Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Performance Enhancement of Floodlight Software Defined Networking Controller using Workload Adaptive Packet Batching

By: Petros Belachew Guyita

Addis Ababa, Ethiopia

July 2019

Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Performance Enhancement of Floodlight Software Defined Networking Controller using Workload Adaptive Packet Batching

By: Petros Belachew Guyita

Advisor: Dr. Eng. Yihenew Wondie

Addis Ababa, Ethiopia

July 2019

Addis Ababa University

Addis Ababa Institute of Technology

School of Electrical and Computer Engineering

Performance Enhancement of Floodlight Software Defined Networking Controller using Workload Adaptive Packet Batching

By: Petros Belachew Guyita

A Thesis Submitted to the Department of Electrical and Computer Engineering in Partial Fulfillment for the Degree of Master of Science in Computer Engineering

Addis Ababa, Ethiopia

July 2019

Addis Ababa University
Addis Ababa Institute of Technology
School of Electrical and Computer Engineering

By: Petros Belachew Guyita

This is to certify that the thesis prepared by *Petros Belachew*, titled *Performance Enhancement of Floodlight Software Defined Networking Controller using Workload Adaptive Packet Batching* and Submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Engineering compiles with the regulations of the University and meets the accepted standards with respect to originality and quality.

Approved by the board of Examining Committee:

	Name	Signature
Dean, School of Electrical and Computer Engineering:	Dr. *Yalemzewud Negash*	_____
Advisor:	*Dr. Eng. Yihenew Wondie*	_____
Internal Examiner:	_____	_____
External Examiner:	_____	_____

Addis Ababa, Ethiopia

July 2019

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university and that all source of materials used for the thesis has been duly acknowledged.

Declared by:

Name: Petros Belachew Guyita

Signature: _____

Date of submission: July 2019

Addis Ababa, Ethiopia

This thesis has been submitted for examination with my approval as a university advisor.

Confirmed by advisor:

Name: **Dr. Eng. Yihenew Wondie**

Signature: _____

Acknowledgments

I first and foremost, praise and give thanks to God, the Almighty, for his blessings throughout my research work.

I would like to extend sincere gratitude to my passionate advisor, Dr. Eng. Yihenew Wondie, for his constructive comments and advice. He is always kind to correct my work, consistently until the end of this thesis.

My acknowledgment shall also pass to Debre Berhan University, which provides me the opportunity to join this program and support me until I finish my study.

I also want to acknowledge all staff of the Addis Ababa Institute of Technology School of Electrical and Computer Engineering for all of their cooperation.

Finally, I would like to thank the Floodlight and Mininet mailing lists in the Open source community. Their prompt answers and guidance helped me a lot to navigate in this thesis, in working with the different tools that I am not familiar with.

Abstract

The innovation of high tech devices with increasing demand for big data processing, made the networking systems unresponsive to the need of users. The capacity of network technologies such as wired, wireless, and cellular networks has been increasing highly due to the high traffic of the network system. Such traffic nature of today's network system is very complex which is very hard to be handled by the conventional networks. These conventional network characteristics could not be adapted to the fluctuating requirements. Due to this, it is very hard to manage the different networking devices; inflexibility to increase in size of the networks, and dependability to the specific vendor's software. Software Defined Networking was made by separating the cumbersome network control from data forwarding devices, apart from traditional networking.

SDN was aimed at making a networking paradigm that responds quickly to the changing network requirements. SDN controller uses an OpenFlow protocol, which handles rules for the traffic that arrives at the switch. Floodlight controller uses static packet batching for supervising the traffic by OpenFlow protocol. The static batching is sluggish to the rapidly expanding traffic and it takes a high time for processing. In this thesis, the Workload Adaptive Packet Batching, which learns the batch size based on the nature of the workloads, was proposed to optimize the performance of the Floodlight SDN controller.

This study implemented and tested on the network that was created by Mininet network emulator. The network consists of the virtual switches and hosts that are managed by the Floodlight controller. After network setup, the performance evaluation was performed using the Cbench tool, which tests for throughput and latency metrics.

The proposed Workload Adaptive Packet Batching achieved an enhanced average throughput of 11% and a latency of 10%. The throughput was improved and the latency was reduced with this proposed mechanism. Therefore, enterprise SDN networks can boost their performance and traffic management by applying the Floodlight controller into their networks.

Keywords: Big data, Software Defined Networking (SDN), OpenFlow, Floodlight controller, Workload Adaptive Packet Batching, Cbench

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ABS	Adaptive Batch Scheduling
ACLs	Access Lists
API	Application Programming Interface
ARP	Adrees Resolution Protocol
Cbench	Controller Benchmark Tool
CLI	Command Line Instruction
CPU	Central Processing System
DSCP	Differentiated Services Code Point
FPI	Fixed Point Iteration
GUI	Graphics User Interface
I/O	Input/output
IDS	Intrusion Detection System
IoT	Internet of things
IP	Internet Protocol
IPS	Intrusion Prevention Systems
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IT	Information Technology
JDK	Java Development Kit
JVM	Java Virtual Machine
L2	Layer 2
L3	Layer 3
LAN	Local Area Network
LLDPs	Link Layer Discovery Protocols
MAC	Media Access Control
MPLS	Multiprotocol Label Switching
NOS	Network Operating System
ONF	Open Network Foundation
OSI	Open System Interconnection

OVS	Open Virtual switch
PCS	Projective Cone Scheduling
QoE	Quality of Experience
QoS	Quality of Service
SDN	Software Defined Networking
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual machine
VRF	Virtual Routing and Forwarding
WAPB	Workload Adaptive Packet Batching
4G	Fourth Generation
5G	Fifth Generation

Chapter 1. Introduction

1.1 Background

The advancement in technology leads to an increase in the number of applications, users, and devices of the network that were emerged before thirty years [1]. Due to this, the networking system could not effectively handle the emerging technologies and innovations to maximize its performance. To overcome the different issues of networking, the use of newly invented technologies could boost the quality of service (QoS) and quality of experience (QoE). One of these technologies is Software Defined Networking (SDN), with the selection of high-speed network parameters, the bottleneck of networking, mainly data centers, and cloud-computing services could be resolved.

Nowadays, the networking system consists of modern devices and more innovations that increase from time to time. The demand for Big data-systems that processes very large data sets including; cloud computing, mobile traffic-traffic from mobile devices also increases highly. With these demands, the capacity of the wired and wireless network technologies, and cellular networks (i.e. 4G and 5G) increase. The demand of the user and capacity of the network makes the networking system need for high-performance network management system. Moreover, the current network traffic pattern is very complex and the traditional network architecture can not handle the current users' demand [1].

The existing network infrastructure and technologies are based on traditional networks. With these types of network, the system is unable to adapt to the changing systems requirements, hard to manage the different types of devices in numbers and complexity, could not increase in size as necessary, and device vendor dependence. The increase in traffic volumes trends and a greater emphasis on network reliability, predictability, and performance, different organizations and vendors have made researchers research for better ways of controlling traffic and programmability [2, 3], in order to ease the different complex tasks of configuration and management. In the end, they have applied programmable control mechanisms by separating the control and the data forwarding devices [4].

A software-defined network (SDN) is a new networking paradigm that allows solving many problems of traditional networks and brings many new opportunities [5]. This approach is based

on separating the networks intelligence of forwarding packet out of the data-forwarding plane and putting it into a logically centralized controller. The controller is accountable for the forwarding mechanisms that reside inside the switch via standard protocols, like OpenFlow. The inspiration of SDN is to perform a network operating system; where network tasks can be performed for each switch elements using the central controller, without adding additional software and providing access for developing applications that control the switches by running it on top of a network operating system [6].

There are different SDN controller's that were developed. Floodlight is one of the controllers that is designed for network agility. The controller uses OpenFlow for traffic orchestration across different devices. For these purposes, there are different mechanisms that are employed for packet processing. One of them is a static packet processing system, which most of the controller uses.

1.2 Motivation of the Study

The need for high performance in data centers and cloud computing providers is becoming a crucial issue. The controllers lack the performance for large-scale data centers, therefore it is very important to enhance the performance of the Floodlight controller, which is an open source controller and implemented by different enterprises. Unlike closed source controllers that are represented as a distributed controller instances, open source controller are represented as, single controller instances. In addition, single controller instances performance can be an issue because using only one controller for the network will result in lower performance relative to many numbers of the distributed controller.

Two major parameters for OpenFlow controller benchmarking are, the average maximum number of flows setup per second that a controller is able to populate, and the time required for the controller to respond to requests from the OpenFlow switches. By such examination, it is possible to know which controller has the highest throughput and lowest delay.

Enhancing the performance of the Floodlight controller using Workload Adaptive Packet Batching has a high impact on the SDN based networks. The data centers and cloud-based providers will get the great benefit of easily managing and installing rules within a short period of time with the high-performance Workload Adaptive Packet Batching Floodlight controller. Generally,

Workload Adaptive Packet Batching will help the networking system to increase thier efficiency and meet their demand.

1.3 Statement of the Problem

Software-defined networking (SDN) is a general term including numerous kinds of network technologies designed at making the network as responsive and dynamic as the virtualized server and storage infrastructure of the modern data center [9].

Responding quickly to changing business requirements is the goal of SDN, which allows network engineers and administrators to work with the underlying network easily. In SDN, a network administrator can character traffic from a centralized control console, without configuring individual switches. In addition, the network administrator can deliver services to wherever they are needed in the network, without regard to number and type of specific devices a server or other device is connected. Moreover, the key technologies provided by SDN are functional separation, network virtualization, and automation through programmability.

Floodlight controller uses a static packet batching process. Static packet batching is simple to implement, especially easy to calculate [10]. However, it has two distinguished drawbacks, the first one; the batches are not responsive to bursts in the number and size of packets. This means it has high latency, it will still take a long time to process before the newly arriving packets can be served, making many packets to wait to get access for the resource. Secondly, the static batches cannot guarantee maximal throughput for that fixed batch size.

In this thesis work, we are proposing Workload Adaptive Packet Batching (WAPB) and implementing it to optimize the performance of the Floodlight controller. In addition, the research also compares the performance of Floodlight and the enhanced version of Floodlight. With this assumption, we have the following research question.

- Can the implementation of Workload Adaptive Packet Batching enhance the performance of the Floodlight SDN controller?

1.4 Objectives

The general objective of this work is to optimize the performance of Floodlight SDN controller using Workload Adaptive Packet Batching.

Our specific objectives include:

- Identify the design features of the Floodlight SDN controller
- Investigate the effect of Workload Adaptive Packet Batching on throughput and latency
- Implement Workload Adaptive Batching to the Floodlight controller
- Compare the performance of Floodlight and the enhanced version of Floodlight.

1.5 Contributions of the Thesis

The controller is essential for SDN to meet its goals and requirements. The selection of the right controller to meet one demand is the first issue in SDN enterprise networking. The Floodlight controller uses static packet pusher that uses static packet processing mechanism. Moreover, it has an impact on the overall performance of the system, because, the efficiency is limited.

In this thesis, the performance of Floodlight will be boosted and network stability will be increased using Workload Adaptive Packet Batching. The Workload Adaptive Packet Batching uses ABS by incorporating Fixed point iteration that learns the existing workload and adjusts its processing ability.

1.6 Research Methodology

This thesis presents Performance enhancement of Floodlight SDN controller using Workload Adaptive Packet Batching method. To conduct this thesis different features of the existing Floodlight controller will be studied and after identifying the feature that needs improvement, a new approach is that is proposed will be presented. This approach is implemented and evaluated for performance. Therefore, the different methodologies for an adaptive packet processing system are reviewed from literatures and compared for the state of the art of the system.

1.7 Significance of the Study

This thesis work will help the network community as a whole and the SDN based service providers and user in providing the different networking services to the user and fulfill their goal. Below, listed some of the significances of the study.

- The researchers will understand the mechanism of enhancing the performance of the SDN controllers using Workload Adaptive Packet Batching.
- This work plays a great role for researcher and network enterprises in order to choose and apply high-speed packet switching architectures.
- This thesis will motivate researchers to look for and work on different methodologies to improve SDN technologies performance using WAPB or other approaches.

1.8 Scope and Limitations

This thesis works on enhancing the performance of Floodlight SDN controller using Workload Adaptive Packet Batching technique. The thesis implements the adaptive packet batching to Floodlight controller and evaluates it's performance. The performance evaluation of this thesis will include only throughput and latency. So in this thesis other metrics such as security, scalability is not included in performance evaluation. Because security and scalability tests require its own metrics and ways for checking the limitation of the network which is different from performance metrics.

The performance of both the enhanced version and Normal controller will be evaluated and compared. Since the testbed configuration of the different controllers is not easy, we will not evaluate and compare with other SDN controllers. In addition, the performance evaluations will not be done on real hardware, which costs a lot to establish an SDN network. Therefore, the evaluation is done on software tools.

1.9 Organization of the Rest of the Thesis

The remaining of this thesis report is organized as follows. Chapter 2 presents a literature review on the theoretical backgrounds of the SDN, Floodlight controller, factors of high-speed networking i.e. packet batching and the domain related to thesis will be covered. Also, Chapter 2 presents a

review of related works and it discusses research works that were conducted on Floodlight controller and system improvement on different approaches. The design of the Workload Adaptive Packet Batching system is presented in Chapter 3. The experimental results, evaluations, test results and discussions are described in Chapter 4. Lastly, in Chapter 5 conclusions and recommendations are made.

Chapter 2. Theoretical Background and Literature Reviews

This Chapter describes the theoretical background information and business domains about SDN, and the different techniques and literature reviews. A first short introduction about the innovation of SDN and the definition of SDN is included in Section 2.1. The traditional network architecture and the SDN architecture are presented in Section 2.2 and 2.3 respectively. In section 2.4, the OpenFlow protocol used in SDN is discussed and Section 2.5 reveals SDN controllers, the architecture, and the design features of the Floodlight controller. Section 2.6 explains the packet batching technique. In addition, Section 2.7 comprises of the review of the different works of the literature that are related to the research domains. Finally, the summary of the Chapter is included in Section 2.8.

2.1 Introduction

Martin Casado invented SDN, during his work at Stanford University. At that time, he was working on a project of virtual network systems [10]. This project finally, helped him to work on a protocol that could talk to and manage switches. In addition, later alongside with his coworkers, they coined the term SDN. Even though it initially stands for a specific meaning, it is the general ways of networking and networking technologies. In addition, this networking technology is aimed to make the networking have the properties of software systems as far as innovation, provisioning speed, and upgrade speed; by making the networks to be as flexible and as agile as compute is. Nowadays the network is not as they aspire, but this what they wish the technology would fit in the near future [11].

According to the definition of Open Networking Foundation "SDN is an emerging network architecture where network control is decoupled from forwarding and is directly programmable. This migration of control, formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity." [12]

2.2 Traditional Network Architecture

Apart from SDN, in traditional network architecture, the control plane and the data-forwarding plane are on the same device as shown in Figure 2.1. In the past, such kind of network architecture

has worked fine, but with today high tech and the virtualized world, it will be perplexing to meet the new virtual requirements. Currently, most of the IT-related departments are on the move to virtualize most of their servers. Such a process is challenging since the demand for both application and user mobility explodes.

