جمهورية العراق- وزارة التعليم
العالي والبحث العلمي
جامعة بابل
كلية تكنولوجيا المعلومات
قسم البرامجيات

نهج تخصيص الموارد المحسن للشبكات المعرفة برمجيا

اطروحة مقدمة الى مجلس كلية تكنولوجيا المعلومات للدراسات العليا بجامعة بابل كجزء من متطلبات درجة الدكتوراة فلسفة في تكنولوجيا المعلومات/ برامجيات

من قبل

حيدر منذر نعمان مدحت

باشراف

ا.م.د مهدي نصيف جاسم علوان

2021 A.D 1443 A.H

Republic of Iraq
Ministry of Higher Education and Scientific Research
University of Babylon
College of Information Technology
Department of Software

An Improved Resource Allocation Approach for Software Defined Networks

A Dissertation

Submitted to the Council of the College of Information Technology for Postgraduate Studies of University of Babylon in Partial Fulfillment for the Degree of Doctorate of Philosophy in Information Technology/ Software

By

Haeeder Munther Noman Madhat

Supervised By

Assist Prof. Dr. Mahdi Nsaif Jassim Alwan

2021 A.D **1443 A.H**

قَالُواْ سُبْحَانَكَ لاَ عِلْمَ لَنَا إِلاَّ مَا
عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ
الْحَكِيمُ

صدق الله العظيم

آية ٣٢ سورة البقرة

Declaration

I hereby declare that this dissertation, submitted to University of Babylon in partial fulfilment of requirements for the degree of Doctorate of philosophy in Information Technology \ Software, has not been submitted as an exercise for a similar degree at any other University. I also certify that this work described here is entirely my own except for experts and summaries whose source are appropriately cited in the references.

Signature:

Name: Haeeder Munther Noman

Date: / /2021

Supervisor Certification

I certify that the dissertation entitled "An Improved Resource Allocation Approach for Software Defined Networks**"** was prepared under my supervision at the Department of Software/ College of Information Technology/ University of Babylon, as partial fulfillment of the requirements of the degree of Doctor of Philosophy in Information Technology / Software.

Signature:
Name. Dr. Mahdi Nsaif Jasim
Title: Assist. Professor
Date: / /2021

The Head of the Department Certification

In view of the available recommendations, I forward the dissertation entitled "An Improved Resource Allocation Approach for Software Defined Networks**"** for debate by the examination committee.

Signature:
Name. Dr. Ahmed Saleem Abbas
Title: Head of Department of Software/ College of Information Technology / University of Babylon Title: Assist. Professor
Date: / /2021

Abstract

The emergence of software defined networking (SDN) offered the opportunity for the managers to own a traffic management technology that included the advantage of low cost, flexible form of operation, and optimal resources utilization. SDN offered an inexpensive, scalable and promising solution for the traditional load balancer limitations through enabling programmability and centralized control. The major challenge for existing load balancing algorithms is the lack of ability to enable traffic engineering (TE) to aggregate and generate bandwidth utilization statistics for traffic analysis and monitoring, which is a further challenging mission if the desired monitoring is fine grained. The conducted work aims to produce a model for enhancing the utilization of server resources by developing a new load balancing algorithms like the adaptive least load ratio (ALLR) algorithm and Hybrid-based (HB) algorithm. The proposed load balancing algorithms model have been tested using different types of network benchmarking tools such as httperf and OpenLoad. Results reveal that The ALLR algorithm improves server average throughput up to 7.25%, server connection time up to 16%, and server connection rate up to 1.2%. Moreover, The ALLR algorithm reduces the server reply time, server CPU utilization, and server average queue length to 19%, 2.5%, 14% respectively as compared with other load balancing algorithms such least connection-based (LCB) least bandwidth-based (LBB). The HB algorithm enhances the load balancing and request handling by increasing the server transactions per second to 22%, and CPU capacity to 5% and reducing server average response time to 17% as compared with weighted round-robin (WRR) and least connection-based (LCB).

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

Abbreviation	Description
SDN	Software defined network
ALLR	Adaptive least load ratio load balancing algorithm
ALLRComputeM	ALLRCompute module of ALLR Algorithm
AMPQ	Advanced message queuing protocol
API	Application programming interface
ASMM	Advanced server monitoring module of HB Algorithm
Cbench	POX controller average throughput and Latency testing tool
CFS	Complete fair scheduling
CLI	Command line interface
DCN	Data centre network
DISCO	Distributed mutli-domain SDN controller
D – ITG	Distributed internet traffic generator tool
DUET	Cloud based load balancing with hardware and software
FTP	File transfer protocol
FV	Hypervisor/proxy between a switch and controller

HTTP	Hyper test transfer protocol
HB	Hybrid load balancing Algorithm
HLBM	Hybrid-based load balancing module of HB Algorithm
httperf	HTTP Server web-performance testing tool
Iperf	Hosts Maximum achievable bandwidth testing tool
IPv4	Internet protocol version 4
ISP	Internet service provider
LLDP	Link layer discovery protocol
MAC	Media access control address
NBI	Northbound interface
NP	Network performance
NVP	Network virtualization platform
ONIX	Distributed control platform for large scale production
ONOS	Open network operating system
OpenLoad	HTTP Server web- application testing tool
OVX	Network hypervisor that can act create multiple virtual programmable networks on the top of single physical infrastructure
POX	SDN-Based platform controller

QoS	Quality of service
SBI	Southbound interface
SSL	Secure sockets layer
SMM	Server monitoring module of ALLR Algorithm
TCP	Transport control protocol
TLS	Transport layer security protocol
VIP	Virtual IP address of the load balancer

List of Symbols

Symbol	Description
b_i	Total amount of data
$B(S_i)$	Bandwidth utilized by every server member in a pool
BT	Business time
CPI	Average cycle per instruction
I	Number of instructions
$L(S_i)$	Load ratio of every server member in a pool
N_{PR}	Cloud based load balancing with hardware and software
N_{PS}	Distributed mutli-domain SDN controller
OT	Observation time
R_N	Number of request
S_m	Server member having least number of active connections
$S_{optimal}$	Optimal server member for handling the incoming request after fulfilling both conditions of hybrid load balancing module of HB load balancing algorithm
T_{fb}	Time of the first byte of response
t_i	Time taken for the destination to send the final packet

T_{lb}	Time of last byte of response
T_T	Thinking time per request
T	Clock cycle time
U_N	Number of concurrent users
$W(S_i)$	Static weight of the server

1.1 Introduction

The Software Defined Network (SDN) emerged as a new generation of network architecture to fix conventional network issues like protection, virtualization, and traffic management [1]. SDN separates network control from forwarding functions to allow the network to be programmable and abstractable from the underlying infrastructure [2]. The separation is a departure from the conventional architecture where complex tasks are abstracted from the forwarding tasks. However, a complex process is automated and handled separately in a centralized SDN controller or Network Operating System to offer the opportunity for managers to own a technology for traffic management [3]. In order to achieve a flexible form of operation and low cost, the controller uses the OpenFlow protocol to connect with the real physical or virtual switch, and the data path uses the flow entries inserted by the controller in the flow table for routing data [4]. The OpenFlow protocol is an open source standard protocol that provides a method for programming data-level switches in a network originally designed for network researchers to test their protocols on real network devices [5]. Moreover, the OpenFlow protocol is the regulator of all the communication between network devices and the controller in SDN, and was first introduced in the year 2009 by the ONF organization [6]. In traditional networks, each switch has its own program by which the switch knows how to handle incoming packets. In SDN after separating the data and control planes, the OpenFlow protocol allows the controller

(control plane) to instruct the switches (data plane) where to send the incoming packets. Therefore, the process of controlling the packets has become centralized so that the network switches can be programmed independently.

1.2 Related Works

In order to select the best server to handle incoming request, various load balancing algorithms has been implemented in [7]. Typically, these algorithms can be divided into two types namely; Static and Dynamic algorithms [8]. The algorithms are pre-configured and assigned by the administrator. The random load balancing algorithm was introduced by Li, J., Chang, X., Ren, Y., Zhang, Z., & Wang, G. [9]**.** The aim was to use a random number generator to select a particular server among the server members to handle client requests. The random algorithm recorded a weak performance as it led to overloading of a single server member while other server members were under-utilized, due to the fact that the Controller selects the server randomly every time a flow is set up, and the current load of each server is not taken under consideration. Round-robin load balancing algorithm was presented by[10]**.** The input traffic is divided in a round-robin (rotational) way. The biggest drawback of using the round robin algorithm was the assumption that servers were similar enough to handle equivalent loads. If certain servers have more CPU, RAM, or other specifications, the algorithm has no way to distribute more requests to these servers. Weighted round-robin was developed by J. S. Sabiya [11] to overcome the limitations of

the round-robin algorithm. The weighted round-robin algorithm identifies weights in proportion to actual server capacity. least delay dynamic weighted round-robin (LDDWRR) was presented by M. S. Sroya and V. Singh [12]. The LDDWRR algorithm relied on assigning various delays within each link based on different speeds. Therefore, the server with the highest speed link achieves the least delay and handles more requests [13]. The advantages and capabilities of machine learning were invested in routing mechanisms to balance the loads between server members by the algorithm proposed by D. Todorov, H. Valchanov, and V. Aleksieva [14]. The results revealed that the algorithm reduced the load and improved routing mechanisms. Based on the mean field game theory that did not require explicit communications led to the proposition of Wardrop load balancing algorithm by Cimorelli, F., et al. [15].The algorithm converges to an arbitrarily small neighborhood of a specific equilibrium among the loads of the providers. The findings approved that Wardrop algorithm is feasible for different SDN scenarios where service requests originate from network nodes and manage by SDN controllers. The performance of several heuristic Load balancing algorithms to raise the ability of SDN to resist the failure among controllers and nodes was evaluated by A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, and A. Rezaee [16]. This method aimed to heuristically search and allocate a minimum number of controllers for individual nodes by following proper node placement to maximize the reliability of controller placement and to achieve certain threshold reliability. The OpenFlow switch migration

from source to target controllers was handled by the self-adaptive load balancing algorithm (SALB). The SALB was proposed by M. Priyadarsini, et al. [17] to dynamically implement the load balancing task among multiple controllers. the concept of combining the hardware and software was achieved by DUET algorithm proposed by R. Gandhi, et al. [18] . The system utilizes the currently existing switches to construct a hardware-typed load balancer to improve the capacity competently and reduce the cost and delay. The routing control platform (RCP) load balancing scheme in SDN was highlighted by P. Vizarreta, et al. [19]**.** The RCP is used for the path computation for Traffic Engineering (TE) systems to alleviate or avoid congestion for operating networks by selecting one ’best’ path for the traffic. RCP is composed of three components: path computations based on a consistent network state, expressive defined routing policies, and regulated communications across the routing protocol layers. The employment of the virtual SDN-Based platform controller vSDN as VNF was adopted by S. Ejaz, et al. [20] to enable traffic sharing for the vSDN when the flow grows or become even**.** the dynamic weighted random selection (DWRS) algorithm was developed by M.L. Chiang, H.S. Cheng, H.Y. Liu, and C.Y. Chiang, [21]. The DWRS algorithm addressed the limitations in random algorithm by considering the server's load in real-time when the requests are dispatched to servers. The underutilized servers are assigned larger weights to give them a higher possibility of handling the incoming requests. the Load variance-based synchronization (LVS) load balancing

was introduced by Z. Guo, et al. [22]. The motivation behind the LVS algorithm was to enhance the execution mechanism of the load balancing process by minimizing synchronization and removing forwarding loops. The appropriateness of server-based load balancing algorithms for server-cluster virtual environment was revealed by Chen, W., Shang, Z., Tian, X., and Li, H., [23]. Initially, the Controller uses the simple network management protocol (SNMP) to collect server status information. Then, the Controller measures the services load in order to match the side-blotched lizard algorithm (SBLA) algorithm and the lightest load server is selected to respond for users. This approach was found to reduce the server's response time but is inappropriate for unstructured data centers and networks. An adaptive load balancing algorithm applicable on fat-tree topology data center network (ALB-DCN) was introduced by S.K. Askar [24]**.** The ALB-DCN algorithm takes action when the received throughput is below the threshold or there is an increase in loss in one or more DCN links. the performance of least connection-based (LCB) load balancing algorithm to distribute the load distribution among servers based on the actual number of connections was estimated by S. Raghul, T. Subashri, and K. R. Vimal [25]**.** The load balancer keeps a record of the number of requests served by every node. The node with the least number of requests served is primarily selected. However, in cases where nodes have similar specifications, a node may be overloaded due to longer lived connections. This limitation is addresses by modulating the weighted least connection-based (WLCB)

load balancing algorithm proposed by R. Abbasi, S. M. M. Gilani, A. Kabir, Q. Nawaz, and M. S. Riaz [26]. The administrator assigns a weight to each application server based on criteria of their choosing to demonstrate the application servers traffic-handling capability. The Loadmaster is making the load balancing criteria based on active connections and application server weighting. The resource adaptive load balancing algorithm based on installing an agent on the application server that reports its current load to the load balancer was settled by K. Govindarajan and V. S. Kumar [27] . The agent reports the current load status of the load balancer and the Controller and utilizes the reported real-time information to select the next server that best handles the request. The evaluation assessment of least bandwidth-based (LBB) load balancing algorithm was approved by M. K. Faraj, A. Al-Saadi, and R. J. Albahadili [28]. The server having the lowest network traffic consumption during the last (n) second is selected to respond to the subsequent request. The least bandwidth-based algorithm is implemented by analyzing traffic information to obtain every stream statistic on each OpenFlow switch. OpenFlow protocol enables the controller to request these statistics through a specific request (StatsRequest). The controller must periodically request the data to update the values on each server

Table (1.1): Literature Survey Categories

Category	Load balancing Approach
Static	Random [9]
	Round-robin [10]

	Weighted round-robin [11]
	LDDWRR [12,13]
Artificial intelligence	Machine learning [14]
Game theory	Wardrop [15]
Optimization	Heuristic [16]
Switch migration	SALB [17]
Hardware & software	DUET[18]
Routing	RCP [19]
Virtual Network Function	V-SDN[20]
Dynamic	DWRS [21]
	LVS [22]
	SNMP,SBLA [23]
	ALB-DCN [24]
	LCB [25]
	WLCB [26]
	Agent [27]
	LBB [28]

1.3 ***Problem Statement***

Many approaches have been adopted to deal with the load balancing task in the SDN-based platform. But the majority of these approaches didn't highlight enough the issue related to the timing factor of the counters statistics of the OVS OpenFlow switch. The OpenFlow switch provides raw byte counters without providing the times of these counters, which may affect the load balancing functionality to perform

incorrectly and send falsified connection requests to different devices that are overloaded.

1.4 ***Motivation***

The quick development of Internet technology makes the server cluster of large Internet service providers (ISP's) more challenging. With the growth of users as well as network bandwidth, servers need to handle a large number of access requests during a short period of time. If the server cannot timely process user access request which leads to the extension of user's waiting time, then it will greatly reduce the Quality of Service (QoS), circumstances like these make servers become the new bottleneck in network and force researchers to start the study of how to improve servers performance.to achieve that enterprises adopt a number of measures, such as improvement of CPU's processing speed, increment of server's cache capacity, deployment of high speed disk array, as well as the construction of the server cluster[29].Simply upgrading the hardware system will not only cause the idle criteria of existing resources, but as the business continue to expand, companies will face the same challenging situation [30]. Through the establishment of the server cluster, companies can forward access request to a pool of servers with the aim to improve the performance of the server to a certain extent. However, such solution also comes up with a new problem, that when the server cluster receives an access request, which server will have the chance to respond. As the control system cannot assign access requests in a reasonable way, then within such conditions load imbalancing will

probably appear [31]. Therefore, there is an urgent need to constantly develop and innovate load balancing algorithms [32].

