2.1.

1.10 Limitation of the Study

Under this research, certain constraints that the writer is unable to show here due to the nature of work needs many computing resources. Some of the limitations faced: as follows: *(1) Due to lack of an OpenFlow device, obligated to work on an Open Virtual Switch (OVS). (2) The bench-marking tools are not availability in freely and old version during testing phase. (3) The circumstance is limits our finding only based on focusing on reactive and hybrid flow mode. Proactive flow mode testing is limited only memory usage in actual device. The downside of this cause also needs more time frame to measure the parameter. In this work, we include a*

test based on previous conducted result. Categorically, the work done in this scope is focused on primarily specific use case.

1.11 Assumption of the Study

The work proposed in this research is assumed to work on any infrastructure since the main aim of the research as stated in section 1.5. SDN can operate as any of the legacy network functionality. The most compelling scene was to apply this to a data center, cloud computing environment, web hosting services, enterprise area, governmental institution, Software-Defined Wide Area Network (SD-WAN), and for others who use many computing devices will benefit from this paradigm. The cost of the device, as well as migrating to this paradigm, may be found to be cost-ineffective. The operation and use of this technology need professional knowledge as it is not ready to use for the day-to-day home or small network. Nowadays, there are other options for small offices and homeowners who care less about spending, but more about technology. Internet-connected smart devices have revolutionized the way to live and work, some security and privacy concern issues that come with it. Therefore, it is recommended to use a cloud-based firewall [9]. The solution proposed aims at helping OpenStack cloud infrastructure integration and how an Openstack environment benefits from this work — the research has done to work with a heavy workload and a very demanding infrastructure that grows exponentially.

1.12 Organization of the Paper

Chapter one discussed the introduction, the motivation, statement of the problems, the research questions, the proposed solutions, the scope and limitation/delimitation, the objective of the study, methodology, and the use cases of application results. The next chapter presents about literature review. It includes a systematic review of paper, an overview of flow writing modes, types of Network operating systems, technology stacks detail architecture and modules, and related works. Chapter three discusses the research methodology and proposes a solution, data collection method, development methods and tools, techniques, and design of the algorithm. Chapter four discusses the proposed implementation and the role that we have made a modification of the hybrid system to be implemented. The fifth chapter describes experiment environments setup, results, and analysis as well as performance evaluation. Chapter six concludes the research based on the findings and proposes future work.

CHAPTER TWO

Literature Review and Related Works

2.1 Introduction

In this part of our proposed work, we find the much-related research papers based on our areas of interest and then study carefully to find the whole system and gap. The whole paper focused on controller performance and OpenFlow rule related issues, which is reactive and proactive flow mode. After looking at the gap seriously, it is a happy ending to draw a conclusion based on the narrow or specific parts of the problem. The SDN framework have a well organized, standardized and frequently revising architecture.

2.1.1 Software-Defined Network and its Architecture

The separation of the data and control planes is undoubtedly one of the essential principles of SDN—and one of its increasingly disputable protocol [10]. Even though it is anything but another idea, the contemporary perspective makes them intrigue bends on an old thought. How much distance away the control plane can place from the data plane?. How many instances expected to exist to fulfill high-availability and resiliency prerequisites. Whether a hundred percent of the control plane can be migrated further away than a couple of inches are on the whole seriously discussed. We like to move toward these thoughts by considering them in a continuum of conceivable outcomes extending between the least difficult, being the accepted with complete dispersed control plane. The semi-or legitimately unified control plane, to at last that design is carefully brought together. Figure 2.1 represents the range of alternatives accessible to arrange layers and a portion of the upsides and downsides of each approach.

SDN Applications are programs that unequivocally, programmatically, and directly convey their desired network behavior and network requirements to the SDN controller through North

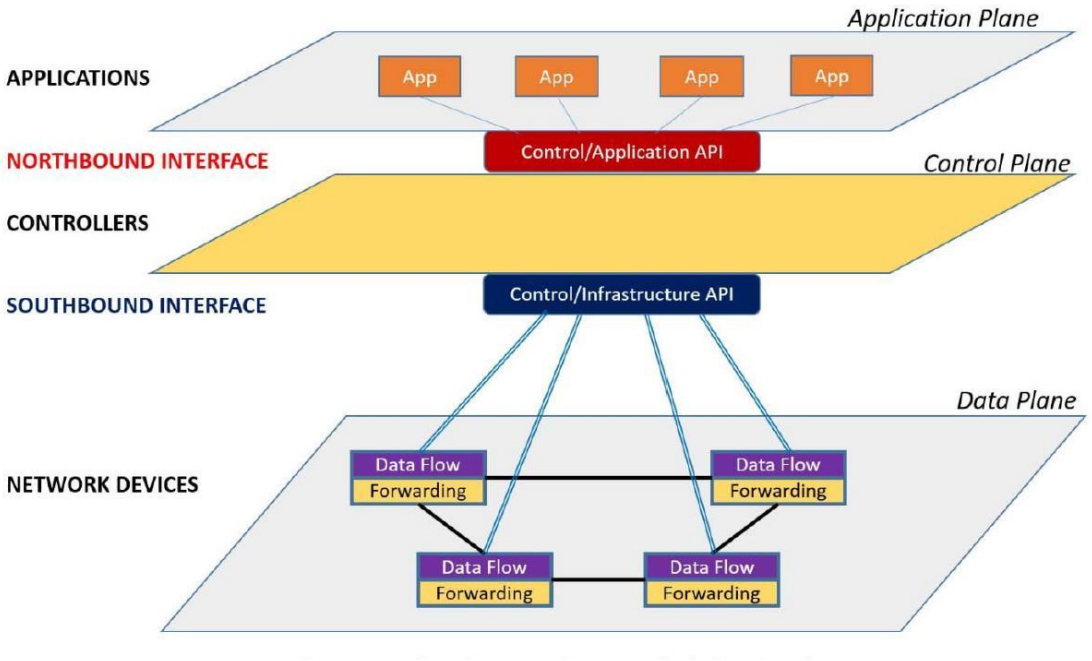


Figure 2.1: Software-Defined Network architecture, source adopted from [11]

Bound Interface (NBI) [2]. Furthermore, forwarding plane advances traffic to the next hop along the way to the selected destination path as per control plane logic. Data plane packets go through the device, switch/router. In this plane, it communicates with the controller using South Bound Interface (SBI) protocols like OpenFlow and a dozen of the non-OpenFlow protocol. SDNs NBI is interfacing between SDN controllers and SDN applications and generally give abstract network view and empower direct expression of requirements network behavior. May those occur at any level of abstraction and crosswise over various arrangements of functionalities. The SDN value lies in the desire that these interfaces have accomplished in an open, non-vendor-specific, and operational friendly way. The control plane is play the orchestration between data and application layer.

The implementation of the SDN control plane can pursue a centralized, various hierarchical, or decentralized structure. Control plane proposals focused on a centralized methodology. Where a solitary control substance has a worldwide perspective on the network. While this unravels the use of the control logic, it has scalability confinements as the size and components of the network to mitigate the scalability issue in the controller. A few methodologies have proposed in controller approach, fully distributed approaches and hierarchical approaches. In hierarchical arrangements distributed controllers work on an apportioned network view, while a consistently concentrated root controller takes choices that require organize network-wide

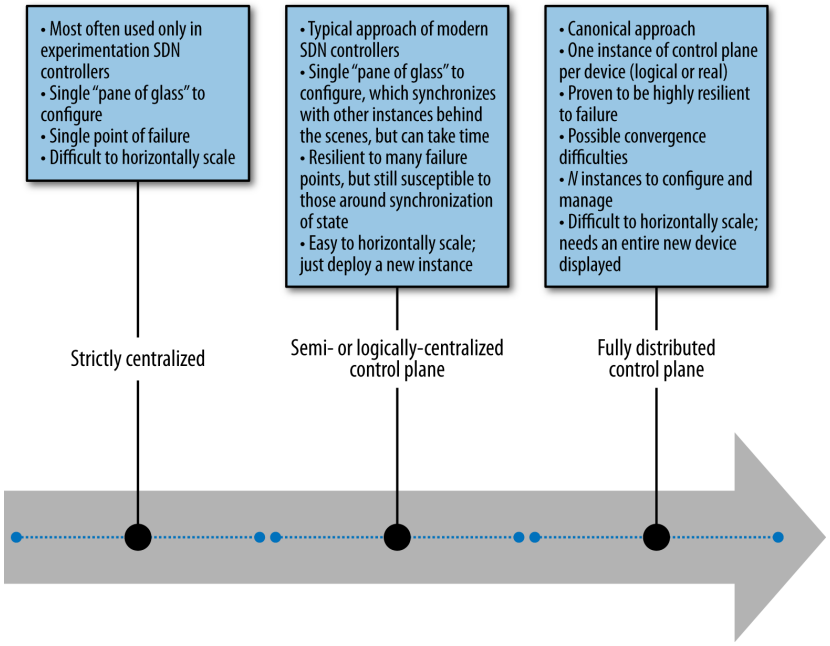


Figure 2.2: Spectrum of SDN controller placement, source adopted from [10]

information [12][13]. In distributed approaches, controllers deal with their nearby view, or then again, they may exchange synchronization messages to their insight [14][15]. Conveyed arrangements are increasingly reasonable for supporting versatile SDN applications, also the spectrum of the controller shown in Figure 2.2. OpenFlow specifications 1.0-1.5 [16] are listed by Open Networking Foundation (ONF). Many existing and newer protocols are work in control plane, instead of decentralized.

The control plane supports almost every existing legacy networking protocol, like Open Shortest Path First (OSPF), Multi-protocol Label Switching (MPLS) or Border Gateway Protocol (BGP). A custom protocol or newly derived protocol without compromising the ecosystem for fine-grained integration. The controller does give leading network functional service, and on top of that, derived services may thrive on getting most out of them. For the sake of comparison, we detailed the controller's lists in Table 2.1 — furthermore, this concise and refined comparison taken from [17]. OpenFlow protocol is the standard framework between data and control plane abstraction of devices.

A communication standard interface oversaw for the ONF that utilized in the middle of the control plane and the data plane. In the SDN network permits the setup of forwarding devices such as switches and routers, the concept of this protocol depicted in Figure 2.3. This protocol gets rid of the issue of having a static network architecture. The OpenFlow (OF) protocol,

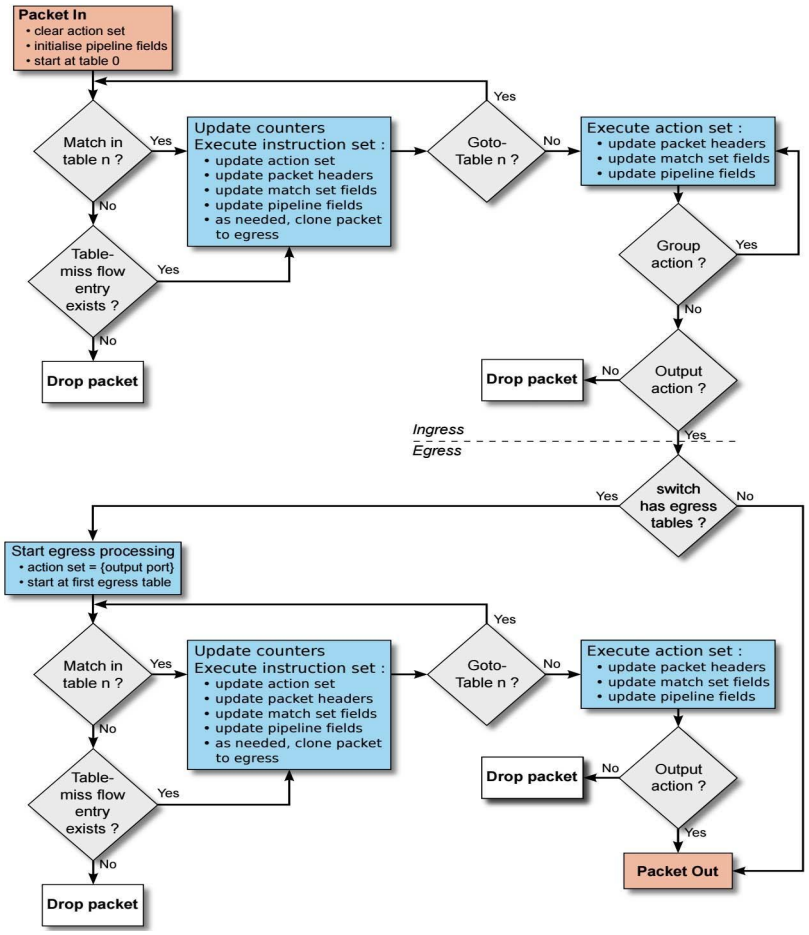


Figure 2.3: OpenFlow protocol detail approach, source adopted from [10]

it is conceivable to make a separate network control policy that can be spread through the whole network, enabling a central controller to remotely deal with the forwarding data in all the device of the data plane [16, 18]. It is a usual approach since this makes the network more dynamic, wiping out the issue of designing every one of the devices and interfaces individually one by one. Another advantage is that it will not vary from a vendor to another, making the procedure simpler. In order to make OpenFlow, a standard protocol that meets the demands of commercial deployments, the Open Networking Foundation (ONF) organization has published several versions of OpenFlow. Furthermore, the OpenFlow protocol overall design, feature, and summarization of the features of versions 1.0–1.4 specification defined in [19],[20]. It gives an overview of current SDN applications as organized by related fields in detail. Moreover, it offers architectural design options for SDN in OpenFlow compliant. OpenDaylight controller is achieving these approach in practical delivering open SDN features than available alternatives listed in Table 2.1.

Table 2.1: SDN Controller's Feature Comparison Table

Name	Language	Arch.	NB API	SB API	EW API	Platform	Interface	License	M.T	Modularity	Consis.	Doc.
Beacon [21]	Java	Centralized	ad-hoc	OF 1.0	-	Lin,Mac,win	CLI, web UI	GPL 2.0	Yes	Fair	No	Fair
Floodlight [22]	Java	Centralized	REST, java Remote Procedure Call (RPC), Quantum	OF 1.0,1.3	-	Lin,Mac,win	CLI, web UI	Apache 2.0	Yes	Fair	Yes	Good
Flow Visor [23]	C	Centralized	JSON RPC	OF 1.0,1.3	-	Linux	CLI	Proprietary	-	-	No	Fair
Maestro [24]	Java	Centralized	ad-hoc	OF 1.0	-	Lin,Mac,Win	web UI	LGPL 2.1	Yes	Fair	No	Limited
NOX [25]	C++	Centralized	ad-hoc	OF 1.0	-	Linux	CLI, web UI	GPL 3.0	Yes	Low	No	Limited
ONOS [26]	Java	Dist. Flat	REST, Neutron	OF 1.0,1.3	Raft	Lin,Mac,win	CLI, web UI	Apache 2.0	Yes	High	Yes	Good
OpenContrail [27]	C, C++, Python	Centralized	REST	BGP, XMPP	-	Linux	CLI, web UI	Apache 2.0	Yes	High	Yes	Good
OpenDaylight [28]	Java	Dist. Flat	REST, Netconf, java API	OF 1.0,1.3	Akka, Raft	Lin,Mac,win	CLI, web UI	EPL 1.0	Yes	High	Yes	Good
POX [29]	Python	Centralized	ad-hoc	OF 1.0	-	Lin,Mac,win	CLI, GUI	Apache 2.0	No	Low	No	Limited
Rosemary [30]	C	Centralized	ad-hoc	OF 1.0,1.3, XMPP	-	Linux	CLI	Proprietary	Yes	Good	No	Limited
Ryu [31]	Python	Centralized	REST	OF 1.0-1.5	-	Linux,MacOS	CLI	Apache2.0	Yes	Fair	Yes	Good
Trema [32]	C, Ruby	Centralized	Ad-hoc	OF 1.0	-	Linux	CLI	GPL 2.0	-	Good	No	Fair

2.1.2 OpenDaylight Controller and its Architecture

The ODL controller is a commercial, collaborative across multi-vendor, open-source platform to accelerate the adoption and innovation of SDN and NFV. ODL is a modular system in nature, so each module and bundle can interact without knowing the detail of the object. Modularity comes at the expense of Open System Gateway Interface (OSGi) nature, and Model-Driven Service Abstraction Layer (MD-SAL). MD-SAL can hold data to be kept centrally in configuration and operational data as a producer/consumer fashion. The above represents a feedback loop for application to observe the state of the network/system with the help of the Yet-Another Next Generation (YANG) modeling definition.

2.1.3 OpenDaylight Technology Stack.

In this section, in-depth structure and overview of the ODL SDN controller's technology stack. Different technologies are used to compose across modules to realize pure SDN features in ODL. To highlight the technology stack used in ODL, MD-SAL is a central concept in ODL, and the rests are listed in Table 2.2.

Table 2.2: Some technology stack using in ODL controller

Technology Name	Description
Java	Programming language
Maven	Project management
YANG	Data modeling language
OSGi	A framework for building modular systems,
Apache Karaf	lightweight OSGi container for loading bundles/modules between different services and deploying to the SAL.
Blueprint Container	Dependency injection across bundles runs in an OSGi framework.