As Figure 2.1 depicts, the traditional network couples both data and control, making the network difficult to automate. As Figure 2.2 describes, the SDN architecture combats the issue with traditional networks. In traditional network infrastructures, networking and network security functions - such as routers, switches, firewalls, and IPS – are implemented as physical appliances or devices. Networking flows are determined by network topology, with each individual network device making local decisions about the best way to move a packet along to its destination. However, with the emergence of cloud-based virtualized server and network environments, the ability to quickly deploy new applications without complex network changes has become a standard requirement. These types of network architectures are ill-suited to meet the requirements of today's enterprises, carriers, and end users. Software Defined Networking (SDN) is an emerging network architecture where network control is decoupled from the network infrastructure and transforming networking architecture [7].

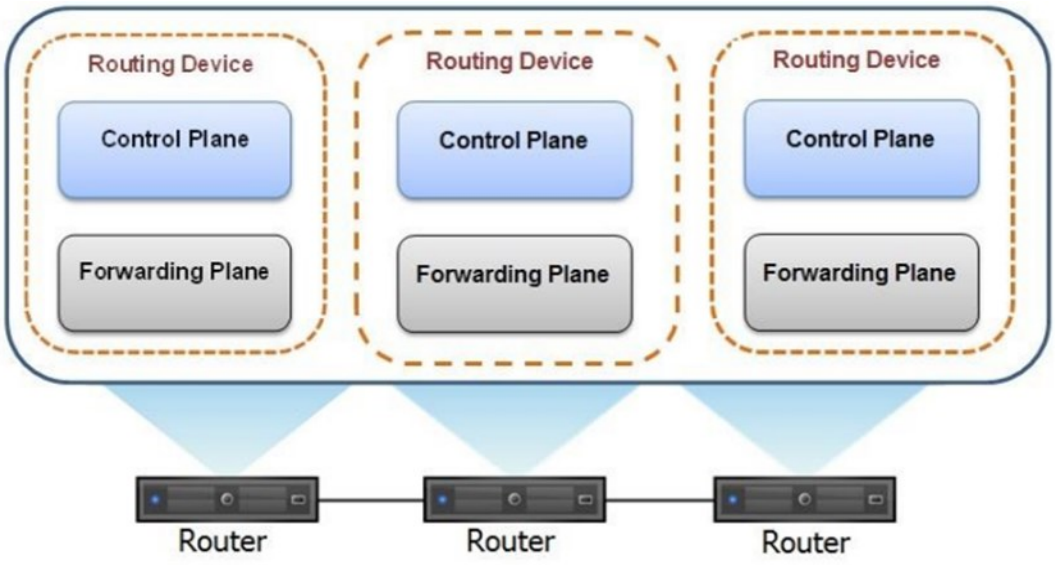

Figure 2.1 Legacy network architecture [7]

2.2.1 Limitations of Traditional Networks

Meeting current market requirements is impossible with traditional network architecture. The traditional network was not built in a way to meet today's requirements for end-users, service providers and enterprises; because of the limitations of traditional network architecture. Some of the limitations are discussed below [13]

- **Management Complexity:**

  The computer network technologies that are mostly being used, had always been built on a set of routing protocols that are engineered to connect hosts in a reliable manner over long distances with high speeds and different network topology. In order to meet the current industry requirements such as high availability, security, and extended connectivity, over the last decades, protocols have been designed in many ways that lead to separation, where each protocol is to solve a specific kind of problems, without implementing the concept abstractions. Such an approach has led to one of the main problems that network administrators are facing nowadays, namely network management complexity. One example is that adding or removing a device on a network has become a burden for network administrators, where several parts of the network switches, routers, firewalls, Web authentication portals, etc. has to be reconfigured, such as Access lists, VLANs QoS policies, routing protocols, and network topologies.

  In addition to the above, equipment vendor dependency and software versions compatibility have to be considered before making any modification to the network. As a result, network administrators keep their network rather static, in order to avoid or minimize the service downtime that can be caused by any change. Such nature of static network design limits the dynamic nature of server virtualization, which in turn increases the number of hosts that need connectivity.

  Before the time virtualization service was introduced, a single server connects to selected clients. While nowadays thanks to the services that virtualization provides, it has become possible for the applications to be spread over several virtual machines that connect to each other. In addition, in many cases, VMs have to migrate to obtain balanced workloads; such

functionality of virtualized platforms put many challenges on the traditional networking that was not designed for such dynamic flow changes.

- **Challenges in configuring policies:**

  In order to maintain an enterprise network policy, network administrators would need to configure many numbers of routers, switches that the network could able to hold. In virtualization, each time the IT adds a VM to the network, normally it takes up to hours if not days, where the network administrator needs to configure and adjust Access Lists (ACLs) across the whole network. With this kind of network complexity that we run today, it has become very difficult for network administrators to maintain a consistent setting for access privileges, QoS, and security.

- **Scalability issues**:

  As the data centers have a high demand to grow rapidly, at the same time, the network has to grow at the same speed. However, in reality, networks have become extremely complex due to the addition of hundreds if not thousands networking devices i.e. switches, routers, firewalls, etc. which are needed to be manageable and configurable. Network administrators have always relied heavily on bandwidth oversubscription in order to scale the enterprise networks, but with virtualization of data centers, traffic patterns have become highly dynamic which resulted in the difficulty in predicting the traffic patterns. Large operators, like Google, Amazon, eBay, and Facebook face even more extremely difficult scalability issues. Such operators are working on huge data processing systems and complex datasets, which are impossible to configure manually

- **Device vendor dependability:**

  Internet service providers and Data centers are always looking forward to implement new features and services to satisfy the changing industry requirements or end user demands. However, the ability to respond is restricted by the equipment vender's life cycles for the produced service and equipment, which in some cases can be about three years or even more. Also, the absence of open standardized interfaces has restricted the ability of network operators to customize the network to their required environments. Such mismatching

between network capabilities and customer requirements has affected the industry badly. Therefore, to tackle these issues the industry has invented an SDN paradigm.

2.3 The SDN Architecture

The traditional network resulted in limitations with the current traffic bursts. And, SDN was emerged in response to the demands of today's market. In SDN networking architecture as Figure 2.2 shows, the control is separated from forwarding, making the network directly programmable. Since the control is abstracted from forwarding functionality, this feature gives the administrators the ability to adjust the network traffic flows dynamically to satisfy the changing requirements.

The data plane is responsible for forwarding packets, whereas the control plane provides the intelligence in designing routes, setting priority and routing policy parameters to meet QoS and QoE requirements and to cope with the shifting traffic patterns. Open interfaces are defined so that the switching hardware presents a uniform interface regardless of the details of internal implementation.

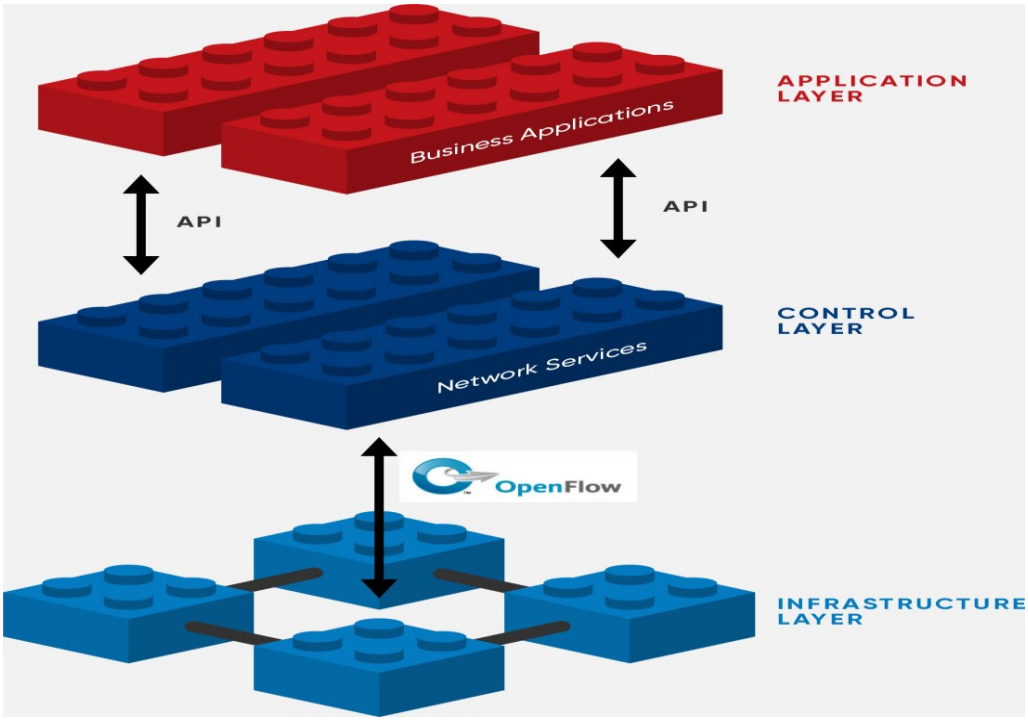

Figure 2.2 SDN architecture [8]

Network operators and managers programmatically configure this abstracted network, rather than manually configuring hundreds or thousands of the devices. In addition to the centralized controller, the network administrators can alter network behavior in real-time and deploy new applications and network services in a matter of hours or days. Rather, the admins can write a program and deploy on the open software devices apart from vendors' proprietary and closed software environments in the middle of the network. These all help the operators and the enterprises to choose SDN as networking paradigm.

2.3.1 Advantages of SDN

Software Defined Networking will change the way that network engineers and designers build and operate their networks to achieve business requirements. With the introduction of SDN, networks have become open standards, nonproprietary, and easy to program and manage. SDN will give enterprises and carriers more control of their networks, allow them to adapt and to optimize their networks to reduce the overall cost of keeping the network. Some of the main benefits that someone gets from SDN can be summarized below [14]:

- **Cost reduction**

  Installing SDN does not take a huge budget and labor. Even there are some SDN products that are open source, which are free for everyone. However, there is a license fee for some SDN solutions. Overall, it reduces the different overhead costs for deployment, Operational, and management perspective.

- **Network management Simplicity:**

  With SDN the network can be viewed and managed as a single node which will transfer complicated default network management tasks to be abstracted in a rather easy to manage interfaces.

- **Fast service deployment:**

  New features and applications can be deployed in a fast manner within hours instead of many days.

- **Automated configuration:**

  Manually configuration of tasks such as assigning VLAN and configuring QoS can be provisioned automatically.

- **Network Virtualization:**

  Since servers and storage, virtualization has become deployed more than before, networks can benefit from SDN to be virtualized as well.

To get the benefits that are mentioned above, SDN should implement desired changes in the network by installing the suitable forwarding rules through a southbound interface that connects the control layer and the infrastructure layer as depicted in Figure 2.2. There are different southbound interface protocols such as OpenFlow [15], Opflex [16], NETCONF [17], ForCES [18], POF [19], etc. There are other options for southbound interface implementation, but OpenFlow is the first choice of researchers and enterprise [15]. The northbound interface is the one who meets the requirement of implementation of business policy over application layer of controllers. Widely used northbound interfaces are for deployment of service policy to define traffic behavior.

2.4 OpenFlow Protocol

Among SDN protocols, OpenFlow is the most commonly used protocol [12] for the southbound interface of SDN, which separates the data plane from the control plane and standardized by the Open Networking Foundation (ONF). It works with the concept of flows, defined as groups of packets having the same header [15], which are implemented by different mechanisms depending on how the network is programmed. OpenFlow's simplicity and flexibility, allied to the high performance at low cost, ability to isolate experimental traffic from production traffic, and to cope with vendors' need for closed platforms [15], are probably among the main reasons for this success. Generally, OpenFlow is an open API that provides a standard interface for programming the data plane switches.

OpenFlow architecture consists of three basic concepts. The first is the network that is built up by OpenFlow-compliant switches that compose the data plane; second is the control plane consists of one or more OpenFlow controllers; and the third is a secure control channel that connects the switches with the control plane [20].

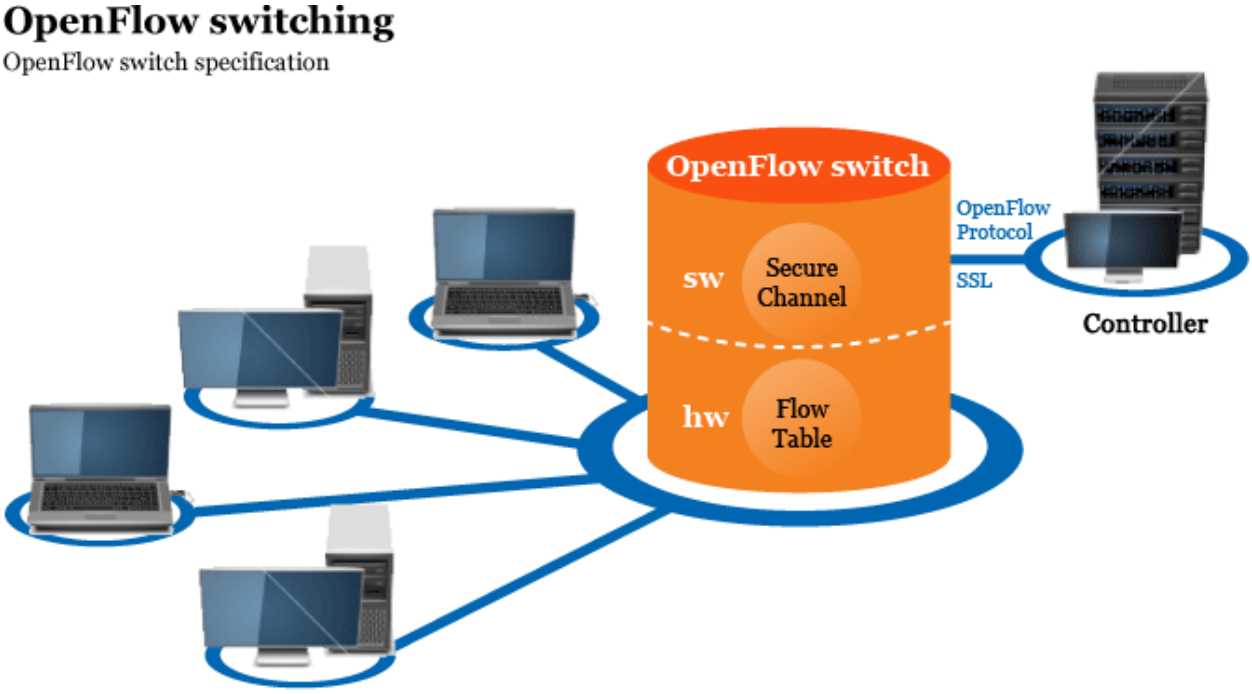

Figure 2.3: OpenFlow Switch Specification [20]

An OpenFlow-compliant switch is a basic forwarding device that forwards packets according to its flow table. This table holds a set of flow table entries, each of which consists of match fields, counters, and instructions, as illustrated in table 2.1. Flow table entries are also called flow rules or flow entries. The "header fields" in a flow table entry describe to which packets this entry is applicable. They consist of a wildcard-capable match over specified header fields of packets. To allow fast packet forwarding with OpenFlow, the switch requires ternary content addressable memory (TCAM) that allows the fast lookup of wildcard matches. The header fields can match different protocols depending on the Open Flow specification, e.g., Ethernet, IPv4, IPv6 or MPLS. The "counters" are reserved for collecting statistics about flows. They store the number of received packets and bytes, as well as the duration of the flow. The "actions" specify how packets of that flow are handled. Common actions are "forward", "drop", "modify field", etc [21].