1.5 ***Main Aims***

a) To design data-collection modules that allow the controller to collect the counters statistics of the OVS OpenFlow switch periodically.

b) To allow the administrator the ability to assign the timing interval of the data-collection modules.

c) To improve SDN performance by proposing new load balancing algorithms based on data-collection modules to address flaws, reduce average latency, and increase productivity.

1.6 Contributions

a) Proposing the adaptive least load ratio load balancing algorithm (ALLR) based on the server monitoring module (SMM) to collect counters statistics of the OVS OpenFlow switch periodically every 5 seconds and the ALLRCompute module (ALLRComputeM) to carry out the load balancing decision.

b) Proposing the hybrid-based load balancing algorithm (HB) based on the advanced server monitoring module (ASMM) to collect per-port and per-flow counters statistics of the OVS OpenFlow switch periodically every 5 seconds and the hybrid load balancing module (HLBM) to carry out the load balancing decision.

1.7 Scope of Work

The work was carried out according to the SDN-Based platform technology that utilizes the OpenFlow protocol version 1.0, Linux operating system version 14.04 LTS, mininet emulator version 2.2.1, SDN-based platform POX (*eel*) controller, and OVS OpenFlow switch version 2.0.2.

1.8 Study Outline

Chapter Two

In addition to chapter one, this chapter covers a detailed overview of theoretical backgrounds like SDN architecture, OpenFlow Protocol, OpenFlow switch, controllers, and a detailed classification of load balancing techniques/approaches in SDN.

Chapter Three

This chapter introduces the design, implementation, and evaluation of proposed novel load balancing algorithms represented by the adaptive least load ratio (ALLR) and hybrid-based (HB) load balancing algorithms.

Chapter Four

This chapter presents the experimental results along with associated discussions and scientific justifications.

Chapter Five

This chapter encloses the conclusions obtained from this work followed by suggestions and future works.

2.1 Introduction

SDN is a modern architecture that separates the control and forwarding functions in the network [33]. The separation is a departure from the conventional architecture where complex tasks are abstracted from the repeated forwarding tasks [34]. A complex process is automated and handled separately in a centralized SDN controller or Network Operating System [35]. The controller uses the OpenFlow protocol to connect with the real physical or virtual switch, and the data path uses the flow entries inserted by the controller in the flow table for routing data [36]. Three methods exist to separate the control plane and the data plane: fully distributed [37], logically centralized [38], and strictly centralized [39]. In the Fully distributed method, switching devices provide one essential feature for forwarding packets, but unfortunately, without a control power, which may lead to a failure point. The logically centralized method includes a remarkable feature signified in the Devices with a partial functionality embedded inside. The strictly centralized method adopts the conventional way of making all machines within all planes route packets through the network. The control plane is the manager that determines where each packet needs to go. The control plane communicates with the data plane to exchange management messages, update the routing table or speak to other control planes to achieve network changes [40]. The data plane is manufactured using various Technologies where its main function is packets routing. If there is no entry in the table for a packet's destination address then, the data

plane forwards it to the control plane for further processing. Application programming interfaces (such as North-bound Open APIs) establish communications between the application layer and control layer and facilitate different management business goals [41].

2.2 SDN and Virtualization

The virtualization technology provides a layer of abstraction between the networking hardware, storage, computing, and applications. The virtual infrastructure is important for the administrators to handle the shared resources around the enterprise in order to allow information technology managers to become more reactive to dynamic organizational requirements and additional leverage infrastructure instruments. The abstraction of the physical network in terms of a logical network is identified as network virtualization, clearly does not need SDN. In the same way, SDN is the separation of a logically centralized control plane from the fundamental data plane, does not imply network virtualization. Interestingly, however, a cooperation between network virtualization and SDN has risen, which has started to stimulate a few new research areas. SDN and network virtualization are related in the ways listed as bellow [42].

2.2.1 *Network Virtualization for Assessing and Testing SDN*

SDN control applications may be tested and evaluated in a simulated environment to isolate the SDN control applications from the underlying data plane [43]. Mininet uses process-based virtualization to

simulate a network of hundreds of hosts and switches on a single Computer [44].

2.2.2 Virtualizing (Slicing) SDN

In conventional networks, virtualizing a router or switch is difficult as each virtual component must execute its control plane software instance. Virtualizing a "dump" SDN switch, in contrast, is reasonably more straightforward. The FlowVisor (FV) framework enables the campus to maintain a testbed for network testing on the top of the same physical device to handle the production traffic. Every slice is handled by a separate SDN controller and holds its own set of networking resources. The FV operates as a hypervisor with the ability to communicate through OpenFlow protocol with each SDN-based platform controller and the underlying OpenFlow switches.

2.3 Differences between SDN and Traditional Network

The major difference that lies between SDN and other conventional network is that SDN is a software based network unlike others. The infrastructure of traditional network is purely physical comprising switches and routers while software based infrastructure operates virtually by control planes. SDN is much easier as the user has to interact with the software rather than any physical gadget. SDN can communicate more conveniently with gadgets as compared to physical infrastructures of conventional networks. The main difference that differentiates SDN from conventional network is the virtualization

ability. An abstraction of physical network is formed by virtualization that helps in provisioning of resources. The data plane guides the data flow in a traditional network which is located within routers or switches. For SDN the control plane is a software entity. This ensures that traffic is controlled with much efficiency and care by centralized user interface. This provides more hold over the network. Also, the configuration settings of a network can be changed and managed remotely. This is a much cost effective approach. SDN is now considered as the best alternative of traditional networks for their provisioning and bandwidth properties. The main differences between SDN and traditional network have been listed in table (2.1).

Table 2.1: Comprehensive Comparison between SDN and Traditional Network

SDN	***Traditional Network***
Programmable, supports business through the flexibility, agility, and virtualization	Inflexible, static, and low agility
Configured using software	Hardware apparatus
The control plane is logically centralized	The control plane is distributed

2.4 SDN Background

Martin Casado's proposed a research project under the name Clean Stat project at Stanford University [45]. The researchers addressed the integrated relationship between network administration and security as

they are both rightly responsible for connectivity regulation where the project represented the start of SDN concept growth [46]. Later, the Ethane project aimed to construct an advanced, more soft-management, tough-security, and efficient new network architecture for companies that allowed managers to quickly and dynamically transfer and change the security policies through centralized controllers [47]. Ethane accomplished this objective by integrating the simple flow switches with the centralized controller to handle the traffic flow in these switches via a secured connection.

2.5 SDN Architecture

The early definitions realized the SDN elements as different planes (application, control, and data). Each plane comes along with its functionality and communicates with each other through open standard interfaces. The SDN architecture is composed of the layers as listed in figure (2.1).

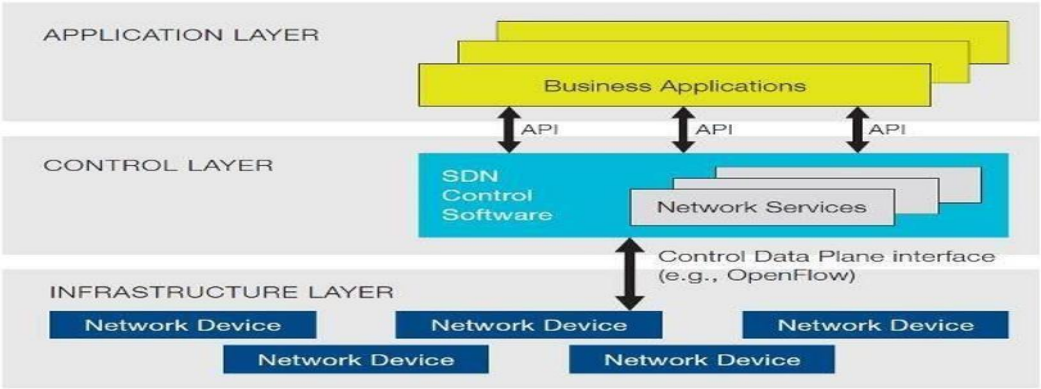


Figure (2.1): SDN architecture

2.5.1 Application Layer

The application layer consists of several networking application services that run on top of the controller. Based on network changes, application services could be used to configure the flows that need to be forwarded. The SDN applications communicate with SDN controller through APIs to manipulate network information. However, The APIs depend on the controller itself; whether or not the controller provides access APIs that allow developers to create their applications or not [48]. This layer consists of the services and applications that network provides to the user, and the most prominent examples of them are routing and QoS (Quality of Service). This layer communicates with the control layer using the application programming interfaces (API's), which are used by engineers to help the network to provide its services and application through programming.

2.5.2 Control Layer

The control layer provides a general view of the network topology to manage the OpenFlow switch via a secured communication channel. The control layer manipulates, controls, and manages the flow tables in the OpenFlow switch. In addition, the controller communicates with two interfaces: the Southbound interface that Communicates via a secured channel with the Data layer and the northbound interface that interacts via API's with the application layer. The API's provide an abstract view of the network and offers specific network features to meet the network

operator's needs. The SDN supports two-controller deployment modes [49]: the centralized mode where a single controller handles the entire SDN. The distributed mode where two or maybe more controllers manage the whole network. Every controller is called the domain controller that manages multiple OpenFlow switches and exchanges information with other controller in the network. An additional type of control mode is the master/slave mode [50], through which the slave controller serves as a backup to the master controller in any failure incident. The OpenFlow specification offers two approaches for managing flow tables: The proactive flow, where the controller sets up the flows in advance [51], and the reactive flow, where the controller responds to PacketIn events and updates the flow table dynamically [52].

2.5.3 Data Layer

This layer consists of networking devices, whether they are physical devices or work by Virtual technology and is responsible for passing on the data. Generally, this layer utilizes the southbound programmable interface (SBI) to interact with the control plane to receive the forwarding rules and apply them correctly on the devices [53].

2.5.4 Southbound Interface (SBI)

Communication through the SBI is very important for programming the behavior of the SDN switches by the controller that is how the SDN tries to do in order to program or configure the network, the most prominent example of the southern interface is the OpenFlow

protocol .However, most projects related to SDN assume that the connection of the controller with the switches is based on the OpenFlow protocol, but it should be noted that the OpenFlow protocol is one of the protocols, but it is fairly common compared to the many possible implementations of interactions with the console[54].

2.5.5 Northbound Interface (NBI)

NBI are software interfaces that support high-level applications, and provide services such as a firewall, load balancing system and communication between applications inside and outside the controller and the controller through this interface. The NBI gives the ability for the rest of the programs to access the controller and use the information and dependencies in it according to their desire, and thus the ability to program the network through software development [55]. The NBI carries out orchestration automation, active data share between systems, and the support of network functions like computing, load balancing, security, routing, and loop avoidance.

2.6 SDN Benefits

This section presents several advantages that SDN offers:

2.6.1 Improving Network Management

Network management and setup has been improved by the addition and removal of devices, which are necessary to modify the network topology. In traditional network, device manufacturers acquire

a different set of control interfaces with a different set of commands. This kind of diversity results in making the manual configuration quite dull and error-prone. Accordingly, many challenges arise in the design and dynamic adjustment of the traditional network. The SDN significant architecture, especially the logical centralized network operating system, enables the controller to acquire an abstract view of all forwarding devices in the network, which facilitates the network administration process as long as the configuration is performed from a single point. Each system should not be given the chance to be individually configured [56].

2.6.2 Improving Network Flexibility

When it comes to traditional networks, performance enhancement is achieved by delivering essential services to the users where the performance is below the required level because of relying on local information without considering the entire network. In SDN, as the controller is in the middle of its architecture, the exchange of complete network information leads to a global optimization performance [57].

2.6.3 Improving Network Development

Application developers and researchers in the legacy network domain, such as proprietary network infrastructure, encountered close connectivity between the data transmission, the control functionality, and common vendors Command Line Interface (CLI) to configure the

networking system. The SDN addresses these issues with a versatile SDN architecture and support for the concept of network programmability.

2.7 Network Performance

Network performance defines the quality of service (QoS) that is related to computer network infrastructure. The network performance consists of quantitative and qualitative measures/metrics, with a mention that is considered a qualitative process from a customer perspective [58]. There are three network performance analysis techniques: The first technique known as measurement reflects the fundamental approach done by software or hardware, or both. But, it is quite expensive. The second technique, known as analytical modeling, mainly focuses on constructing a mathematical model to introduce a tool for network testing before real implementation, then solving any possible issues. The third technique, usually known as simulation, is regarded as the most suitable technique to investigate network problems and verify performance that maps the network's substantial part to a virtual model [59]. The network metrics measurement is used to outline the network's condition and regarded as one of the critical elements in the network management procedure. The SDN categorizes three types of metrics: topology, traffic, and performance [60].

2.7.1 Topology Metrics

The topology metrics discuss the network information details such as the number of nodes, the links between nodes, and the switches' state

if they are active or not. The SDN controller initially explores the network topology's global view based on link layer discovery protocol (LLDP) [61]. After the OpenFlow switches receive the LLPD packets, the controller sends messages (packets) to all forwarding infrastructure devices (OpenFlow switches) in the data plane that directly connects with the controller. Later, the OpenFlow switches respond by sending a help message known as the Packet_in message due to the absence of any forwarding information in the flow table to the controller. The controller afterwards carries out the construction of the network topology [62].

2.7.2 Traffic Metrics

Traffic metrics analyze the network users' behavior by detecting the proposed number of packets or the expected duration of time for packets to proceed over the network. There are two different traffic types in the SDN [63]: The control flow generated from the transmission of data between the application and the network [64] besides the traffic flow generated from the transmission of OpenFlow switches data [65].

2.7.3 Performance Metrics

Performance metrics include a wide range of metrics to evaluate the performance of an SDN-based platform networks.

a) Utilized Bandwidth

Bandwidth capacity is an important consideration. Bandwidth utilization means the maximum data rate a link can transfer. The bandwidth utilization is calculated according to the formula listed as bellow:

Utilized Bandwidth $=\sum_{i=1}^{t} R_x + T_x$ ..... (2.1)[66]

Where $\boldsymbol{T_x}$ is the transmitted bytes, and $\boldsymbol{R_x}$ is the received bytes from the interface port during a predefined time interval ***t***.

b) Dropped Packets

Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. The reason for packet loss could be the inefficiency of a component such as a loose cable connection, a faulty router, or a lousy Wi-Fi signal. However, Unsuccessful packets lead to network slowdowns and cause bottlenecks. The dropped packets is calculated according to the formula listed as bellow:

Dropped Packets $=\sum_{i=1}^{t} N_{PS} - N_{PR}$ ..... (2.2)[67]

Where $\boldsymbol{N_{PS}}$ refers to the sent packets and $\boldsymbol{N_{PR}}$ refers to the received packets is the number of the packets received from an interface port during a predefined time interval ***t***.

c) Average latency of the POX controller

Defined as the time difference between the Packet_in messages sent from the OVS OpenFlow switch and the Flow_Mod messages received by the OVS OpenFlow switch from the POX controller.

d) Average latency of the POX controller:

$< Packet\_in\ messages\ \ Flow\ \_Mod\ messages >_t$ .....(2.3)[68]

The Packet_in message: In an OpenFlow SDN, when an OVS switch receives a packet on a port, it will try to match the packet to a flow entry in the switch's default flow table. If the switch cannot locate a flow that matches the packet, it will by default send the packet to the controller as a packet-in for closer inspection and processing.

Flow _Mod message: is the message sent from the controller to the switch as a response to the Packet_in message. It contains a data entry rule (input) that is entered into the flow table of the switch. It differs from the Packet_out message in that it contains more fields such as idle and hard_time out.

e) Average throughput of the POX controller

Refers to the summation of the Flow_Mod messages received by the OVS-OpenFlow switch from the POX controller.