2.1.3.1 Model-Driven Service Abstraction Layer (MD-SAL)

In the ODL project, MD-SAL plays a core functional role. It helps in interfacing various layers and modules through a well-defined API. The MD-SAL uses YANG as the modeling language for interface, data definition, and messaging and data-centric run time for such services that depend on YANG modeling. YANG tools are used to compile the YANG template

Table 2.3: Detail differences of AD-SAL and MD-SAL, source adopted from[33]

AD-SAL (API-Driven SAL)	MD-SAL (Model-Driven SAL)
APIs, request routing between consumers and providers, and data adaptations are all statically defined at compile/build time.	APIs, request routing between consumers and providers are defined from models, and data adaptations are provided by 'internal' adaptation plugins.
Both NB/SB APIs even for functions/services that are mapped 1:1 between SB/NB Plugins	Allows both the NB/SB plugins to use the same API generated from a model. One plugin becomes an API (service) provider; the other becomes an API (service) Consumer.
Dedicated REST API for each NB/SB plugin.	Common REST API to access data and functions defined in models
provides request routing (selects an SB plugin based on service type) and optionally provides service adaptation, if an NB (Service, abstract)API is different from its corresponding SB API.	provides request routing and the infrastructure to support service adaptation, but it does not provide service adaptation itself; service adaptation is provided by plugins
Request routing is based on plugin type: the SAL knows which node instance is served by which plugin, and when an NB Plugin requests an operation on a given node, the request is routed to the appropriate plugin and node.	Request Routing in the MD-SAL is done on both protocol type and node instances, since node instance data is exported from the plugin into the SAL
AD-SAL is stateless	Statefull
Limited to flow-capable device and service models only.	Model agnostic. It can support any device and/or service models and is not limited to flow-capable device and service models only.
Usually provide both asynchronous and synchronous versions of the same API method.	Service model APIs only provide asynchronous APIs, but they return a 'java.concurrent.Future' object, which allows a caller to block until the call is processed and a result object is available. Same API can be used for both synchronous and asynchronous approach. Thus MD-SAL encourages asynchronous approach to application design but does not preclude synchronous applications.

and produce java classes/interfaces, and consequently construct a REST API Doc Explorer. Besides, MD-SAL data stores can be used to store data generated by models. MD-SAL storage is used to exchange data between the providers and consumers with respected logic [34]. The adjustment in to usefulness is not a piece of MD-SAL; It is given by modules to play out a model-to-show mapping between two APIs. Henceforth, MD-SAL does not have any explicit module APIs; this infers its capacity to adjust any SBI or NBI plugins loaded into the ODL. Full features of MD-SAL are described in Table 2.3. The rests are well supported by related works.

2.2 Related Works

In recent sections, we tried to investigate the technology behind the framework of the ODL controller. In this part how the research world trying to accomplish the mentioned problem in recently. The researcher proposes a method to use the SDN control channel and controller resources efficiently while protecting the controller from potential control channel floods [35]. This method presents the OpenFlow switch to forward the first packet, only belonging to a flow. Meanwhile, the rest of the packets belonging to the stream are buffered until the controller responds. Although the proposed method implemented using OVS with Data Plane

Development Kit, it can also apply to any available OpenFlow hardware switches. So the research presented an excellent opportunity, but how much it affords the amount of data that keeps in memory expensive architecture? It should be addressing this issue when the packet growth instantaneously. In our proposed work trying to avoid memory consumption by keeping unrelated service to use reactive flow, not to use the flow for longer time.

As a standard mechanism to reduce packet-in packet size in a single controller environment. To mitigate the SDN controller's load, who proposed a plugin to reduce the packet's size, called packet-in Buffer Plugin (PIBP) [36]. This plugin has located between OpenFlow switches and SDN controllers. As opposed to the standard SDN operation processes, switches initially send packet-in to PIBP, when PIBP receives a packet-in packet, it will reserve raw packet data and only send packet-in's header to the controller. After the controller completes its processing, it sends the packet-out message back to PIBP to regroup. Finally, PIBP sends the packet back to the OpenFlow switch. The above mentioned work violates the SDN architecture and s too much cost ineffective, to solve this issue we reduce the packet-in message to controller by maintain related service flow keep in memory for longer proactively.

The approach using buffering of the first arrival packet-in message of original raw data to the controller buffer proposed in work [37]. Moreover, it gives the buffer-id and then returns to the switch with flow-mod. The deducted and minimized packet circulates the whole intermediate switch until destination switch is the destination, that is, the last switch, then the controller, it looks the buffer-id and amends the raw data and sends it to the destination switch. Of course, this method reduces the packet size to float a small amount of information to mitigate the load between switch and controller, packet processing time [latency], and bandwidth requirement optimization. However, what is the critics? Every intermediate switch located in between sender's and receivers should be intercommunication with the controller, it also made a zigzagging scenario around in critical nanoseconds of time environment. The above approach it costs overall latency of the packet arrival, we proposed solution does not get involve in intermediate switch, the controller take action based on available topology by itself.

The interested solution appears to presents and compares the locally and cloud-hosted network controller. That override the limitation of the locally hosted controller and showed greatness in many factors of scalability, reliability; to enhance processing demand of the packet header for

a different reason for the additional application concept, for Quality of Services (QOS), load balancing, a large volume of data processing, firewall and so on [38]. A Cloud-based controller is much more powerful, and it takes the full advantage of cloud computing characteristics offers, instead of putting high end IT infrastructure to simple network solutions. On the other hand, the locally-hosted controller has it is edging than a remotely-hosted controller because it does not depend on remote support. The main disadvantage of the cloud-based controller is not much feasible. At this moment, the reliability of the network not guaranteed everywhere. Soon, if we lose connectivity to the remote controller, the whole system will disrupt unless a backup solution. In this solution the reactive flow will suffer a lot, the main aim is to offload the processing burden put on the controller. As a matter of fact the above issue, packet-in is the primary concern regards to computing power. Packet-in is reduced to solve the processing power bottleneck as described in our hybrid solution.

2.2.1 Summary of Related Work

Table 2.4: Short Notation of related work

Paper	Findings/contribution	Critical Remarks
[35]	The method presents the OF switch to forward the first packet belonging to a flow; meanwhile, the rest of the packets buffered until the controller responds.	Expensive memory architecture.
[37]	Deducted and minimized packet circulates the whole intermediate switch until destination switch reached, that is the last switch.	Additional controller overhead cost, flow table does not get an extra edge.
[36]	Proposed a plugin to reduce the packet size, called Packet-in Buffer Plugin (PIBP). This plug-in located between OpenFlow switches and SDN controller	Cost of time and additional resources, also out of SDN standard.
[38]	It hosted the controller in the cloud. For the benefit of controller processing power, high-end services running in remotely like firewall	Reactive flow entry got to suffer in each time flow rule request, if the connectivity loss, the reactive mode does not work completely.
[39]	Proposes an in-switch dynamic flow aggregation (IDFA) mechanism which can be dynamically triggered. Redundant flow entries inserted to speed up flow aggregation convergence time, it resides in an OpenFlow switch.	Switch has additional work to do by compressing entry in TCAM, and every time flow-mod happen, based on a dynamic threshold. It added extra time of recalculation workload in the switch.

2.3 Knowledge Gap

The way we look after the study, we have identified every work we have investigated. So, these gaps are listed as follows; (1) when the flow model using a proactive approach. Practically does not exist almost everywhere; it is difficult to find proactively as it the only way to work fine because of the limitation of the hardware capability, even virtually every SDN. The controller comes up with reactive mode pre-configured, for example, in ODL controller, (2) we conduct our efforts on reactive flow mode. Because the only chance is in practical works excellent in a currently available device without any limitation of hardware capability, but the performance of the network is slow down in advance. The method downside, that the first arrival of flow, to wait until getting a response from the controller, to find where about to go. Moreover, this causes the forwarding devices buffer may run out of memory. The controller may be crowd by request and drops the appeal. As a result, the switch/router resend the request. After some idle time, the rule has expired and request again. So, there are options, of course, to combine these two methods. In the next chapter, we try to illustrate hybrid flow mode methodology, proposed design methods, research strategy, and development tools and techniques.

CHAPTER THREE

Hybrid Flow Mode Design and Methodology

3.1 Introduction

To maintain how the method for organization, simulation, and examining and comparing the available tools that we reviewed. We analyzed the past overview and kept up the good behavior. Searching the best fit technology and distinguish which method to choose for our preferred methodology. The result gives us the best output and flexibility for our development and design strategy. During the earlier stage of the research phase, we used the scientific research review technique to obtain relevant information by reviewing the paper and defining the gap. We collect the resource from different sources to gain an extra edge. In our research, we followed the inductive (bottom-up) approach to conduct, and quantitative research mechanism. To work with this previously done research, we typical emphasis on the existing use case thoroughly. There are lots of use cases in our research to conduct; cloud infrastructure like OpenStack, internet service provider, and data center to name. In this step, a critical scientific paper review is the primary source of our use cases. Additionally, on top of the paper review, a different source of data is used, and this approach is found in Figure 1.2

3.2 Philosophy of a Study

Currently, in SDN, there are two kinds of primary flow processing modes to forward the packet that defines how incoming flow of data to handle, those are reactive and proactive flow modes. Hybrid flow mode is an implementation based on reactive and proactive modes which are designs depending on per service requirements and policies.

3.2.1 Reactive Flow Mode Setup

In this mode of flow setup, the first packet of the first flow is compared/searched within the entire flow table against the rule defined by the central node to forward the incoming data flow. Packet-in event is triggered if a rule is not present, the packet-in forward to the controller to determine the path of the destination. Then the controller decides and returns with the route and writes back to the device packet-out/flow-mod message for packet destination to serve as future reference. In reactive mode, the controller has the power to enforce logic than the proactive since switches are merely dumb and not capable of intelligence.

3.2.2 Proactive Flow Mode Setup

By convention, all flow rules are written on the device rule table predominantly, and there is no packet-in message to the controller. In this case, the packet forwarding is so fast to ensure latency, but the device memory falls short due to device incapability. In short, the entire rule is pre-populated before the packet has arrived at the forwarding device. In the real world, it is not feasible to use this mode.

3.2.3 Hybrid Flow Mode Setup

In this flow setup mode, the developer will decide which mechanism of flow mode to which network mode and segment to write a static rule and dynamic rule based on a business model of the organization as per requirement. It was better to be flexible and perform better than the two-mode if implemented clearly. By default, almost every NOS available on the market, neither open source nor commercial, has to be reactively configured. It is the best way to get out of these two methods refined service, from reactive and proactive. Thus, we aim to build a solution to overcome the problem mentioned above.

Applications can also implement a hybrid approach where some traffic handled proactively, and some managed reactively. A "true" proactive use, it can not learn anything about the flows in the network. Besides, only what found in the flow counters, such as from an orchestration system. In practice, few applications are likely to implement a completely pure proactive model [40].

3.3 Research Strategy

This section shows how advanced system solution is designed and implemented, besides of this, how to analyze the research based on the previous conducted research and currently available technology. At this stage of the study, all mechanics of every step and procedure will clearly be defined based on the identified problem domain and characteristics. So in this step, we have demonstrated a method of approach, flowchart of the design system, the proposed system test, and evaluation mechanism, comparison of the reactive and hybrid system in action, and defining the tools we used in every step. The tasks mentioned-above in the research shows some brief steps that we get around to the strategy of our research.

- The first and for most important step, we clearly defined the identified problem to be solved.
- We identified the parameters to design the proposed solution.
- To collect the relevant resource accurately by analyzing written articles, research papers, a dedicated forum about the problem we identified, visiting official sites of the technology to look around specifications, methodology, technology stack, conventions, and archetype. Additionally, some influential books written by experts and authors are reviews. After all of these, some available experts on the Internet Relay Chat (IRC) channel to discuss the best possible ideas around.
- We identified the parameters to design the proposed solution.
- Implement the proposed solution.
- Prepare for the conduct of the experiment using different mechanisms that suit the problem we identified and document the result we obtain from the analysis.

Analyze the result to check what is wrong or whether the intended task works or not and look in detail of our conclusion for our algorithm. Our algorithm may trigger another issue that needs a solution for the future that we clearly show this without any hesitation to fellow researchers and reviewers ethically manner.

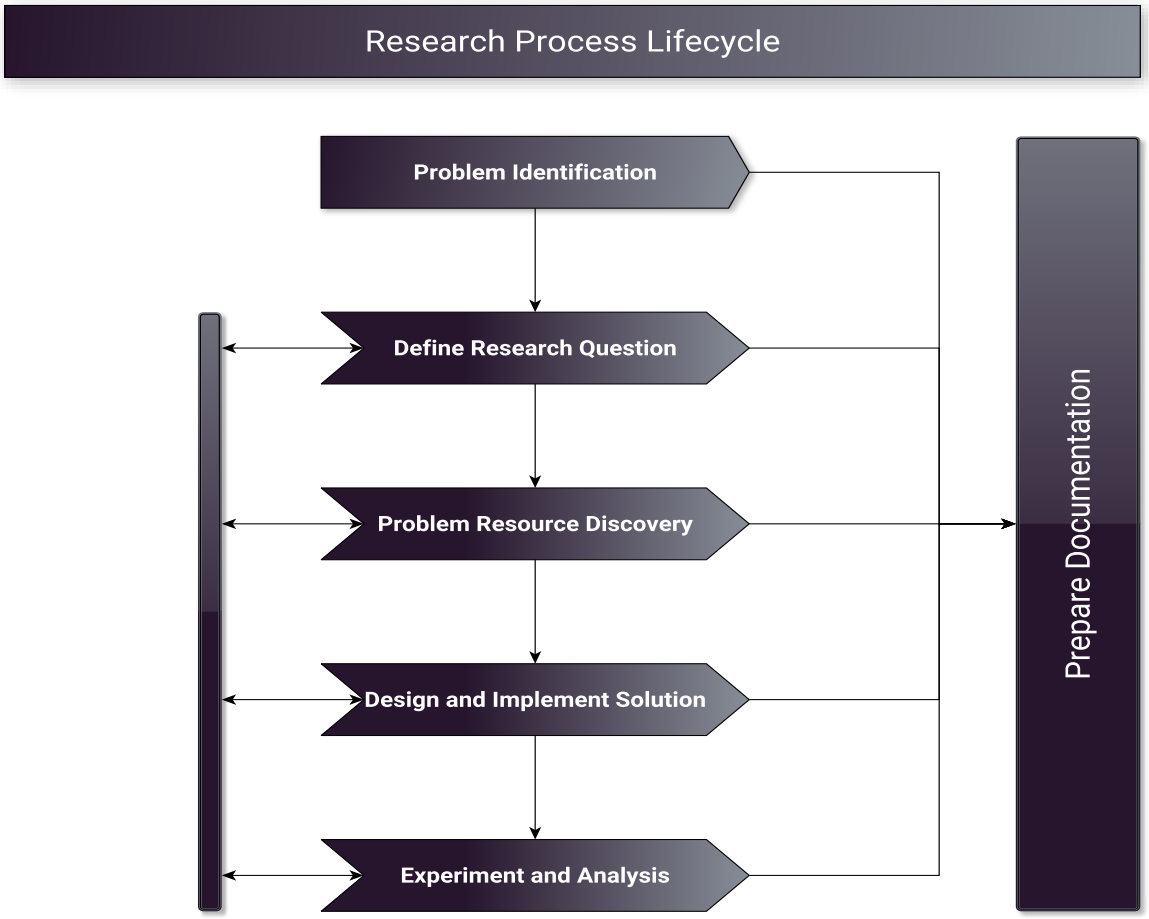


Figure 3.1: Research process Life-cycle

3.4 Data Collection Instrument

In this research, what we are going to do is to focus on various case studies. In this particular research, we use data based on existing use cases. Precisely, in our case, it is on the public and private cloud service provider, data center, medium and large scale enterprise, universities, a research institute. For this matter, we followed the pattern in our case study to conduct by looking at the exact method and previously published journals and to analyze and make a decision afterwards.

3.5 Proposed Solution

The solution of defined problems in here by combining the reactive and proactive flow setup mode, in simple term hybrid mechanism or static/dynamic flow rule in the SDN-enabled switch/router in networking device, by the mechanism of static/proactive flows, will be set up in the same subnet of the network segment. Outside the subnet, it will use a dynamic flow

setup mode. In this step flow programming, the module controls which device resides in which subnet and writes the rule accordingly. Let us assume that hosts A and B is located on the same subnet. Furthermore, host C is located on a different subnet. Here the scenario is if host A wants to send a packet to host B, then the controller extracts a packet header and detects the sender's/receiver's address to decide a subnet. Since switch S1 is the interconnection between the hosts A, B, and C, so the rules incorporated in host A, and B wrote in the static rule. Furthermore, reverse flow rules written accordingly. If A wants to send data to host C, then the rules are written in reactive mode because the subnet is different from A and B. It expires based on specified time-out values. A $\rightarrow$ B rules stored in switch S1 is permanent, unless we provide different mechanisms to eliminate unused or idle rule for a different logic defined in the central node. A straightforward example is when a node is lost/detached from the switch, the Link Layer Discovery Protocol (LLDP) gets device failerity during a routine check and triggers every rule incorporated to the detached device removed from every switch in the network.

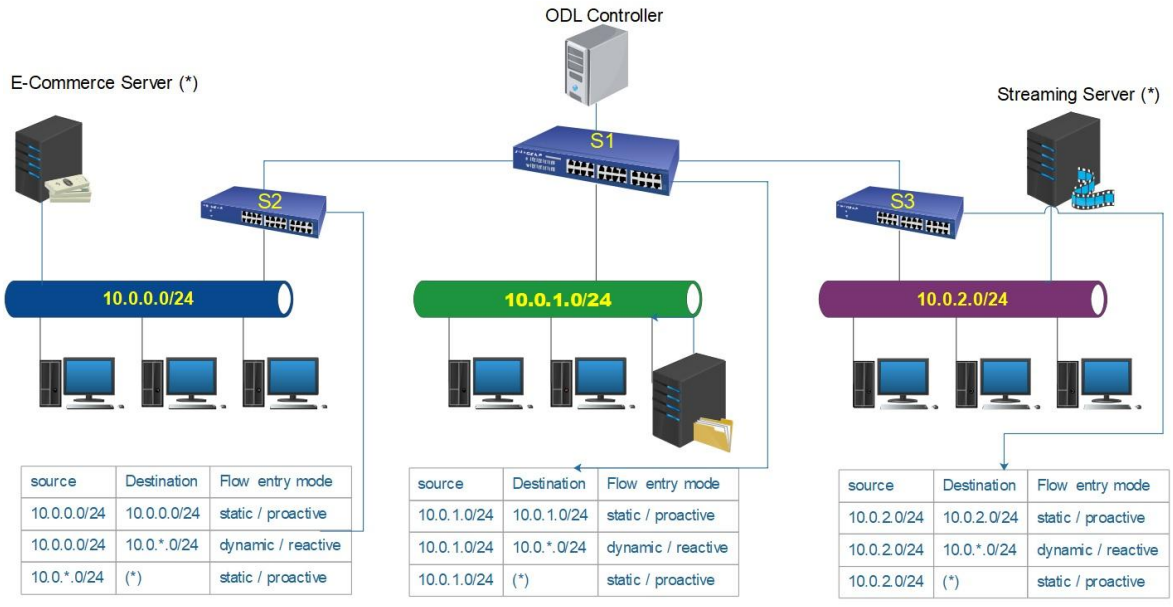


Figure 3.2: Proposed Research Model

Besides this, the rule defined permanently has not been kept for long unless it is acting upon, so we should find another solution to counterfeit the unused rules or not frequently-used rules based on statistical data analysis. OpenFlow table has a counter matching entry. Categorically matching action and counter fields in the attribute. The counter entry is subject to hold how much data transmitted using a single rule. Based on the counter we designed algorithm to

remove the rule, we call it undesirable entry, this algorithm run based on the predefined time interval and later administrator may change the range as they wish. Furthermore, there are other aspects to enhance the hybrid solution to make it viable. In addition to subnetting, another expansion is, if there is mission-critical application/service running hosts can be written the rule proactively by triggering the setting once, because the host may reside on another physical location or limitation of subnetting members.