Table 2.1: Flow table entry for OpenFlow 1.0 [21]

Header Fields	Counters	Actions

A software program, called the controller, is responsible for populating and manipulating the flow tables of the switches. By insertion, modification, and removal of flow entries, the controller can modify the behavior of the switches with regard to forwarding. The OpenFlow specification defines the protocol that enables the controller to instruct the switches, using a secure control channel.

2.4.1 Flow Tables and Packet Forwarding Mechanisms

An OpenFlow switch consists of one or more flow table(s) [21] that comprises different sets of entries each of them consists of fields named match, action, and counters as shown in table 2.1. The switch is responsible for processing packets, which are compared against their respective flow table. If a packet header matches a flow entry, an action for that entry is performed on the packet (e.g., the action might be to forward a packet out a specified port). If no match is found, the packet is forwarded to the controller over the secure channel. The counters are reserved for collecting statistics about flows. They store the number of received packets and bytes, as well as the duration of the flow.

When a switch receives a packet, it parses the header field and checks it against the rules in the flow table. If a match exists, an action in the flow table is considered. If several matches are found, packets are matched against a specific flow entry based on prioritization, i.e., the flow entry with the highest priority is selected. Then the switch updates the counters of that flow table entry. Finally, the switch forwards the packet out to a port. If the incoming packet didn't match any flow entry in the flow table, the switch would forward the packet to the controller to calculate which logic must be implied to this packet and similar future packets. The process of packet forwarding mechanism can be illustrated in the flowchart Figure 2.4

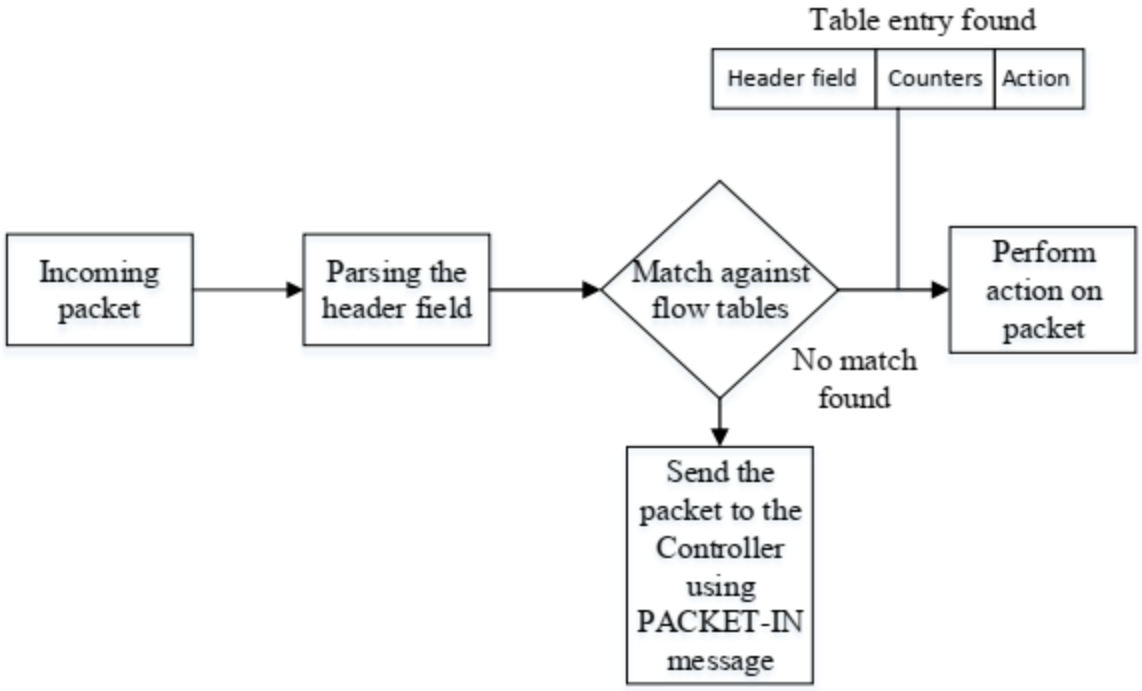

Figure 2.4: OpenFlow switch basic packet processing mechanisms [15]

2.4.2 Matching Flows

Packet header fields are extracted from the packet in order to foster table lookup. The header field in the flow table consists of different fields on which the incoming packets are compared to, these includes:

- Incoming switch port
- IEEE 802.3 Ethernet source and destination address
- IEEE 802.3 Ethernet type
- IEEE 802.1Q VLAN ID and priority
- IP source and destination address
- IP proto field
- IP Type Of Service (TOS) bits
- TCP/UDP source and destination ports

The incoming packets can be matched against various fields on every layer of the OSI model for a packet, ranging from the data link to the transport layer as well as on the incoming switch port. In order to match all the header fields, the special value 'any' can be used in the flow table.

2.4.3 Actions Performed on the Flows

An action set is associated with each packet generated. The default value for action is empty. The default value can be modified using write action instruction or clear action instructions that are associated with a particular match. If an ingress packet matches one of the match fields in the flow table, an action should be implied to that packet. The OpenFlow switch must support forwarding the packets action to each physical port. Additionally, there are virtual ports defined by the OpenFlow standard as special targets that the packets can be forwarded to.

Beside the forwarding action, there are other actions in the flow table:

- **DROP**: A required action indicated by an empty action list. All packets match an empty action list are dropped.
- **Enqueue**: This optional action can be used to put packets in a queue, which is associated with a port in order to provide QoS.
- **Modify-field**: This optional action is employed to modify a specific header field for the incoming packet. The following modification can be made:
    - Set VLAN ID and priority.
    - Strip VLAN header.
    - Modify the Ethernet source and destination MAC address.
    - Modify IP source and destination.
    - Modify IP TOS bits
    - Modify the transport layer source and destination ports.

2.4.4 Flow Types

Flows are packets with the same header. Flows populated from the OpenFlow controller can be classified into types: microflows and aggregated flows [21].

- **Microflows**: This type of flow is useful when a small number of flows needs to be installed in the switch, e.g. campus network. The flow tables in this type contain one entry per flow and exact matching is needed to perform an action.
- **Aggregated**: It is useful for big networks that require a large number of flow table entries, e.g. backbone networks. In this type, one flow entry (wildcarded) covers a large number of flows, each of which must belong to a specific category.

2.5 SDN Controllers

The controller is the core and the main part of the Network Operating System (NOS) in SDN networks that is a software platform over which, all the network control applications are deployed. It is placed between the network infrastructure and the application layer. This network operating system is responsible for coordinating and managing the resources of the whole network and revealing an abstract unified view of all components to the applications executed on top of it. This idea is analogous to the one followed in a typical computer system, where the operating system lies between the hardware and the user space and is responsible for managing the hardware resources and providing common services for user programs. Similarly, network administrators and developers are now presented with a homogeneous environment easier to program and configure much like a typical computer program developer would.

The controllers mostly include the different modules that provide different networking services for the deployed applications, including routing, multicasting, security, access control, bandwidth management, traffic engineering, QoS, processor and storage optimization, energy usage, and all forms of policy management, tailored to meet business objectives. The network services provided by them consist of network applications running upon the controller platform.

The logically centralized control and the generalized network abstraction SDN offer makes the SDN model applicable to a wider range of applications and heterogeneous network technologies compared to the conventional networking paradigm. For instance, consider a heterogeneous environment composed of a fixed and a wireless network comprised of a large number of related network devices (routers, switches, wireless access points, middleboxes, etc.). In the traditional networking paradigm, each network device would require individual low-level configuration by the network administrator in order to operate properly. Moreover, since each device targets a

different networking technology, it would have its own specific management and configuration requirements, meaning that extra effort would be required by the administrator to make the whole network operate as intended. On the other hand, with the logically centralized control of SDN, the administrator would not have to worry about low-level details. Instead, the network management would be performed by defining a proper high-level policy, leaving the network operating system responsible for communicating with and configuring the operation of network devices [22].

The controllers can be classified into two main categories [22] Open source controller and Commercial closed source distribution controller. The Open source controllers are available for research and development; therefore, it is represented as a single controller instance [1]with the ability to develop various APIs on their platform to perform certain tasks. There are many Open source OpenFlow controllers; the major distinction between them is the programming language they are written in and the design choice they use for developing the controller.

Even though the SDN technology has revolutionized the networking system, selecting a specific controller for a specific application needs a careful evaluation of the performance. Therefore, enhancing the performance available controller's using a given metrics has a great advantage for boosting the efficiency of the given network system.

2.5.1 Floodlight

Floodlight is the SDN controller, which is brought by Big Switch Networks that supports OpenFlow protocol for traffic management across the network. It is an open source, Java language based and multithreaded, which makes it a cross-platform controller and it is considered as one of the enterprise-class OpenFlow controller. Floodlight being apache-licensed makes developers and users use it for different purpose [23].

The performance comparison of different controllers indicates that Floodlight controller has lower performance [54]. So, studying the architecture of floodlight, identifying the responsible module

[1] Single instance controller refers to the ability of impending a one controller in a system, but it is not refers for the impeding of cluster controller that can be considered for other cases.

for performance degradation, and proposing and implementing that technique will result in performance improvement.

2.5.1.1 Floodlight Architecture

The Floodlight controller has modular architecture as shown in Figure 2.4 that makes it convenient for working with it. It comprises of the three most classifications, the core, Rest API, and Java API. Each of these classifications comprises of different services and modules. In the following Section, we will see the services classifications with some of the modules [24].

Core Architecture

The core architecture consists of the core, internal, and services of the common interest to SDN applications. The core network applications oriented to manage and orchestrate the operation of the control platform, including managing communication between other network services and shared data resources. It consists of different modules such as Floodlight-provider, module manager, switch manager topology management, device/end-station management, path/route computation, infrastructure for web access (management), counter store (OpenFlow counters), and a state storage system, that is well stitched by a module-management system and many more.

Below we will describe some of the important components of the controller architecture [23]

- **Floodlight-provider**

  Floodlight-provider manages the connections to routing devices by using the OpenFlow secure channel. This module specialized for translating received OpenFlow messages into events, which may be processed by other modules. In addition, it also ensures the ordering of the received messages before passing it to the interested modules.

- **Link Discovery Manager**

  Link Discovery is responsible for maintaining the link state information by using LLDPs packets, in simple terms, a link is set to be established between two switches if an LLDP is sent out of a port of one switch, and the same LLDP is received on the port of another switch.

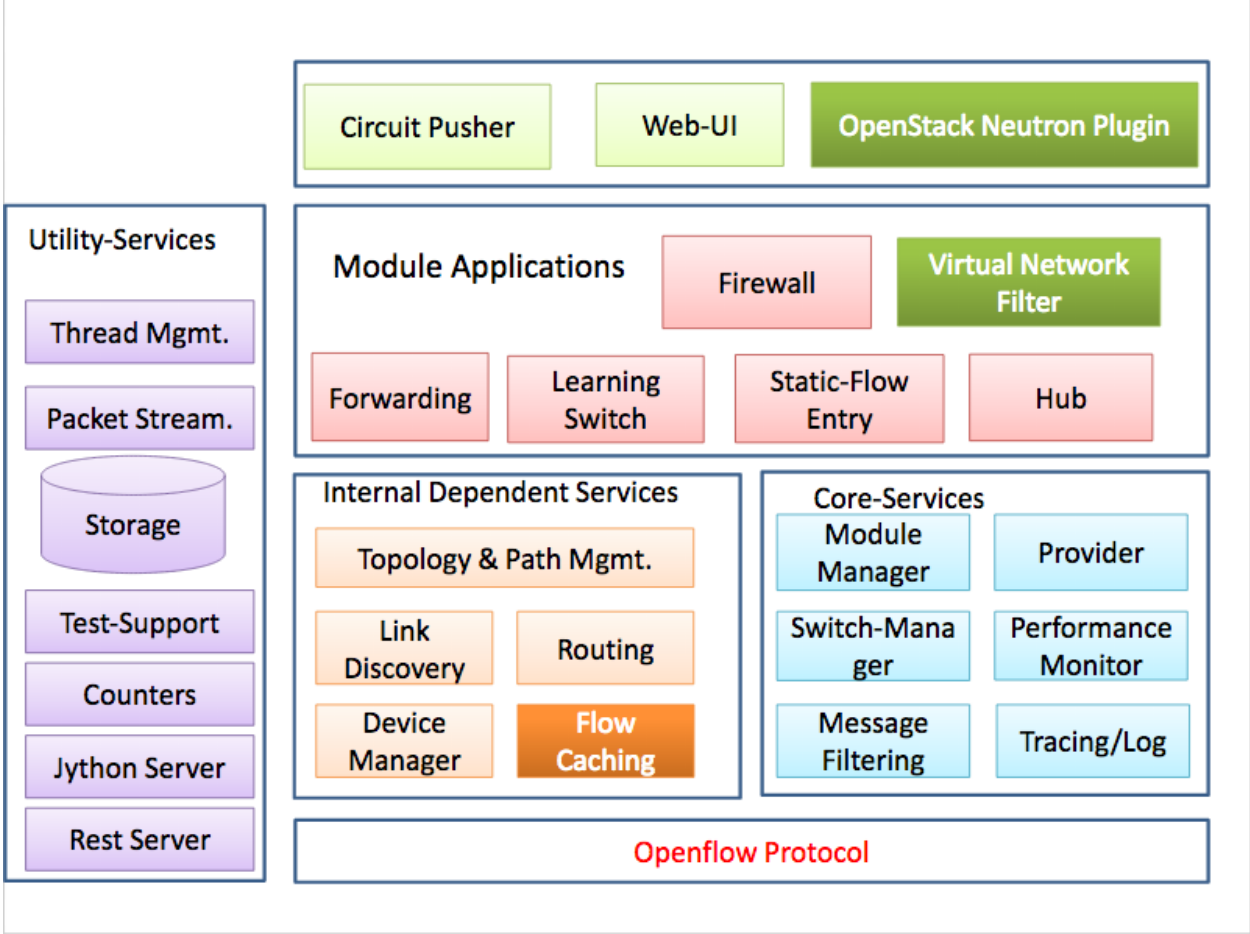

Figure 2.5: Floodlight controller architecture [31]

- **Topology management**

  It communicates the shortest path by using Dijkstra's Algorithm that is used to find the shortest path between one node and the other nodes.

- **Device-Manager**

  The Device Manager keeps track of the devices or end-stations via PacketIn[2] requests, rather the information (MAC, VLAN, etc) present in the PacketIn, with this, it can also learn to which port of which switch a particular device is connected.