Average throughput of the POX controller:

$\sum_{I=1}^{n} Flow\ \_Mod\ message$ ..... (2.4)[68]

Flow _Mod message: is the message sent from the controller to the switch as a response to the Packet_in message. It contains a data entry rule (input) that is entered into the flow table of the switch. It differs from the Packet_out message in that it contains more fields such as idle and hard_time out.

f) Average Throughput of Server

Average throughput is the ratio of the measure of data processed over a period, by that period of time

Average Throughput of Server= $\frac{1}{k}\sum_{i=1}^{k}\frac{bi}{ti}$ ..... (2.5)[67]

Where ***bi*** denotes the total amount of data, and ***ti*** is the time taken for the destination to get the final packet, ***k*** is the total number of application traffic.

g) Server Reply Time

It is sometimes called Server Transfer Time which equals the time between the first byte of response and the last byte of response. Typically, it is less than Server Response Time and impacted by the user requests.

Server Reply Time= $T_{fb}+T_{lb}$.....(2.6)[69]

Whereas $\boldsymbol{T_{fb}}$ is the time of the first byte of the response and $\boldsymbol{T_{lb}}$ is the last byte of the response. In general, the reply time is less than the average response

h) Server CPU Utilization

Represent percentage of CPU capacity during a specific period of time

$$\textbf{\textit{Server CPU Utilization}} = \frac{B.T}{O.T} \quad \text{.....(2.7)[69]}$$

Whereas ***O.T*** denotes the observation time defined as the total server monitoring time and ***B.T*** that denotes the Business time defined as the total time the server is being on during OT.

i) *Server Average Queue Length*

Is the number of requests waiting for a service in the queue of type (*M/M/1*) generated between the controller and server members in the pool. However, the server average queue length calculation is based on the computation of server CPU capacity from equation (2.7).

$$\textbf{\textit{Server Average Queue Length}} = \frac{U}{1-U} \quad \text{.....(2.8)[69]}$$

Where ***U*** is the Server CPU utilization (Usage)

j) *Server Transactions per Second*

Refers the number of transactions processed by the server per second. However, it may also refers to the average throughput of the server.

$$\textbf{\textit{Server Transactions per Second}} = \frac{\sum TPS1+TPS2+\cdots TPSN}{T} \quad \text{.....(2.9)[69]}$$

Where ***TPS*** is the server transactions per second and ***T*** is time driven to complete the transactions in seconds.

k) Server Average Response Time

Average response time is one of the critical metrics in load balancing systems. It is interpreted as the amount of time taken to return the results of a request to the user. The response time is affected by various factors, for instance, bandwidth, the number of users who access the system at the same time, the number of requests and average thinking time. To get faster responses, a high number of requests per second must be processed. Thus, we can calculate the response time as follows:

l) Server Average Response Time $= \frac{Un}{Rn} - T_T$ .....(2.10)[69]

Whereas $\boldsymbol{T_T}$ is the thinking time per request (in general the thinking time =3 seconds at start), $\boldsymbol{U_n}$ is the number of concurrent users, and $\boldsymbol{R_n}$ is the number of requests per second

m) Server CPU Capacity = $\frac{Server\ Average\ Response\ Time}{1-Server\ Average\ Response\ Time}$ .....(2.11)[69]

2.8 *OpenFlow Protocol*

The OpenFlow protocol provides a structured way to handle the OpenFlow switch traffic and exchange information between the OpenFlow switches and the controller [70]. The OpenFlow protocol does not consider the formulation of forwarding decisions (i.e., actions) of the header fields where the controller makes these decisions and downloads or installs them into the flow tables of OpenFlow switches [71]. The process begins by looking-up flow tables and matching the header fields of the incoming packets with the pre-calculated forwarding decisions, In case of a match, then the associated rules are followed. If no match is found, then the packet is forwarded to the controller for further processing, and the packet returns back to the OpenFlow switch where

the controller performs the installation of the relevant flow into the flow table [72]. The lookup process begins at the first table when a new packet arrives and terminates with a match or a miss at one of the pipeline tables. Miss action occurs when there is no instruction to apply for the packet. The combination of different matching fields could establish a flow rule. If there is no default rule, then the packet mostly ends up being discarded. Generally, a standard rule is installed to notify the OpenFlow switch to forward the packet to the controller [73]. The flow table performs several actions such as: Packets forwarding to the outgoing ports, Packets encapsulation and transmit to the controller, Packets dropping, and Packet sending to the usual pipeline of processing.

2.8.1 OpenFlow Protocol Messages

OpenFlow protocol messages refer to the messages exchanged between the controller and the OpenFlow switch to provide a secured OpenFlow channel on the top of the secure socket layer (SSL) or Transport Layer Security (TLS) [74]. OpenFlow protocol supports Three different types of messages: asynchronous [75], symmetric[76], and controller-to-switch [77]. The asynchronous message is sent without any reference from the controller and consist of several status messages. The packet-in message is an excellent example of an asynchronous message that occurs when a packet arrives at the OpenFlow switch and does not match the stream's input. The Symmetric messages is simple, helpful, and sent from the OpenFlow switch or controller without request. The symmetric message includes the Hello messages between the controller

and the OpenFlow switch when the connection is established. Echo request and reply messages are other examples of symmetric messages for both the controller and the OpenFlow switch to evaluate a controller-switch connection's latency or bandwidth or verify that the system is under operation. The controller-Switch message is initiated by the controller with a response from the OpenFlow switch where the controller handles the OpenFlow switch's logical state. The controller-to-switch message involves the packet-out message when the OpenFlow switch sends a packet to the controller.

2.8.2 Flow types

According to the number of hosts, Flows could be classified into micro flows [76] and aggregated flows [77]: in Micro-Flow, the flow table encloses a single entry for each flow configured in an individual mode by the controller to keep track of the flow entry condition. In the Aggregated-Flow, the single flow entry covers a large group of flows with permission for wildcard flow entries. The flow table is appropriate for a large number of flows and includes one entry per flow type [78]. The flow entries population mechanism consecutively is similarly classified into a reactive-flow [79] and proactive-flow [80] modes. In the reactive-flow mode during which the controller is triggered to insert flow entries where every flow needs a short additional time for flow setup. In the Proactive-Flow mode**,** the controller pre-populates the flow tables without additional flow setup time. Proactive-Flow usually involves aggregate (i.e., wildcard) rules where the processing load of the

controller reduces since the traffic is not interrupted during a connection loss and the query is not being sent every time to the controller[80].

2.9 *Classification of SDN Load Balancing Techniques / Approaches*

This section introduces the majority of the techniques, approaches, and algorithms related to load balancing mechanisms in an SDN-based platform.

2.9.1 *Slice Technique*

The slicing technique emerged as a result of virtualizing the SDN network. The slicing technique introduced a virtual layer mounted on the top of the physical network infrastructure by constructing several virtual networks where different types of virtual resources in digital networks require to be observed and supervised. The transparent proxy achieves the control mechanism that connects the OpenFlow switches with multiple controllers for each Virtual Network, and a slice is generated for proxy controllers such as FV and OVX. However, The FV slices process depends on the packet header field and transmits the packets based on their respective policies [81].

2.9.2 Wildcard Technique

The wildcard technique is based on the customer's IP address to route the incoming requests. Many rules are composed due to inserting different rules for each packet flow in the flow table. Therefore, Any Packet_in message that needs to be manipulated by the controller causes

a heavy load. The Wildcard technique suggests an algorithm based on proactive mode to compute the concise wildcards rules adjusted automatically for different load balancing strategies to address the loading issues. The algorithm inserts new flow entries without interfering with the controller [82].

2.9.3 Genetic-Based Technique

The genetic-Based technique redirects many traffic flows from clients to servers to achieve optimal load balance. The Genetic-Based assumes several N flows with a different load for each flow, the additional workload for each server pool server, and a fitness function that minimizes the servers' coefficient. The Genetic-Based architecture consists of an SDN controller, OpenFlow switch, and three modules mounted on top of controller: monitor, decision, and flow control [83].

2.9.4 *L2 Direct Server Return Approach (L2DSR)*

The L2DSR forwards the packet while substituting its MAC address as the source MAC address to improve the performance of the OpenFlow switch. The system architecture consists of a load balancer controller (LBC) that carries out the servers' separation process from the network to prevent conflictions. The core idea is that initial requests is passed through the load balancer. But, the response goes directly from the real server to the client. Thus, bypassing the load balancer [84].

2.9.5 *Flow-Oriented Approach*

The flow-oriented approach focuses on policies that organize the entire data flow to the required servers. The Flow-Oriented approach assumes that communication establishes between clients and servers when a client sends requests to the server. The flow remains active even after a specific time of inactivity. [85]. The summarization of the approaches/techniques including operating layer as well flow table management mode involved in SDN load balance is listed in table 2.2.

Table 2.2: Summarization of Approaches/Techniques in SDN Load Balancing

Approaches/ Techniques	***Layer***	***Flow Table Management***
Slices	Control	Reactive
Wildcard	Data	Proactive
Genetic based	Application	Reactive
L2DSR	Data	Reactive
Flow-oriented	Application	Reactive

2.10 SDN-Based Platform Load Balancing System Components

The following components are essential to perform load balancing tasks within SDN standards:

2.10.1 Open-Source Controllers

The SDN controller is responsible for maintaining the stability and implementing the policies and rules in the network as well as distributing instructions to different network devices using the OpenFlow protocol

[86]. The controller has a global view of the network components and is usually referred to as the NOS (Network Operation System) [87]. The SDN controller defines the flows that occur in the data plane as each flow in the network must first get permission from the controller according to the network policy, if the controller allows the flow it computes the path for that flow and adds an input to this flow in each of the switches along the path, and as a result the switches can simply manage flow tables that can be filled with inputs only by the controller [88]. A controller is an application in a software-defined network architecture that manages control flows in a network in order to improve network management and performance [89]. Thus it is an application that is downloaded to a specific server to become a controller that uses the (OpenFlow- OVSDB) protocols to communicate and tell the switches where to send packets. The controller can be divided into three components as shown in figure (2.2).

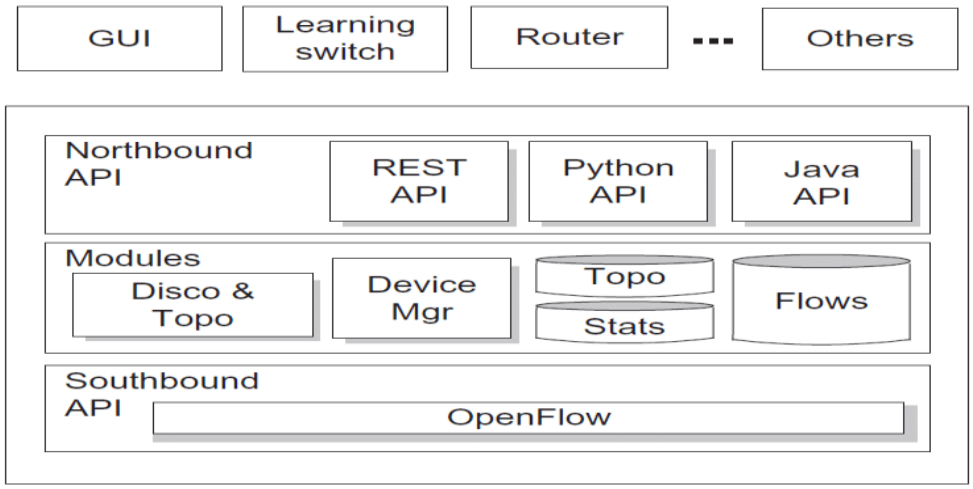


Figure (2.2): Open-Source controller architecture

The first component is the northbound interface (NBI). The NBI is the northern programming interface that supports high-level applications, provides services such as a firewall, load balancing system, and communication between applications outside the controller and the controller. The second component is the Controller Kernel, which is a set of applications, components, or software modules that are present in the controller by default (build-in), which differ from one controller to another as it provides the basic functions of the controller. Most of the controllers contain some similar components, that is, each controller comes with some of the common and most popular applications are: a learning switch application, a simple firewall application, a simple load balancing application and some other applications. Key features offered by the kernel applications of the SDN controller include: Discovering end-user devices**,** Network devices discovery, and Topology management of network devices, and Flow Management: Its function is to maintain and implement a database of flows that are managed by the console. The third component is the southbound interface (SBI), which provides control functions to the infrastructure layer (data level switches) such as switches, where this interface is usually represented by a protocol such as OpenFlow and OVSDB, this interface collects information from data level switches and provides input to the streams required for the SDN traffic process. Moreover, the controllers could be classified into two categories:

a) Open-Source Controllers

Which are development projects that publish their code on platforms such as Github, and are often developed by people. Several types of Open-Source controllers listed as bellow:

1- POX

POX is a component-based open flow controller used for prototyping. The elements and activities of the entire network are recognized as separate components that are isolated and used each time a situation needs it. The POX is located between network components and applications. POX may be used as a basis for ongoing work to aid in the emerging disciplined construction of SDN applied in distribution and troubleshooting models, debugging, network utilization, controller design, and programming model [90]. There are several factors behind the utilization of POX like POX owns extra python programming modules to run different applications like a hub, switch, load balancer, and a firewall, POX is easy to set up, install, and run. POX is supported by several platforms like Linux, MAC, and windows, and POX contains re-usable sample components for topology discovery and path selection.

2- NOX

First-generation, Written with C++, developed by Nicira, and donated to the research community in 2008, thereby becoming

open source. The controller registers for events, and the programmer writes event handlers [91].

3- RYU

Written with Python, centralized, and aims to be an operating system for SDN-Based platform. The advantages of Ryu include the support of OpenFlow 1.0, 1.2, 1.3, 1.4, and OpenStack [92].

4- Floodlight

Written in Java. Floodlight support OpenFlow 1.0 with a wide range of features and a very supportive community that could be incorporated to build a solution that best meets a particular organization's. Moreover, the Floodlight is maintained by the big switch systems and forks from Beacon OpenFlow Java controller. The advantages of Floodlight include good documentation, the integration with REST API, and the support of OpenStack and Multi-tenant clouds [93].

5- OpenDayLight

Written in Java. OpenDayLight is a joint project of the Linux Foundation endorsed by Cisco. OpenDayLight is a common industry-supported platform that is robust, extensible open-source codebase, and distributed. The advantages of OpenDayLight include the industry acceptance and the integration with OpenStack as well as cloud applications [94].

b) The other type of controllers is the Commercial (Closed Source)**,**where most of the well-known network companies now have their own controllers, for example Cisco has Cisco ACI and Juniper has Contrail [95].

2.10.2 Open vSwicth (OVS)

OVS generally refers to the OpenFlow switch in SDN. OVS is an open-source, multi-layer virtual switch designed to establish a connection between virtual and physical interfaces. The OVS is driven by the increasing demands in virtual environments and enables the exchange of OpenFlow messages with the controller via a secured communication channel to carry out essential tasks like configuration and management. The OVS in mininet consists of the following components: the ovs-vswitchd that is programmed by the controller via the OpenFlow protocol or is programmed periodically by some tool in order to assign flow rules to the switch to pass data, The kernel module that Saves the flow rules sent from the previous component for the purpose of not consulting the controller every time a packet arrives at the controller and thus not visible to the controller. The OVS operates in two modes: the ***standard mode*** which is the default mode that operates as a traditional switch (L2_swicth) and uses The ARP protocol in the process of learning and passing data between computers and therefore does not depend on the controller. The other mode is the ***secure mode*** that depends on the controller in the process of assigning flows in order to pass the data [96] as shown in figure (2.3).