3.6 Development Tools and Techniques

As previously defined in the gap and methodology, We use different tools to implement the thesis and for testing purpose. The following are some of them in the consecutive subsection.

3.6.1 Resource Emulation and designing tools

Table 3.1: Some technology stack using in OpenDaylight controller

Ref.	Tools	Description
[41]	Mininet	Simulation of network topology
[42]	OVS	A multilayer virtual network switch
[43]	Oracle VirtualBox	Data modeling language
[44]	IntelliJ IDEA Comm. Ed.	Java programming language editor (IDE).
[45]	ODL	Open-source SDN controller.
[46]	Gnuplot	Command-line driven plotting program.
[47]	Edraw Max	Designing diagram
[47]	Latex	Document processor
[48]	OpenSuse Leap	Open-source linux Operating System

3.6.2 Algorithm

The proposed system works by borrowing the two mechanisms, a method using reactive and proactive as we mentioned to eliminate the limitation of TCAM in OpenFlow switch in SDN, the flow chart Figure 3.3, shows in the clear chart. The first system startup, the SDN controller, writes the initial flow for switches to know-how about the address of the controller and related attributes. On the other hand, the device wants to communicate with others; it requested a

packet-in message to the controller. The packet is processed, and the segment is determined. The segment is the same, the flow mode is categorized as proactive to eliminate the request from the same segment in the future, as discussed in section 1.3. The segment is different, reactive flow mode will be executed. However, these proactive flow entry examined later based on specific time intervals, which is predetermined by the administrator. Based on usage analysis, it will get rid off these static flows at a certain point.

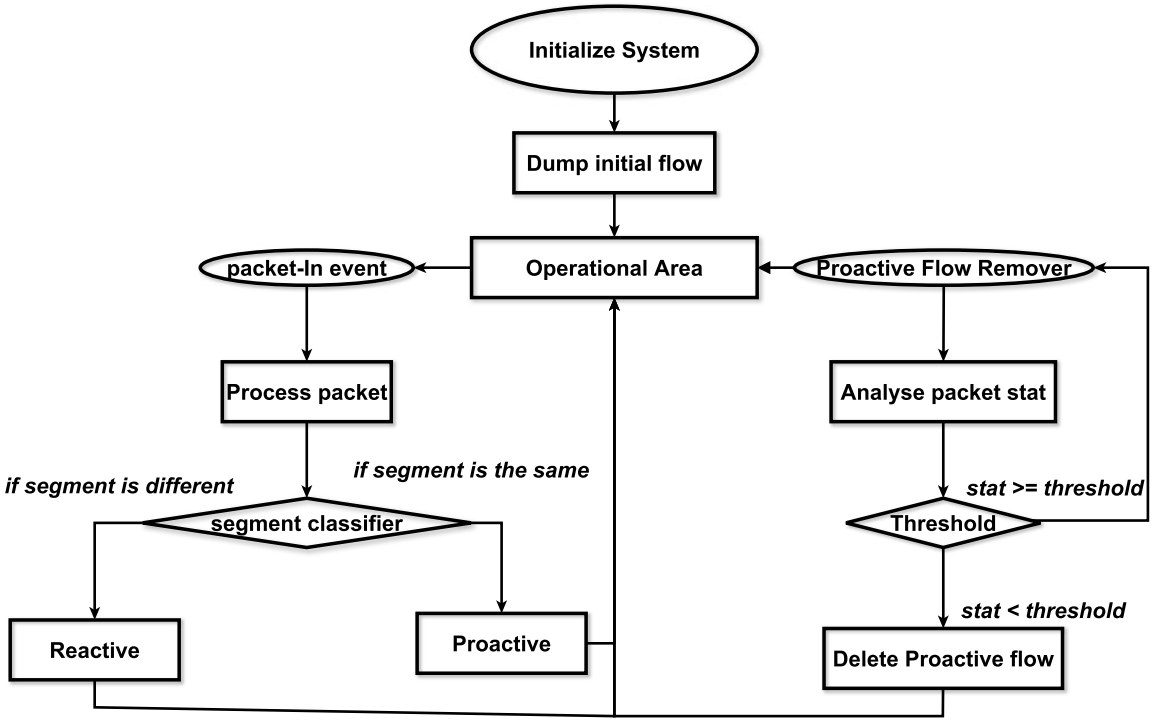


Figure 3.3: Proposed System Flow Chart

Snap of the algorithms will look like the following, which identifies a segment of the network. Besides, list a mission-critical and latency-sensitive application/service — the next part details on algorithm description.

3.6.2.1 Algorithm in Depth

- Initialize the object and related fields, then write rules, if the condition holds match value.
- Load main configurations in L2SwitchProvider class from a file; hybridcondition is one of this entry to compare.
- The value of hybridCondition is 2, initializes flowIdleTimeout and flowHardTimeout members. Variable to flowWriterServiceImpl object from mainConfig values.

- HybridFlowWriter: this class object created using InventoryReader and flowWriterServiceImpl as an argument, then register as notificationListener to act upon when switch adds/remove notifies from system inventory.

L2SwitcMainProvider: is a class that loads when the controller starts and employing the "Blueprint container" as initial dependency injection with a parameter of DataBroker and SalFLowService with init() method call binding. In the constructor space, all class-dependent variables initialized, refer to Algorithm 3.1.

Algorithm 3.1 L2Switch Main Provider Algorithm

```
1:  procedure INIT()                                   ▷ called at startup
2:      condition ← loadConfig                  ▷ loading config. parameters
3:      if condition ≡ 2 then
4:          idleTimeOut ← hybridIdleTimeOut
5:          hardTimeOut ← hybridHardTimeOut
6:          HFW ← init()
7:          reg ← HFW                    ▷ Register as notification listener
8:      else
9:          idleTimeOut ← reactiveIdleTimeOut
10:         hardTimeOut ← reactiveHardTimeOut
11:         RFW ← init()
12:         reg ← RFW                    ▷ Register as notification listener
    End procedure
```

FlowWriterServiceImp: initializing FlowWriterServiceImp class object, the actual flow writer algorithm resides in this class. DataBroker and salFlowService parameter passed through the class constructor argument, and load InitialFlowWriter mainConfig to dump initial flow like send to the controller rule and drop the packet, refers to Algorithm 3.2.

Algorithm 3.2 MAC-to-Ip flow Flow Writer Service Implementation Algorithm

```
1:  destIp ← IpExtract
2:  procedure ADDMACTOIPFLOW(srcMac,destIp,destNodeCR)
3:      condition ← loadConfig                  ▷ loading config. Parameters
4:      if destMac ≡ null then
5:          Return
6:      tableKey ← buildusingflowtableid
7:      flowPath ← buildflowpath
8:      fBody ← createMacToIpFlow(tableKeyId, priority, sMAc, dIp, destNodeCR)
9:      writeFlowToConfigData(flowPath, fBody)
    End procedure
```

Algorithm 3.3, Which provides network address segmentation to define the sender belonging to the network prefix, and it is the central part of the algorithm to decide reactive or proactive flow mode. Based on this algorithm, the network prefix once extracts from the packet header; it goes to the next part.

Algorithm 3.3 Network Address Determination Part

```
1: procedure IpPrefixToNetAddress(Ipv4Prefix)
2:     ipv4PrefixEntry ← prefixValue
3:     ipv4NetValue ← fromipv4PrefixEntry
4:     prefixLenValue ← fromipv4PrefixEntry
5:     result[] ← fromNetValue.length
6:     if prefixLenValue ≡ 0 then
7:         for all i ∈ 4 do                ▷ four octet value of network ipaddress
8:             result[i] ← 0                ▷ if prefix=is 0 ipaddress=0.0.0.0/0
9:             End loop
           return result                    ▷ returns only 0.0.0.0/0 and halts.
10:    result ← ipv4NetValue    ▷ array of ipv4NetValue value and length
11:                  ▷ below Sets all bytes after the end of the prefix to 0
12:    lastByteIndex ← (prefixLenValue − 1)/Byte.SIZE
13:    for all lastByteIndex ∈ 4 do
14:        result[i] ← 0      ▷ rest octet of netAddress=0,like 192.168.0.0/16
15:        End loop
16:    bytelastByte ← ipv4NetValue[lastByteIndex]
17:    bytemsb ← (byte)0x80
18:    intlastBit ← (prefixLenthValue − 1) % Byte.SIZE
19:    bytemask ← 0
20:    for all i = 0 ∈ Byte.SIZE do
21:        if i ≡ lastBit then
22:            mask | = (msb >> i)
23:        End loop
24:    result[lastByteIndex] = (byte) (lastByte& mask)
25:    return ipv4PrefixFor(result, prefixLenValue)
   End procedure
```

Algorithm 3.4, provides the way to write flow to the switch table entry as an input to decide the future flows handled in the rule manner, which is in the match field the source the address defined in layer two(L2) scheme and the destination match address(field) is defined in layer three (L3) address scheme. Algorithm 3.5, is packet

Algorithm 3.4 MAC-to-Ip flow writer

```
1: destIp ← IpExtract
2: procedure addMacToIpFlow(srcMac,destIp,destNodeCR)
3:     condition ← loadConfig                ▷ loading config. Parameters
4:     if destMac ≡ null then
5:         Return
6:     tableKey ← buildusingflowtableid
7:     flowPath ← buildflowpath
8:     fBody ← createMacToIpFlow(tableKeyId, priority, sMAc, dIp, destNodeCR)
9:     writeFlowToConfigData(flowPath, fBody)
   End procedure
```

receiver when the packet is sent, it filtered accordingly, and is directed to flow writer service algorithm. The packet extracts according to the protocol it belonged to, then traces with loop, and also it initializes the destMac(destination MAC address) and IP address.

Algorithm 3.5 Flow Writer Algorithm

```
1: procedure onArpPacketReceived(packt)        ▷ waiting for event to
   happen: switch packet-in
2:     condition ← loadConfig                  ▷ loading config. parameters
3:     rawPkt ← null
4:     rethPkt ← null
5:     arpPkt ← null
6:     if pkt ≡ null then
7:         return
       End if
8:     for all pkt ∈ packetReceived(getPacketChain) do
9:         if pkt ≡ instanceof(RawPacket) then
10:            rawPkt ← pkt
11:        else if pkt ≡ instanceof(EthernetPcket) then
12:            ethPkt ← pkt
13:        else if pkt ≡ instanceof(ArpPcket) then
14:            arpPkt ← pkt
15:        End loop
16:    if rawPkt ≡ null || arpPkt ≡ null || ethPkt ≡ null then
17:        return
18:    if arpPkt ≡ ipv4 then
19:        FWSI.setArpPacket ← arpPkt
20:    destMac ← getMac
21:    if destMac ≠ null then
22:        fRS.addBidirectFlows(srcMac, ingress, destIp, destNodeCon)
   End procedure
```

Algorithm 3.6 is used to remove proactive rules based on a statistical analysis of transferred packet count. It is not supposed to be live forever unless the flow is using over the threshold. This threshold determined by a business rule. Since the flow is beyond a certain threshold to pass lifeline, Otherwise the device request for packet-in the future if the flow arrives again.

Algorithm 3.6 Proactive Remover

```
1: procedure Future<RpcResult<RemoveFlowOutput>> re-
   moveFlow(DataBrokerdataBrocker)                ▷ called at
   startup
2:     nodeIdent ← InstanceIdentifierBuilder
3:     readOnlyTransaction ← dataBrocker.newReadOnlyTransaction()
4:     dataObjectOptional ← LogicalDatastoreType.OPERATIONAL
5:     if dataObjectOptional ≡ isPresent then
6:         nodes ← dataObjectOptional.get()
7:     close transaction
8:     if node ∉ null then
9:         builder ← RemoveFlowInputBuilder(flowBody)
10:        builder.setNode ← nodeIdent
11:        setFlowRef ← flowPath
12:        setFlowTable ← tableId
13:        setTransactionUri ← flowBodyUrivalue
14:        setStrict ← true
15:    for all next ∈ matchAvaileable do
```

3.6.3 Monitoring and Bench-marking Tools

The experiment environment for this study is painful as of requirement goes wild for even to measure the latency. Bear in mind, we appropriately measured to fulfill the achievement in hardware and software that we conquer, there is some software which is commercial

and open-source that are not even thinking, because of the price and resource requirement with respectively [17]. For example, PktBlaster, OFNet, and so on. Some best testing tools described below as a comparison.

Table 3.2: Monitoring and bench-marking

Ref.	Monitoring & Bench-marking tools	Advantage/disadvantage
[49]	Wireshark	Network monitoring tool
[50]	CBENCH	Its and old measurement tool, lacks in latest version of controller
[51]	WCBENCH	Extension of CBENCH using python, not compatible with OF 1.3
[52]	Enhanced CBENCH*	Derived from CBENCH, automated script code using python and drawing plot tools.
[53]	PktBlaster	All-in-one test solution, and the very best features but its commercial and not feasible to measure due to resource requirement.
[54]	HCprobe	Extension of CBENCH, additionally and reliability measurement. But its lack of operation on latest controller.

3.6.4 Data Analysis

In the previous subsections, we described the method of implementation, designing methods and tools, and so on, In this subsections, we try to discuss how the experiment and testing are going to be. After all the relevant modules and plugins are developed and deployed, it is ready to emulate the host and devices at the first stage. It then sends the generated traffic using Cbench and different selected traffic generators in the system. After all this, we collect the data based on different criteria and analyze the output of the system. In this case, our output is system performance, and latency is measured accordingly. In the SDN environment, the measurement tools have many difficulties in maintaining the exact outcome because a lot of technologies and tools apply at once. It is worth some effort to establish the game-changing techniques. As of now, CBENCH is still the real deal in this area to conduct latency. Moreover, throughput by stressing the controller environment. What to expect:

I. What is the latency threshold to be in the range?

II. Based on the hardware used and what the performance to be like is?

III. What is the unexpected result in the output?

IV. Everything is measured and documented for further analysis of manual intervention, decide which is fixable or not for re-evaluating the code and structure, if it is not proper report for a fellow researcher, it is ethically broken research.

In our case, we try to measure the latency and throughput by sending a packet-in message to the SDN controller. The controller processes the packet-in message and returns packet-out or flow-mod to the switches. In the latency case, the switch waits for a reply and send the next message. However, in throughput, the switch does not wait until the reply has arrived. The packet responses are collected and analyzes based on minimum, maximum, average, and standard deviation. We focused on only an average to calculate the result based on the packet we get from the controller.

3.6.5 Reliability and Validity

In this research work as much as we concern, we try to find the best way to solve the problem due to a focus on reliability and validity. The main issue is to ensure the above. Without these measurements, the research is not going to be the real one. In our domain, we tried to keep everything in honesty and manner the result to fellow researchers and reviewers and Investigate properly based on valid written data and analysis result in a standard format. So in this platform, we conduct on the actual platform to make things valid and easy to deliver and integrate into the system in real-time. As of now, OpenDaylight is the leading platform in terms of productivity and availability for the enterprise or personal, service providers, or production environment free of charge. Every six month have a significant upgrade and fix lots and lots of bugs, due to the technology at the early stage of a mass adaptation.

3.7 Summary

To validate our methods in our research, we conducted various types of proving mechanisms. Analysis and conclusion part of our research shows how the methods are valid and improved in aspects of the aim we described in chapter one subsection 1.5.2. Since we used the approach, a pattern of the use case followed by our hypothesis to come up with an improved solution to the existing problem.

CHAPTER FOUR

Implementation

4.1 Design Overview

After we profoundly examined the current system and the newly proposed system architecture in previous chapters, including the development method, we defined the proposed design, experiment, and test of the system in detail. experiment and test of the systems. At the early stage of the study stated in section 1.3 problem statement of the study is defined, and then in the previous chapter research methodology is presented by examining the current system in order to get an in-depth understanding of our development method, defining the design of the proposed system method, describing the proposed system's testing method, and comparing and contrasting available development tools in order to select appropriate tools for the proposed system has been described.

In this part of the research, we do have to set up the environment according to the research needs. Our research conducted in a real environment except for simulating the network resources, like hosts and switches. For this purpose, in this thesis, we used a real and actual platform to conduct our research, and it works as expected, the Network Operating System of our choice has deliberately chosen to show our research. ODL controller is the choice we have made to make it happen, the reason we choose the ODL platform is that it is the best controller available platform in the current design aspects as discussed the feature in Table 2.1.

4.2 Implementation

This thesis aims to develop one module in the OpenDaylight infrastructure, as well as to provide a solid idea for developing ODL applications/modules. OpenDaylight is a considerable controller that depends on a large number of third party technologies. Despite having some

official documentation and developer guides, it lacks related documentation that explains the whole developing process. Moreover, how does the controller internally manages everything results in a significant entry barrier for the new developer to participate, which presents many problems in understanding the whole infrastructure and, as a consequence, is developing an application in the field.