- **Topology-Service**

  The Topology Service computes topologies based on link information from the Link-Discovery-Manager. It manages the topology of the different number of devices that are connected to the SDN-controller (Floodlight).

- **Packet Streamer:**

  Packet streamer can forward OpenFlow packets to any connected monitoring device in the network, using static flow entry that works on the principles of fixed size. It provides an interface to specify (in terms of one or multiple fields in the packet) the interested OpenFlow messages, typically termed as a filter.

- **Static Flow Entry:**

  An application for installing specific flow entry that includes match and action columns for a specific switch that is enabled by default.

- **Forwarding:**

  It is forwarding the application for packets, by default that supports topologies for its utilities. Packets are transferred from the source to its destination in order to share the information across the network.

- **Firewall:**

[2] PacketIn messages are messages that are sent from the controller to the switch that usually consists of a header and a buffer id.

An application to apply Access Control List rules to restrict specific traffic based on the specified match.

2.5.1.1.1 Rest and Java -API Based Applications

The northbound interface that connects application and controller in Floodlight uses both Java and Rest-API. The northbound interfaces include the Java and Rest API that is used to facilitate the communication between the controller and the applications.

Floodlight comes with few applications that use the exposed REST APIs. Circuit Pusher utilizes Floodlight rest APIs to establish a path between any two IP-addressable devices by adding flow-entries in all the switches that constitute the path. In addition to circuit-pusher, Floodlight can be run as the network backend for OpenStack using a Neutron plugin. There are two main components to this solution: a RestProxy to realize the connectivity between the Floodlight controller and OpenStack Neutron. VirtualNetworkFIlter to realize the Neutron APIs. The VirtualNetworkFilter module, which implements L2-address based network isolation, is not dependent on OpenStack-Neutron and can be activated via a configuration file. However, the RestProxy plugin was designed to run as part of OpenStack's Neutron service.

2.5.1.2 Floodlight Design Features

Floodlight SDN controller includes the different features that make it preferable by SDN community [24]. Some of these features are described as follows.

1. **OpenFlow Support**

The controller does not only support OpenFlow networks, but it can support non-OpenFlow networks also. This makes it manage multiple combinations of devices through OpenFlow and other protocols that are described in Section 2.3.1. Currently, there are different versions of OpenFlow, starting from OpenFlow V1.0 to V1.6 each of them has different features. The Floodlight controller currently supports v1.4 that enhances switch synchronization for multiple switch configuration-using bundle and enhanced flow table scalability for MAC learning/forwarding using a synchronized table.

2. **Network Virtualization**

One of the most important benefits of an SDN is network virtualization. Network virtualization is nothing new. One type of network virtualization that has been in production networks for decades is a virtual LAN (VLAN). VLANs partition an Ethernet network into as many as 4,094 broadcast domains and have been a convenient means of isolating different types of traffic that share the same switched LAN infrastructure. Another type of network virtualization that has been in production networks for a decade is Virtual Routing and Forwarding (VRF) instances. VRF is a form of Layer 3 network virtualization in which a physical router supports multiple virtual router instances, each running its own routing protocol instance and maintaining its own forwarding table

3. **Performance**

As previously described, one of the key functions of an SDN controller is to establish flows. As such, two of the key performance metrics associated with an SDN controller are the flow setup time and the number of flows per second that the controller can set up. These performance metrics heavily influence when additional SDN controllers are deployed. For example, if the switches in a production SDN initiate more flows that can be supported by the existing SDN controller(s), more controllers must be added.

Flows can be set up in one of two ways: proactively or reactively. Proactive flow setup occurs before the packet arrives at the OpenFlow switch, and so when the first packet arrives at the OpenFlow switch, the switch already knows what to do with the packet. This results in negligible setup delay and no real limit on the number of flows per second that the controller can support. Ideally, the SDN controller pre-populates the flow tables to the maximum degree possible.

In contrast, reactive flow setup occurs when the OpenFlow switch receives a packet that does not match the flow table entries and hence the switch has to send the packet to the controller for processing. Once the controller decides how to process the flow that information is cached on the OpenFlow switch and the SDN controller determines how long to keep the cache alive. The time associated with reactive flow setup is the sum of the time it takes to send the packet from the OpenFlow switch to the SDN controller; the processing time in the SDN controller and the time it takes to send the flow modification message back to the OpenFlow switch and to populate the flow table on the switch. The key factors that affect flow setup time include the processing power of the switches that are attached to the controller and processing and I/O performance of the Controller.

4. **OpenStack support**

OpenStack is an Open source framework developed for cloud-based services where virtual servers and other services are made to be available for the customers.

5. Routing

Routing is the process by which the networking system decides where to send a packet. It also discovers the path using the combination of Link discovery and management, topology management, and a routing engine.

2.6 Packet Batching

Batch is normally related to huge data-processing systems and could take days for processing. Now batch is an integral part of different modern systems. Batch jobs process different complex tasks in today high tech systems [25]. Some perform a single function, whereas others perform more functions that are traditional [25]. Such functions include processing large numbers of records, performing complex calculations, and compiling statistical information. Batches are normally run in sequence slowly. Smart batching that are implemented for networking systems use the parallelization technique to save time and other resources. So, implementing packet batching to the SDN controllers will result in easily processing the high traffic in the current networks

Packet Batching is a technique used by OpenFlow controllers – where multiple bytes are read from or written to the underlying network using a socket buffer; this feature helps prevent the overhead of a socket (read/write) system call for a large number of pending requests, thereby improving the overall throughput of the controller.

There are two types of Packet Batching methods

1. Fixed packet batching
2. Workload adaptive packet batching

If the size each batch is fixed, the fixed batching policy will be used for this type of approaches. The scheduling is done every fixed time slot T and for some appropriate batch size C. In the extreme case, and indeed for many algorithms, T = 1 is assumed, scheduling just one-time slot at

a time. The general algorithms schedule batches of fixed time slots using a pre-calculated batch size that is fixed. These are called policies/algorithms fixed batch schedules (FBS) [26].

In the Workload Adaptive Packet Batching methedology, the batch length increases as workload increases. With contrast to fixed packet batching which has fixed batch length, which could not necessarily serve all the workloads waiting for the single batch, it serves all the workloads. It uses Adaptive Batch scheduling (ABS) [26].

2.7 Related Works

The innovation of SDN technology has attracted many researchers and the Data centers to work on the controller's design and improvement. In this Section, the researches that are done on the Floodlight SDN controller design principles, like the architectural evaluation of SDN controllers, hardware mechanisms to improve SDN controller's performance, and Floodlight QoS module are reviewed. Additionally, adaptive batch processing system will be analyzed and presented.

2.7.1 Floodlight SDN Controller Design Principles

Syed Abdullah et. al. [27] identified the most performance bottlenecks and best architectural pillars for designing and developing SDN controllers. They had evaluated the performance of OpenFlow controllers; NOX [28], Beacon [29], Maestro [30] and Floodlight [31] .This work is aimed at the evaluation of detailed system input-output (I/O) and overall process efficiency of the controllers with a different number of devices. Moreover, they used the result of the evaluation to identify new architectural guidelines, which could be used to enhance or design new OpenFlow controllers. Depending on the result, they have analyzed and made recommendations. First, controllers designed for high throughput (i.e., processing a high amount of traffic) should use packet batching and switch partitioning. Secondly, the controller designed for delay-sensitive control plane applications should use the packet-batching and task batching technique that could adapt the behavior of the workload nature. Floodlight controller with static packet batching design affect the performance of the system, this why the researchers come with a new design idea. The authors come with a good performance proliferation but have not implemented it with a mechanism that could adapt automatically as they recommended.

MR-Flood [32] is a system that combines Hadoop MapReduce framework and Floodlight SDN controller. MapReduce, the big data analytics framework that processes huge data across large clusters motivated the authors, which results in performance degradation in the system. The researcher proposes a bandwidth and latency aware MapReduce service composition and self-configuration for big data Analytics with SDN, by extending and leveraging Hadoop MapReduce framework and Floodlight SDN controller. Then, the researcher has implemented a MapReduce framework into the SDN controller to utilize the bandwidth effectively across all the devices in data centers. It was designed in the way that MapReduce could be used as an application that was

installed in the controller. Moreover, using the abstraction of SDN they have gathered the network parameters such as bandwidth, latency, and CPU to provide a better resource to the device clusters without degrading the performance. All this was possible by using the Floodlight SDN controller.

The researchers [33] have proposed hardware mechanisms to improve the performance of the SDN controllers. Since the control and the data path are separated, it causes the control path delay on packets and the flows forwarding, because the control has to send a signal from the remote device. In addition, they have examined the packet processing of the switches and packet forwarding behavior in the kernel module, to meet the demand of increasing traffic patterns and huge data processing network. They have identified two factors that affect Open Vswitch. Firstly, in the switch architecture, the daemon is situated in the user space having low space for flows storage. Secondly, the data path that is situated in a kernel module has a low processing ability than a controller that is a little bit faster. Depending on these factors, network performance is limited by buffer size, delay, bandwidth, and packet loss. In order to gain the advantage of SDN, designing the new software stack that makes TCAM bigger memory is advisable. And designing the architecture of the processor to increase the management of CPU processing ability for the software switches and to handle many control path request is recommended. They have work in such a way that the performance could be enhanced, but the solution is limited to hardware design and implementation, which takes a large amount of budget.

Floodlight QoS module [34] was developed to manage the high traffic control using the SDN controller. This module of packet-switched networks comprises of different factors Such as the reliability and availability of service, delay, scalability, effectiveness, the grade of service, etc., which has an impact on the role of packet handling mechanisms. These properties of packet-based processing can cause low throughput, delay, loss of packets, corruption errors, latency, jitter, disordered delivery and more. The QoS was developed as part of Floodlight in the application module. The proposed QoS mechanisms have used the approach of a software-defined QoS module, DSCP (Differentiated Services Code Point) and common queuing techniques within OVS. The DSCP contains an 8-bit value in the IP header that declares the different variety of services.in the given network section. The module performs actions such as matching, classification, flow insertion, flow deletion, and policy handling. The module is responsible for tracking and storing a variety of services with their associated DSCP values, applying policies that either take advantage

of a class of service or apply a type queuing technique on queues that are attached to ports on a switch.

2.7.2 Adaptive Batch Processing Systems

The dynamic scheduling network application systems [35], includes the models for data scheduling in high-performance packet switches. The scheduling algorithms assign incoming tasks over many amounts of queues with the objective of guaranteeing properties such as throughput maximization, QoS, and load balancing. The author has mainly developed two algorithms Projective Cone Scheduling (PCS) and Adaptive Batch Scheduling (ABS). The PCS algorithm is designed in the way that the throughput is maximized-the incoming jobs will be served immediately. That means the PCS maps each service configurations among the different workload nature. The algorithm also makes a system configuration to make the system stable throughout the lifetime. In order to make the system stable, it includes the necessary features such as, as workload arrival patterns change, the choice of service configurations must change, and if one or more workload queues increases faster than the other queues, service attention will shift toward those queues. However, the propagation delay incurred in the network, cause contention over multiple periods. Therefore, the researcher has introduced ABS algorithm that the length of each schedule is adapted to the demand of the system. The researcher has implemented the algorithms using local search over a large volume of computational complexity and under incomplete, delayed, and noisy information

The researchers [36] have proposed an algorithm that adapts the workload nature of different streaming systems based on the ABS algorithm [35] that is discussed above. The MapReduce-style that is processing a batch of jobs have motivated the researchers, which works on stream processing systems on continuous and huge datasets. Implementation was supported using Spark Streaming's sliding window-based operations that continuously generate the count of records. The algorithm was evaluated on different volumes of streaming workloads under different combination of data rates and operating conditions. Even though, the stream data operation is complex they have achieved system stability and performance improvements. The nature of data streaming is somewhat related to SDN controller traffic pattern, in which both of them work on complex traffic. Therefore, ABS is adopted in designing workload adaptive packet batching system for Floodlight controller.

The incremental data processing systems includes batched stream processing systems such as Comet [37] and Spark Streaming [38]. They collect received streaming data into batches and periodically process those using MapReduce-style batch computations. In both systems, the periodicity of the batch computation is left to the developer. Incremental bulk data processing systems such as CBP [39], Percolator [40], and Incoop [41] allows updated view of processed data set to be maintained by incrementally and efficiently recomputing the updates to the input data. In such systems, the recomputation is triggered whenever an update to the input dataset is detected. Percolator briefly discusses the adjustment of the batch size of data updates based on server load. In general, this has not been well explored in such systems, and our adopted algorithm usefully works with incrementing batch size up to the availability of system resources.

Other stream processing systems such as Borealis [42], TimeStream [43], Naiad [44], and Storm [45] are based on the continuous operator model. In this model, the streaming computation is expressed as a graph of long-lived operators that exchange messages with each other to process the streaming data. For efficiency, many of these systems employ batching of messages between pairs of operators. Even though such type of system that exchange messages with each other are not the focus of this thesis, our adaptation technique works by collecting statistics from previous batch intervals to make the system more repost.

2.8 Summary

The increase in number and type of modern networking devices, technologies, and computing need leverages the network that was designed prior for fulfilling the demand. Since the traditional networks could not cope up with the technological innovation and need, SDN is invented to address the issues with such kind of networks. SDN manages the networking system by separating control and data. This separation gives SDN the chance to orchestrate the traffic easily by using programmability. The SDN controller software enables to control the underlying networking devices without manually configuring the individual devices. The application that is installed on the controller modifies the flow table rules automatically using the OpenFlow protocol.

The OpenFlow switch contains one or more switches that comprise different sets of entries each of them consists of fields named match, action, and counters. The match field matches the flow entries depending on actions and flow table rules. The action field performs different actions such

as enqueue, drop, and modify on the flows. The counter counts and stores the flows in size and type. Therefore, in SDN when a packet arrives at the switch the switches matches them against different OpenFlow rules and if there is no match it sends back to the controller to calculate the logic that could be applied for such types of packets.

Floodlight controller is one of the controller that implement OpenFlow protocol using the static packet batching method. The static batching approach uses a fixed scheduling policy, where the job of the controller process is limited. Effectively pulling the power of SDN, the networking technologies must imitate with the dynamic and growing traffic.

The researches that have currently done on SDN suggests that there are bottlenecks with the performance of the system. The bottlenecks are nature of the controller design methodology such as packet batching, inability to aware to bandwidth and latency, the low memory storage for flows and rules, and the slower data path that is unable to cope up with the speed of processor which is much faster than it. And the dynamic scheduling network application systems revealed that the increase in traffic pattern leads to, making the system unstable due to contention. These mentioned bottlenecks are the result of buffer size, delay-bandwidth, and packet loss. Because the system is operating on modern networking system that depends on complex traffic. The researchers have used different approaches to overcome the factors that limit the SDN networks.