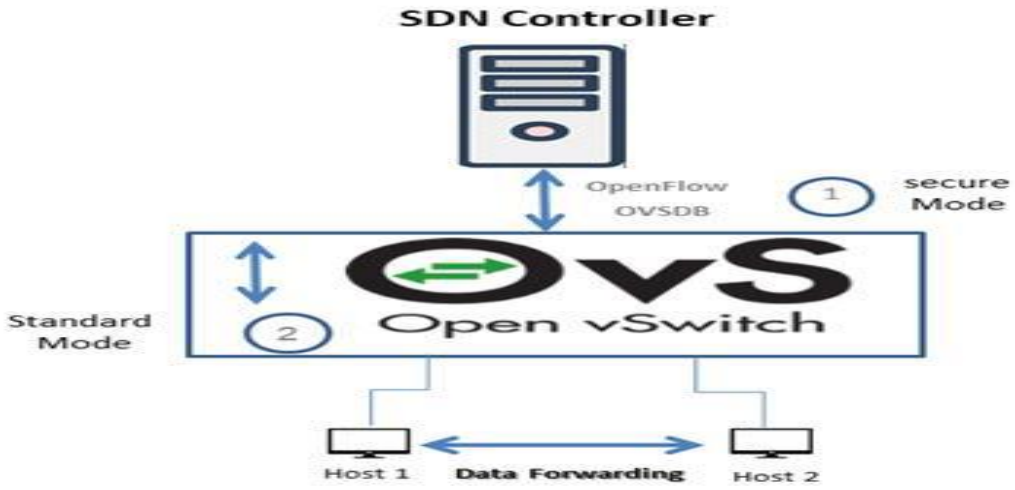


Figure (2.3): OVS OpenFlow switch operation modes

Moreover, the OVS is configured in two ways: by the controller, and manually by the command line interface utility (CLI), which consists of: the **ovs-ofctl** administrative tool that uses the OpenFlow protocol to set the OVS and the **ovs-vsctl** tool that uses the OVSDB protocol to set the OVS as well [97] as shown in figure (2.4).

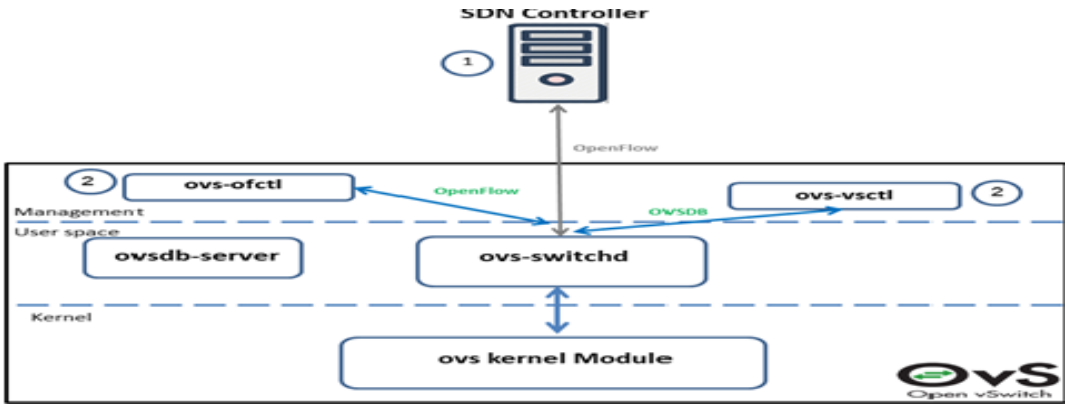


Figure (2.4): OVS OpenFlow switch configuration modes

The OVS mechanism for the data passing process goes through the following steps as shown in figure (2.5).

a) The first packet that arrives from the flow will be sent to the controller.

b) The controller is consulted to calculate the path for the flow and send instructions (actions) which may be one or more to the ovs-vswitchd or the ovs-vswitchd is programmed manually to calculate the path.

c) The instructions are sent to the OVS.

d) The rest of the packets belonging to the same flow are processed by ovs-kernel component.

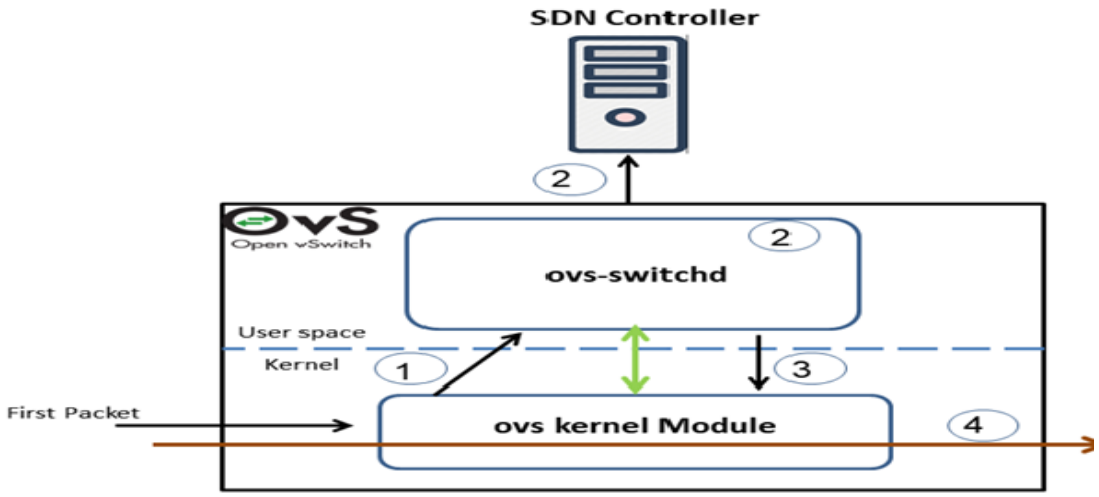


Figure (2.5): OVS OpenFlow switch data processing mechanism

Flow table in OpenFlow switches consist of flow entries that determines how the incoming packets manipulate and forward to the desired destination. However, each flow entry consists of match fields, counters, and several sets of instructions to apply on matched packets, and

associated actions that direct the packets to a group that performs extra processing, such as link aggregation, flooding, multipath forwarding, and rapid rerouting. The switch contains flow tables that in-turn contain flow entries where each flow entry consist of:

a) Header fields or match fields, with information found in packet header, ingress port, and metadata, used to match incoming packets.

b) Counters, used to collect statistics for the particular flow, such as number of received packets, number of bytes, and duration of the flow

c) A set of instructions or actions to be applied after a match that dictates how to handle matching packets. For instance, the action might be to forward a packet out to a specified port as shown in figure (2.6).

2.10.3 ***Virtual IP Address (VIP)***

The VIP represents the load-balancing instance that the external clients use to obtain the services provided by the service pool. The VIP contains a TCP or UDP port number such as port 80 for web traffic and an IP address that should be publically accessible (default is 10.0.1.1) through which one or more active servers are hosted on this IP address. A VIP will have at least one real server assigned to it, to which it will dispense traffic, usually there are multiple real servers, and the VIP spreads the traffic among server members ports.

2.10.4 Servers

Servers usually refer to the Apache HTTP web-server members that are defined as the set of one or more server member with one or more associated IP addresses connected through software, hardware, and network equipment. Moreover, The Apache server members form the server pool, avoid being publicly accessible, and behave if they are a single system. It is necessary to understand that the Apache server is the most commonly used Web server on Linux systems [98]. Apache server member most important parameter is the ***MaxClients*** that sets a limit on the total number of server processes or simultaneously connected clients that can run at one time. The main purpose of this directive is to keep a runaway Apache HTTP Server from crashing the operating system. For busy servers, this value should be set to a high value. However, the Apache's server member default MaxClients is set to ***256*** [99]. The ***MaxClients*** sets the limit on the number of simultaneous requests that can be supported. Definitely not more than the aforementioned child processes can be created. To configure more than 256 clients, then the HARD_SERVER_LIMIT must be edited. Any connection attempts over the MaxClients limit is queued [100].

2.10.5 Load Balancing Application

Load Balancing means reassigning the full load for the server members in the pool to make resource utilization more efficient and improve requests' response time. Simultaneously, it eliminates the

condition where some servers are under loaded while others are overloaded. However, there are two classes inside the POX load balancing application: The MemoryEntry class that represents the abstracted flows in the OpenFlow switches, and the iplb class includes the balancer logic. When it comes to the iplb class, it stores all of the Servers' IP addresses, active flows, the methods involved in the Server's selection, and the methods that handle the controller's packets. The load balancing component receives two addresses: the IP address on which the balancer is running (most often 10.0.1.1) and the Servers IP list. The POX uses the information mentioned above and sends an ARP Message to check if a server crash occurred (the host that has not reacted for some time to this message has possibly crashed and must be deleted from the list). When the balancer IP address receives a TCP request, it forwards it to the active servers' IP address in the Server's list through a load balancing algorithm (static or dynamic). When this process runs, a flow copy is created in the controller in a specific period**.** Different algorithms have been implemented in the SDN load balance framework for selecting the best server to handle the incoming requests. Generally, there are three types of load balancing algorithms: static [101], dynamic[102]. The static does not consider the load status of server, so it is suitable for the condition that load status can be predicted in advance. Moreover it is appropriate for platforms where the load changes at a low rate. The preliminary information about system resources is required to ensure those load shifting decisions do not rely on the system's current state. The

dynamic load balancing algorithm refers that the load balancer can dynamically distribute the load to the optimal server according to the load status of the server. The dynamic distribution of the pre-programmed load balancing patterns makes the dynamic load balance strategies more efficient than static. The appropriate selection of load balancing algorithm is essential to achieve maximum flow and minimum response time, energy, and resources overload. Two methods are available to implement the dynamic load balancing scheme: centralized and distributed. In the nondistributed approach, a centralized node receives and distributes all requests to the servers. While in the distributed approach, nodes corporate with the request's distribution.

2.11 Experimental Environment

SDN is an ever-developing technology. Therefore, it is a challenge to deal with such a system directly. The solution may differ when the emulator is used. The emulator is a mixture of software and hardware resources to represent the SDN in a virtual environment. The analysis is conducted using the following tools:

2.11.1 Oracle VM VirtualBox

VirtualBox is a powerful virtualization product for X86 and AMD64/intel64 dedicated to enterprise and domestic usage. The Virtual Box operates as a hypervisor to run an operating system mounted on top of the existing virtual machine. The VirtualBox includes useful functions

to control virtual machines and runs on several operating system platforms such as Windows, Linux, Macintosh, and Solaris [103].

2.11.2 Mininet

Mininet is a network emulation orchestration system that runs routers, SDN switches, different end-hosts, and links between all the devices on a Linux kernel system. Hosts in mininet behave nothing, but like a real machine in a way, the client may run different arbitrary programs. The Mininet is used exclusively in constructing virtual SDN, which consists of an OpenFlow controller, a flat Ethernet network of multiple OpenFlow-enabled Ethernet switches, multiple hosts connected to switches, and Built-in functions support different types of controllers and OpenFlow switches. The Mininet Python APIs enormously help create complex custom scenarios compared to any other network emulator. Mininet is easy to use, open-source and equipped with many inbuilt features. Mininet features include: The operation in a Quick way, the creation of custom network topologies, the ability to run Real-Programs, and the customization of packet forwarding. [104].

2.11.3 Evaluation Suite for Mininet

Several types of network benchmarking tools have been utilized in mininet to generate the traffic load in SDN-based platform environment. The first tool is the Distributed Internet Traffic Generator ***(D-ITG)*** [105]. The D-ITG is defined as a distributed internet traffic generating tool that supports both IPV4 and IPV6 traffic generation. D-ITG could replicate

suitable stochastic processes for both PS (packet size) and random variables IDT (inter-departure time). Additionally, this tool is used to create multiple flows that play a major role in evaluating the network's performance. D-ITG includes two flow modes. The first mode is the Single Flow Mode, where the ITGSend generates only one traffic flow according to the given command-line options in which a dedicated thread handles the flow. At the same time the ITGRecv component uses a different thread to set up and organize the generation process on a separate channel. The second mode is the Multi-Flow Mode, where the ITGSend allows the multi-flow mode to create multiple flows simultaneously, and a single thread manages each flow. The second tool is the ***httperf***. The httperf is a software benchmarking tool that evaluates the performance of the HTTP web server. Subsequently, several metrics are measured directly through this tool, like server throughput, reply time, and connection time while other metrics are obtained in an indirect way, such as Server CPU utilization and Queue Length. Generally, it comes up with several advantages like the support of HTTP 1.1 protocol, the save and generation of server overload, and the extensibility to statistics collectors and workload generators. The third tool is the ***OpenLoad*** that performs load testing of web server applications. OpenLoad measures and records metrics such as the server average response time and server transactions per second. This tool is easy to use and provides a near real-time performance measurement of applications under test. The last tool is the ***Cbench***, which is one of the main open-

source licensed benchmarking tools that is designed to evaluate the average throughput and average latency of the SDN-based platform POX controller. The Cbench tool pretends to act as a switch, establishes working sessions with the controller, sends packets, and waits for the output messages.

2.12 Summary

This chapter reviews the SDN-Based platform theoretical background, which is relevant to the primary concern of the study. The SDN architecture is analyzed by identifying the layers, interfaces, and metrics. A taxonomy of techniques, approaches, and algorithms performing the load balancing in the SDN-Based platform environment is also presented

3.1 Introduction

This chapter aims to analyses the problem that was highlighted in chapter 1 and conducts a deep investigation to show the impact of the user's requests. The focus was on the impact of the different types of requests that use the proposed system design. This scheme used different algorithms that aim to select the best server for handling the incoming request. In order to analysis the problems, various experiments are performed with statistical analysis.

3.2 *Proposed Model Architecture*

The proposed model architecture consists of three major component: the ***First component*** represents the POX controller performance evaluation based on examining the effect of QoS metrics and measuring the ability of the POX controller to produce a throughput and latency while increasing the load on the linear topology and at what point of network load (number of OpenFlow switches) the POX controller stop responding. The ***Second component*** represents the proposed system design that consist of the proposed load balancing algorithms (ALLR and HB). The ALLR and HB cooperate with the POX controller as load balancing algorithms to determine the most suitable server for handling the traffic. The ***Third component*** represents the tests related to applying the ALLR and HB load balancing algorithms on the network topology assigned to each one of them. The proposed model architecture diagram is illustrated in figure (3.1).

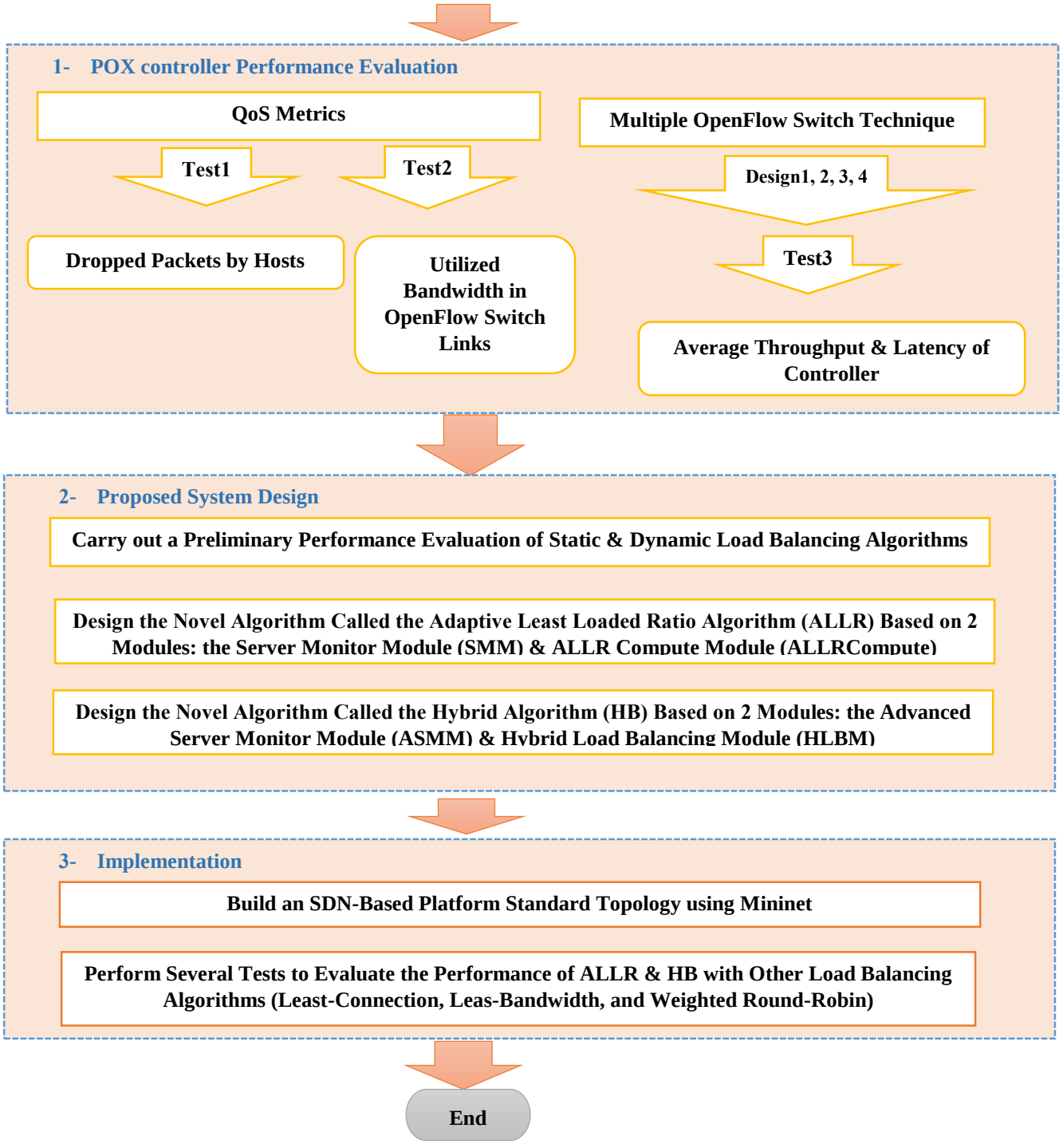


Figure (3.1): Proposed Model Architecture

3.3 *POX Controller Performance Evaluation*

In order to evaluate the performance of the SDN-based platform POX controller, two techniques have been utilized:

3.3.1 QoS Metrics Technique

This technique offers a way to mark traffic prioritization over networks to guarantee a certain level of performance. Moreover, QoS makes sure that routing priorities do not change. Two QoS metrics have been examined: the dropped packets by hosts and the utilized bandwidth by OVS OpenFlow switch links represented by algorithm (3.1) and algorithm (3.2).