Most of the OpenDaylight documentation is specific for a concrete platform, so it does not provide a general view that is internally performing in the OpenDaylight. Furthermore, OpenDaylight is still a young controller, and as that it is continually changing, complicating the documentation process since well-trained developers are more focused on developing. OpenDaylight is aware of the entrance barrier that it presents for newcomers and has started to make some Summits are presenting low-level documentation. However, it is not enough to break that barrier. So the challenge is even running so high to involve. Before going to describe the implementation, let up mention the hardware that we use across the implementation.

```
Samsung
PROCESSOR: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz
HARDDISK: 1TB SSD
RAM: 8GB DDR3
GPU: Nvidia GK107M [GeForce GT 750M]
2GB dedicated memory
```

To make things clear in this research, we are working on the OpenDaylight controller release version, and the module is l2switch module with version l2switch-release-oxygen-sr4. L2switch, one of over 70 available official modules with it is own feature in the platform and Table 4.1 shows some of modules. L2switch modules have the following implementation bundles inside, let us look one by one.

I. arphandler: Handles the decoded ARP packets.

II. addresstracker: Learns the Addresses (MAC and IP) of entities in the network.

III. hosttracker: Tracks the locations of hosts in the network.

IV. packethandler: Decodes the packets coming to the controller and dispatches them appropriately.

Table 4.1: Some of OpenDaylight modules, source adopted from [55]

Feature Name	Feature Description	Karaf feature name
Authentication	Enables authentication with support for federation using Apache Shiro	odl-aaa-shiro
BGP	Provides support for Border Gateway Protocol (in-cluding Link-State Distribution) as a source of L3 topology information	odl-bgpcep-bgp
DIDM	Device Identification and Driver Management	odl-didm-all
DLUX	Provides an intuitive graphical user interface for OpenDaylight	odl-dluxapps-applications
Fabric as a Service (Faas)	Creates a common abstraction layer on top of a physical network so northbound APIs or services can be more easily mapped onto the physical network as a concrete device configuration	odl-faas-all
LL2 Switch	Provides L2 (Ethernet) forwarding across connected OpenFlow switches and support for host tracking	odl-l2switch-switch-ui
LACP	Enables support for the Link Aggregation Control Protocol	odl-lacp-ui
NetIDE	Enables portabilty and cooperation inside a single network by using a client/server multi-controller architecture	odl-netide-rest
OF-CONFIG	Enables remote configuration of OpenFlow datapaths	odl-of-config-rest
OVSDB OpenStack Neutron	OpenStack Network Vir-tualization using OpenDaylight's OVSDB support	odl-ovsdb-openstack
Topology Processing Framework	Enables merged and filtered views of network topologies	odl-topoprocessing-framework
VTN Manager	Enables Virtual Tenant Network support	odl-vtn-manager-rest
VTN Manager Neutron	Enables OpenStack Neutron support of VTN Manager	odl-vtn-manager-neutron

V. loopremover: Removes loops in the network.

VI. l2switch-main: Installs flows on each switch based on network traffic.

4.3 Enhancing l2switch-main Algorithm

In the l2switch module, the l2switch-main bundle is our point of interest to enhance the algorithm in a way we discussed earlier in the problem statement, to implement hybrid flow rule in OpenFlow enabled switch virtual or actual network device by combining the two methods, reactive and proactive. Further, we decomposed the module into different classes to get into the detailed code. This module has the following class before we modified.

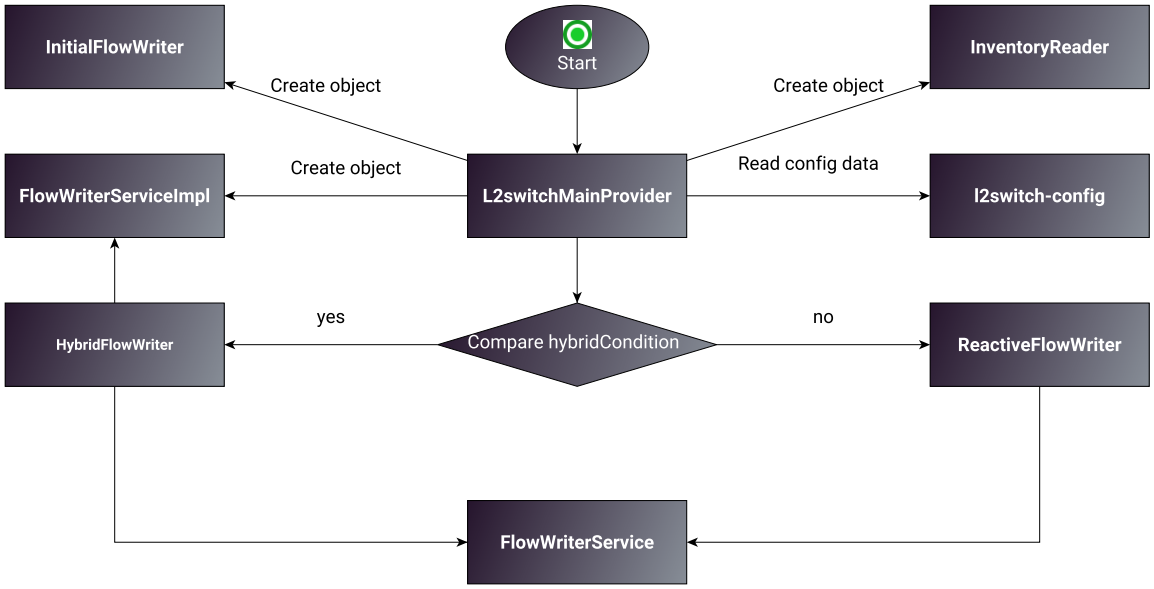


Figure 4.1: OpenDaylight's l2switch module Architecture, source adopted from [28]

4.3.1 L2switch-config.yang

This yang file can define every network-related state, Data modeling language for network devices. There are three significant elements of NETCONF (Network Configuration Protocol); to define the state data configuration, to define the signature of Remote Procedure Call (RPC) invokes on network components through the NETCONF protocol, and to define the event notifications format emitted by network components.

Modification: we added in this yang file code to initiate at module startup configuration.

```
leaf hybrid-mode-defined {
type uint16;
default 2;
}
leaf hybrid-flow-idle-timeout {
type uint16;
default 0;
}
leaf hybrid-flow-hard-timeout {
type uint16;
default 0;
}
%\end{verbatim}
```

Listing 4.1: Modification in l2switch-config.yang

hybrid-mode-defined leaf is to define the behavior of the module to change from reactive/proactive mode to hybrid. To disable the feature put any number to this variable. Hybrid-flow-hard-timeout means the flow will remove after x seconds, regardless of how many packets it is forwarding, and hybrid-flow-idle-timeout in the other hand removes if the flow does not forward a packet for x seconds. To make flexible, the administrator can change both parameters to a higher number for a long time. Nevertheless, they do not change the mode. It is reactive mathematically. Zero means the switch does not become eligible to remove flow from internal memory unless the controller triggers removal after the node removed from the network. This yang file is not accessible once building the system, instead it done by using the REST call. The modified file shows in Listing 4.1, and the REST call to modify these parameters shown in Listing 4.2. The rest of the yang file also found in Listing A.1.

```
Make the following REST call
URL: http://{{LOCALIP}}:8181/restconf/config/l2switch-config:l2switch-config/
Content-Type: application/json
Body:
{
"l2switch-config":
{
"configName":value //like "hybrid-flow-idle-timeout":10000
}
}
//Expected Result: 201 Created, then restart the Controller
//or create xml file in opendaylight/datastore/initial/config directory.
//l2switch-config_l2switch-config.xml with the entry
<?xml version="1.0" encoding="UTF-8"?>
<l2switch-config xmlns="urn:opendaylight:packet:l2switch-config">
<configName>value</configName>
</l2switch-config>
%\end{verbatim}
```

Listing 4.2: REST call to yang setting

4.3.2 ReactiveFlowWriter.java

This class listens to a certain type of packets and writes a mac to mac flows by implements the ArpPacketListener.java class and override onArpPacketReceived as an argument of ArpPacketRecieved class object. It is pretty that writes the flow to mac-to-mac flows on a

switch in a reactive manner, reactive-flow-hard-timeout, and reactive-flow-idle-timeout entry have 300 and 600 seconds respectively flow lifeline, and the expired if certain criteria met. This class is never touched, and it is as it is. Code is available Listing A.2.

4.3.3 FlowWriterService.java

It is a simple java interface class which defines *addMacToMacFlow* and *addBidirectionalMacToMacFlows* method which later implements in *FlowWriterServiceImpl* class. In this class, nothing we need to modify. The code found in Listing A.3.

4.3.4 L2SwitchMainProvider.java

This class acts as a registrar for the rest of the module. It is called when the module starts using blueprint dependency and starts automatically upon first, then creates an object from all classes, load main configuration files compare the value if the hybrid is configuring, it creates an object from *HybridFlowWriter* class or *ReactiveFlowWriter* as config says and register as notification listener to provide notification service on flow packet-in events. Full code in Listing A.5.

Modification: We put the condition in the main configuration and load condition and implement the rough code into logic in Listing 4.3.

```
1  if(hybridCondition == 2) {
2  fWSI.setFlowIdleTimeout(mainConfig.getHybridFlowIdleTimeout());
3  fWSI.setFlowHardTimeout(mainConfig.getHybridFlowHardTimeout());
4  HybridFlowWriter hFW = new HybridFlowWriter(inventoryReader, fWSI);
5  hybridFlowWriterReg = notificationService.registerNotificationListener(hFW);
6  }
7  else
8  {
9  fWSI.setFlowIdleTimeout(mainConfig.
10 getReactiveFlowIdleTimeout());
11 fWSI.setFlowHardTimeout(mainConfig.getReactiveFlowHardTimeout());
12 ReactiveFlowWriter rFW = new ReactiveFlowWriter(inventoryReader, fWSI);
13 reactFlowWriterReg = notificationService.registerNotificationListener(rFW);
14 }
```

Listing 4.3: modification of l2switchMainProvider

4.3.5 InitialFlowWriter.java

Which drops all packet to all switches, and adds a flow. It registers as an ODL Inventory listener so that it can add flows once it gets a new node, that switch added to the network. Nothing to provide other than this feature of initializing flows. This flow defines the controller address, and port at the initial switch does not know where to send packet-in messages. No modification is available for this class.

4.3.6 InventoryReader.java

InventoryReader reads the OpenDaylight-inventory tree in the MD-SAL data store. To provide topology information; node, host, host connector, and so on. We can not touch this class as our thesis part.

4.3.7 FlowWriterServiceImpl.java

FlowWriterServiceImpl is the class that was heavily modified, the main purpose of this class algorithm put in together, redirected from **HybridFlowWriter** and **ReactiveFlowWriter** to provide flow write service. These both class nothing to provide except setting the flow behavior and listen as a service. We call this class an implementation class. It determines the host location of subnet or prefix to write reactive/proactive mode if the flow is belongs to the same subnet/prefix, the flow mode will be proactive/static and if not reactive mode.

Addition:

```
1  if (ipv4PrefixDst.getValue().compareTo(ipv4PrefixSrc.getValue()) == 0 )
2  {
3  mb = createMatch33(); to init building match value(call)
4  match = mb.build();
5  this.setFlowHardTimeout(0);
6  this.setFlowIdleTimeout(0); mb.getLayer3Match(), mb.getEthernetMatch());
7  }
8  else {
9  this.setFlowHardTimeout(mainConfig.getReactiveFlowHardTimeout());
10 this.setFlowIdleTimeout(mainConfig.getReactiveFlowIdleTimeout());
11 match = createIpAddressMatch(mb, destMac, ipv4PrefixDst).build();
12 }
13 public MatchBuilder createMatch33() {
14 MatchBuilder match = new MatchBuilder();
15 Ipv4MatchBuilder ipv4Match = new Ipv4MatchBuilder();
```

```
16 try {
17 ipv4Match.setIpv4Destination(new Ipv4Prefix(dstIp.getValue() + "/32"));
18 Ipv4Match i4m = ipv4Match.build();
19 match.setLayer3Match(i4m);
20 EthernetMatchBuilder eth = new EthernetMatchBuilder();
21 EthernetTypeBuilder ethTypeBuilder = new EthernetTypeBuilder();
22 ethTypeBuilder.setType(new EtherType(0x800L));
23 EthernetSourceBuilder esb = new EthernetSourceBuilder();
24 esb.setAddress(this.sourceMac);
25 eth.setEthernetSource(esb.build());
26 eth.setEthernetType(ethTypeBuilder.build());
27 match.setEthernetMatch(eth.build());
28 } catch (Exception e) {
29 LOG.info("'\nbulletproof:{}'\n", e.toString());
30 }
31 return match;
32 }
```

Listing 4.4: Creating Flow MatchBuilder in FlowWriterServiceImpl.java

This l2switch module does not have the implementation to write mac-to-IP flow before, it is mac-to-mac, but now it is capable of writing destination IP address as match criteria. That is why IP based subdivision for flow writing criteria not happen. As a case from this perspective, traffic engineering based on IP address matches effective in the SDN platform and also includes in the l2switch module. So based on our implementation, we got good results as a positive side, and in the next chapter, try to discuss the results and analysis part.

CHAPTER FIVE

Result Analysis, and Discussion

5.1 Background

In this chapter, an elaboration on the previous chapters iterated on critical paper reviews of portion, methodology theories, identified use case, and relevant points raised in detail. So, in short, a review on some works done in OpenFlow enabled switches memory constraints and related issues on proactive flow mode. In reactive flows, frequent packet-in message disrupts the controller performance in many cases. An SDN controller lacks the power to process a high demand of packet-in message due to the dynamic nature of flows in reactive mode. The proactive mode has its disadvantages in nature, TCAM being increasingly inadequate due to its size.

This problem is obtained by carefully studying and examining different use of case studies. The problem persists in small and large scale enterprises, internet service providers, cloud computing environments, and data centers.

OpenStack is a well-known cloud computing operating system in IaaS architecture. The main reason to address the problem stated in the above paragraph is to ensure the OpenStack cloud operating system. Underlay network infrastructure not accustomed to managing by OpenStack. Instead, it addresses only overlay infrastructure. When OpenStack is integrated with SDN and operates in accordance, it gets the most out of this proposed solution more than anything. Virtual devices have only handled with OpenStack without the integration of the OpenDaylight (ODL) controller. To overcome this problem, it proposed to use a hybrid flow mode enabled to OpenFlow enabled SDN switches.

To address a specific solution to the matter, the same network subnet/prefix of the part has the more chance to communicate frequently in the respective domain. Therefore, it adopts the flow in the switch to have proactive flow mode to reduce the controller dependency, and in the other part of the subnet utilizes reactive flow mode. At last, there is one big issue that arises in proactive flows that may live longer without used to some degree in this case. The flow removes periodically based on packet usage analysis to free up unused rules. By introducing a hybrid flow mode in the SDN OpenFlow enabled switches, It is enabled to achieve the stated goal to reduce latency and maximize the throughput anticipated. Besides, The improvement lowers the rate of a packet-in message into the controller, reducing the Distribute Denial of Service (DDOS) attack on the system (caused due to memory overflow), and enhancing better performance and room for more bandwidth.

5.2 Performance Evaluation

In this section, a detailed performance analysis of reactive and hybrid implementation provided. First, pronounced the parameters of testbed and then presented throughput and latency results in the last part of this section. The implementations briefly discussed. An essential issue not to be forgotten in this part is that in proactive mode, there is no packet-in event. Therefore, it is not considered in the evaluation of latency and throughput measurement. However, to put some conclusion on proactive flow mode, there is no controller at this point of distribution that uses this approach in real-time, unless manually override the configuration to use. Proactive flow mode is rare in a production environment — currently, proactively used in a small office and home use, due to memory abundance. In our use case, it is not the choice to draw in context.

The output result measured in contentiously using a loop for 20 times for consistency reason to the specified array of a switch for 1000 millisecond long, for example, if some switches is {1 4 8 16}, the test will perform 20 times for every number in the array, and each test is 1000 millisecond in length. The rest of the parameters is described in detail in Table 5.1. For the latency test, CBENCH is running in the same machine whose controller is running because it does not contribute too much as a disrupt of throughput. Because it only generates a packet-in request to the controller. Moreover, waits for a reply, instead of generating another one—this behavior guarantees to the machine no overloaded memory demand. In the case of throughput, the test will perform on another machine to offload the massive memory demand

Table 5.1: Latency testing parameters list

Parameters	Value	Description
c/--controller	"localhost"	hostname of controller to connect to
-p/--port	6653	Controller port number
-l/--loops	20	loops per test
-m/--ms-per-test	1000	test length in ms
-s/--switches	1-64	fake $n switches/in the form of 1 2 3 4 5 6 ...64, 1 2 4 8 16 32 64 ...
-D/--delay	0	delay starting testing after features_reply is received (in ms)
-w/--warmup	1	loops to be disregarded on test start (warmup)
-M/--mac-addresses	100000	unique source MAC addresses per switch
-r/--ranged-test	"off"	test range of 1..$n switches
-t/--throughput	-t	test throughput instead of latency/blank by default for latency
-C/--cooldown	0	loops to be disregarded at test end (cooldown)
-i/--connect-delay	0	delay between groups of switches connecting to the controller (in ms)
-I/--connect-group-size	1	number of switches in a connection delay group
-L/--learn-dst-macs	"on"	send gratuitious ARP replies to learn destination macs before testing
-o/--dpid-offset	1	switch DPID offset

in the process. Also, when a throughput test is performing, there is a lot of buffer full error and dropped packet return with zero value, so many times that may contribute to the resulting inconsistency. One thing to mention in this contributing factor is Java Virtual Machine (JVM) is the main factor of massive demand in memory constraint. As far as we concerned, we tried hard to make conservative in a heavily controlled environment.