The performance evaluation that identifies the factors that obstructs the performance of SDN controller, has recommended implementing techniques that could go in line with the current traffic nature like packet batching mechanism. The other researchers have successfully implemented MapReduce on the Floodlight controller in order to design and develop a system that could be aware of bandwidth and latency. And the research that studied the packet processing and packet forwarding nature of the switch comes with the way to increase throughput and decrease delay by increasing the TCAM memory and new architecture for the processor. More studies were done on making the network system adapt to the existing demand. They have developed PCS and ABS algorithms in order to divide the arriving system tasks to the existing queues and to make the system to serve the tasks immediately. This helped them to design a mechanism to adapt to the work nature of the different applications. Moreover, The incrementally growing and continuous operator models depend on continuously growing the number of buffers depending on the tasks that are arriving at the system. But, it makes the system unstable because there is no match between

the arriving tasks and the system serving ability. This all methodologies mentioned here above, have some guidelines to design a new system to grow with the nature of traffic but does not address the adaptability concept in the way that it could be implemented to Floodlight controller. So, in this thesis, we will design an approach to make the batch size that could grow with the system and the traffic pattern of different SDN networks.

Chapter 3. System Design and Implementation

3.1 Introduction

As discussed in Chapter 2, we had considered different approaches of the Floodlight, SDN controller, and adaptive batch processing systems. In this Chapter, the Workload Adaptive Packet Batching technique is proposed in order to enhance the performance of the Floodlight controller. The following Sections comprise the discussion of the details of the proposed system with the description. In Section 3.2, the proposed system is presented. Section 3.3 discusses the software tools used in building and testing the system. Finally, Section 3.4 covers the summary of this Chapter.

3.2 Proposed System Design

Figure 3.1 and Figure 3.2 as shown below describes the proposed system. The first one consists of the different components of the Floodlight controller and the workload adaptive algorithm. The system consists of the SDN infrastructure and the control layer. The application layer, which is on top of the control layer, is hidden from this design because it contains the different applications that are user dependent. The job is used in this paper to describe any traffic pattern that arrives at the switch or networking devices. This may include tasks that a user is requesting to be processed, data computation, and analysis that is done on a huge amount of datasets.

The Floodlight controller manages the OVS via the OpenFlow protocol rules. The job first arrive at the infrastructure layer; specifically the OVS .Upon receiving the job, OVS manager of the switch creates the batch size according to the workload adaptive algorithm for the batch of the tasks as depicted in Figure 3.2. Moreover, the switch's flow table will install a rule to forward packets from the batch of tasks to the created batch sizes. The Floodlight controller is responsible for coordinate the whole system.

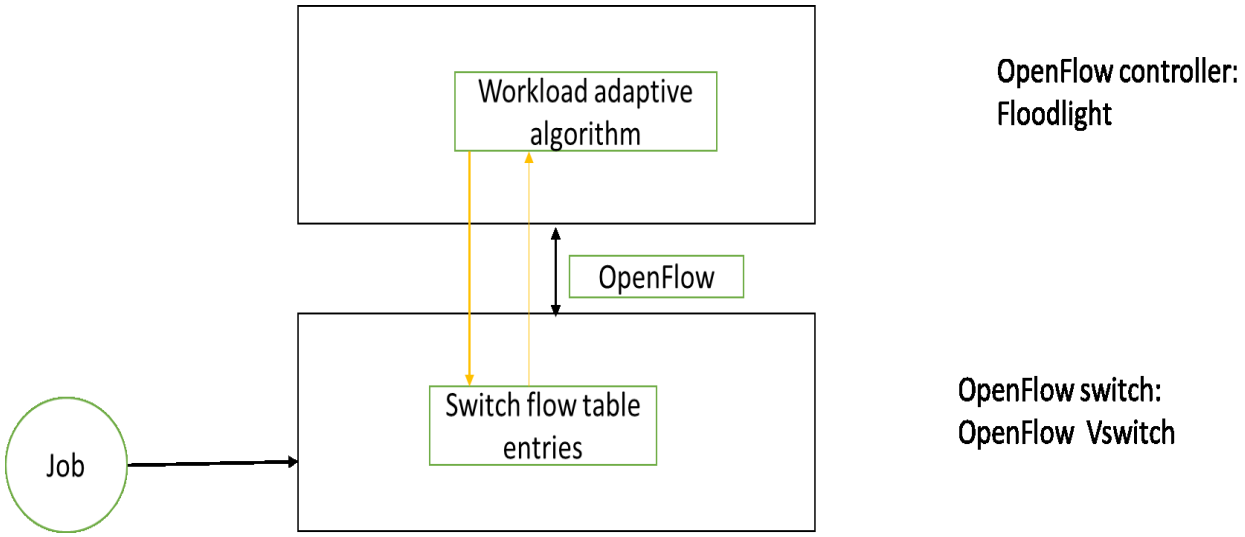

Figure 3.1: The proposed system design

The WAPB technique is the technique that inherits the nature of high-speed packet switching network optimization architectures [26, 27, 35, 36, 46]. The Floodlight controller implements the WAPB, in order to effectively handle the network traffic and increase its overall efficiency. The flow table uses different rules in order to process the data packets that arrives at the device. There are three different pillars in this methodology as figure 3.2 describes: namely, batch module, batch processor, and the next batch calculator. The packets that arrive have to wait in the batch module in order to get access. The processor module receives the tasks from the batch module and process them. The next batch calculator identifies the maximum batch size that could results in high throughput using the FPI [46], which is an optimization technique. The aim is to minimize the number of tasks waiting in the batch module and serve as soon as possible. If the FPI converges, it means the system has identified a balanced batch size by learning the number arriving packets in the batch module.

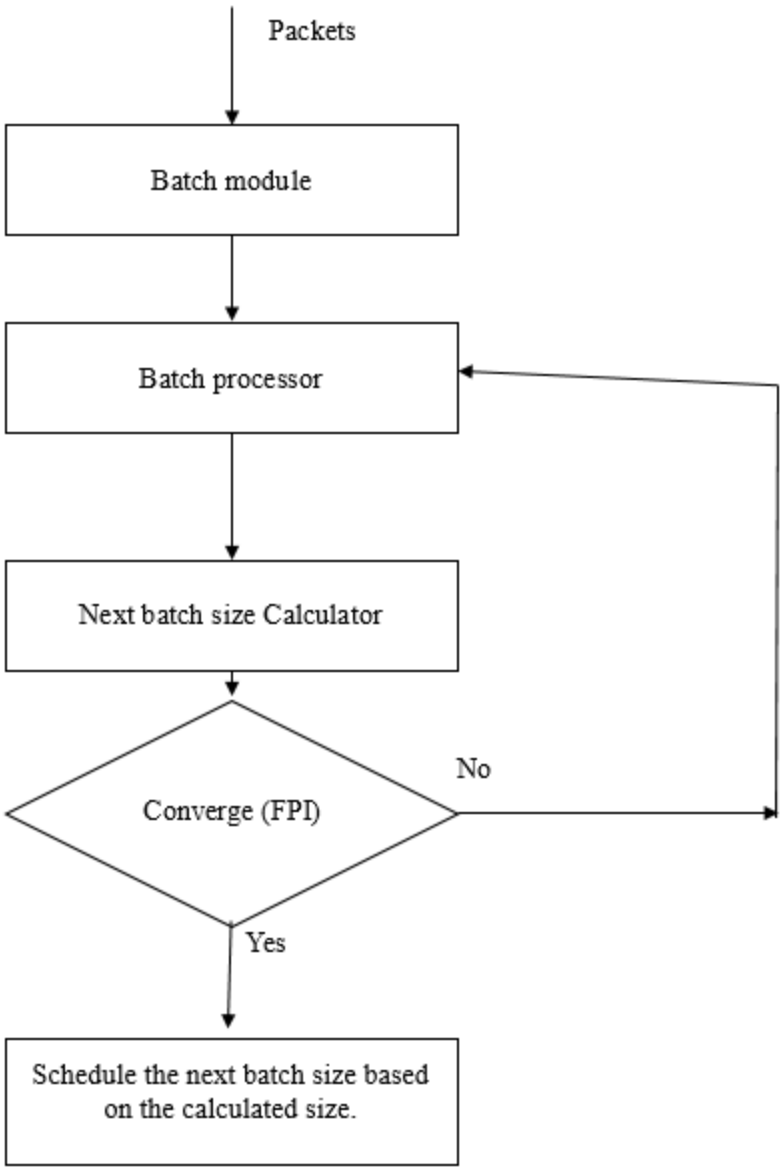

Figure 3.2: Workload adaptive packet batching algorithm flowchart

Batch module

The batch module is intended to organize the jobs into the different batches. The batch size is left for the controller to figure out depending on the nature of jobs. The big issue is to identify the first batch size, after that the next batch calculator will be expected to identify it. So, the first batch size that is in time of microseconds will be the time it takes to process that batch of jobs that arrive initially. After the initial batch, the next batch sizes will be calculated based on the size of the first two batches. This module works on the first two batches alone and waits for the next batch

calculator to work on the robust batch size. Then, it categories the jobs into varying batch sizes depending on the calculated size.

Batch Processor

Batch processor receives a different batch of jobs and sends them for processing to the Floodlight controller. The time it takes to process a batch is kept that will help for FPI to check the convergence of the batches

The existence of continuously varying workload nature of the Floodlight SDN controller may be challenging, to attain the stable batch size. However, at an optimal batch size, batch-processing time is equal to the batch size. At this stage, the incoming jobs are served immediately, so that there wll be no queues that are waiting, that could affect the controller's adaptability.

Fixed Point Iteration (FPI)

The Fixed-point Iteration method [46] is an iterative algorithm to find the solution x of a fixed-point equation f(x) = x, for a function f. The algorithm is very simple and operates as follows.

1. Start with an initial guess x_1

2. Iterate using $x_{n+1:}= f(x_n)$ for n = 1; 2;

This method of finding fixed points is applicable to a type of functions that satisfy conditions imposed by the Contraction Mapping Theorem [46].

In our system, we wish to find the point x according to the Contractive Mapping Theorem so that $x_{n+1} = C \times x$ (x represents the batch size) where $0<C<1$. For the purpose of this explanation, let us assume we know the $x_{n+1,}$ solving the fixed point of the function $f(x) = x_{n+1}/C$, so that $f(x) = x_{n+1}/C = x$.

We can then iterate using the Contractive Mapping Theorem as follows:

1. Start with an initial guess x_1

2. Iterate using $x_{n+1:}= f(x_n) = x_{n+1}/C$, for n = 1; 2;:::.

At any batch interval x_n, the next batch size will be determined by

$$\text{Batch processing time} = C \times \text{Batch size}, \text{ Where}$$

$$Batch\ size = \frac{1}{C} * \ Batch\ processing\ time$$

The Contractive mapping theorems constant C will depend on the following two cases

1. If the processing time is decreasing, the batch size will be increased using the following equation.

$$Batch\ size = \frac{1}{C} * Batch\ processing\ time$$, where C=0.55

2. If the processing time is increasing, the batch size will be decreased using the following equation.

$$Batch\ size = C * Batch\ processing\ time$$, where C=0.55

The constant C was taken as 0.55. It depends on the system and configured as a value that will help the system to adjust to the incoming traffic. It increases the batch size if the processing time is decreasing and decreases the batch size when the processing time is increasing.

Finally, using the fixed-point iterations there is no manual batch size configuration because the system will adapt the workloads. The algorithm automatically adjusts the batch size based on the convergence of the fixed point iteration as depicted in Figure 3.2 This makes the algorithm very robust with respect to changing operating conditions.

3.3 Software Tools

In this thesis we will use different software tools, some of them are Floodlight controller from Big switch, Mininet, and Cbench. And with these, we use Eclipse ant for developing and building the Floodlight according to the methodology proposed in Section 3.1.

3.3.1 Floodlight

As explained in section 2.5.1, the Floodlight controller is an enterprise OpenFlow controller. Some of the main features it provides are:

- Apache-licensed, multi-purpose Open source software [47].
- The community of Developers including Big switch [48] is actively engaging in it.

- Floodlight is easy to use, build and run so that it is widely used by the network community to easily learn and work with SDN.
- It is designed in the way to scale to the widely increasing number of OpenFlow enabled devices [47].

3.3.1.1 Floodlight Controller Installation and Running

Since Floodlight is written in Java and thus runs within a JVM or eclipse during the build. We will need the following Prerequisites [49]:

- Linux Ubuntu 14.04 LTS.
- JDK and Ant which can be installed by running the following command:

  sudo apt-get install build-essential default-JDK ant python-dev eclipse

After preparing all the prerequisites to run Floodlight it is time to download and run the controller by using the following commands

git clone git://github.com/Floodlight/Floodlight.git
cd Floodlight
git submodule init
git submodule update
ant
sudo mkdir /var/lib/Floodlight
sudo chmod 777 /var/lib/floodlig

3.3.2 Mininet

Mininet is a software emulator that creates a variety of network. Mininet instantaneously creates Virtual Network on a machine that can be VM, cloud or native running on the real kernel, switch and application code**.** Using the Mininet CLI and python API [50], it helps us to interact with the network easily. Moreover, it can be customized, shared with others, or deployed on real hardware; which makes it different from software simulators. For these reasons, Mininet is suitable for development, teaching, and research purpose. Mininet is also designed for OpenFlow and Software-Defined Networking, to develop, share, and experiment with different networking

systems. Moreover, Mininet is developed and supported, and released under a permissive Open Source license, which gives freedom for developers to easily access and use the software.

Mininet supports OpenFlow protocol that provides an interface between the control plane and the data-forwarding plane. OpenFlow protocols are used to control the packet flow as per the API is written on the controller. Mininet also has support for a variety of topologies and ensures the availability of custom topologies.

3.3.2.1 Mininet Installation

The Mininet installation procedure can be found in the Mininet official page [51].After installing Mininet, running needs to set the following parameters for running and creating networks.

- Select the type of switch: Open vSwitch
- The OpenFlow controller address that the network will connect to IP address and port at which the Floodlight controller is running
- Specify network topology: Include the number of switch and hosts for the network

Mininet Installation Procedure

To install natively from source, first you need to get the source code:

git clone git://github.com/mininet/mininet

The above git command will check out the latest and utmost Mininet. Running the last tagged/released version of Mininet - or any other version – may need to check that version out explicitly:

```
cd mininet
git tag  # list available versions
git checkout -b 2.2.1 2.2.1  #  Mininet version to be  installed
cd ..
```

After the source directory has set, the command to install Mininet is:

mininet/util/install.sh [options]

Typical install.sh options include:

- -a: install everything that is included in the Mininet VM, including dependencies like Open vSwitch as well the additions like the OpenFlow Wireshark dissector and POX. By default, these tools will be built in directories created in your home directory.
- -nfv: install Mininet, the OpenFlow reference switch, and Open vSwitch
- -s mydir: use this option before other options to place source/build trees in a specified directory rather than in your home directory.

Therefore, from the above options one could wish to use one (and only one) of the following commands: But, in this thesis, the second option was used that is enough to create the virtual network. The other options, the controller we will be using Floodlight controller and the default is not necessarily needed.