Algorithm (3.1): QoS Metrics – Dropped Packets
Input:
-No. of OpenFlow Switches -No. of Hosts -No. of Packets
Output:
-Measure the Dropped Packets by Hosts Based on Packet Size. The dropped packets refer to the number of packets loss or drop during their travel across a computer network. The unsuccessful packets lead to network slowdowns and cause bottlenecks.
Process:
1- Initialize the L2_learning module for the SDN-based platform POX controller /* this component invoked by the POX controller enables the OpenFlow switch to act as a type of L2 learning switch */ 2- Construct a network topology /* 8-OVS OpenFlow switches 8-hosts with IP's from 10.0.0.1 to 10.0.0.8 8-links Between OVS OpenFlow switches and hosts

Single Remote POX controller with listening port 6633 between the POX controller and OVS-OpenFlow switch and loopback address 127.0.0.1*/

3- Link up OVS OpenFlow switches and hosts and select the Complete Fair Scheduling algorithm (CFS)
4- Test connection between hosts
/* a ping is used between a source host(i.e. 10.0.0.1) and destination host(i.e. 10.0.0.8) to check network connectivity */
5- Set Packet_size ⟵ 64 Bytes
6- While packet_size < 1500 Bytes do
/* the 1500 Bytes represent the Maximum Transmission Unit (MTU) for TCP protocol. */
7- Packet_size ⟵ Packet_size + 64 Bytes
8- Execute the "Distributed Internet Traffic Generator (D-ITG)" to determine the dropped packets between hosts.
9- End while
10- Stop the POX controller and clear the network topology
11- End

Algorithm (3.2): QoS Metrics – Utilized Bandwidth

Input:

-No. of OpenFlow Switches
-No. of Hosts

Output:

-Calculate the utilized Bandwidth in OVS OpenFlow switches. The utilized bandwidth represent the actual bandwidth consumed by OVS OpenFlow switches

Process:

1. Initialize mininet libraries
2. Initialize the L2_learning switch for the remote POX controller
/* this component invoked by the POX controller enables the OpenFlow switch to act as a type of L2 learning switch */
3. Construct topology consisting of 8 OVS OpenFlow switches and 8 hosts
/* 8-OVS OpenFlow switches
8-hosts with IP's from 10.0.0.1 to 10.0.0.8
8-links Between OVS OpenFlow switches and hosts

Single Remote POX controller with listening port 6633 between the POX controller and OVS-OpenFlow switch and loopback address 127.0.0.1*/

4. Link up OVS OpenFlow switches and hosts and select the Complete Fair Scheduling algorithm (CFS)
5. Test connection between hosts
   /* a ping is used between a source host(i.e. 10.0.0.1) and destination host(i.e. 10.0.0.8) to check network connectivity */
6. Set the initial Allocated_Bandwidth ⟵ 10 Mbps
7. While Allocated_Bandwidth < 300 Mbps do
   /* the range of allocated bandwidth is ruled by the queuing bandwidth allocation management approach for slow to moderate application up to 300Mbps*/
8. Allocated_Bandwidth ⟵ Allocated _Bandwidth + 10 Mbps
9. Execute the " iperf" test to determine the utilized bandwidth in OVS OpenFlow switches
10. End while
11. Stop the POX controller and clear the topology
12. End

3.3.2 ***Multiple OVS OpenFlow Switches Technique***

The multiple OpenFlow switches technique aims to examine the ability of the POX controller to process the Packet_in message flows that do not have a database in the OpenFlow switches in order to acquire the necessary full knowledge of SDN components. The Multiple OVS OpenFlow Switches Technique has been represented by algorithm (3.3).

Algorithm (3.3): Multiple OpenFlow Switch Technique

Input:

-No. of OpenFlow Switches

-No. of Hosts

-Design_No.

Output:

- Evaluate the Ability of POX to Process the Flows of Designs 1, 2, 3, and 4 in terms of Average Throughput and Average Latency. The average POX controller average throughput is defined as the number of flow requests a POX controller can process per unit time.

Process:

1. Initialize mininet libraries
2. Initialize the L2_learning switch module for remote POX controller
   /* this component invoked by the POX controller enables the OpenFlow switch to act as a type of L2 learning switch */
3. Initialize CPU-time per host to 10%
4. For Design_No. ⟵ 1 to 4
5. Case Design_No. of
6. Case :1
7. Construct design1 (1 OVS Open-Flow switch)
   /* 1-OVS OpenFlow switches
   1 -hosts with IP's from 10.0.0.1 to 10.0.0.8
   1-link Between OVS OpenFlow switches and hosts
   Single Remote POX controller with listening port 6633 between the POX & OVS-OpenFlow switch and loopback address 127.0.0.1*/
8. Test connections between the hosts
   /* a ping is used between a source host(i.e. 10.0.0.1) & destination host(i.e. 10.0.0.8) to check network connectivity*/
9. Execute the "Cbench" test to determine the POX controller average throughput (flows/sec) & average latency (msec).
10. End Case 1
11. Case 2:
12. Construct design2 (4 OVS OpenFlow switches)
    /*4 -hosts with IP's from 10.0.0.1 to 10.0.0.8
    4-link Between OVS OpenFlow switches & hosts
    Single Remote POX controller with listening port 6633 between the POX controller & OVS-OpenFlow switch and loopback address 127.0.0.1*/
13. Test connections between the hosts
    /*a ping is used between a source host(i.e. 10.0.0.1) and destination host(i.e. 10.0.0.4) to check network connectivity*/

```
14.         Execute the "Cbench" test to determine controller average throughput (flows/sec) &
            latency (msec).
15.     End Case 2
16.     Case 3:
17.       Construct design3 (16-OVS OpenFlow switches )
          /*16 -hosts with IP's from 10.0.0.1 to 10.0.0.8
            16-link Between OVS OpenFlow switches and hosts
            Single Remote POX controller with listening port 6633 between the POX controller
            & OVS-OpenFlow switch and loopback address 127.0.0.1*/
18.       Test connections between the hosts
19.       /* a ping is used between a source host(i.e. 10.0.0.1) and destination host(i.e. 10.0.0.4)
          to check network connectivity */
          Execute the "Cbench" test to determine the POX controller average throughput
          (flows/sec) and average latency (msec).
20.     End Case 3
21.     Case 4:
22.       Construct design4 (64-OVS OpenFlow switches )
23.       Test connections between the hosts
          /* a ping is used between a source host(i.e. 10.0.0.1) and destination host(i.e. 10.0.0.4)
          to check network connectivity*/
24.       Execute the "Cbench" test to determine the POX controller average throughput
          (flows/sec) and average latency (msec).
25.     End Case 4
26. End for
27. Collect and Compare results of  to evaluate the performance of designs 1,2,3, and 4
28. Stop the POX controller  and clear the design topologies
29. End
```

3.4 Proposed System Design

This section represent the core part of the methodology, which is categorized into three subsections: a preliminary performance evaluation

of load balancing algorithms running under SDN-based platform infrastructure, the adaptive least load ratio (ALLR), and hybrid based (HB) load balancing algorithms.

3.4.1 *Preliminary Performance Evaluation of Load Balancing Algorithms*

This section investigate the characteristic, behavior, and performance of load balancing algorithms: random, round-robin, weighted round-robin, least connections-based, and least bandwidth-based.The evaluation process adopts the httperf benchmarking tool to examine the actual reaction and efficiency of the load balancing algorithm to distribute the traffic to the Apache HTTP webservers based on server average throughput metric represented by algorithm (3.4).

Algorithm (3.4): Preliminary Performance Evaluation of Load Balancing Algorithms

Input:

-No. of Apache HTTP Servers
-No. of Hosts
-Send Buffer
-Receive Buffer
-Algorithms_No.

Output:

- Select the Maximum Outperforming Algorithms in terms of Server Average Throughput.

Process:

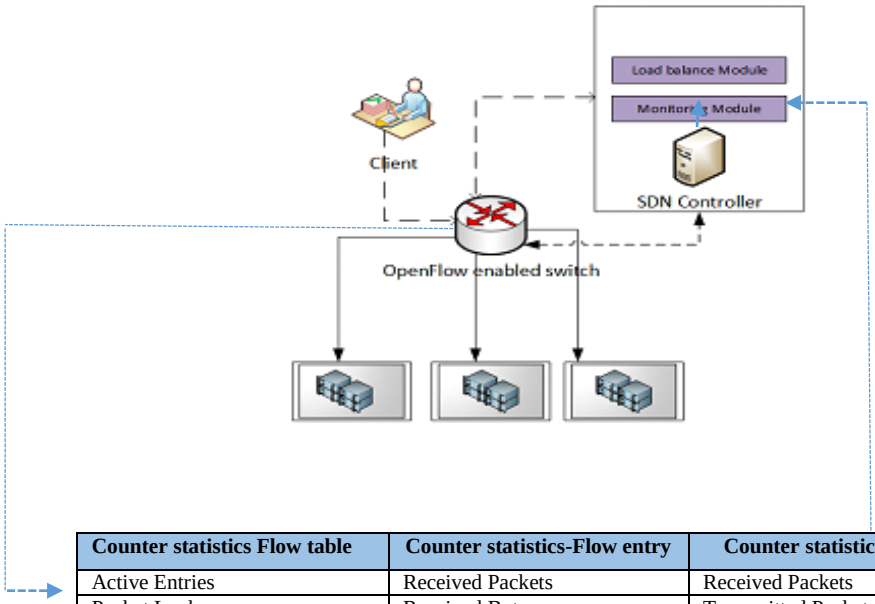
1. AlgorithmNamesArray[1,…5] = ”Random”, “Round-Robin”, “Weighted Round-Robin”, “Least Connection-Based”, “ Least Bandwidth-Based”
2. If DIP =VIP
3. /* DIP is a 64 bit value assigned to switches in a straight forwarding way and is communicated from the switch to the controller during handshaking by way of the

ofp_switch_features. Switches connect to POX and the communication might go either from the controller to a switch, or from a switch to the controller. When communication is from the controller to the switch, this is performed by controller code which sends an OpenFlow message to a particular switch (more on this in a moment). When messages are coming from the switch, they show up in POX as events for which you can write event handlers. There are essentially two ways you can communicate with a datapath in POX: via a Connection object or via an OpenFlow Nexus which is managing that datapath.

4. For Algorithm_No. ⟵ 1 to 5
   /* ”Random”, “Round-Robin”, “Weighted Round-Robin”, “Least Connection-Based”, “ Least Bandwidth-Based” are evaluated under the same load conditions where the requests range from 0-180 and network topology consist of single remote POX controller , 8-OVS OpenFlow swicthes, and 8-hosts*/
5. For J ⟵ 1 to 180
   /* the number of requests is generated using the httperf benchmarking tool and the number of requests has been limited to 180 request/sec due to the MAX_CLIENTS parameter of the Apache HTTP servers (256 request/sec)*/
6. Request[J] ⟵ Packet_in Message
   /* no rules regarding path forwarding for the packets exist in the OpenFlow swicth, thus requests are forwarded to the POX for decision making*/
7. Select the AlgorithmNames[Algorithm_No.] to handle the incoming requests
8. According to the policy select the server to handle the incoming request
9. Push flow entry
10. Evaluate the performance in terms of server average throughput
11. End for
12. Compare performance the of ”Random”, “Round-Robin”, “Weighted Round-Robin”, “Least Connection-Based”, “ Least Bandwidth-Based “in terms of server average throughput
13. End for
14. Select the Max outperforming algorithms
15. End if
16. Else
17. Flood the traffic
18. End

3.4.2 ***The Adaptive Least Load Ratio (ALLR) Algorithm***

The ALLR load balancing algorithm include two modules: the server monitoring model (SMM) and the adaptive least load ratio compute module (ALLRComputeM). The SMM operates on the top of the POX to collect OpenFlow switch per-port counters statistics periodically. The SMM reports the statistics to the ALLRComputeM to perform advanced computations related to making the load balancing decision. The schematic diagram of ALLR algorithm has been illustrated in Figure (3.2).



Counter statistics Flow table	Counter statistics-Flow entry	Counter statistics-port
Active Entries	Received Packets	Received Packets
Packet Lookups	Received Bytes	Transmitted Packets
Packet Matches	Duration (Sec)	Received Bytes
	Duration (NanoSec)	Transmitted Bytes
		Receive Drops
		Collisions

Figure (3.2): ALLR Algorithm schematic Diagram

a) Adaptive Least Load Ratio Computation Module (ALLRComputeM)

This module performs the task of load balancing by relying on the server load ratio of each server member in the pool. The client requests demand a text-type pages with small sizes as they consume less bandwidth and avoid the need for server-side scripting (e.g., JSP, ASP, and PHP). In contrast, pages of big sizes containing images may require additional resources that cause extra time and consume high bandwidth. The ALLRComputeM performs the following tasks:

1- Computing the Load of Servers

The clients send the same type of requests to the server pool, which provides the same type of service. The work assumes a single pool (P) allocated with a single service type (HTTP). The pool consists of server members that hold different load, L= (L (S_1), L (S_2)….L (S_N)).The client send the request R_i to the VIP where R is the set of requests that need to be scheduled, R= (R_1, R_2……R_N).

The load of a pool of server members is calculated according to equation (3.1):

$$L(P_{SALLR}) = \sum_{i=0}^{n} L(S_i) \quad \ldots\ldots (3.1)$$

The load of each server member in the pool is calculated according to equation (3.2):

$$L[s] = \sum_{i=0}^{n} B(S_i)/W(S_i) \quad \ldots\ldots (3.2)$$

In equation (3.2), the B (S_i) refers to the utilized bandwidth of

every server member in the pool calculated by equation (3.3)

$$B(S_i) = \sum_{t=1}^{5} Received_{Bytes} + Transmit_{Bytes} \quad \ldots\ldots\ldots\ldots\ldots (3.3)$$

The $Received_{bytes}$ Represents the received bytes and $Transmit_{Bytes}$ represents the transmitted bytes from OpenFlow switch ports reported uniformly and periodically from SMM. The addition of both counters during the TimeInterval (assumed to be 5 seconds) passed between each counter value's points represents the utilized bandwidth. The TimeInterval to collect link information is assumed to be 5 seconds in order to bypass an overhead condition for the POX controller. $W(S_i)$ Is defined as the static weight assigned by the administrator for every server member in the pool. Weights generally reflect the capabilities of each server to process tasks, like server CPU and RAM. The list of IP servers is traversed only once, and the time consumed by this algorithm is O (n).