5.2.1 Latency Comparison

When we talk about measuring latency, it is the total time taken from packet-in to packet-out measured — in other terms, a two-way cost of time. Here the real deal in the SDN controller latency measurement approach is discussed, CBENCH is a tool that measured latency and throughput in ODL. The SDN controller is not measured on real/virtual switch using the mininet network emulator and CBENCH. It is mimicking the switch illustrated by Table 5.1. A real data center or production-grade network, traffic patterns are not that simple to measure with CBENCH accordingly because the pattern is not predictable suggested in this study[56]. The main aim of this study is to make an impact on the latency perspective and reduce the time gap in the latency of interactive service/application within the respectable time slot. We try to discuss in the problem statement and Figure 5.1 shows the detail we made progress, in

the resulting latency is reduces as we wish, the graph shows the controller repossessions per request reduced in excellent form. When the controller connects, a few switches latency is not significant. The reason is that a controller is under the capability of the performance. Moreover, the switch does not generate enough amount of packet to try the best of the controller. Furthermore, the conclusion shows that switches increase the controller performance better in terms of latency. Because the overall packet generation is higher compared to fewer switches, the time taken by request/response is divide by the total packet. This situation is much like in idle state to dictate in a small demand of work environment than in a tight situation. Nevertheless, this situation does not end in this way. In this research, it shows the analogy of the number of switches increasing around 64 and above as shown in Figure 5.3.

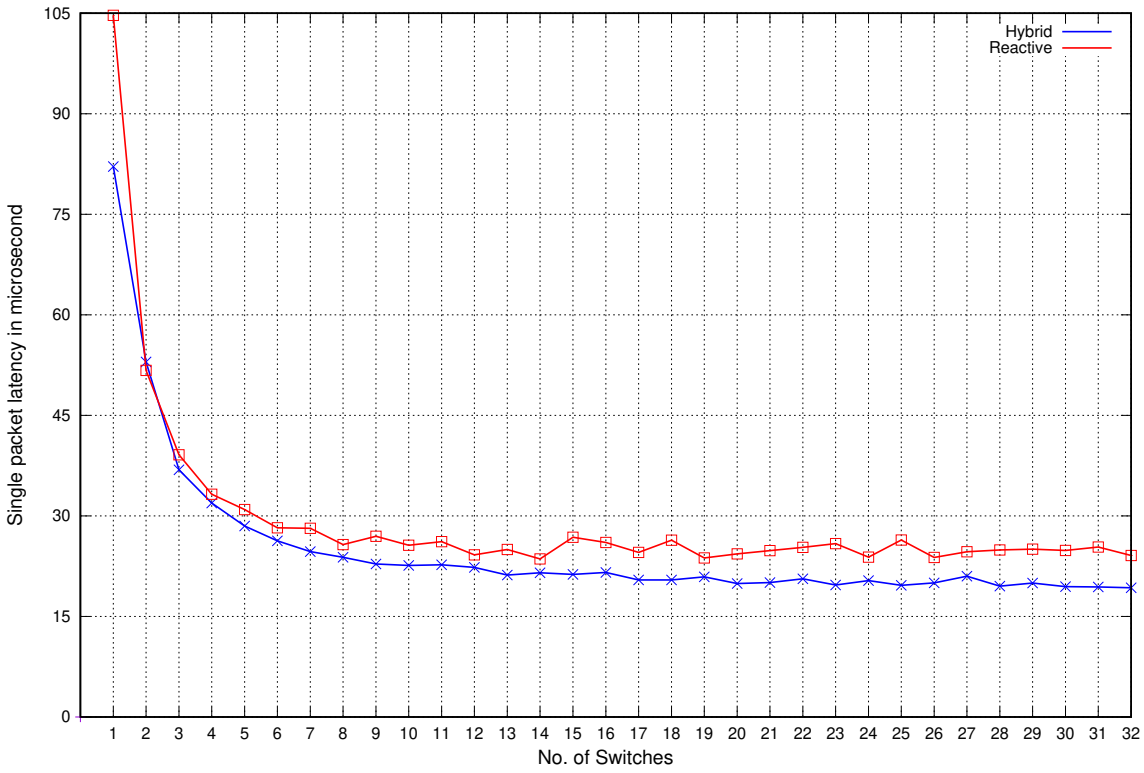


Figure 5.1: Latency graph based on the parameters Table 5.1 regards to time measurement

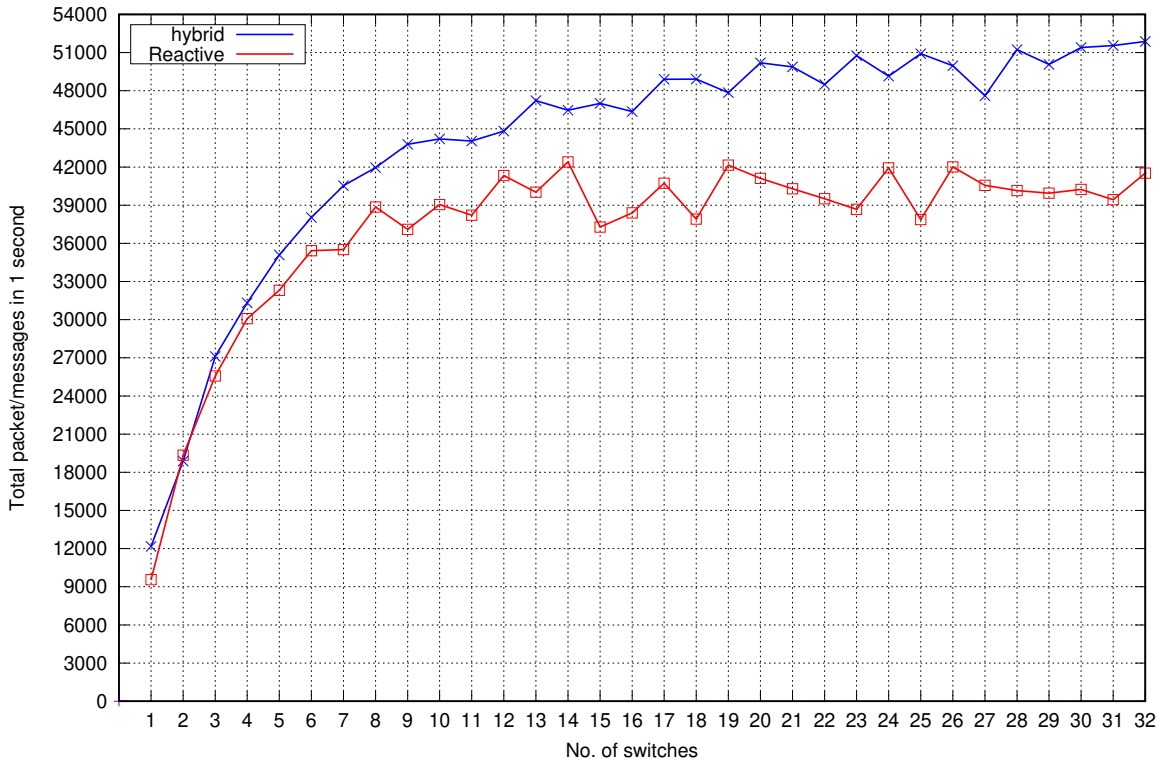


Figure 5.2: Latency graph based on the parameters Table 5.1 regards to packet measurement

Now the switch continues to grow in the fold from 32 to 64; the thing we discussed appears clearly. The controller now feels the stress so quickly. Hence, it is unable to respond to the request as previously performed. Which is the controller peak, and after that is the solution to the problem. Of course, this is the environment that resources are very much restrictive; in reality, the number of switch increase to a thousand to get a step we reach. The solution is everything is a single controller can not afford anymore; having a multi-controller is mandatory at this point. Let us rewind to Figure 5.3. After the switch grows to 64, the latency falls steep on the graph. However, this graph based on the packet-in count. The lines appear to go upward, shown in Figure 5.1. In the CBENCH tool, the result displayed in packet-in serves as numbers. Y-axis represents the number of a packet, and the X-axis represents the number of a switch. We did convert time in the microsecond to show latency in recognizable time-variant for inspection. Of course, if the packet increases, the result means good in latency. Latency calculated as $\frac{time}{total\ packet}$, one packet takes a fraction of microsecond to get a response.

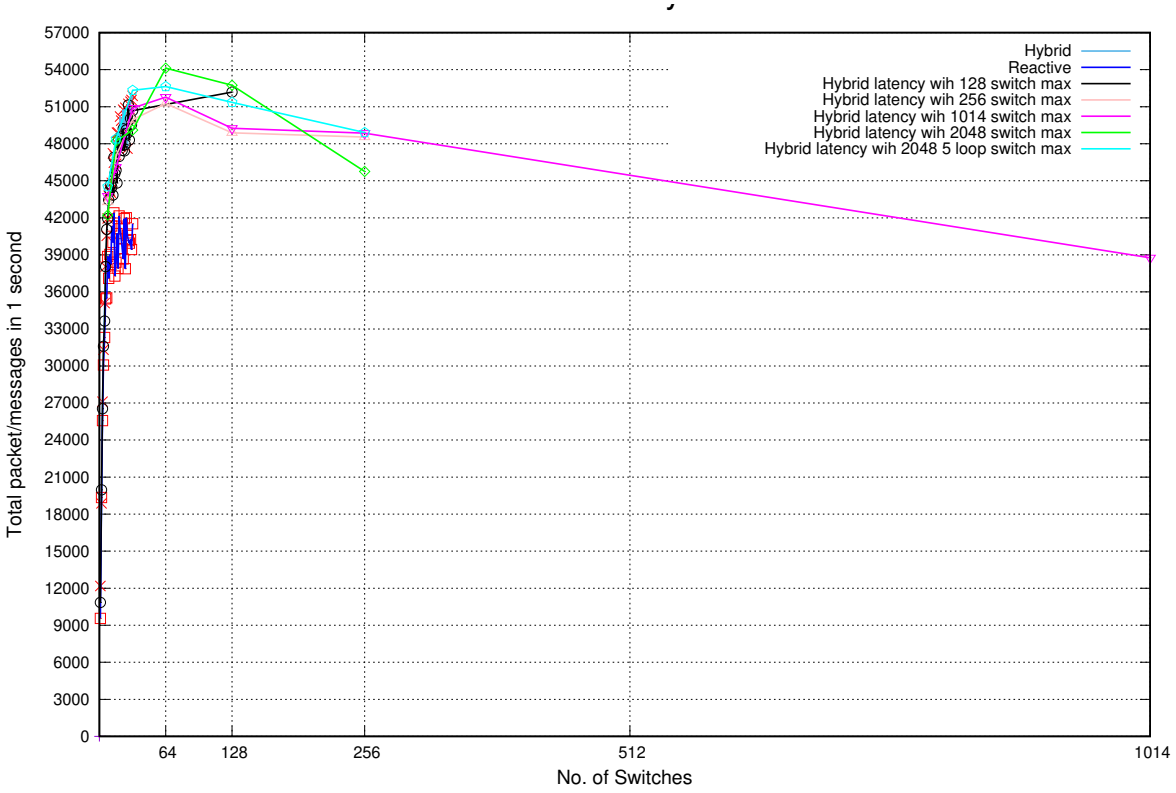
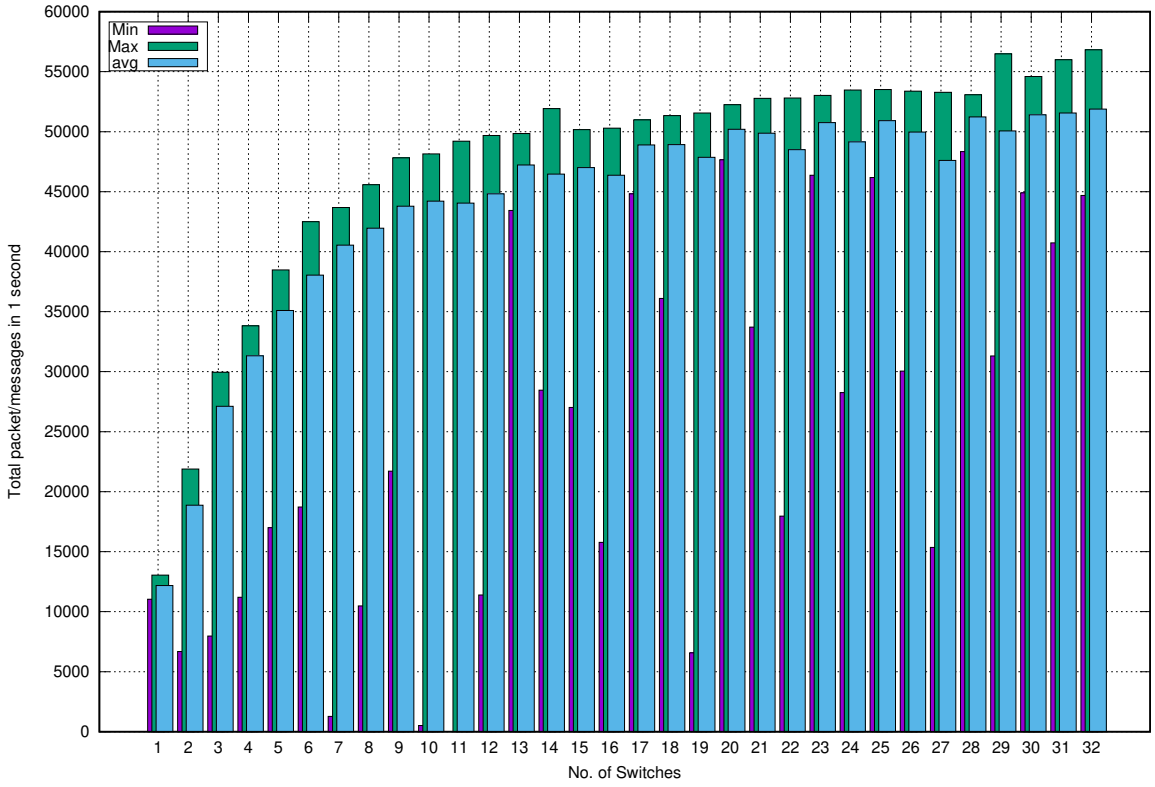


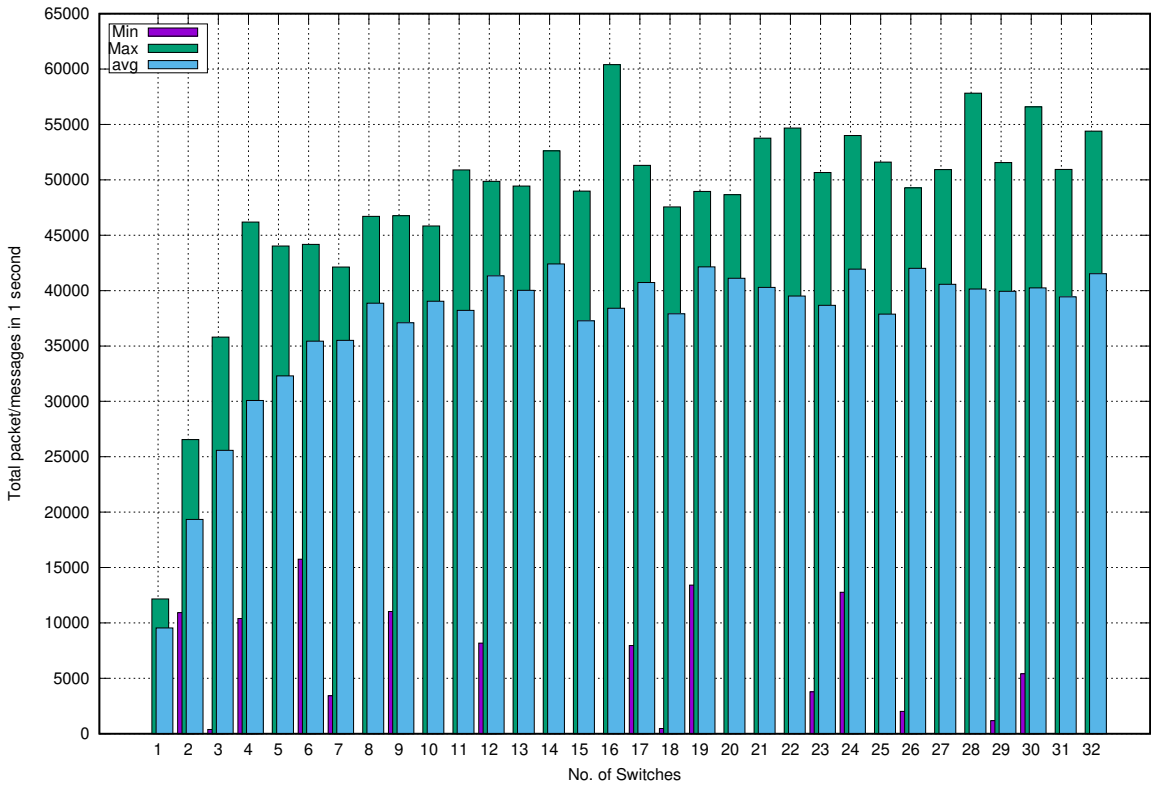
Figure 5.3: Maximum pressing test in latency mode

When we measure a latency, there is a lot of unpredictable requests that has dropped due to some memories leak in the controller side, memory contention. Moreover, the system used to run a controller is not a commercial-grade to run smoothly; the system that we use is multi-service running at the time. So that is why the latency result shows some luck in the continuity department. The resulting Figure 5.1 shows the controller's initial phase; the result is insufficient regards to packets dropped in the process. This drop packet affects overall behavior because the dropped packet is counting as zero to calculate on average. The Controller warming up and the number of the switch increases the improvement is stable. It shows healthy requests and smooth performance. The dropped packet has still affected the result. It contributes to the graph scattered due to the zero return of the failed request.

Figure 5.4: Detailed packet delivery in latency mode.