- To install everything (using your home directory): *install.sh -a*
- To install everything (using another directory for build): install.sh -s mydir -a
- To install Mininet, user switch, and OVS (using your home dir): install.sh -nfv
- To install Mininet, user switch, and OVS (using another dir:) install.sh -s mydir -nfv

After the installation has completed, test the basic Mininet functionality:

sudo mn --test pingall

3.3.3 Cbench (Controller Benchmark Tool)

The Open Networking Foundation has defined, "(controller benchmarker) is a program for testing OpenFlow controllers by generating packet-in events for new flows. Cbench emulates a bunch of switches which connect to a controller, send packet-in messages, and watch for flow-mods to get pushed down." [52] Using Cbench tool two main benchmarking tests can be done, throughput and latency by counting the number of flows in flow table entry and time. The detailed description of how Cbench works to test throughput and latency; and the Cbench algorithm is explained in Section 4.4.1.

3.3.3.1 Cbench Installation and Running

The instruction for installing the Cbench tool can be found in benchmarking controller performance on Floodlight documentation [53].

Cbench installation and configuration procedure

sudo apt-get install autoconf automake libtool libsnmp-dev libpcap-dev

git clone git://gitosis.stanford.edu/oflops.git

cd oflops; git submodule init && git submodule update

git clone git://gitosis.stanford.edu/openflow.git

cd openflow; git checkout -b release/1.0.0 remotes/origin/release/1.0.0

wget http://hyperrealm.com/libconfig/libconfig-1.7.2.tar.gz

tar -xvzf libconfig-1.7.2.tar.gz

cd libconfig-1.7.2

./configure

sudo make && sudo make install

cd ../../netfpga-packet-generator-c-library/

sudo ./autogen.sh && sudo ./configure && sudo make

cd ..

sh ./boot.sh ; ./configure --with-openflow-src-dir=<absolute path to openflow branch>; make

sudo make install

cd cbench

Running Cbench needs to declare different parameters. The options of these parameters and the description for each option is included in Table 3.1.

Table 3.1: Cbench Running Options

Option	Description	Default value
-c/--controller	<str> hostname of controller to connect to	("localhost")
-d/--debug	enable debugging	(off)
-h/--help	print this message	
-l/--loops	<int> loops per test	(16)
-M/--mac-addresses	<int> unique source MAC addresses per switch	(100000)
-p/--port	<int> controller port	(6633)
-r/--ranged-test	test range of 1..$n switches	(off)
-s/--switches	<int> fake $n switches	(16)
-t/--throughput	test throughput instead of latency	
-C/--cooldown	<int> loops to be disregarded at test end (cool down)	(0)
-D/--delay	received (in ms) <int> delay starting testing after features_reply	(0)
-i/--connect-delay	to the controller <int> delay between groups of switches connecting	(in ms)
I/--connect-group-size	<int> number of switches in a connection delay	(1)
-L/--learn-dst-macs	macs before testing send gratuitous ARP replies to learn the destination	(on)
-o/--dpid-offset	<int> switch DPID offset	(1)

3.4 Summary

This chapter covered the design of the WAPB for Floodlight SDN controller based on ABS algorithm. The system comprises of different modules within the Floodlight controller. The WAPB, which will be implemented to the controller layer, is the one that is responsible for enhancing the performance of the Floodlight controller. In addition, open vSwitch is situated in the infrastructure or network devices layer that handles the network traffic according to the

supervision of the controller. The WAPB applies the batch policy, that adapts the workload nature of the network, to enhance the performance of the system. In order to do this, the underlying network is created using the Mininet emulator.

The network that is created by the Mininet will be evaluated for the performance under the different number of device configurations. The Cbench tool evaluates the performance of the system. Throughput and latency are the two main metrics that are used to evaluate system performance. Throughput measures the average flow setups per second across the different networks that are connected to the controller. While latency measures the time it takes to install the flows in each switch.

Chapter 4. Evaluation Results and Discussions

4.1 Introduction

This Chapter describes the implementation details and experimental results of the proposed design for WAPB Floodlight controller. Different experiments were performed to verify the performance of the proposed approach. Section 4.2 and 4.3 presents the implementation and the experimental setup of the proposed system respectively. Section 4.4 describes the Floodlight controller evaluation benchmarking methodology. Section 4.5 presents the test results of the systems. Finally, discussions are made in the last sections of this Chapter.

4.2 Implementation

The following commands are used to run the Floodlight controller. During these processes, it loads the different modules that are indicated on the dashboard in Figure 4.1.

sudo a –c 0-3 java -jar target/Floodlight.jar

The Floodlight controller as Figure 4.1 shows consists of a GUI that can be opened in any browser with the controller address after the controller runs with the command written above. It shows the network topology that is created by Mininet, switches, and hosts of the specified network.

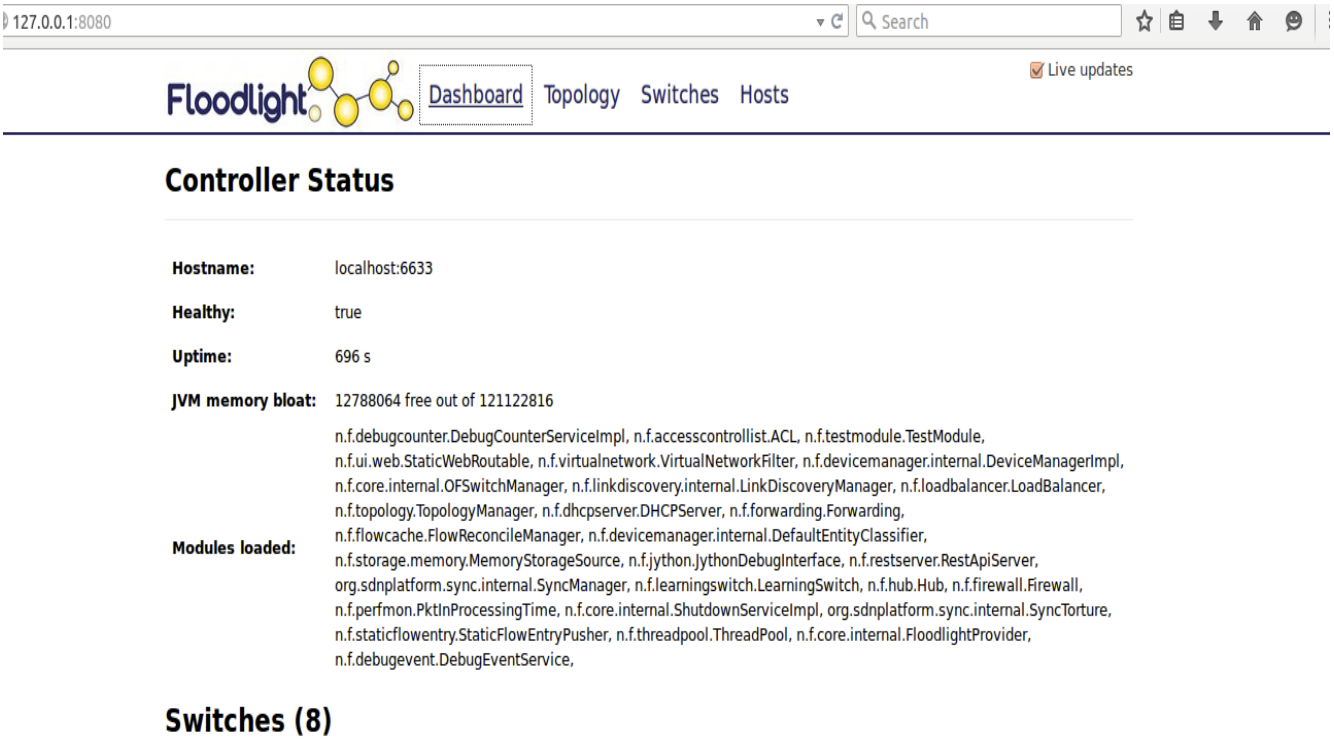

Figure 4.1: Floodlight controller dashboard

4.3 Experimental Setup

The test system comprises of

- Processor: Intel i7-6500U CPU 2.5GHz
- RAM: 8 Gb
- Operating system: Linux Ubuntu 14.04, x64 based
- The OpenFlow emulation software-Mininet that will be used to create the test network, covered in the Section 3.3.2.

The switch that is used in this testbed is open vSwitch version 2.11.9 with the following parameters

- Port:-virtual ports (48 ports)
- Bandwidth: 100 Mb
- Rate: 10 Mbps

The Floodlight controller is a software that controls the devices of the network. Therefore, the controller does not have performance parameters like hardware, because it is a type of network operating system and depends upon the underlying hardware. The controller solely purpose is to manage the network and network devices using the OpenFlow protocol.

The testbed that is used to evaluate the performance of the system composed of a controller and "N" number of switches that are connected directly to the controller as shown in Figure 4.3. Each of switches has 48 hosts (can emulate any number of virtual hosts) devices that are connected to the switch. These switches and host devices are emulated with Cbench tool. The detailed explanation of the Cbench tool is included in Section 3.3.3.

The network was created by Mininet using the following command as shown in Figure 4.2

Sudo mn –topo linear,8 –controller=remote,ip=127.0.0.1,port=6633 –switch ovsk, protocol = openflow13

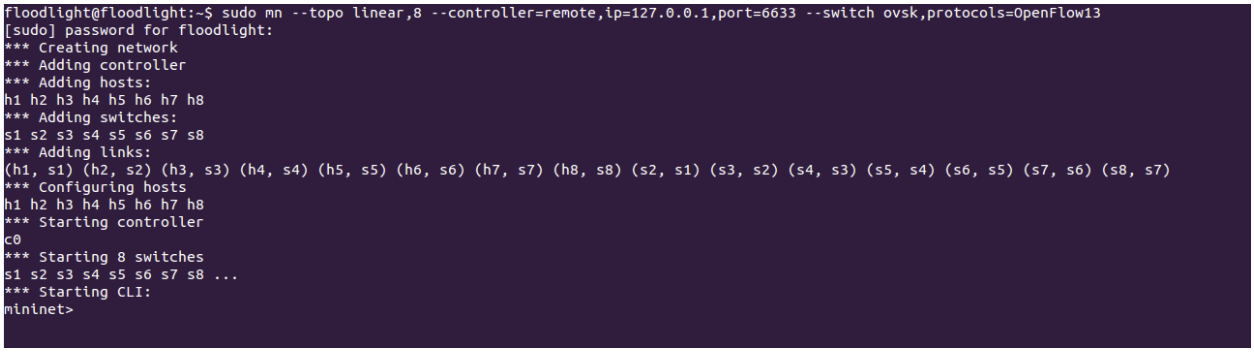

Figure 4.2: Mininet network emulation

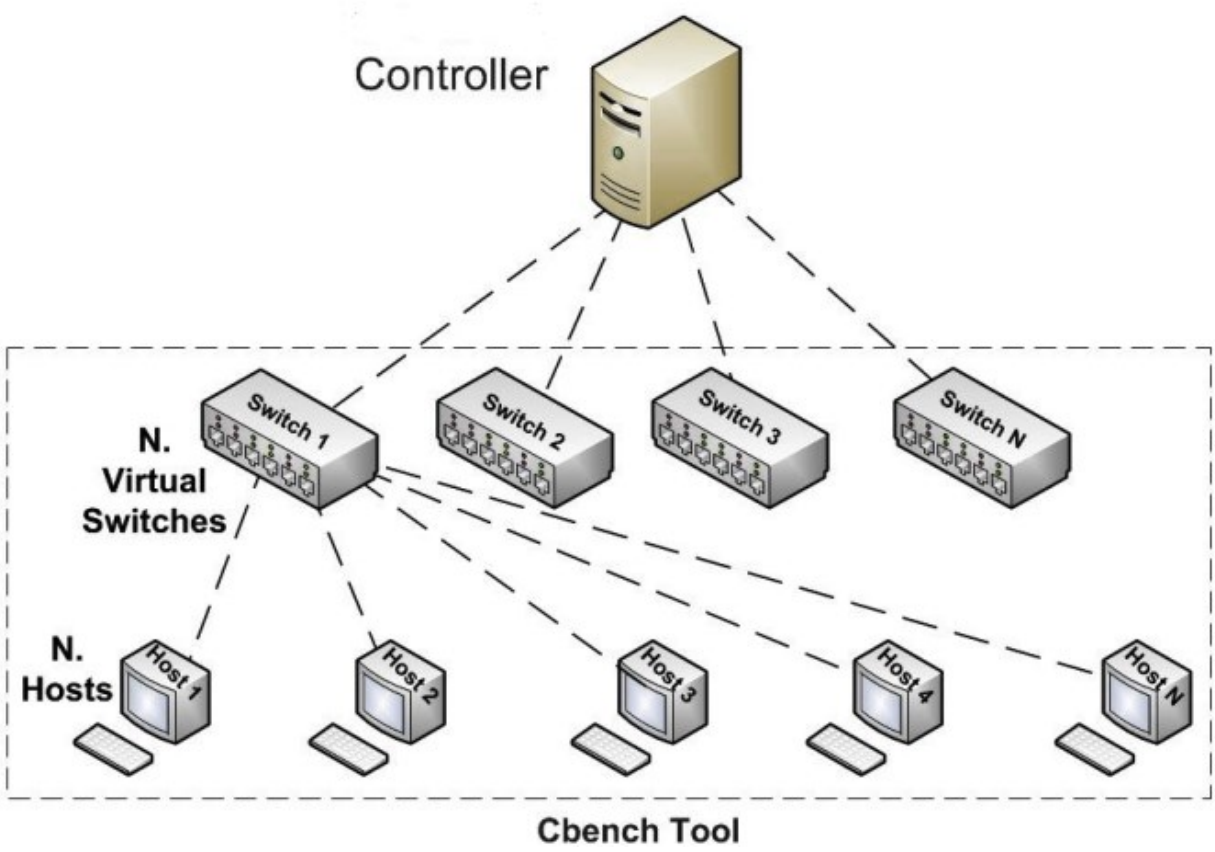

Figure 4.3: Experimental setup

The commands that are presented above are used to create the network at some remote controller. Let us discuss each of the terms on by one. Sudo mn command gives the administrator privilege for the users on Mininet and instructs Mininet to create the specified network on the controller's address that is specified. Topo linear command specifies the topology of the network in this case linear and creates a network of 8 switches with it. Moreover, the controller's, IP, and port specify the type of controller to be connected for the network, the IP address and the port at which the controller is located. The controller as specified 'remote', indicates that the default controller that is inside Mininet is not used but, the control that is located remotely outside the Mininet at the specified address will be used for the network management purpose.

The traffic that was used for testing the network is generated by the controller which is default traffic [55]. The following are the traffic that is generated by the controller to test the network.

1. ARP (Adrees Resolution Protocol):- It is a protocol used by IPv4 to map IP network address to hardware address. In this address resolution mechanism, ARP messages are communicated between the sender and receiver.
2. UDP (User Datagram Protocol):- It is a connectionless protocol used primarily for establishing low-latency and loss-tolerating connections between applications on the internet.
3. TCP (Transmission Control Protocol):- It is a connection-oriented protocol for exchanging messages and it is also the error-free transmission of data and it supports the retransmission of dropped packet and the acknowledgment of all received packets.

Three different types of traffic are used for testing the network. ARP used for mapping the address, UDP for sending any number of traffic without waiting for the response from the receiver, and TCP used for adding the reliability to the sent data. This variety of traffic helps for testing the network under different workload nature.