2- <u>Selecting the Optimal Server</u>

According to equation (3.3) and (3.2), the server with the least load is selected to handle the incoming clients' requests. According to equation (3.4):

$$S_m = Min[L(S_i)] \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.4)$$

S_m Represents the server member with the least load among other server members in the pool. Therefore, every time server's load changes, the information is updated. The ALLRComputeM has been represented by algorithm (3.5).

Algorithm (3.5) : Adaptive Least Load Ratio Computation Module (ALLRComputeM)

Input:

- ServersIDArray = {S1, S2, …, Sn}
- ServersWeightArray = {ServerWeight (1), ServerWeight (2),…, ServerWeight(n)}
- LoadServersIDArray = {L1, L2……Ln}

Output:

- Selects the Best Server to Handle the Incoming Request

Process:

```
1.  SelectedServerNo:Integer
      /*declare the SelectedServerNo variable that refer to the number of server that handles  the
       incoming client request*/
2.   TimeInterval:integer
     /*declare the time interval variable for running the ALLRComputeM*/
3.  Type1: SwitchStatisticsType
4.     Record PerPort
       /*define per-port counters statistics record fields of the OVS OpenFlow switch*/
5.     ReceiveBytes: integer
       /*declare the ReceiveBytes variable that refer to the received bytes from the each server  by
       the OVS OpenFlow switch via OVS OpenFlow switch port*/
6.     TransmitBytes: integer
        /*declare the TransmitBytes variable that refer to the transmit bytes  from the OVS
        OpenFlow  switch via OVS OpenFlow switch port  to each server*/
7.     ReceivePackets: integer
       /*declare the ReceivePackets variable that refer to the received packets from each server  by
       the OVS OpenFlow switch via OVS OpenFlow switch port*/
8.     TransmitPackets: integer
       /*declare the TransmitBytes variable that refer to the transmit bytes  from the OVS OpenFlow
       switch via OVS OpenFlow switch port  to each server*/
9.     End record
10. End Type1
11.  ServerStatistics: ServerStatisticsType
12.  Type2
13.    ServerWeight:integer
       /* declare the ServerWeight variable that refer to weight of every server member in pool*/
```

14. NoOfServers:integer
    /* declare the NoOfServers variable that refer to the no. of server members in the pool*/
15. LoadServer:integer
    /*declare the LoadServer variable that refer to the load value for every server member in the pool periodically every 5 seconds */
16. UtilizedBand:float
    /*declare UtilizedBand variable that refer to utilized bandwidth for every server member in the pool periodically every 5 seconds*/
18 End Type2
19. ServerInfo:Type2
20. Call ServerMonitoringModule (SwitchStatistics)
    /* Call algorithm (3.6) that collect per-port counters statistics of the OVS OpenFlow switch periodically every 5 seconds*/
21. For i⟵ 1 to NoOfServers
22. Allocate ServerInfo.ServerWeight(Si)
    /*Assign weight for every server member in the pool by the administrator*/
23. End for
24. For i⟵ 1 to NoOfServers
25. ServerInfo.UtilizedBand(Si) ⟵ $\sum_{i=1}^{No.of.Servers}$ $(Received_{bytes}) + (Transmit_{Bytes})$ from SMM
    /*calculate the utilized bandwidth for every server member in the pool by summing the received bytes and transmit bytes periodically based on reports uploaded from SMM*/
26. End for
27. For i ⟵1 to NoOfServers
28. ServerInfo.LoadServer(Si) ⟵ ServerInfo.UtilizedBand(Si)/ ServerInfo.ServerWeight(Si)
    /*calculate the load for every server member in the pool by dividing the utilized bandwidth for every server member and the weight of every server member periodically*/
29. End for
30. SelectedServNo ⟵ FindMin(ServerInfo.LoadServer(Si))
    /*compare the load values for every server member in the pool and select the least value to be the server the handles the incoming client request*/
31. End

b) Server Monitoring Module (SMM)

The health of the server member has always been considered as crucial part of the load balancing mechanisms. Thus, there is an urgent need to report server metrics such as the bandwidth utilization to the POX controller periodically to perform further computations. Without this feedback statistics, load balancing functionality may operate inappropriately and send falsified requests to connect with overloaded devices. The bandwidth is defined as measuring how much data could be sent and received through a link and is measured in bits, megabits, and gigabits per second. There are two approaches for the SMM to collect the per-port counters statistics: approach one frequently reports the per-port counters statistics to ensure that the utilized bandwidth in ALLRComputeM is calculated within real-time. Approach two reports per-port counters statistics less frequently to avoid networking overhead. However, approach one leads to more errors due to the occurrence of timestamps. Approach two reduces overhead but is less accurate. The solution is to delegate the selection of the time interval for collecting statistics to the administrator. Regarding the ALLR algorithm, it is agreed to assume the time interval equal to 5 seconds. The SMM module executes a runnable class, running a data collection function at the interval set to 5 seconds by passing out OVS port number (OpenFlowPortNo.).The OpenFlowPortNo guarantees the restoration of the entire set of counters statistics per-port of the OVS OpenFlow switch such as the received packets, transmit packets, received bytes, transmit bytes, and

the collisions. The SMM is integrated into the ALLRComputeM by implementing a thread to handle the request response statistics. The SMM has been demonstrated according to algorithm (3.6).

Algorithm (3.6): ServerMonitoringModule (SMM)

Input:

- OVS OpenFlowPortNo.//OpenFlow switch Port Number
- TimeInterval//time adopted to collect counters statistics per-port of the OpenFlow switch (5 seconds)

Output:

- OVS OpenFlow switch per-port counters statistics periodically (5 seconds)

Process:

```
1. Type: SwitchStatisticsType
2.     Record Per Port
         /*define per-port counters statistics record fields of the OVS OpenFlow switch*/
3.       Received Bytes: integer
          /*declare the ReceiveBytes variable that refer to the received bytes from the each server by
          the OVS OpenFlow switch via OVS OpenFlow switch port*/
4.       Transmit Bytes: integer
          /*declare the TransmitBytes variable that refer to the transmit bytes from the OVS OpenFlow
          switch via OVS OpenFlow switch port to each server*/
5.       Receive Packets: integer
          /*declare the ReceivePackets variable that refer to the received packets from each server by
         the OVS OpenFlow switch via OVS OpenFlow switch port*/
6.       Transmit Packets: integer
          /*declare the TransmitBytes variable that refer to the transmit bytes from the OVS OpenFlow
         switch via OVS OpenFlow switch port to each server*/
7.     End Record
7. SwitchStatisticsM: SwitchStatisticsType
8. Collect SwitchStatisticsM.Port for Every Server Connected to the OpenFlow Switch using
   OpenFlowPortNo.
```

```
    /* collect the per-port counters statistics of OVS OpenFlow switch periodically every 5 seconds*/
10. Update SwitchStatisticsM.FlowTable, SwitchStatisticsM.FlowEntry, SwitchStatisticsM.Port
11.  Sleep (TimeInterval)
    /* the TimeInterval is 5 seconds*/
12. End
```

c) ALLR Flow-Sequence Diagram

This section introduces the flow-sequence diagram of the ALLR load balancing algorithm, which identifies all the steps involved in the ALLR algorithm represented in figure (3.3).

1- System Configuration

The server pool, the VIP of the pool, and the traffic type should be initially configured by the administrator. However, the configuration of the server pool is based on the number of server members. The proposed system uses a single pool due to the fact that single service and a single traffic type is provided by the server members.

2- Addition of Server Members to the Pool

The HTTP servers (HTTP Server1, HTTP Server2, and HTTP Server3) are launched into the pool based on their layer three static IP address associated and the VIP port number.

3- Generating Requests

The three distinguishing characteristics of httperf tool include the generation and sustain of server overload, the support for the HTTP/1.1 besides SSL protocols, and the extensibility to new workload generators and performance measurements. the httperf tool is used on the client machine to generate workload requests represented by a total number of connections set to 100, and a number of client requests from 0 up to 180 (req/sec) with a uniform increase of 20 (req/sec) per connection .The aim is to evaluate the HTTP server members' performance in terms of server average throughput, server reply time, server connection time, server connection rate, server queue length and server CPU capacity.

4- Receive and Process Packet_in Messages

The flow table of the OpenFlow switch is initially empty and contains one single action known as the controller that sends any packet to the controller. However, before sending the Packet_in message, all the clients and servers must be notified by the MAC address of the VIP, which may be obtainable through the handshaking process by way of ofp_switch_features_ request and Ofp_switch_features_ reply messages.

5- Load Balancing Mechanism

When the SMM reports the per-port counter statistics periodically every 5 seconds, the ALLRComputeM carries out the

computation of the least load server member to handle the incoming client requests according to equation (3.4).

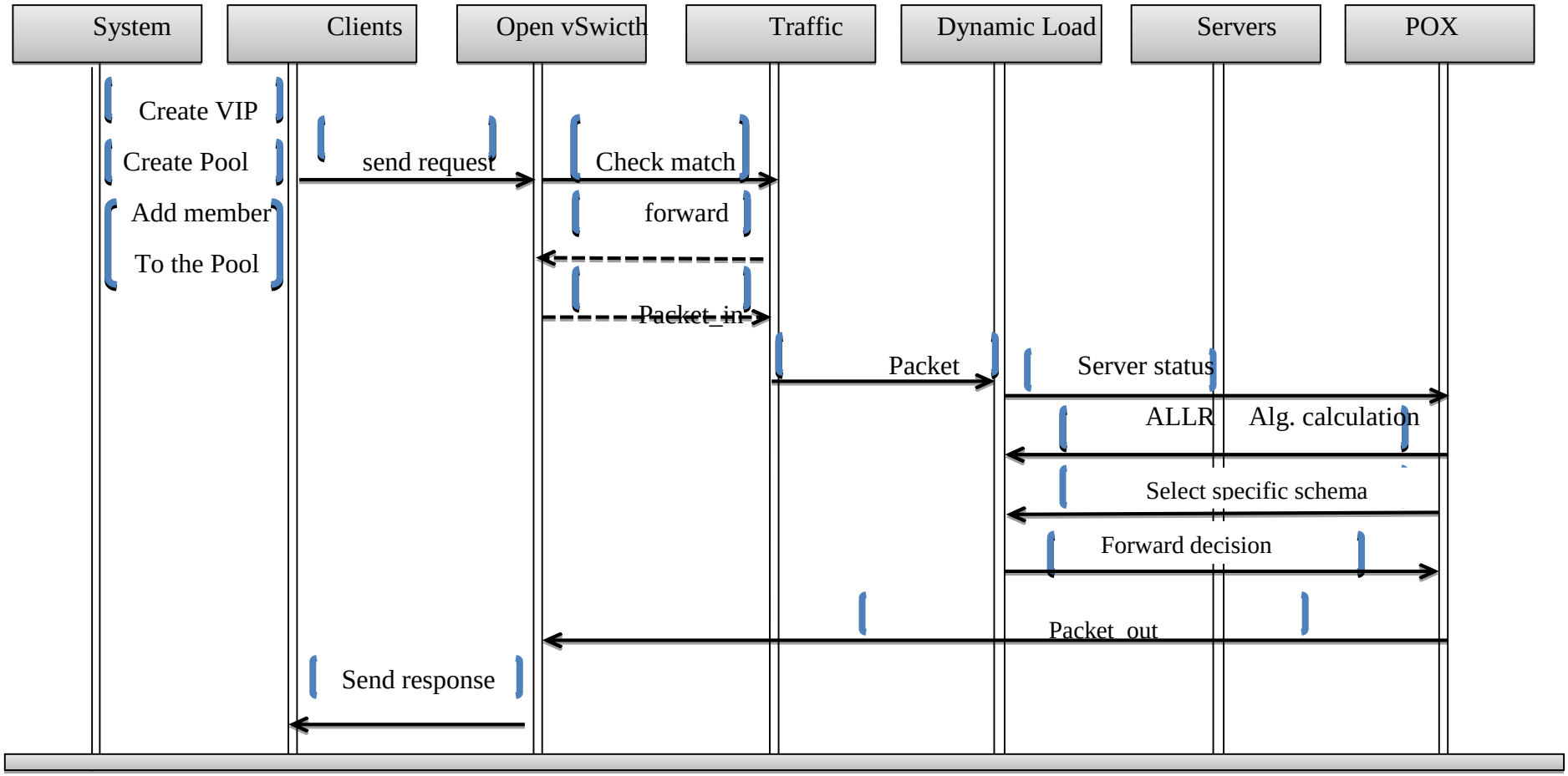
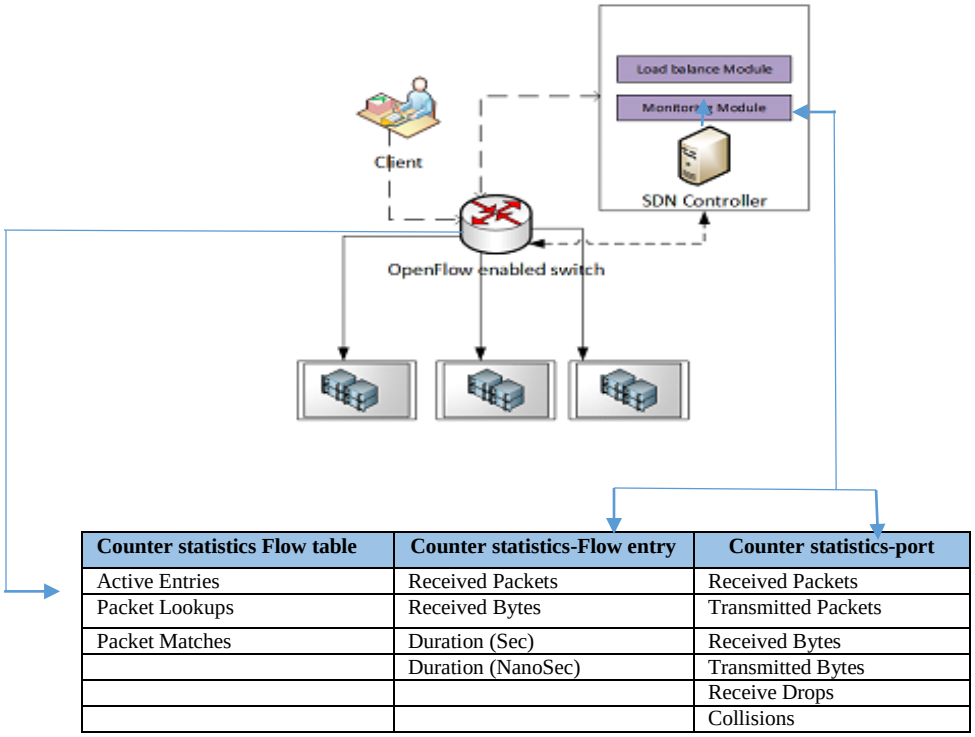


Figure (3.3): ALLR Algorithm Flow Sequence Diagram

3.4.3 ***The Hybrid-Based (HB) Algorithm***

The HB load balancing algorithm includes two functional models: the Advanced Server Monitoring Module (ASMM) and the Hybrid Load Balancing Module (HLBM). The modules usually mount on the top of the POX controller. The ASMM tracks per-port and per-flow counters statistics of the OVS OpenFlow switch and forwards it periodically to the HLBM. The HLBM is considered the core module responsible for making the load balancing decisions. The schematic diagram of HB algorithm is illustrated in Figure (3.4).