(a) Hybrid mode detailed test



(b) reactive mode detailed test

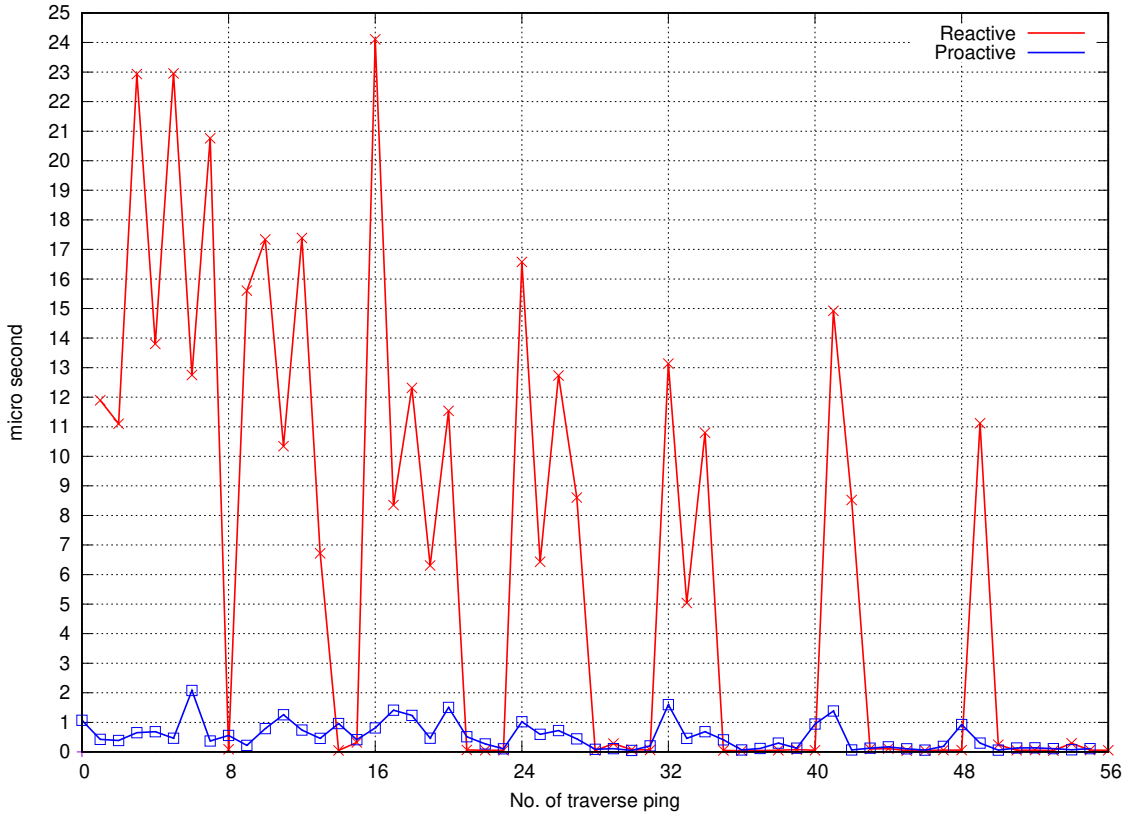


Figure 5.5: Latency difference between reactive and proactive, pinging traverse host

The latency (Round Trip Time) of the two generic mode shown Figure 5.5, proactive mode is the dominant in latency. But the problem is in memory consumption. This graph illustrates the first round ping, which means the reactive mode sends the packet-in request message to the controller but proactive does not. In this ping test, sequentially execute the manual command in mininet terminal. So the packet-in triggers every time, a barrier request/reply (notification of completed operation) is follows at each step. In case, if the second round ping is executes, the packet-in message does not required by a switch unless idle-timeout/hard-timeout is caused. So that, latency is the same as proactive.

Table 5.2: packet lost/dropped in latency mode

No. of sw	Sent Packet	Replied	Lost/ drop	Latency		
				Min	Max	Avg
1	254518	231354	23164	11038.01	13051.07	12176.53
2	394605	358685	35920	6685.46	21883.05	18878.15
3	566708	515119	51589	7968.32	29952.36	27111.55
4	654664	595070	59594	11206.67	33825.93	31319.47
5	733612	666823	66789	17002.94	38481.68	35095.94
6	795211	722816	72395	18730.23	42493.33	38042.95
7	847412	770268	77144	1278.89	43674.42	40540.42
8	884704	425654	459050	10477.16	45577.62	41959.63
9	915311	831976	83335	21716.27	47820.44	43788.19
10	924169	840016	84153	510.81	48144.54	44211.39
11	920625	836801	83824	0	49204.14	44042.16
12	937010	851689	85321	11387.97	49684.77	44825.71
13	987021	897136	89885	43432.77	49844.71	47217.68
14	971225	882779	88446	28456.93	51926.64	46462.04
15	982619	893129	89490	27014.55	50163.02	47006.79
16	969161	880896	88265	15773.42	50291.81	46362.92
17	1022168	929070	93098	44820	50981.74	48898.44
18	1022528	929388	93140	36103.11	51338.38	48915.18
19	1000410	909281	91129	6569.67	51555.71	47856.87
20	1049407	953816	95591	47658.91	52245.51	50200.85
21	1042583	810277	232306	33707.41	52770.98	49873.95
22	1013707	921378	92329	17969.61	52805.71	48493.57
23	1060964	964305	96659	46366.9	53024.79	50752.92
24	946231	933861	12370	28253.86	53462.51	49150.6
25	1064288	967331	96957	46179.06	53506.7	50912.15
26	1044600	949411	95189	30049.31	53379.57	49968.98
27	995261	995261	0	15353.24	53277.71	47609.7
28	1071042	1071042	0	48335.51	53077.55	51234.37
29	1046621	1046621	0	31313.65	56488.12	50066.31
30	1074515	976600	97915	44895.22	54590.8	51400.01
31	1077610	979418	98192	40736.17	55997.52	51548.34
32	1048497	985687	62810	44686.55	56817.62	51878.26

(a) Hybrid mode detailed packet lost

No. of sw	Sent Packet	Replied	Lost/ drop	Latency		
				Min	Max	Avg
1	199654	181482	18172	0	12165.31	9551.68
2	404292	367484	36808	10921.93	26557.34	19341.24
3	534661	485868	48793	382.57	35804.64	25572.01
4	628786	571523	57263	10405.55	46182.24	30080.17
5	675312	613806	61506	0	44018.5	32305.57
6	740610	673159	67451	15752.3	44170.38	35429.4
7	742305	674684	67621	3433.17	42132.85	35509.67
8	812336	738373	73963	0	46708.28	38861.72
9	775494	704841	70653	11030.82	46757.53	37096.91
10	816163	741824	74339	0	45837.5	39043.37
11	799187	726194	72993	0	50899.33	38220.75
12	864026	785316	78710	8184.68	49865.25	41332.43
13	836724	760401	76323	0	49439.31	40021.09
14	886498	805753	80745	0	52616.56	42408.05
15	779233	708255	70978	0	48983.43	37276.56
16	802730	729607	73123	0	60392.34	38400.39
17	851439	773749	77690	7960.18	51311.77	40723
18	792393	720177	72216	470.81	47560.49	37904.06
19	881112	800835	80277	13408.53	48951.96	42149.2
20	859774	781103	78671	0	48653.64	41110.71
21	842355	765572	76783	0	53761.08	40293.24
22	826041	750788	75253	0	54676.87	39515.15
23	808425	734766	73659	3782.28	50658.57	38671.89
24	876611	796722	79889	12767.14	53997.41	41932.76
25	791812	719622	72190	0	51606.25	37874.85
26	878080	798048	80032	2014.29	49278.16	42002.53
27	848023	770725	77298	0	50925.83	40564.46
28	839713	762823	76890	0	57827.29	40148.59
29	835019	758904	76115	1182.57	51560.78	39942.33
30	841185	764513	76672	5416.98	56588.7	40237.51
31	824271	749148	75123	0	50945.08	39428.84
32	867995	788868	79127	0	54401.08	41519.35

(b) reactive mode detailed packet lost

The histogram Figure 5.4a and Figure 5.4b dictate how much packet is lost/dropped in the process of hybrid and reactive, respectively. If the switch generates 1000 packet and gets 700 responses, it only calculates the response from the 700 packets. If it gets a small number, it may have a significant impact on the average. The process of stress testing is simple under the parameters taken if an array switch is {1, 2, 3, 4}. CBENCH picks the first member, which in this case is 1, and it generates 20 times testing for 1000ms long, then picks the next value 2. This process continues until it done. In Table 5.2, it is clearly shown how many packets are lost/dropped in hybrid and reactive mode. This problem occurs due to memory contention, and incapable buffer size primarily depends on the machine where they are running. In latency mode, there is enough use of a buffer because the response is adequate every time the packet generates. Therefore, it waits for an unspecified time to generate again that is why latency of a small packet produced through the process.

5.2.2 Throughput Comparison

As discussed in the previous section, a lot about latency, and the results be states. Throughput testing resource-hungry and tricky to measure relative to latency. In this test, the machine needs a lot of buffer space; it is a huge limit in throughput case; this ends the CBENCH generates a packet without getting proper response. Furthermore, at this time, even the number of packet dropped has dramatically increased and returned zero value, either buffer nor connection error. CBENCH is a single-threaded application even though it makes less packet production. That is why much packet generation not shown. In a real environment, it is not affected by this problem. Since CBENCH is the only viable option for the research purpose testbed right now. There are other commercial alternative options not afford to use due to cost and heavy resource requirements. Moreover, that is not Applicable due to resource utilization, which makes it challenging to conduct the other option. Hence, looking at Figure 5.6 it is evident that it shows the analogy in which the throughput is improved. However, the current configuration of the testbed does not allow using more switches at once to show more results at some point until the controller slows down the responses due to resource constraint.

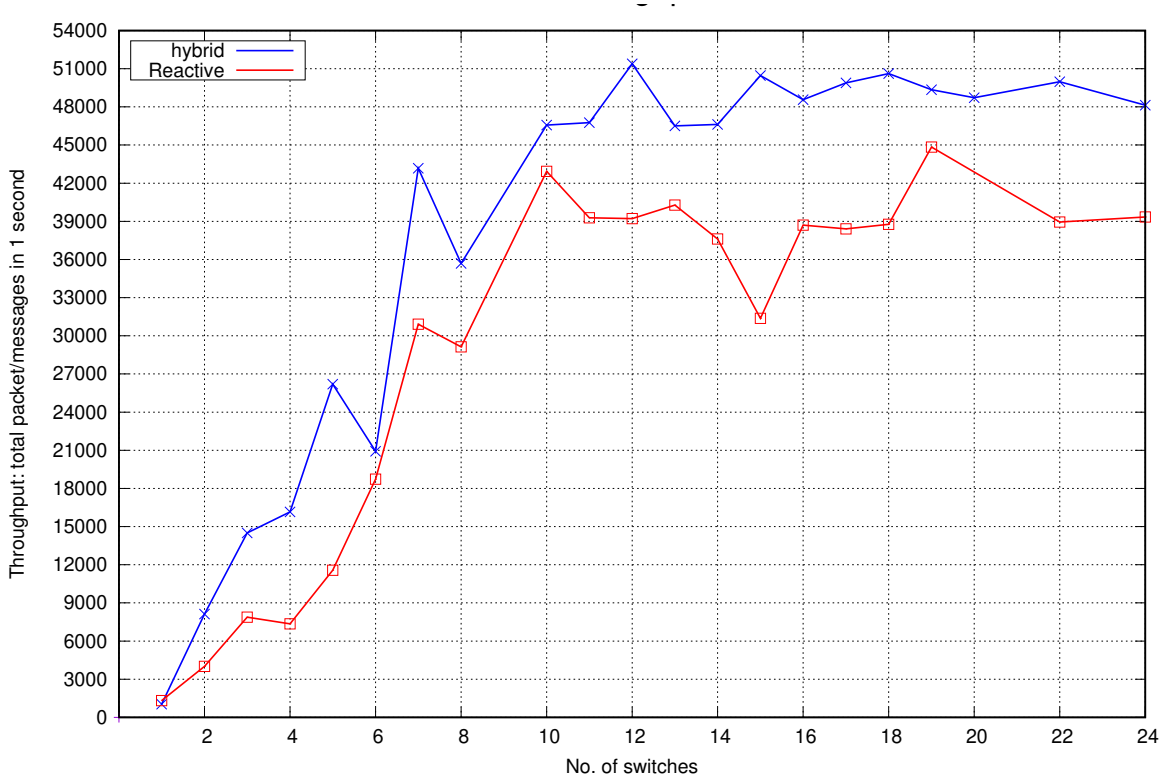
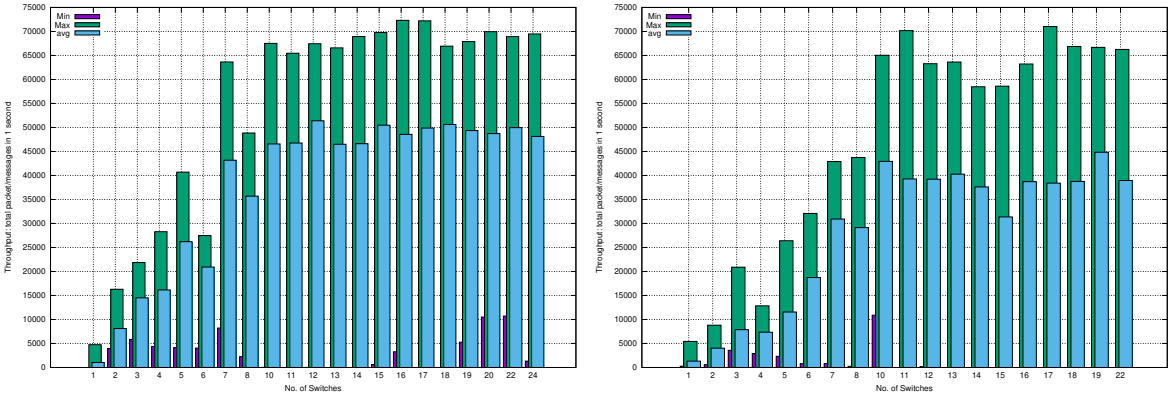


Figure 5.6: Throughput Result Graph

Figure 5.7a and Figure 5.7b show the packet response. The switch is frequently returned zero

for minimum, that indicates how much packet is lost/dropped in the process, which makes throughput test inconsistent. These two graphs draw a clear picture in mind. As the number of switches decrease to 24, the level is satisfactory to conduct with the available resources in hand. Practically, there are so many packets generated in the process. Nevertheless, The graph shows only the responses coming from the controller to the switch labeled as Y-axis. Beyond that our controlled environment is not capable to manage.

Figure 5.7: Detailed packet delivery in Throughput mode.



(a) Hybrid mode detailed test

(b) reactive mode detailed test

5.2.3 Memory Consumption/Management

Despite their flexibility, for individual benefit of application that run on SDN management layer, it uses proactive control logic. There are two factors affecting the OpenFlow switch: *(1) flow definition — granularity of the switch forwarding policies. (2) flow entry timeout — impacts the delay that new flow entry in switch.*

Table 5.3: Proactive control logic in SDN networks: number of flow-table entries in access network Table 5.3a and core network Table 5.3b switch for different flow definitions [57].

Flow definition	Flow table size	
	UniBS2009	DSLAM
/24 dst	27664	38964
/24 dst + /24 src	56669	164501
/32 dst	31213	61613
/32 dst + dst port	84597	237718
/32 dst + /32 src	64430	182356
/32 dst + /32 src + dst port	107025	336599

(a) Access Network

Flow definition	Flow table size	
	PAIX2005	CAIDA2013
/16 dst	2914	1979
/16 src	1720	5662
/16 src + /16 dst	127039	293384
/24 dst	74152	97152
/24 src + /24 dst	1267187	1094808
/32 dst	739376	389636
/32 src + /32 dst	1933482	1578977

(b) Core Network

The current switch technology technology in line speed is limited to hundred's of update/s compared to a 1OGb/s and 40Gb/s traffic in a core network link accommodation. The scenario considering where SDN is used to implement QOS and firewalling policies to an access network switch flows traversing. According to [57] the actual traffic trace collected from DSLAM an Internet Service Provider (ISP) and campus edge router — the trace lasted one day long and three consecutive working days respectively.

Switch/Modules	# flows v1.0	# flows v1.3
Compatible mode – "allow v1 modules" – A chassis where v1 as well as v2 modules are present may execute in this mode. Non-compatible mode – "no allow v1 modules" – A chassis that only has v2 modules may execute in this mode.		
8200/5400 v1 modules 3500 series 6600 series 6200 series	Total: 64 K	Total: 64 K
	Hardware: TCAM – 1.5K per slot	Hardware: Standard match mode TCAM – 1.5 K per slot
	Software: Total minus Hardware	Software: Total minus Hardware
8200/5400 v2modules 3800 series 2920 series (Flow numbers will be lower for this series)	Total: 64 K 16 K for 2920	Total: 64 K (may be higher) 16 K for 2920
	Hardware: TCAM - 2000 per slot TCAM – 500 per slot for 2920	Hardware: Compatible mode Standard match mode TCAM – 1.5 K per slot
		Non-compatible mode TCAM – 4K per slot
	Software: Total minus Hardware	Software: Total minus Hardware

Table 5.4: Switch modules scalability, source adopted from [1]

Obviously, flow table sizes depend on the switch model. HP switches are shown in Table 5.4 to hold specific for number of flow entries accordingly, obtained from Hp switch administrator's guide. From this we have to look Table 5.3 number of flows table size requirement in proactive mode, and compare with OpenFlow switch capacity from the list to observe inadequate memory consumption.

5.2.4 Key Note

In all the above, we discussed latency and throughput by stressing the controller with detail. Thus, we find exciting results throughout the process and have a clue about results. In latency aspects the result is 18.4% and in throughput 23.5%. The overall improvement is as shown in Table 5.5a and Table 5.5b in the latency and throughput test respectively. Due to the lack of the right balance hardware, it is difficult to measure the specific performance of the controller and it is shown in throughput test. For example, switch iteration number 9,20 and 21 are does not perform a lack of buffer and lost connection in that case. Of course, we conduct a different test in time to obtain consistent value, but the overall character does not change. In proactive

we obtained the memory requirements of proactive flow mode, to conclude the test.