We have used a ping command to send a packet in a host-host connection in the network setup. As illustrated in Figure 4.4, the first host sends a request for a MAC address and the switch sends the PacketIn messages to the controller. Then the controller responds with the packetOut message instructing the switch to forward packetOut messages to all ports. When the controller receives the reply it sends a message for the switch to install new flows.

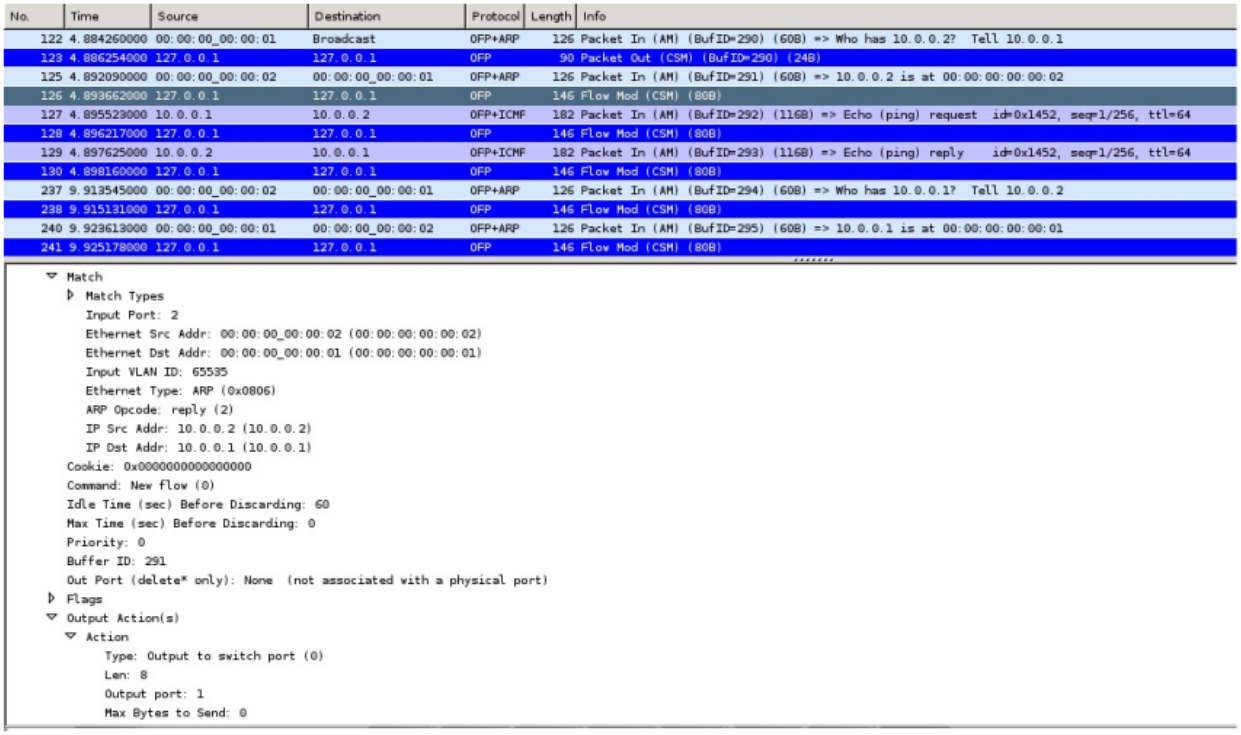

Figure 4.4: Wireshark capture of the packet exchange between two hosts

The test runs on the same device with the learning-switch[3] application and the Cbench evaluation tool as described in section 4.4, to evaluate the throughput and latency. The Cbench tool emulates 64 switches connected directly to the controller. Each switch has 48 connected devices and consequently, these devices are connected directly to the controller.

There has been a test performed, on increasing the number of switches available (64 Switches) , in order to send the maximum number of Messages PacketIn to evaluate the controller's performance.

[3] Learning-Switch tries to learn host locations and install flows to decrease the overhead, but a hub floods a packet out for each packet-in

4.4 Floodlight Controller Evaluation Benchmarking Methodology

To evaluate the performance, two types of tests are conducted on the Floodlight controller; they are flow setup throughput and latency. To measure the performance of the controller these two mechanisms are used.

4.4.1 Cbench Algorithm

To perform throughput and latency tests on the controller Cbench uses the following algorithm [53].

Algorithm 4.1: Cbench Algorithm

```
Pretend to be n switches (n=16 is default)
Create n OpenFlow sessions to the controller
if latency mode (default):
    for each session:
        1) send up a packet in
        2) wait for a matching flow mod to come back
        3) repeat
        4) count how many times #1-3 happen per sec
else in throughput mode (i.e., with '-t'):
    for each session:
        while buffer not full:
            queue packet_in's
            count flow_mod's as they come back
```

4.4.2 Throughput

In the throughput mode, the controller will be made to handle a large amount of traffic in which the switch continuously sends the PacketIn messages to the controller within a period of time and watches for the total number of flows installed per seconds on that switch.

4.4.3 Latency

Latency measures the ability of OpenFlow controllers to measure the speed of incoming packetIn's by intention to quickly reply to switches. In latency mode, the switches generate a packetIn message and wait for the response from the controller before sending the next message. This procedure will be done as quickly as possible. Then the total number of responses per millisecond was then counted and calculated to measure the average per packet latency the controller took to process.

4.4.4 Floodlight Controller Benchmarking Parameters

For evaluating the performance of the controller, the following parameters were used for testing both latency and throughput:

- The number of OpenFlow-enabled switches: the number of OpenFlow switch that will establish a connection with the controller.
- Number of hosts: number of hosts that will connect to switch.
- OpenFlow Version: OpenFlow protocol version that controller will use for connection setup, OpenFlow protocols such as 1.1, 1.2, 1.3, and 1.4
- Test loops: the number of frequencies the test needs to be repeated.
- Test Duration: the duration of test iteration, articulated in seconds.

As Figure 4.5 and 4.6 depicts, fourteen test loops were conducted, from them, the first four were the warmup test that are ignored in the average test calculation. Then, the average of these ten tests appeared in the overall test result. Moreover, for measuring throughput some delay is added to wait for some handshake processes from the controller and the switch before starting the tests. So, in this evaluation, a delay of 60 milliseconds is used.

```
floodlight@floodlight:~$ taskset -c 0-3 cbench -c localhost -p 6633 -m 1000 -l 14 -w 4 -M 1000  -s 8 -t -D 60
cbench: controller benchmarking tool
   running in mode 'throughput'
   connecting to controller at localhost:6633
   faking 8 switches offset 1 :: 14 tests each; 1000 ms per test
   with 1000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 60 ms delay after features_reply
   ignoring first 4 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
```

Figure 4.5: Cbench running in throughput mode

```
floodlight@floodlight:~$ taskset -c 0-3 cbench -c localhost -p 6633 -m 1000 -l 14 -w 4 -M 1000  -s 8  -D 60
cbench: controller benchmarking tool
   running in mode 'latency'
   connecting to controller at localhost:6633
   faking 8 switches offset 1 :: 14 tests each; 1000 ms per test
   with 1000 unique source MACs per switch
   learning destination mac addresses before the test
   starting test with 60 ms delay after features_reply
   ignoring first 4 "warmup" and last 0 "cooldown" loops
   connection delay of 0ms per 1 switch(es)
   debugging info is off
```

Figure 4.6: Cbench running in latency mode

4.5 Test Results

Throughput and latency are the two main parameters that were used to evaluate the performance of a system. We created the network using Mininet based on learning-switch application. The evaluation was done for both the enhanced version and the normal Floodlight controllers separately. On each evaluation, the parameters are tested on the increasing number of switches by making the number of host's constant.

4.5.1 Throughput

The first parameter to evaluate performance is throughput. Throughput measures the average number of flows that are installed in each switch.

The throughput or network throughput is the number of transactions per second that an application can handle. There is a different form of measuring this parameter, one would be using bits per second (bit/s or bps) and another way to measure the throughput performance would be through data packets per second. So in these evaluation data packets per second were used for evaluating the throughput of the controller.

The results of flow rate measurement for the Floodlight controller as represented by Figure 4.7, clearly demonstrate that the connection of one switch results in a maximum of 57950 flows/sec (flow rate) with WAPB and 52349 flows/sec with normal Floodlight evaluation. The throughput was dropped after the 8 switch connections.

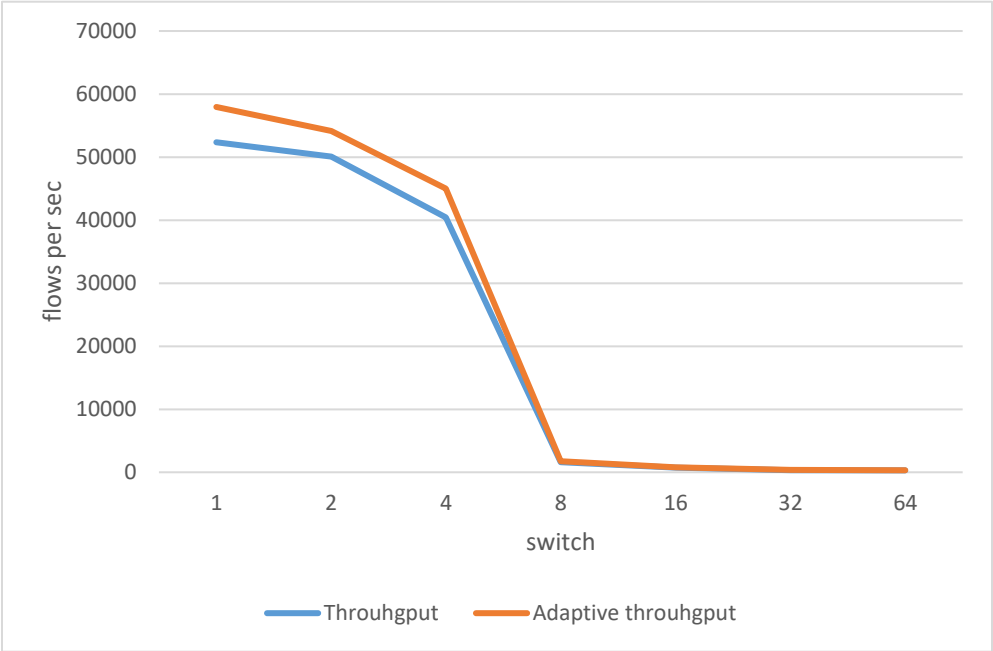

Figure 4.7: Throughput performance test

4.5.2 Latency

The latency results represent the average number of milliseconds that a flow consumes to be installed on each switch.

Latency in a packet-switched network is the time required for a packet to arrive at its destination through the network. The physical environment or devices forming the network (Switches, routers, etc…) can cause the delay. There are two ways to measure the latency: the first is to measure the time it takes for a packet from source to destination (One-Way) and the second is to measure the time it takes a packet to go and return (Round-Trip). This last form of measuring the latency, Round-Trip, is the most used because it allows a single device to measure the latency of a network which is used in this paper.

The results of latency measurement for the Floodlight controller, with the connection of one switch as depicted by Figure 4.8, resulted in a maximum of 18 flows per milliseconds for WAPB and 21 flows per milliseconds for normal Floodlight controller. The results also dropped after the connection of eight switches to the controller for both cases.

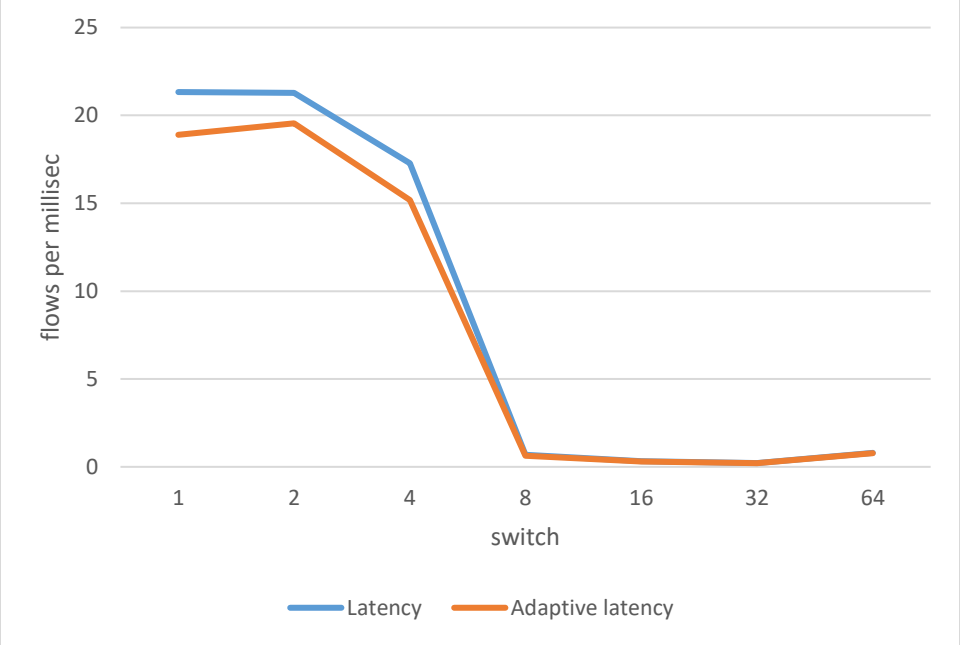

Figure 4.8: Latency performance test

4.6 Discussions

We have proposed and implemented the Workload Adaptive Packet Batching technique of high packet switching architecture to Floodlight SDN controller. The test results were presented in the last section with throughput and latency results. In this section, the test results were discussed with the comparison between the modified version, Workload Adaptive Packet Batching, and the normal Floodlight controller.

As depicted in Figure 4.7, the throughput result for both the Adaptive and normal Floodlight controllers had a maximum flow rate with the connection of one switch. This implies that the tradeoff for installing flows in that single switch from the controller is low. So that the controller

could install many numbers of flows relative to increasing number of switches. As the number of switches connected to controller increase, the throughput drops slightly until eight switches. After that, the throughput was dropped severely. This is due to the increase in the number of traffic request that is above the batch size that the controller could handle.

The latency result, as Figure 4.8 illustrates, the controller has populated a maximum number of flows per milliseconds with the connection of one switch to the controller. The flows that were populated within a milliseconds time is high because there are no other devices request for a task to process so that, this one switch resulted in the high flow installation. As the number of the switch connected to the controller's increase, the latency of the system dropped abruptly. This is due to a high number of transactions that were processed, making the milliseconds time to be sliced to the different request from the different number of switches. These behaviors that are discussed for throughput and latency is seen in both the normal and the Workload Adaptive Packet Batching version of the Floodlight controllers.

The modified version, WAPB based Floodlight controller as seen in Figure 4.7 compared to the normal Floodlight has average throughput increment of 11%. And as depicted in Figure 4.8, the latency result of the WABP, based Floodlight controller has an average latency decrement of 10 % than the normal controller. This result is because of the implementation of the Adaptive packet batching mechanism, which adapts the workload nature of the system and calculates the batch size that could make the system process stable.