Counter statistics Flow table	Counter statistics-Flow entry	Counter statistics-port
Active Entries	Received Packets	Received Packets
Packet Lookups	Received Bytes	Transmitted Packets
Packet Matches	Duration (Sec)	Received Bytes
	Duration (NanoSec)	Transmitted Bytes
		Receive Drops
		Collisions

Figure (3.4): HB Algorithm schematic Diagram

a) Hybrid-based Load Balancing Module (HLBM)

The HLBM is responsible of carrying out load balancing decision by relying on the server engagement factor for each server member in the pool. Similar to what has been covered in the ALLR load balancing algorithm, The client requests demand a text-type pages with small sizes as they consume less bandwidth and avoid the need for server-side scripting (e.g., JSP, ASP, and PHP). In contrast, pages of big sizes containing images may require additional resources that cause extra time and consume high bandwidth. The ALLRComputeM performs the following tasks:

1- Computation of Server Members Load

The clients send the same type of requests to the server pool, which provides the same type of service. The work assumes a single pool (P) allocated with a single service type (HTTP). The pool consists of server members that hold different load, L= (L (S_1), L (S_2)….L (S_N)).The client send the request R_i to the VIP where R is the set of requests that need to be scheduled, R= (R_1, R_2……R_N). The load of a pool of server members is calculated according to equation (3.5):

$$L(P_{SHB}) = \sum_{i=0}^{n} L(S_i) \quad \text{(3.5)}$$

The load of each server member is calculated based on determining the active number of connections for each server member in the pool. The active number of connections usually

refers to the number of client requests being served by the server member. The server member with least number of connections S_m is selected according to equation (3.6):

$$S_m = \text{Min}[\text{Conn.} \sum_{i=1}^{n} (S_i)] \quad (3.6)$$

Then, the server engagement ratio for each server member is calculated according to equation (3.7):

$$\text{ServerEngagementRation}[S] = \sum_{i=1}^{n} [(S_m + \text{ActiveNoOfConn}\ (S_i))/W(S_i)] \quad (3.7)$$

$W(S_i)$: refers to the static weight to each server member assigned by the administrators.

2- Selecting the Optimal Server

The server member with the least value of the engagement factor $S_{optimal}$ is selected to handle the incoming clients' requests according to equation (3.8):

$$S_{optimal} = \text{Min}\ (\text{EngagementFactor}[S]) \quad (3.8)$$

The HLBM has been presented by algorithm (3.7).

Algorithm (3.7) : Hybrid-based load balancing Module (HLBM)
Input:
- ServersIDArray = {S1, S2 …Sn}
- ServersWeightArray = {ServerWeight (1), Server-Weight (2),…, ServerWeight(n)}
Output:
- The Best Server to Handle the Incoming Request Among a Pool of n Server members
Process:
1. LoadRatio: float
/*declare the LoadRatio variable that refer to load value for every server member in the pool*/

```
2.  SelectedServNo: integer
    /*declare the  SelectedServNo variable that refer to the number of the selected server member to
    handle the incoming client requests*/
3.  NoOfServers: integer
    /* declare the NoOfServers variable that refers to the number of server members in the pool*/
4. Type1: Type
5.      ServerNo.: integer
        /*declare the  SelectedServNo variable that refer to the number of the server member in pool*/
6.      ServerNoOfConnections: integer
        /*declare the  ServerNoOfConnections variable that refers to the number of active connections
        for each server member in the pool*/
7.      ServerWeight: integer
        /*declare the  ServerWeight variable that refers to the static weight assigned for each server
        member in   the pool*/
8. End Type1
9.   Servers: Array[No.of.Servers] of Type1
10. ServersLoadRatio: Array[NoOfServers]
11. Type2: SwitchStatisticsType
12.     Record Per Flow Entry
            /* define per-flow counters statistics record fields of the OVS OpenFlow switch*/
13.         ReceivedPackets: integer
            /*declare the Receivedcounters variable that refer to the number matched received packets by
            the OVS OpenFlow switch*/
14.         ReceivedCounters: integer
            /*declare the Receivedcounters variable that refer to the count of packets that match the
             flow entry of the OVS OpenFlow switch*/
15.         Time: time
16.     End Record
17.     Record Per Port
            /*define per-port counters statistics record fields of the OVS OpenFlow switch*/
```

```
18.     ReceivedBytes
        /* declare the ReceiveBytes variable that refer to the received bytes from the each server by
        the OVS OpenFlow switch via OVS OpenFlow switch port*/
19.     TransmitBytes
        /*declare the TransmitBytes variable that refer to the transmit bytes from the OVS
        OpenFlow switch via OVS OpenFlow switch port to each server*/
20.     ReceivedPackets
        /*declare the ReceivePackets variable that refer to the received packets from the each
        server by the OVS OpenFlow switch via OVS OpenFlow switch port*/
21.     TransmitPackets
        /*declare the TransmitPackets variable that refer to the transmit packets from the OVS
        OpenFlow switch via OVS OpenFlow switch port to each server*/
22.   End Record
23. End Type2
24. SwitchStatistics: SwitchStatisticsType
25. Call AdvancedServerMonitoringModule (SwitchStatistics)
    /* Call algorithm (3.7) that collect per-port counters statistics of the OVS OpenFlow switch
    periodically every 5 seconds*/
26. For i ← 1 to NoOfServers
27.   Call FindServerMinConn (NoOfServers, Servers, Var SelectedMinServerConn.)
      /*Call function FindServerMinConn to select the server member with least no. of connections
      namely SelectedMinServerConn.*/
28.   MinConnServer ← SelectedMinServerConn
      /* assign the value of SelectedMinServerConn. To the MinConnServer*/
29. End For
30. For i ← 1 to NoOfServers
31.   LoadRatio[i] ← Servers[i].ServerNoOfConnections + MinConnServer /
                                                    Servers[i].ServerWeight
      /*calculation of the load value represented by the server engagement factor of each server
      member in the pool*/
```

```
32. End for
33. SelectedServNo ← FindMin (Servers[i].LoadRatio)
    /*select the server with least value of server engagement factor */
34. forward the request to (SelectedServNo)
35. Function FindServerMinConn (N, Servers, Var SelectedMinServerConn)
    /* function to calculate the number of active connections for each server member in the pool and
    selecting the server member with the least no. of connections*/
36. n← NoOfServers
    /*declare the number of server members in the pool*/
37. ServerMinConn ← 0
38. For m ← 0 to n-1
39.     if Servers[m].Conn < ServerMinConn
40.         Then ServerMinConn ← Servers[m].conn
41.     End if
42. End for
43. Return ServerMinConn
44. End
```

b) Advanced Server Monitoring Module (ASMM)

This module is an advanced version of the server monitoring module (SMM) of the ALLR load balancing algorithm. The difference lies in expanding the scope of this module to include ports, flow entries, of the OVS OpenFlow switch counters statistics. The OVS OpenFlow switch operating under OpenFlow protocol provides two types of counters statistics: The first type is the statistics per flow entry in a flow table that consists of the received bytes, received packets, and duration. The second type is the statistics per port

consisting of received packets, transmitted packets, received bytes, transmitted bytes. The ASMM executes a runnable class, which involve the running of a data collection function according to time interval set to 5 seconds by passing out two parameters: the first parameter is the OVS port number (OpenFlowPortNo.) that is used to retrieve the counters statistics related to OpenFlow switch ports such as the received packets, transmitted packets, received bytes, transmitted bytes, received drops, transmitted drops, and the collisions. The second parameter is the Flow entry number (FlowEntryNo.) that is used to retrieve the counters statistics related to flow entries such as received bytes, received counters, time and OpenFlow table statistics such as active entries, packet lookups, and packet matches. The ASMM is incorporated into the HLB by using a thread that handles the request and response statistics. The ASMM has been introduced by algorithm (3.8).

Algorithm (3.8): AdvancedServerMonitoringModule (ASMM)

Input:

-OpenFlowPortNo. //OpenFlow Switch with Port Number

- OpenFlowEntryNo. //OpenFlow Switch Flow Entry Number

-timeInterval //Time Rate to Collect OpenFlow Switch traffic information periodically

Output:

-Collect per-port and per-entry counters statistics of OVS OpenFlow Switch periodically Every 5 Seconds

Process:

```
1. ┌─ Record Per FlowEntry
   │    /*declare per-flow counters statistics record fields of the OVS OpenFlow switch*/
2. │    ReceivedPackets: integer
```

```
    /* declare the Receivedcounters variable that refer to the number matched received packets
    by the OVS OpenFlow switch*/
3.  Receivedcounters: integer
    /*declare the Receivedcounters variable that refer to the count of packets that match the flow
    entry of the OVS OpenFlow switch*/
4.  Time: time
5. End Records
6. Record PerPort
    /*define per-port counters statistics record fields of the OVS OpenFlow switch*/
7.  ReceivedPackets: integer
    /*declare the ReceivePackets variable that refer to the received packets from the each server
    by the OVS OpenFlow switch via OVS OpenFlow switch port*/
8.  TransmitPackets: integer
    /*declare the TransmitPackets variable that refer to the transmit packets from the OVS
    OpenFlow switch via OVS OpenFlow switch port to each server*/
9.  ReceivedBytes: integer
    /*declare the TransmitBytes variable that refer to the transmit bytes from the OVS OpenFlow
    switch via OVS OpenFlow switch port to each server*/
10. Transmit Packets: integer
    /*declare the TransmitPackets variable that refer to the transmit packets from the OVS
    OpenFlow switch via OVS OpenFlow switch port to each server*/
11. End record
12. SwitchStatisticsM: SwitchStatisticsType
13. Collect SwitchStatisticsM.Port for Server Connected to OpenFlow Switch using Switch
    OpenFlowPortNo.
14. Collect SwitchStatisticsM.FlowEntry for Server Connected to OpenFlow Switch using Switch
    OpenFlowEntryNo
15. Update SwitchStatisticsM.FlowTable, SwitchStatisticsM.FlowEntry, SwitchStatisticsM.Port
16. Sleep (TimeInterval )
17. End
```

c) HB Flow-Sequence Diagram

Introduces the flow-sequence diagram of the HB load balancing algorithm. The diagram exhibits all the steps and operations involved in load balancing Similar to what has been suggested for ALLR algorithm in section 3.4.2

3.5 *Summary*

This chapter presents the dissertation methodology that involves two phases: The first phase evaluates the performance of POX controller based on the implementation of several aspects and techniques. The second phase witnesses the proposition of two novel load balancing algorithms, the ALLR and HB. The ALLR algorithm adopts a data-collection module that acts as an agent to collect raw byte counters statistics inside OpenFlow switches used later in further computations related to load balance. The HB algorithm similarly adopts an advanced data collection module that acts as an advanced agent to collect all of the raw byte counters statistics inside OpenFlow switches.

4.1 Introduction

In this chapter, the evaluation of the proposed model architecture is introduced by performing experiments using different types of network topologies. Special tools for testing the network performance such as iperf, D-ITG, Cbench, httperf, OpenLoad have been utilized to clarify how the ALLR and HB algorithm could develop resources utilization by reducing the latency and improving throughput.

4.2 Simulation Setup

Several tools (hardware and software) are used for testing proposed model architecture as shown in table 4.1 model

Table 4.1: Parameter Values of Emulation

Tool	***Function***	***Version***
Mininet	Networking Emulator	2.2.1
OVS Switch	OpenFlow Virtual switch	2.3
POX	SDN- controller	1.0
Oracle VM virtual box	Software virtualization system	6.0
Linux	O.S	14.04
Python	Programming language	2.7.6
Processor	Intel Corei7	
RAM	8 Gbyte	

4.3 SDN POX Controller Performance Evaluation

QoS metrics Offer a way to mark traffic prioritization over networks to guarantee a certain level of performance. QoS also makes sure that routing priorities do not change.

4.3.1 Impact of QoS Metrics-Utilized Bandwidth

Utilized bandwidth is the maximum data rate an OpenFlow switch link could transfer. The utilized bandwidth is useful in monitoring bandwidth usage to provide information for the administrators whether they have adequate bandwidth to fit the needs of their applications. The utilized bandwidth may be affected by bandwidth management. The bandwidth management follows two techniques for bandwidth allocation: policing and queuing. Policing is out of the study scope. The queuing technique assumes three values for bandwidth allocation: 100 Mbps for slow applications, 300 Mbps for medium applications, and 500Mbps for critical applications. Referring to the applications provided by the Apache HTTP webservers, the first and second values are chosen and the third is excluded; 500 Mbps.The L2_learning switch module is the default application module invoked by the POX controller with the assumption of no delay in links that connect the OpenFlow switches with the hosts and with the POX controller. The evaluation process is done using the iperf network benchmarking tool.Based on the results shown in the figure (4.1) as the allocated bandwidth is between 10 Mbps and 30 Mbps, the utilized bandwidth is minimum and constant. When the allocated bandwidth reaches the boundaries of 40 up to 50 Mbps, no observed increment in the utilized bandwidth is noticed. The paradigm shift occurs from 100 up to 300 Mbps of the allocated bandwidth to record a clear and effective utilized bandwidth up to 85.5, 176, and 256 Mbps of the allocated bandwidth. As the allocated bandwidth increases,

the data flow over a fixed interval increases, which causes a consistent growth in the utilized bandwidth. The parameter values for the evaluation process is explained in table (4.2).

Table 4.2: Parameter Values of QoS Metrics –Utilized Bandwidth

Parameter	***Value***
SDN-Based Platform Controller	POX
Module invoked by the controller	L2_learning switch
POX loopback address	127.0.0.1
SDN-Based platform Virtual Switch	Open vSwicth (OVS)
VIP (Virtual IP)	10.0.1.1
Southbound Protocol	OpenFlow Version 1.0
Allocated Bandwidth Range	1-300 Mbps
Listening Port B/W POX and OVS	6633
Port B/W the Sender and the Receiver	5001
Tool	iperf
Link Delay	No
Allocated Bandwidth for each link	1Mbps (default)
No. of Apache HTTP Webservers	3
No. of Clients	9
Link Delay	No