Table 5.5: Overall Improvement Result

Nos	Average Latency		Improv. In (%)
	Hybrid	Reactive	
1	12176.53	9551.68	27.4805060471
2	18878.15	19341.24	-2.39431391162
3	27111.55	25572.01	6.02041059737
4	31319.47	30080.17	4.11999001335
5	35095.94	32305.57	8.6374269205
6	38042.95	35429.4	7.37678312362
7	40540.42	35509.67	14.1672676767
8	41959.63	38861.72	7.97162348964
9	43788.19	37096.91	18.0372974461
10	44211.39	39043.37	13.2366135403
11	44042.16	38220.75	15.2310197995
12	44825.71	41332.43	8.45166858082
13	47217.68	40021.09	17.9819939937
14	46462.04	42408.05	9.55948222095
15	47006.79	37276.56	26.102810989
16	46362.92	38400.39	20.735544613
17	48898.44	40723.61	20.0739325418
18	48915.18	37904.06	29.0499751214
19	47856.87	42149.2	13.5415856054
20	50200.85	41110.71	22.1113670866
21	49873.95	40293.24	23.777462423
22	48493.57	39515.15	22.7214625277
23	50752.92	38671.89	31.2398230343
24	49150.6	41932.76	17.2128903511
25	50912.15	37874.85	34.4220505164
26	49968.98	42002.53	18.9665955836
27	47609.7	40564.46	17.3680113084
28	51234.37	40148.59	27.6118787733
29	50066.31	39942.33	25.3464933067
30	51400.01	40237.51	27.7415277436
31	51548.34	39428.84	30.7376529464
32	51878.26	41519.35	24.9495957909
	Overall improvement		18.4%

(a) Latency

Nos	Average Throughput		
	Hybrid	Reactive	Improv. in (%)
1	1036.42	1324.83	-27.8275216611
2	8120.55	4014.7	50.5612304585
3	14506.38	7874.34	45.7180909365
4	16155.18	7356.38	54.4642647126
5	26203.17	11563.35	55.8704156787
6	20926.13	18731.69	10.4866021572
7	43172.56	30919.4	28.3818240104
8	35688.58	29134.92	18.3634652878
9	-	-	-
10	46570.96	42934.16	7.80915832527
11	46758.26	39277.75	15.9982642639
12	51385.19	39219.25	23.6759657792
13	46505.79	40281.89	13.3830647754
14	46616.27	37611.8	19.3161529226
15	50467.09	31371.21	37.8382823341
16	48563.38	38706.8	20.2963220435
17	49887.81	38403.56	23.0201526184
18	50619.53	38762.82	23.4231925899
19	49348.46	44838.05	9.13992047574
20	-	-	-
21	-	-	-
22	49973.87	38945.27	22.0687331199
24	48129.82	39344.99	18.2523641269
	Overall improvement		23.5%

(b) Throughput

5.3 Sampling Strategy

According to the CBENCH tool characterization, the strategy of sampling depends on result obtained through a repetitive sample to ensure optimum results. Our experiment is implemented in a virtual environment. This is because of the lack of OpenFlow enabled switch which is a constraint to conduct this study in real environment. To get required result in this test, the packet that includes the response in the sampling phase as well as the first warm-up phase are disregarded as they do not qualify to determine required characteristics of parameters.

The test is tried for same number of repetitions for each switch. Up to 24 switches for throughput and up to 32 switches are used as a trial for latency because of the limiting factor of the test environment do not capable to support large buffer size that the experiment requires.

The sampling has taken in CBENCH benchmarking tool which collects the logs that are generated packets until pressing test to the controller. Each generated and sent packet has its own logs for latency and throughput. Based on that, it analyzes logs and draws the result.

5.4 Summary

In this chapter, the experiments made discussed. Performance evaluation of latency and throughput, the objective is to show that we analyze what get in the result concerning a better goal with tangible results and perform accordingly. In the next chapter, the conclusion, and future work as well as the recommendations as per the study stated.

CHAPTER SIX

Conclusion and Future Works

6.1 Conclusion

In the previous chapter, we discussed every result in a study that has discovered. So in this chapter, we are going to conclude based on our findings related to the objective that pinpointed earlier. We examine the current status of the flow writing mechanism. In every aspect of proactive and reactive mode to gain insight into the existing strength, weakness, and analysis techniques (SWAT) into OpenFlow enabled SDN switches in one by one and determined which mechanisms are the advantage or disadvantage. Therefore, come up with the hybrid flow writing mode to solve the problem, to minimize the knock-on effect and maximizes the actual good behavior. By looking for suitable case studies in the real world to determine which is the best way to collect the data, analyses, and conclude based on what we have found. Every case we look in the literature review has a piece of extraordinary evidence to quantitatively measuring. Based on the methodology we obtained from the use of a case study. To critical evaluate the use of a hybrid flow writing mechanism at the SDN OpenFlow switch, we use latency and throughput observation to verify and test in virtualization using CBENCH SDN bench-marking tool, and get the meaningful achievement. Throughout our study, we aimed at starting by list the objective, and we met our initial objective

6.2 Limitation

It is evident from the results that the algorithm has performance and latency enhancement. We initially found that for each packet-In, switch generated a packet-in request. However, the evaluation of this conduct did not fluently forward. We likewise saw that numerous ODL tests neglected to produce any results. The number of failed tests for ODL in latency mode. It is evident that a massive number of tests are failing, and the number of failed tests are increasing

with the number of switches. We accept that ODL may have memory spills, which can have discovered with the assistance of code profiling. Another conceivable explanation might be that the OpenFlow module utilized with ODL is breaking down. Since OpenFlow protocol released the latest version, but all available controllers on the market still not agreed to use the recent. Because there are need to be adopted many criteria and much trickery in Software or Hardware, especially Hardware part is more expensive to practice. Besides the problems mentioned above, the non-existence of real OpenFlow enabled switch in our hands plays essential roles that restrict in some additional feature tests — the lack of tools to measure the SDN controller. It also overrules our ability to conduct further. If any tools have, it is heavy on demands in hardware requirements, and a commercially available tool that is not to use for the consolidated test, real-time network simulation, controller's switch discovery time.

6.3 Recommendation

In practical, SDN is everywhere that needs a network of computers has interconnected. To be precise in-home, small office, medium, and large scale infrastructure is concerned with this technology. In a small network solution — as little as interaction is best suited for proactive flow mode recommends. Because of this is the OpenFlow TCAM well handles small communication in a better way, on the other. If the device may increase, it will become busy and incapable of handling the flow entry due to size limitation. However, when the device increase, the service/application response matters a lot; it recommends using the hybrid model.

6.4 Future Work

The way we conduct our research is in productive, environment, flexible, emergence, and needs attention to be in touch with future technology; it is open for innovation in everywhere. Moreover, continue where we stopped for the time being, by doing our research, we have found a way forward to a fellow researcher to do their job in this field. Our research is not going to be solved everything, even in this small title, so we have discovered and got a problem in our work. Some of them, the problem must solve the hardware utilization of the controller is a significant issue, in it needs to be more efficient than before. The other one is we do not include L4-L7 OSI layer protocol, we tried L2 and L3 to enhance, but the rest is excellent for traffic engineering to be proper maintaining in the future.

REFERENCES

[1] H.-p. Company, "HP Switch Software OpenFlow Administrator ' s Guide K / KA / WB 15 . 14," no. October 2013.

[2] ONF, "SDN Architecture Overview," *Open Networking Foundation*, pp. 1–5, 2013. [Online]. Available: www.opennetworkfoundation.org

[3] A. Hemid, *Facilitation of The OpenDaylight Architecture*. Mayo, 2017.

[4] "Ieee xplore digital library." [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp [Accessed: 2019-12-02].

[5] "Sciencedirect.com | science, health and medical journals, full text articles and books." [Online]. Available: https://www.sciencedirect.com/ [Accessed: 2019-12-02].

[6] "Home - springer." [Online]. Available: https://link.springer.com/ [Accessed: 2019-12-02].

[7] "Acm digital library." [Online]. Available: https://www.acm.org/publications/digital-library [Accessed: 2019-12-02].

[8] "Google scholar." [Online]. Available: https://scholar.google.com/ [Accessed: 2019-12-02].

[9] S. Shirali-Shahreza and Y. Ganjali, "Protecting home user devices with an SDN-based firewall," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 1, pp. 92–100, 2018.

[10] T. D. N. Gray and Ken, *Software Defined Network*, 1st ed. O'RELLY, 2013. [Online]. Available: http://www.it-ebooks.info/

[11] "Understanding the sdn architecture and sdn control plane." [Online]. Available: https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture/ [Accessed: 2019-12-02].

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

[13] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 116–123, 2014.

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

[15] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 18–33, 2015.

[16] ONF, "OpenFlow Switch Specification 1.3.3," pp. 1–164, 2013.

[17] L. Zhu, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN Controllers : Benchmarking & Performance Evaluation," pp. 1–14, 2019.

[18] "Open Source Guides for the Enterprise – The Linux Foundation." [Online]. Available: https://www.linuxfoundation.org/resources/open-source-guides/ [Accessed: 2019-10-22].

[19] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," pp. 302–336, 2014.

[20] K. Y. Wang, S. J. Kao, and M. T. Kao, "An efficient load adjustment for balancing multiple controllers in reliable SDN systems," in *Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018*. Institute of Electrical and Electronics Engineers Inc., jun 2018, pp. 593–596.

[21] "Home - beacon - confluence." [Online]. Available: https://openflow.stanford.edu/display/Beacon/Home.html [Accessed: 2019-12-02].

[22] “Floodlight OpenFlow Controller -Project Floodlight.” [Online]. Available: http://www.projectfloodlight.org/floodlight/ [Accessed: 2019-12-02].

[23] “FlowVisor – GENI: geni.” [Online]. Available: https://groups.geni.net/geni/wiki/FlowVisor [Accessed: 2019-12-02].

[24] “Maestro platform by zhengcai.” [Online]. Available: http://zhengcai.github.io/maestro-platform/ [Accessed: 2019-12-02].

[25] “noxrepo/nox: The NOX Controller.” [Online]. Available: https://github.com/noxrepo/nox [Accessed: 2019-12-02].

[26] “ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out.” [Online]. Available: https://onosproject.org/ [Accessed: 2019-12-02].

[27] “Juniper/contrail-controller: Contrail controller.” [Online]. Available: https://github.com/Juniper/contrail-controller [Accessed: 2019-12-02].

[28] OpenDaylight Project, “OpenDaylight Project,” 2013. [Online]. Available: https://wiki.opendaylight.org/view/Main_Page [Accessed: 2019-10-22].

[29] “oxrepo/pox: The POX network software platform.” [Online]. Available: https://github.com/noxrepo/pox [Accessed: 2019-12-02].

[30] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, “Rosemary: A robust, secure, and high-performance network operating system,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’14. New York, NY, USA: ACM, 2014, pp. 78–89. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660353

[31] “Ryu SDN Framework.” [Online]. Available: https://osrg.github.io/ryu/ [Accessed: 2019-12-02].

[32] “Trema, SDN Controller.” [Online]. Available: https://github.com/noxrepo/nox [Accessed: 2019-12-02].

[33] “Model-Driven Service Abstraction Layer,” 2013. [Online]. Available: http://sdntutorials.com/catagory/md-sal/ [Accessed: 2019-10-22].

[34] “Model-Driven-Service Abstraction.” [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:FAQ [Accessed: 2019-10-22].

[35] A. V. Atli, M. S. Uluderya, S. Tatlicioglu, B. Gorkemli, and A. M. Balci, “Protecting {SDN} controller with per-flow buffering inside {OpenFlow} switches,” in *2017 {IEEE} {International} {Black} {Sea} {Conference} {Communications} {Networking}*. IEEE, 2017, pp. 1–5.

[36] S. Wang, C. Chang, Y. Hsieh, and C. Chou, “Design and implementation of a packet-in buffer system for sdn switches,” in *2017 IEEE Symposium on Computers and Communications (ISCC)*, July 2017, pp. 955–960.

[37] K. Lai, S. Kao, and M. Kao, “Using the depository buffer to mitigate the communications load between sdn switches and the controller,” in *2018 IEEE International Conference on Applied System Invention (ICASI)*, April 2018, pp. 589–592.

[38] K. Basu, M. Younas, A. W. Wan Tow, and F. Ball, “Performance comparison of a sdn network between cloud-based and locally hosted sdn controllers,” in *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, March 2018, pp. 49–55.

[39] T.-y. Chao and K. Wang, “In-switch Dynamic Flow Aggregation in Software Defined Networks,” pp. 0–5, 2017.

[40] OpenDaylight, “OpenDaylight SDN Controller Platform (OSCP):Overview,” 2013. [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_SDN_Controller_Platform_(OSCP):Overview [Accessed: 2019-10-22].

[41] Mininet Team, “Mininet: An Instant Virtual Network on your Laptop (or other PC),” 2018. [Online]. Available: http://mininet.org [Accessed: 2019-05-30].

[42] “Open vSwitch, virtual switch,” 2013. [Online]. Available: https://www.openvswitch.org/ [Accessed: 2019-10-22].

[43] i. GmbH, “Oracle VM VirtualBox,” 2014. [Online]. Available: https://www.virtualbox.org/wiki/innotek [Accessed: 2019-10-22].

[44] “intellij community edition – ide.” [Online]. Available: https://www.jetbrains.com [Accessed: 2019-10-22].

[45] O. Belkadi and Y. Laaziz, “A Systematic and Generic Method for Choosing A SDN Controller,” vol. 5, no. 11, pp. 239–247, 2017.

[46] “command-line driven plotting program – gnuplot.” [Online]. Available: https://www.gnuplot.info [Accessed: 2019-10-22].

[47] “All-In-One Cross-Platform Diagram Software for Flowchart, Org Chart and Mind Map.” [Online]. Available: https://www.edrawsoft.com/ [Accessed: 2019-10-22].

[48] “opensuse – open-source linux distribution.” [Online]. Available: https://www.opensuse.org [Accessed: 2019-10-22].

[49] “Network sniffer tool.” [Online]. Available: https://www.wireshark.org [Accessed: 2019-12-02].

[50] R., Sherwood, K.-K., and Yap, “cbench.” [Online]. Available: https://github.com/andi-bigswitch/oflops/tree/master/cbench{%}0A [Accessed: 2019-11-01].

[51] “dfarrell07/wcbench: CBench, wrapped in stuff that makes it useful.” [Online]. Available: https://github.com/dfarrell07/wcbench [Accessed: 2019-12-02].

[52] “Enhamced cbench:modified the original cbench SDN controller benchmarking tool/OpenFlow Controller.” [Online]. Available: https://gist.github.com/soyo42/9837240 [Accessed: 2019-11-02].

[53] V. T. R. Technologies, “PktBlaster SDN Datasheet.” [Online]. Available: http://www.veryxtech.com/wp-content/uploads/2015/10/Datasheet-PktBlaster-SDN-Controller-Test5.pdf [Accessed: 2019-11-03].

[54] “ARCCN/hcprobe: Framework for testing OpenFlow controllers.” [Online]. Available: https://github.com/ARCCN/hcprobe [Accessed: 2019-11-02].

[55] R. Nitrogen and O. Project, “OpenDaylight Documentation Documentation,” pp. 143–147, 2018. [Online]. Available: https://docs.opendaylight.org/en/stable/nitrogen/developer-guide/

[56] Z. K. Khattak, M. Awais, and A. Iqbal, “Performance Evaluation of OpenDaylight SDN Controller,” 2014.

[57] M. Dusi, R. Bifulco, F. Gringolit, and F. Schneider, “Reactive Logic in Software-Defined Networking : Measuring Flow-Table Requirements,” pp. 340–345, 2014.

Appendix A

List of Java Codes

A.1 l2switch

```
1  module l2switch-config {
2    yang-version 1;
3    namespace "urn:opendaylight:l2switch:l2switch-config";
4    prefix "l2switch-config";
5  description "This module contains the base configuration for main implementation."
6    older version is 2014-05-28;
7    revision 2014-05-28 {
8      description "Initial module draft.";
9    }
10   container l2switch-config {
11   leaf is-learning-only-mode {
12     type boolean;
13     default false;
14   }
15   leaf is-install-dropall-flow {
16     type boolean;
17     default true;
18   }
19   leaf dropall-flow-table-id {
20     type uint8;
21     default 0;
22   }
23   leaf dropall-flow-priority {
24     type uint16;
25     default 0;
26   }
27   leaf dropall-flow-hard-timeout {
28     type uint16;
```

```
29      default 0;
30    }
31    leaf dropall-flow-idle-timeout {
32      type uint16;
33      default 0;
34    }
35    leaf reactive-flow-table-id {
36      type uint8;
37      default 0;
38    }
39    leaf reactive-flow-priority {
40      type uint16;
41      default 10;
42    }
43    leaf reactive-flow-hard-timeout {
44      type uint16;
45      default 300;
46    }
47    leaf reactive-flow-idle-timeout {
48      type uint16;
49      default 600;
50    }
51    leaf hybrid-mode-defined {
52      type uint16;
53      default 2;
54    }
55    leaf hybrid-flow-idle-timeout {
56      type uint16;
57      default 0;
58    }
59    leaf hybrid-flow-hard-timeout {
60      type uint16;
61      default 0;
62  }}}
```

Listing A.1: l2switch-config.yang

A.2 ReactiveFlowWriter

```
1  package org.opendaylight.l2switch.flow;
2  public class ReactiveFlowWriter implements ArpPacketListener {
3    private final InventoryReader inventoryReader;
4    private final FlowWriterService flowWriterService;
5  public ReactiveFlowWriter(InventoryReader inventoryReader,
6    FlowWriterService flowWriterService) {
7    this.inventoryReader = inventoryReader;
8    this.flowWriterService = flowWriterService;
9  }
10 private boolean ignoreThisMac(MacAddress macToCheck) {
11   if (macToCheck == null) {
12     return true;
13   }
14   String[] octets = macToCheck.getValue().split(":");
15   short firstByte = Short.parseShort(octets[0], 16);
16   return (firstByte & 1) == 1;
17   }
18
19 @Override
20 public void onArpPacketReceived(ArpPacketReceived packetReceived)
21 {
22 if (packetReceived == null || packetReceived.getPacketChain()
23  == null) {
24   return;
25 }
26 RawPacket rawPacket = null;
27 EthernetPacket ethernetPacket = null;
28 ArpPacket arpPacket = null;
29 for (PacketChain packetChain : packetReceived.getPacketChain()) {
30 if (packetChain.getPacket() instanceof RawPacket) {
31   rawPacket = (RawPacket) packetChain.getPacket();
32 } else if (packetChain.getPacket() instanceof EthernetPacket) {
33   ethernetPacket = (EthernetPacket) packetChain.getPacket();
34 } else if (packetChain.getPacket() instanceof ArpPacket) {
35   arpPacket = (ArpPacket) packetChain.getPacket();
36 }
37 }
38 if (rawPacket == null || ethernetPacket == null ||
```

```
39 arpPacket == null) {
40   return;
41 }
42 if (arpPacket.getProtocolType().equals(KnownEtherType.Ipv4)) {
43   FlowWriterServiceImpl.setArpPacket(arpPacket);
44 }
45 MacAddress destMac = ethernetPacket.getDestinationMac();
46 if (!ignoreThisMac(destMac)) {
47   writeFlows(rawPacket.getIngress(),
48   ethernetPacket.getSourceMac(),
49   ethernetPacket.getDestinationMac());
50   }
51 }
52 public void writeFlows(NodeConnectorRef ingress, MacAddress srcMac,
53 MacAddress destMac) {
54   NodeConnectorRef destNodeConnector = inventoryReader
55   .getNodeConnector(ingress.getValue().firstIdentifierOf
56   (Node.class), destMac);
57   if (destNodeConnector != null) {
58     flowWriterService.addBidirectionalMacToMacFlows(srcMac,
59     ingress, destMac, destNodeConnector);
60     Ipv4Address(arpPacketIp.getDestinationProtocolAddress()),
61     destNodeConnector);
62   }
63 }
64 }
```