The Workload Adaptive Packet Batching methodology has a great benefit for the Floodlight controller as well as SDN based networking companies and cloud-computing providers. It helps them to optimize their productivity by improving system performance. Increasing throughput and reducing latency helps them to reduce different costs. Additionally, helps them to work easily with tiresome network management tasks, related to configuring the different devices to the network with the OpenFlow protocol. Therefore, this Workload Adaptive Packet Batching implementation to Floodlight SDN controller facilitates the network management. Moreover, it is very easy to deploy new services with this new Floodlight controller. In addition, cloud-based storage and severer can respond quickly to the request from the users with the implementation of Workload Adaptive Packet Batching to SDN controller.

In general, as the network's size increases, the biggest is the number of data plane requests to be satisfied. This means that in a large network, the time to process all these requests at once increase, which leads to a decreased throughput. To manage this increasing traffic, adaptive algorithms play a great role in boosting the performance of the SDN networks.

A research question was presented in the first Chapter. According to the experiments, our contributions with regard to this question was summarized below

- Can the implementation of Workload Adaptive Packet Batching enhance the performance of the Floodlight SDN controller?

The implementation of Workload Adaptive Packet Batching into Floodlight SDN controller resulted in enhancing the performance of the controller. As figure 4.7 and 4.8 illustrates, both the latency and the throughput results for the same testbeds have shown a remarkable improvement.

Workload Adaptive Packet Batching has an impact on SDN networks performance optimization. This approach could be used with other techniques to optimize the performance of other closed or open-source SDN controllers.

The implementation of Workload Adaptive Packet Batching has an impact on throughput and latency. It first adjusts itself to the workload nature of the system and later updates the workload to the capacity of queue size of the switch. Therefore, the size of the batch increases as many queues being congested at the OpenFlow switches.

The latency of Workload Adaptive Packet Batching based Floodlight SDN controller was decreased because the wait time for a packet to get access for the resources is minimized. The delay was reduced due to batch sizes that learns the system nature and grows with it.

The throughput of Workload Adaptive Packet Batching based Floodlight SDN controller improved, due to the rise in the ability of the system to process any requests at a time. Since there is no many queues saturation, the Floodlight controller process job requests immediately as much as it can. Throughput maximization was employed as the result of batch size adaptability.

Chapter 5. Conclusion and Recommendations

5.1 Conclusion

In this thesis, we have proposed and implemented the Workload Adaptive Packet Batching for Floodlight SDN controller; the study of different SDN technologies specifically, Floodlight controllers design features and OpenFlow protocol. Cbench tool was used for evaluating the performance of the system. Literature reveals that the current Floodlight controller was built with fixed packet batching that limits the system's ability with the mechanism, which could not process the workloads at hand with fixed batch size. The Workload Adaptive Packet Batching was used, in order to optimize the performance of the Floodlight controller.

The proposed adaptive technique was implemented on the Floodlight SDN controller; that is the main part on which different network control applications are deployed. The controller lies between the infrastructure layer, that consists of the different network devices and the application layer. Therefore, the controller is responsible for managing the resource of the whole network using the OpenFlow protocol.

The Controller administers the underlying networking device such as Open Vswitch by updating the flow entry table that holds a different set of information about the packets. When a job was assigned to switch, the controller will handle according to the Workload Adaptive Packet Batching methodology. The jobs will be processed by adaptively matching the batch size to the workload. Therefore, the tasks waiting for a switch resource will be processed immediately as much as possible, depending on the batch size.

The designed Workload Adaptive Packet Batching methodology for Floodlight controller was implemented by building a modified version of it as described above. After that, the performance of both normal and WAPB based Floodlight controller was evaluated on the same device and with the same metrics. The SDN network was created on the learning-switch application, to evaluate their performance. Then, separately running of the different versions of the Floodlight controller and testing was done using the Cbench tool. In Cbench two metrics, namely throughput and latency were tested. Throughput measures the total number of flows that the switch installs within a seconds interval. While latency measures the time, it takes the flow to be installed on the switch.

As illustrated in Figure 4.7, we conclude from the results of the Floodlight OpenFlow controller's evaluation that, the Workload Adaptive Packet Batching Floodlight shows the highest throughput, achieving 57950 flows per second. This result obtained when only one switch was connected to the controller. The throughput result drops severely upon the increasing number of switch connections to the controller. Moreover, throughput result for Normal Floodlight controller varies only by the number of flows setup. The connection of one switch with the normal controller yields the throughput of 52349 flows/sec The Workload Adaptive Packet Batching Floodlight controller has an average of 11% throughput enhancement. The overall throughput result for both versions of the Floodlight controller was dropped after the one switch connection due to the buffer congestion, which could not able to increase the batch size after many switch connection. This indicates that the Floodlight controller is not such much scalable and the TCAM storage size that holds the number of flows setup is small which limits the system throughput results.

Latency tests for the controller was performed with different numbers of connected switches to see how these controllers' response time could be affected as more workload was added, see Figure 4.7. The tests showed that WAPB Floodlight has the highest latency of 18 flows per millisecond and 21 flows per milliseconds for Normal Floodlight controller. The latency result for both of the controller drops for increasing connection of switches to the controller. Generally, the WAPB Floodlight has an average of 10% latency improvement than the normal Floodlight controller.

The implementation of Workload Adaptive Packet Batching to Floodlight controller results in an average enhancement of 11% throughput and 10% latency. The throughput was improved and the latency was reduced with this proposed mechanism.

In our test, the communications between the switches and the controller are done through the loopback interface where, where there is no delay while packets traversing the virtual ports as compared to the hardware ports. Therefore, the latency results are measured higher in real hardware implementation due to virtual ports compared to it.

5.2 Recommendations

We have designed and implemented the WAPB for enhancing the performance of the Floodlight SDN controller. Moreover, with this methodology, we have optimized the performance of the

controller. However, we consider that incorporation of the following ideas, as a future work would result in a better result:

- We have evaluated and compared the performance of the Floodlight controller with the optimized one. Furthermore, comparing the Floodlight with other controllers and including the features of the other controller in addition to adaptive batching techniques would improve the performance and overall system stability.
- Scalability benchmarking test in addition to performance test will determine the maximum number of OpenFlow switch connections to the controller without affecting the performance of the system.
- Designing a software or hardware based mechanisms to use the TCAM effectively, flow table storage space in order to store as much as many numbers of flows in order to satisfy the growing traffic demand.

References

[1] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, 1st edition. Reading, Addison-Wesley Professional, 2015. [E-book] Available: Safari e-book.

[2] B. Anwer, M. Motiwala, M. bin Tariq, and N. Feamster. "SwitchBlade: A Platform for Rapid Deployment of Network," In *ACM SIGCOMM*, 2010.

[3] S. Han, K. Jang, K. Park, and S. Moon. "PacketShader: a GPU-accelerated software router," In *Proc. ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[4] N. Feamster, J.Rexford, E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," 2013.

[5] A. Lara, A. Kolasani and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2013.

[6] M. Betts,et.al. , "SDN architecture," *Open Networking Foundation*, Issue no.1, pp.13-33, June 2014.

[7] Open Networking Foundation. Leveraging Disaggregation to Build Innovative Open Source Solutions for Operator Networks. [Online]. Available: https://www.opennetworking.org. [Accessed: Feb. 13, 2018].

[8] Open Networking Foundation, SDN definition. [Online]. Available: https://www.opennetworking.org/sdn-definition/. [Accessed: Feb. 13, 2018].

[9] .Infinite technology consulting (2016, Aug.5). Software Defined Networking [Online]. Available: http://www.itc1.net/software-defined-networking-sdn/. [Accessed: Feb. 19, 2018].

[10] M. Casado, N. McKeown, "The Virtual Network System," In *Proceedings SIGCSE*, 2005.

[11] S.M. Kerner (2013, Apr 29). Enterprise Networking Planet: OpenFlow SDN Inventor Martin Casado on SDN, VMware, and Software Defined Networking Hype [online]. Available: http://www.enterprisenetworkingplanet.com/netsp/openflow-inventor-martin-

casado-sdn-vmware-software-defined-networking-video.html. [Accessed: March 1, 2018].

[12] N. McKeown et al, “OpenFlow: Enabling innovation in campus networks,” In ACM SIGCOMM - Computer Communication *Review*, vol. 38, no. 2, p. 69–74.

[13] Open Networking Foundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, Apr. 2012.

[14] N. Sharma (2015, January 20). Eight Big Benefits of Software-Defined Networking. [Online]. Available: https://www.serverwatch.com/server-tutorials/eight-big-benefits-of-software-defined-networking.html. [Accessed: Jun. 10, 2018].

[15] W.Braun and M.Menth, “Software-Defined Networking Using OpenFlow: Protocols, Applications, and Architectural Design Choices,” pp 5-7, May 2014.

[16] Smith et al, M. (2014). OpFlex control protocol, Internet Engineering Task Force. [Online].Available: http://tools.ietf.org/html/draft-smith-opflex-00. [Accessed: Jun. 15, 2018].

[17] Enns, R., Bjorklund, M., Schoenwaelder, J., Bierman, A. (2011). Network configuration protocol (NETCONF). *Internet Engineering Task Force*. [Online] Available: http://www.ietf.org/rfc/rfc6241.txt. [Accessed: Jun 15, 2018].

[18] Doria et al, A. (2010). Forwarding and control element separation (ForCES) protocol specification. *Internet Engineering Task Force*. [Online] Available: http://www.ietf.org/r/fc/rfc5810.txt. [Accessed: Jun 15, 2018].

[19] Song, H., “Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane,” in *Proceedings of ACM SIGCOMM Workshop Hot Topics Software Defined Networking* II. p. 127–132, 2013.

[20] A. Jin (2016, Feb.2). OpenFlow switch with hardware flow table [online]. Available: http://learn.linksprite.com/project/openflow-switch-with-hardware-flow-table/. [Accessed: Feb. 20, 2018].

[21] D. Erickson, "The beacon OpenFlow controller," In *Proceedings of the second ACM SIGCOMM Workshop on Hot Topics in Software-Defined Networking*, pp. 13-18, 2013.

[22] Project Floodlight (2012). Floodlight. [Online].Available: http://Floodlight.openflowhub.org/. [Accessed: Jan. 9,2018].

[23] S. Rao (2015,Jan 6). SDN Series Part Five: Floodlight, an OpenFlow Controller [Online]. Available: https://thenewstack.io/sdn-series-part-v-Floodlight/. [Accessed: May 28, 2018].

[24] A. Metzler et al. Ten Things to Look for in an SDN Controller [Online]. Available: https://www.necam.com/docs. [Accessed: Sept. 5, 2018].

[25] D. Ingram, *Design-Build - Run: Applied Practices and Principles for Production-Ready Software Development*, 1st edition. Reading, Wiley Publishing, Inc., 2009. Available: Safari e-book.

[26] I.Elhanany and M. Hamdi, *High-performance Packet Switching Architectures*, A. Doyle and K.Brown, Germany, 2007.

[27] S. A. Shah, "An Architectural Evaluation of SDN Controllers," In *2013 IEEE International Conference on Communications,* 2013.

[28] Nox homepage. Nox. [Online]. Available: http://noxrepo.org/. [Accessed: Nov. 3, 2018].

[29] Beacon homepage. Beacon. [Online]. Available: https://OpenFlow.stanford.edu/display/Beacon/Home. [Accessed: Nov. 3, 2018].

[30] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable OpenFlow controller," Rice University, Tech. Rep., 2010.

[31] Floodlight homepage. Floodlight. [Online]. Available: http://Floodlight.OpenFlowhub.org/. [Accessed: Nov. 3, 2018].

[32] I.T Akhthar, "Tenant-Aware BigData Scheduling with Software-Defined Networking," MSc Thesis, Dept Info. Sys. and Comp. Eng. , Tecnico, Lisboa,2016.

[33] A. L Aliyu, and. Bull, and A. Abdallah, "Performance Implication and Analysis of the OpenFlow SDN Protocol," in *31st International Conference on Advanced Information Networking and Applications Workshops (WAINA),*2017.

[34] R. Wallner and R. Cannistra, “An SDN Approach: Quality of Service using Big Switch’s Floodlight Open-source Controller,” in *Proceedings of the Asia-Pacific Advanced Network* v. 35, p. 14-19,2013.

[35] K. Ross, “Dynamic Scheduling in Queueing Systems with Applications to Communication Networks, ” Ph.D. dissertation, Dept. Manag. Sci and Eng., Stanford, 2004.

[36] T. Das and et al., ‘Adaptive Stream Processing using Dynamic Batch Sizing’, California, UCB/EECS-2014-133, 2014.

[37] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou, ”Comet: batched stream processing for data-intensive distributed computing,” In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 63–74. ACM, 2010.

[38] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, “Discretized streams: Fault-tolerant streaming computation at scale,” In *Proceedings of the 24th ACM Symposium on Operating Systems Principles. ACM*, 2013.

[39] D. Logothetis, C. Olston, B. Reed, K. C. Webb, and K. Yocum, ”Stateful bulk processing for incremental analytics,” SoCC, 2010. [Online ISBN 978-1-4503-0036-0]. Available: http://doi.acm.org/10.1145/1807128.1807138. [Accessed: Sep. 19, 2018].

[40] D. Peng and F. Dabek, “Large-scale incremental processing using distributed transactions and notifications,” In *OSDI*, 2010.

[41] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, “ Incoop: MapReduce for incremental computations,” In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 7. ACM, 2011.

[42] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al, “The design of the Borealis stream processing engine,” In CIDR, volume 5, pages 277–289, 2005.

[43] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, “Timestream: Reliable stream computation in the cloud,” In *EuroSys* vol. 13, 2013.

[44] D. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad, “A timely dataflow system,” In *SOSP*, vol. 3, 2013.

[45] Anonymous. Storm. [Online].Available: https://github.com/nathanmarz/storm/wiki. [Accessed: Oct 28, 2018].

[46] J. Hunter and B. Nachtergaele , *Applied analysis*, July, 2005.

[47] Anonymous. Floodlight [Online].Available: http://www.projectFloodlight.org/Floodlight/. [Accessed: March. 24, 2018].

[48] R. Sherwood (2014, Jul.18). Big Switch Networks. [Online] Available: http://www.bigswitch.com. [Accessed: March. 24, 2018].

[49] Q. Wang .Floodlight installation Guideline [Online] Available: https://Floodlight.atlassian.net/wiki/spaces/Floodlightcontroller/pages/1343544/Installation+Guide. [Accessed: March. 25, 2018].

[50] Anonymous, Mininet [Online].Available: http://mininet.org/. [Accessed: Nov.9, 2018].

[51] Anonymous, Get started with Mininet [Online].Available: http://mininet.org/download/. [Accessed: Nov.9, 2018].

[52] Anonymous. Cbench New [Online]. Available: https://Floodlight.atlassian.net/wiki/spaces/Floodlightcontroller/pages/1343657/Cbench+New. [Accessed: Oct. 24, 2018].

[53] B. Heller. Cbench [Online]. Available: https://github.com/mininet/oflops/tree/master/cbench. [Accessed: Jan. 5, 2019].

[54] A. Shalimov and et al, Advanced Study of SDN/OpenFlow controllers, Jan, 2014

[55] R. Izard. How to Process a Packet-In Message [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller. [Accessed: June. 10, 2019].