Listing A.2: ReactiveFlowWriter.java

A.3 FlowWriterService

```
1 package org.opendaylight.l2switch.flow;
2 public interface FlowWriterService {
3   void addMacToMacFlow(MacAddress sourceMac, MacAddress destMac,
4   NodeConnectorRef destNodeConnectorRef);
5   void addBidirectionalMacToMacFlows(MacAddress sourceMac,
6   NodeConnectorRef sourceNodeConnectorRef,
7   MacAddress destMac, NodeConnectorRef destNodeConnectorRef);
8 }
```

Listing A.3: FlowWriterService.java

```
1  public class InitialFlowWriter implements DataTreeChangeListener<Node> {
2    private static final Logger LOG = LoggerFactory.getLogger(InitialFlowWriter.class);
3    private static final String FLOW_ID_PREFIX = "Asni-L2switch-";
4    private final ExecutorService initialFlowExecutor = Executors.newCachedThreadPool();
5    private final SalFlowService salFlowService;
6    private final AtomicLong flowIdInc = new AtomicLong();
7    private final AtomicLong flowCookieInc = new AtomicLong(0x2b00000000000000L);
8    private short flowTableId;
9    private int flowPriority, flowIdleTimeout, flowHardTimeout;
10 public InitialFlowWriter(SalFlowService salFlowService) {
11   this.salFlowService = salFlowService;
12 }
13 public void setFlowTableId(short flowTableId) {
14   this.flowTableId = flowTableId;
15 }
16 public void setFlowPriority(int flowPriority) {
17   this.flowPriority = flowPriority;
18 }
19 public void setFlowIdleTimeout(int flowIdleTimeout) {
20 this.flowIdleTimeout = flowIdleTimeout;
21 }
22 public void setFlowHardTimeout(int flowHardTimeout) {
23   this.flowHardTimeout = flowHardTimeout;
24 }
25 public ListenerRegistration<InitialFlowWriter>
26 registerAsDataChangeListener(DataBroker dataBroker) {
27   InstanceIdentifier<Node> nodeInstanceIdentifier = InstanceIdentifier
28   .builder(Nodes.class).child(Node.class).build();
29   return dataBroker.registerDataTreeChangeListener(new DataTreeIdentifier<>
30   (LogicalDatastoreType.OPERATIONAL, nodeInstanceIdentifier), this);
31 }
32 @Override
33 public void onDataTreeChanged(Collection<DataTreeModification<Node>> changes) {
34   Set<InstanceIdentifier<?>> nodeIds = new HashSet<>();
35   for (DataTreeModification<Node> change: changes) {
36     DataObjectModification<Node> rootNode = change.getRootNode();
```

```
37    final InstanceIdentifier<Node> identifier = change.getRootPath()
38    .getRootIdentifier();
39  switch (rootNode.getModificationType()) {
40  case WRITE:
41    if (rootNode.getDataBefore() == null) {
42    nodeIds.add(identifier);
43    }
44  break;
45  default:
46  break;
47  }
48  }
49  if (!nodeIds.isEmpty()) {
50  initialFlowExecutor.execute(new InitialFlowWriterProcessor(nodeIds));
51  }
52  }
53  private class InitialFlowWriterProcessor implements Runnable {
54    private final Set<InstanceIdentifier<?>> nodeIds;
55    InitialFlowWriterProcessor(Set<InstanceIdentifier<?>> nodeIds) {
56    this.nodeIds = nodeIds;
57  }
58  @Override
59  public void run() {
60    if (nodeIds == null) {
61      return;
62    }
63  for (InstanceIdentifier<?> nodeId : nodeIds) {
64  if (Node.class.isAssignableFrom(nodeId.getTargetType())) {
65  InstanceIdentifier<Node> invNodeId = (InstanceIdentifier<Node>) nodeId;
66  if (invNodeId.firstKeyOf(Node.class, NodeKey.class).getId().getValue()
67  .contains("openflow:")) {
68  addInitialFlows(invNodeId);
69  }}}}
70  public void addInitialFlows(InstanceIdentifier<Node> nodeId) {
71  LOG.debug("adding initial flows for node {} ", nodeId);
72  InstanceIdentifier<Table> tableId = getTableInstanceId(nodeId);
73  InstanceIdentifier<Flow> flowId = getFlowInstanceId(tableId);
74  //add drop all flow
75  writeFlowToController(nodeId, tableId, flowId, createDropAllFlow(flowTableId,
```

```
76 flowPriority));
77 LOG.debug("Added initial flows for node {} ", nodeId);
78 }
79 private InstanceIdentifier<Table> getTableInstanceId(InstanceIdentifier<Node>
80  nodeId) {
81 // get flow table key
82 TableKey flowTableKey = new TableKey(flowTableId);
83   return nodeId.builder()
84   .augmentation(FlowCapableNode.class)
85   .child(Table.class, flowTableKey)
86   .build();
87 }
88 private InstanceIdentifier<Flow> getFlowInstanceId(InstanceIdentifier<Table>
89  tableId) {
90 // generate unique flow key
91 FlowId flowId = new FlowId(FLOW_ID_PREFIX + String.valueOf
92 (flowIdInc.
93 getAndIncrement()));
94 FlowKey flowKey = new FlowKey(flowId);
95 return tableId.child(Flow.class, flowKey);
96 }
97 private Flow createDropAllFlow(Short tableId, int priority) {
98 // start building flow
99 FlowBuilder dropAll = new FlowBuilder().setTableId(tableId).setFlowName("dropall");
100 // use its own hash code for id.
101 dropAll.setId(new FlowId(Long.toString(dropAll.hashCode())));
102 Match match = new MatchBuilder().build();
103 Action dropAllAction = new ActionBuilder().setOrder(0)
104   .setAction(new DropActionCaseBuilder().build()).build();
105 // Create an Apply Action
106 ApplyActions applyActions = new ApplyActionsBuilder().setAction(ImmutableList
107 .of(dropAllAction))
108   .build();
109 // Wrap our Apply Action in an Instruction
110 Instruction applyActionsInstruction = new InstructionBuilder()
111   .setOrder(0).setInstruction(new ApplyActionsCaseBuilder()
112   .setApplyActions(applyActions).build()).build();
113 // Put our Instruction in a list of Instructions
114 dropAll.setMatch(match) //
```

```
115     .setInstructions(new InstructionsBuilder() //
116     .setInstruction(ImmutableList.of(applyActionsInstruction)) //
117     .build()) //
118     .setPriority(priority) //
119     .setBufferId(OFConstants.OFP_NO_BUFFER) //
120     .setHardTimeout(flowHardTimeout) //
121     .setIdleTimeout(flowIdleTimeout) //
122     .setCookie(new FlowCookie(BigInteger.valueOf
123     (flowCookieInc.getAndIncrement())))
124     .setFlags(new FlowModFlags(false, false, false, false, false));
125     return dropAll.build();
126 }
127 private Future<RpcResult<AddFlowOutput>>writeFlowToController
128 (InstanceIdentifier<Node> nodeInstanceId,
129 InstanceIdentifier<Table> tableInstanceId,
130 InstanceIdentifier<Flow> flowPath,Flow flow) {
131   LOG.trace("Adding flow to node {}",nodeInstanceId.
132     firstKeyOf(Node.class, NodeKey.class).getId().getValue());
133     final AddFlowInputBuilder builder = new AddFlowInputBuilder(flow);
134     builder.setNode(new NodeRef(nodeInstanceId));
135     builder.setFlowRef(new FlowRef(flowPath));
136     builder.setFlowTable(new FlowTableRef(tableInstanceId));
137     builder.setTransactionUri(new Uri(flow.getId().getValue()));
138     return salFlowService.addFlow(builder.build());
139 }}}
```

Listing A.4: InitialFlowWriter.java

A.4 L2SwitchMainProvider

```
1 public class L2SwitchMainProvider {
2 private static final Logger LOG = LoggerFactory.getLogger(L2SwitchMainProvider.class);
3 private Registration topoNodeListherReg, reactFlowWriterReg, hybridFlowWriterReg;
4 private HashMap<InstanceIdentifier<FlowCapableNode>, HashSet<Flow>>
5   flowTable = new HashMap<InstanceIdentifier<FlowCapableNode>,
6 HashSet<Flow>>();
7   private final DataBroker dataService;
8   private final NotificationProviderService notificationService;
9   private final SalFlowService salFlowService;
```

```
10    private final L2switchConfig mainConfig;
11    private int hybridCondition;
12  public L2SwitchMainProvider(final DataBroker dataBroker,
13  final NotificationProviderService
14  notificationService,
15  final SalFlowService salFlowService, final L2switchConfig config) {
16    this.dataService = dataBroker;
17    this.notificationService = notificationService;
18    this.salFlowService = salFlowService;
19    this.mainConfig = config;
20  }
21  public void init() {
22  // Setup FlowWrtierService
23  // if-else addition
24  FlowWriterServiceImpl fWSI = new FlowWriterServiceImpl(salFlowService, dataService);
25  hybridCondition = mainConfig.getHybridModeDefined();
26  if (hybridCondition == 2) {
27    fWSI.setFlowIdleTimeout(mainConfig.getHybridFlowIdleTimeout());
28    fWSI.setFlowHardTimeout(mainConfig.getHybridFlowHardTimeout());
29  }
30  else {
31    fWSI.setFlowIdleTimeout(mainConfig.getReactiveFlowIdleTimeout());
32    fWSI.setFlowHardTimeout(mainConfig.getReactiveFlowHardTimeout());
33  }
34  fWSI.setFlowTableId(mainConfig.getReactiveFlowTableId());
35  fWSI.setFlowPriority(mainConfig.getReactiveFlowPriority());
36  // Setup InventoryReader
37  InventoryReader iR = new InventoryReader(dataService);
38  // Write initial flows
39  if (mainConfig.isIsInstallDropallFlow()) {
40    LOG.info("L2Switch will install a dropall flow on each switch");
41    InitialFlowWriter iFW = new InitialFlowWriter(salFlowService);
42    iFW.setFlowTableId(mainConfig.getDropallFlowTableId());
43    iFW.setFlowPriority(mainConfig.getDropallFlowPriority());
44    iFW.setFlowIdleTimeout(mainConfig.getDropallFlowIdleTimeout());
45    iFW.setFlowHardTimeout(mainConfig.getDropallFlowHardTimeout());
46    topoNodeListherReg = iFW.registerAsDataChangeListener(dataService);
47  }
48  else {
```

```
49   LOG.info("Dropall flows will not be installed");
50 }
51 if (mainConfig.isIsLearningOnlyMode()) {
52   LOG.info("L2Switch is in Learning Only Mode");
53 }
54 else {
55    // Setup reactive flow writer
56   LOG.info("L2Switch will react to network traffic and install flows");
57    // 2-line modifications
58    if (hybridCondition == 2) {
59   LOG.info("Hybrid condition is satisfied: {}",hybridCondition);
60    HybridFlowWriter hFW = new HybridFlowWriter(inventoryReader, fWSI);
61    hybridFlowWriterReg = notificationService
62    .registerNotificationListener(hFW);
63 }
64 else {
65   LOG.info("Hybrid condition is not satisfied: {}",hybridCondition);
66    ReactiveFlowWriter rFW = new ReactiveFlowWriter(inventoryReader, fWSI);
67    reactFlowWriterReg = notificationService
68    .registerNotificationListener(rFW);
69 }
70 }
71 LOG.info("L2SwitchMain initialized , this is blue.asni@gmail.com");
72 }
73 public void close() {
74 if (reactFlowWriterReg != null) {
75   reactFlowWriterReg.close();
76 }
77 if (hybridFlowWriterReg != null) {hybridFlowWriterReg.close();}
78 if (topoNodeListherReg != null) {topoNodeListherReg.close();}
79 LOG.info("L2SwitchMain (instance {}) torn down.", this);
80 }
81 }
```

Listing A.5: L2SwitchMainProvider.java

Appendix B

Controller Test Statistics

B.1 Hybrid Latency

FROM_SWITCH: MSG[PacketInMessage] -> +30852507 | 30852507
FROM_SWITCH_TRANSLATE_IN_SUCCESS: no activity detected
FROM_SWITCH_TRANSLATE_OUT_SUCCESS: MSG[PacketInMessage] -> +30852507 | 30852507
FROM_SWITCH_TRANSLATE_SRC_FAILURE: no activity detected
FROM_SWITCH_PACKET_IN_LIMIT_REACHED_AND_DROPPED: no activity detected
FROM_SWITCH_NOTIFICATION_REJECTED: no activity detected
FROM_SWITCH_PUBLISHED_SUCCESS: MSG[PacketInMessage] -> +30852507 | 30852507
FROM_SWITCH_PUBLISHED_SUCCESS: MSG[MultipartReplyMessage] -> +529 | 529
FROM_SWITCH_PUBLISHED_FAILURE: no activity detected
TO_SWITCH_ENTERED: MSG[TransmitPacketInput] -> +11440 | 11440
TO_SWITCH_ENTERED: MSG[FlowModInputBuilder] -> +30855681 | 30855681
TO_SWITCH_ENTERED: MSG[MultipartType] -> +529 | 529
TO_SWITCH_ENTERED: MSG[SetConfigInput] -> +529 | 529
TO_SWITCH_DISREGARDED: MSG[AbstractService] -> +45 | 45
TO_SWITCH_RESERVATION_REJECTED: no activity detected
TO_SWITCH_READY_FOR_SUBMIT: MSG[] -> +30867605 | 30867605
TO_SWITCH_READY_FOR_SUBMIT: MSG[] -> +529 | 529
TO_SWITCH_SUBMIT_SUCCESS: MSG[FlowModInputBuilder] -> +30413840 | 30413840
TO_SWITCH_SUBMIT_SUCCESS: MSG[SetConfigInput] -> +529 | 529
TO_SWITCH_SUBMIT_SUCCESS_NO_RESPONSE: MSG[TransmitPacketInput] -> +11396 | 11396
TO_SWITCH_SUBMIT_FAILURE: no activity detected
TO_SWITCH_SUBMIT_ERROR: MSG[FlowModInputBuilder] -> +424007 | 424007
TO_SWITCH_SUBMIT_ERROR: MSG[TransmitPacketInput] -> +4 | 4
REQUEST_STACK_FREED: MSG[RpcContextImpl] -> +30849776 | 30849776
OFJ_BACKPRESSURE_ON: no activity detected
OFJ_BACKPRESSURE_OFF: no activity detected

B.2 Reactive Latency

FROM_SWITCH: MSG[PacketInMessage] -> +22492145 | 22492145
FROM_SWITCH_TRANSLATE_IN_SUCCESS: no activity detected
FROM_SWITCH_TRANSLATE_OUT_SUCCESS: MSG[PacketInMessage] -> +22492145 | 22492145
FROM_SWITCH_TRANSLATE_SRC_FAILURE: no activity detected
FROM_SWITCH_PACKET_IN_LIMIT_REACHED_AND_DROPPED: no activity detected
FROM_SWITCH_NOTIFICATION_REJECTED: no activity detected
FROM_SWITCH_PUBLISHED_SUCCESS: MSG[MultipartReplyMessage] -> +516 | 516
FROM_SWITCH_PUBLISHED_SUCCESS: MSG[PacketInMessage] -> +22492145 | 22492145
FROM_SWITCH_PUBLISHED_FAILURE: no activity detected
TO_SWITCH_ENTERED: MSG[TransmitPacketInput] -> +10612 | 10612
TO_SWITCH_ENTERED: MSG[SetConfigInput] -> +528 | 528
TO_SWITCH_ENTERED: MSG[FlowModInputBuilder] -> +26054049 | 26054049
TO_SWITCH_ENTERED: MSG[MultipartType] -> +528 | 528
TO_SWITCH_DISREGARDED: MSG[AbstractService] -> +21845 | 21845
TO_SWITCH_RESERVATION_REJECTED: no activity detected
TO_SWITCH_READY_FOR_SUBMIT: MSG[] -> +26043344 | 26043344
TO_SWITCH_READY_FOR_SUBMIT: MSG[] -> +528 | 528
TO_SWITCH_SUBMIT_SUCCESS: MSG[SetConfigInput] -> +527 | 527
TO_SWITCH_SUBMIT_SUCCESS: MSG[FlowModInputBuilder] -> +25591547 | 25591547
TO_SWITCH_SUBMIT_SUCCESS_NO_RESPONSE: MSG[TransmitPacketInput] -> +10608 | 10608
TO_SWITCH_SUBMIT_FAILURE: no activity detected
TO_SWITCH_SUBMIT_ERROR: MSG[FlowModInputBuilder] -> +425586 | 425586
REQUEST_STACK_FREED: MSG[RpcContextImpl] -> +26028268 | 26028268
OFJ_BACKPRESSURE_ON: no activity detected
OFJ_BACKPRESSURE_OFF: no activity detected