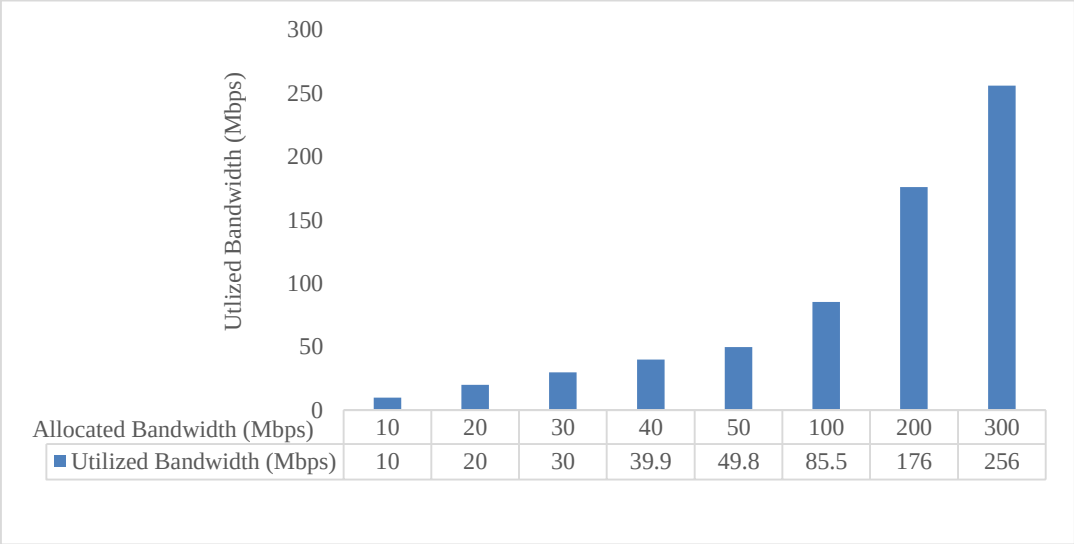


Figure (4.1): Utilized Bandwidth in OpenFlow Switches vs. Allocated Bandwidth

4.3.2 Impact of QoS Metrics- Dropped Packets

Dropped Packets metric is defined as the measure of packets dropped by nodes for different reasons and plays an important role in the computation of delivered packets, network reliability, and network throughput. Dropped Packets is demanded by the network administrators and internet service providers (ISP's) to identify network clusters prone to congestion. One of the major causes of dropped packets is link congestion. In some cases, even if the link can technically handle the amount of traffic reaching it, it has been configured to drop packets after a certain limit. Another cause of packet loss similar to network congestion is Over-Utilized devices. This means that a device is operating at a capacity it was not designed for. In a network, packets may arrive faster than they can be processed/sent out. To handle this type of situation, many devices have buffers where they hold packets

temporarily until they are able to be processed and sent out. However, in the case of an Over-Utilized device, the buffer will probably fill up quickly, resulting in excess packets being dropped. Another cause of dropped packets is Faulty hardware. This could be a component of a device or the whole device itself. The effects of packet loss vary depending on the protocol/application concerned. TCP is generally designed to handle dropped packets because of the acknowledgment and re-transmission of packets – if a packet gets lost (i.e. no acknowledgment is received for that packet), it will usually be re-transmitted. UDP, on the other hand, does not have inbuilt re-transmission capability and may not handle packet loss as well. However, irrespective of the protocol/application, too much drop of packets is definitely a problem. According to figure (4.2), as the packet size is from 64 up to 1536-byte. Then, at 128 bytes, there is a sharp increase in the dropped packets, which turn out to be moderate at 384 and 924 bytes. However, there is no significant difference regarding dropped packets at 192, 256, 512, and 768 bytes. At 1536, the dropped packets reaches another rise due to the fact that packets with sizes close to the value defined by the maximum transmission unit (MTU) of the TCP protocol (1500 bytes) get fragmented, re-transmitted and eventually a rise in dropped packets. The parameter values for the performance evaluation process is described in table (4.3)

Table 4.3: Parameter Values of QoS –Dropped Packets

Parameter	***Value***
SDN-Based Platform Controller	POX
Module invoked by the controller	L2_learning switch (default)
VIP (Virtual IP)	10.0.1.1
SDN-Based platform Virtual Switch	Open vSwicth (OVS)
Link delay	No
Port B/W POX and OVS	6633(default)
Packet Size Range	(0-1500) Byte
No. of Transite Packet	1000
Tool	D-ITG (Single-Mode)
Link Delay	No
No. of Apache HTTP Webservers	3
No. of clients	9

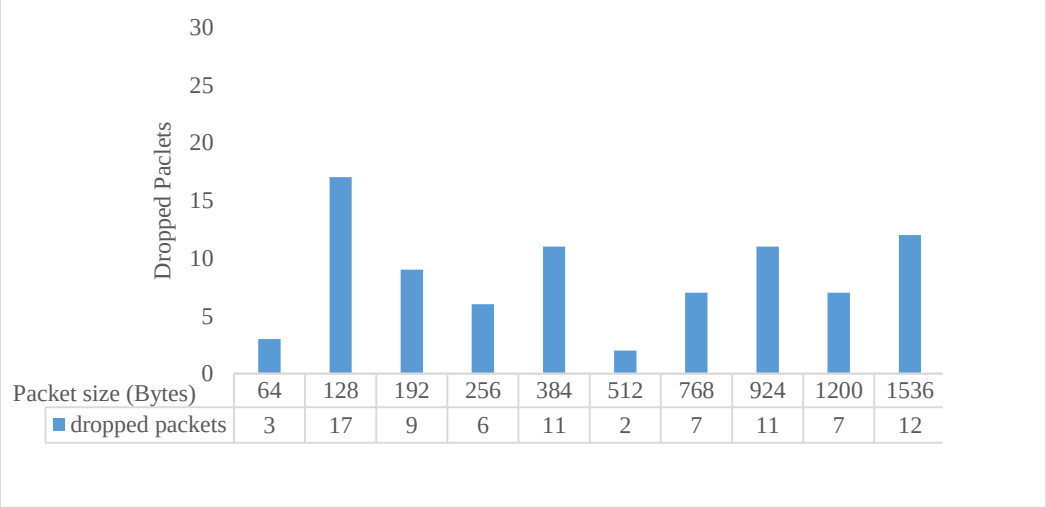


Figure (4.2): Dropped Packets by Hosts vs. Packet Size

4.3.3 Impact of Multiple OpenFlow Switch Technique on POX

The multiple OpenFlow switches technique aims to examine the ability of the POX controller to process the Packet_in message flows that do not have an entry in the OpenFlow switches. This procedure acquires the necessary full knowledge of SDN components that assist in proposing and implementing applications**.** The parameter values for case designs1, 2, 3, and 4 is explained in table 4.4.

Table 4.4: Parameter Values of Multiple OpenFlow Switches Technique

Parameter	***Value***
SDN-Based Platform Controller	POX
Module invoked by the controller	L2_learning switch (default)
Loopback IP	127.0.0.1
SDN-Based platform Virtual Switch	Open vSwicth(OVS)
Case Design 1	1-OVS, 1-Host
Case Design 2	4-OVS, 4-Host
Case Design 3	16-OVS, 16-Host
Case Design 4	64-OVS, 64-Host
Link delay in Case designs 1,2,3,4	No
Listening Port	6633
Tool	Cbench

Results from figure (4.3) indicate that increasing the number of OpenFlow switches from 1 to 4, 16, and then to 64 with the aid of

utilizing the (forwarding.l2_learning) application module by the POX controller leads to a significant decline in POX controller's average throughput that reflects the ability to handle the incoming flows from the OpenFlow switches from 4300 (flows/sec) to 2650, then to 1730 and ends at 0.001 (flows/sec). In a similar behavior results from Figure (4.4) reveal that increasing the number of OpenFlow switches from 1 to 4, 16, and then to 64 while using the default application module of the POX controller (forwarding. L2_learning) leads to a significant increase in the average latency of the POX controller from 0.19 to 1.5, then to 2 and ends up to 2.15 msec. The gradual drop in the POX controller average throughput with the corresponding steady increase in average latency is a result of the reduction in the number of (Flow_Mod) messages sent by the POX controller as the buffer is filled with (Packet_in) messages that have not been processed. Therefore, the time interval between sending the packet (Packet_in message) and waiting for the (Flow_Mod) response messages increases resulting in the degradation of POX controller performance. Multiple controller SDN architectures will be also efficient to manage larger networks, decrease transmission loss and avoid fault tolerance. As the network expands the load on the single controller increases, so it becomes difficult for single controller architecture to efficiently manage the network. Multiple controllers do not help to balance the load between them, it will tackle the network congestion issue by distributing the load between them.

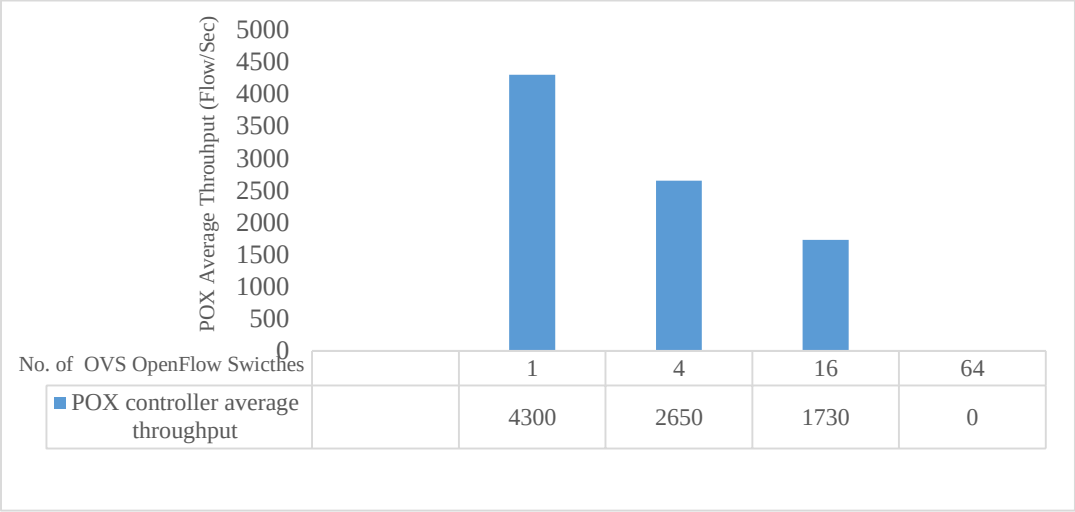


Figure (4.3): Average Throughput of POX according to No. of OpenFlow Switches

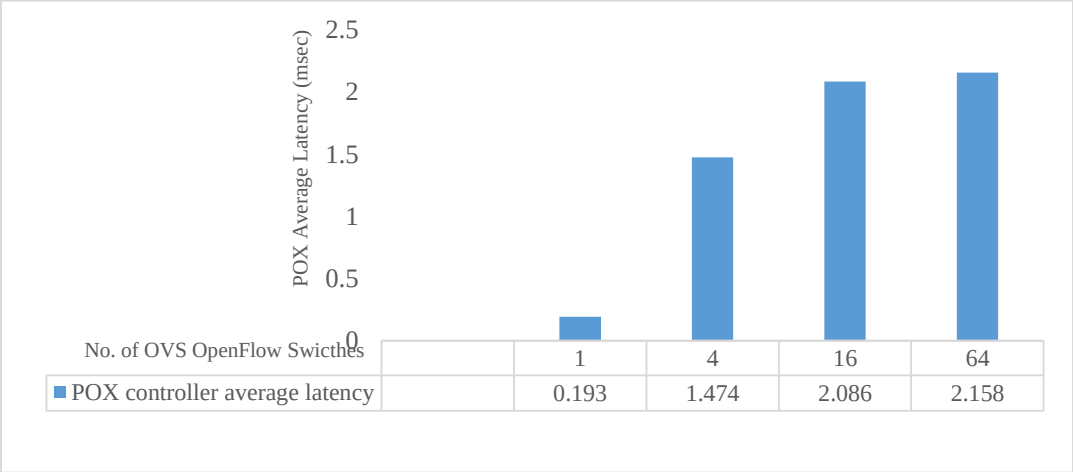


Figure (4.4): Average Latency of POX vs. No. of OpenFlow Switches

4.4 Proposed System Design

This section presents the core of the proposed system architecture that analyses and evaluates the performance of the ALLR and HB algorithms.

4.4.1 Preliminary Performance Evaluation of Load Balancing Algorithms

This section is prior to the introduction of the proposed load balancing algorithms. It aims to conduct a preliminary analysis for a set of different pattern of working load balancing algorithms in SDN to select the algorithms that will be compared with ALLR and HB to evaluate the performance through certain specifications and conditions. By referring to table (4.8) all the parameters in the experimental analysis are constant except the parameter that relates to the number of client requests per second that starts from 0 up to 180 (request/second) with a gradual increase of 20 (request/second) per connection. However, the network performance may be checked through specific metrics, and the throughput is one of them. The throughput is defined as the amount of transferred data over the network in a particular period of time. Moreover, the server average throughput is one of the metrics used to mitigate the congestion control and ensure systems quality of service (QoS).

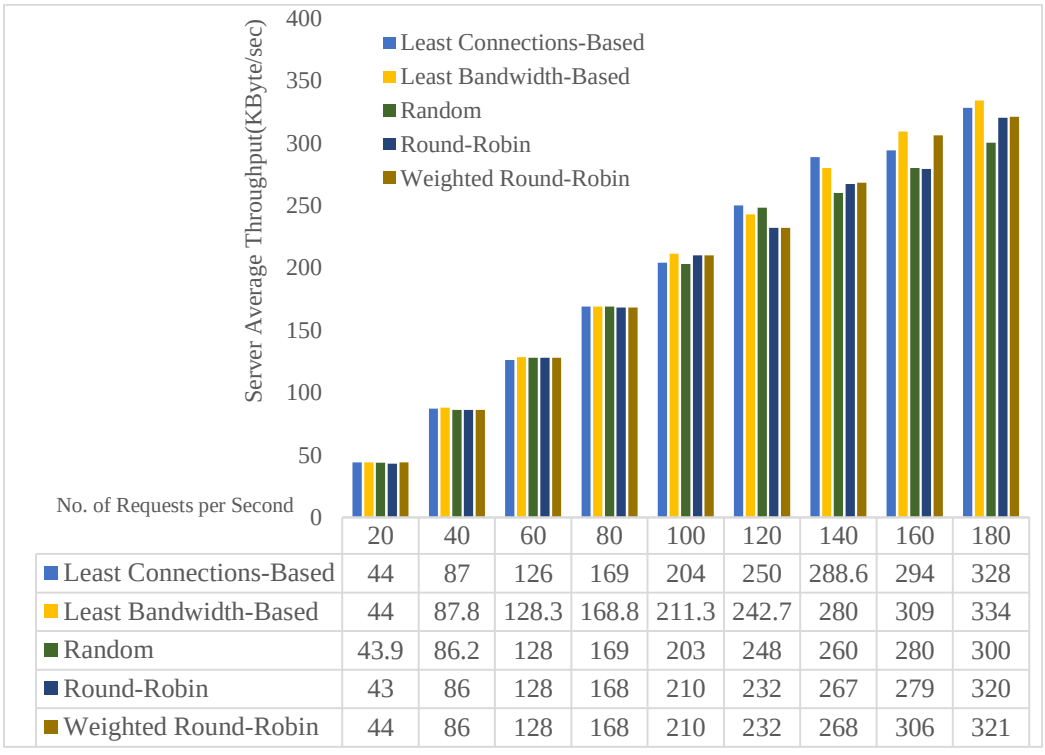
Table 4.5: Parameters of Load Balancing Algorithms Running on POX

Parameter	***Value***
tool	httperf
No. of HTTP web servers	3
Client Requests Range	(0-180) request/second
No. of HTTP clients	9
SDN-based Platform Controller	POX

Virtual SDN Switch	Open vSwicth (OVS)
Communication Port B/W POX & OVS	6633
HTTP web servers listening port	80
Virtual IP (VIP)	10.0.1.1
Send Buffer	4Kbyte
Receive Buffer	16Kbyte
Loopback Address	127.0.0.1
Algorithms invoked by POX	1- Random 2- Round-Robin 3- Weighted Round-Robin 4- Least Connection-Based 5- Least Bandwidth-Based

Outcomes in Figure (4.5) supports the previous studies related to this topic, which states that Dynamic load balancing algorithms outperform static ones. The dynamic least connection-based improves the server average throughput up to 4%, 3%, and 1% when client requests per second is (1-180) compared with static random, static round-robin, and static weighted round-robin respectively. Moreover, the dynamic least bandwidth-based algorithm improves the same metric up to 5%, 4%, and ?% when compared with the same static load balancing algorithms and number of client requests. The random algorithm records the worst performance as it leads to the overload of a single server while other servers are under-utilized. The static round-robin outperforms static random due to the equal division of requests among servers regardless of their capacity. However, it comes with no priority. Static weighted

round-robin occupies a good rank within the static balancing schemes as it solves the priority limitation of round-robin. Dynamic load balancing algorithms demonstrate an obvious improvement regarding performance compared to static ones. This goes back to the servers' continuous monitoring and selecting the best available server for directing client requests. Within the dynamic least connection-based allocation, the requests are directly forwarded to the server having the least number of connections. Whereas within the dynamic least bandwidth, requests are forwarded immediately towards the server, which utilized the lowest traffic among the other server members.



No. of Requests per Second	20	40	60	80	100	120	140	160	180
Least Connections-Based	44	87	126	169	204	250	288.6	294	328
Least Bandwidth-Based	44	87.8	128.3	168.8	211.3	242.7	280	309	334
Random	43.9	86.2	128	169	203	248	260	280	300
Round-Robin	43	86	128	168	210	232	267	279	320
Weighted Round-Robin	44	86	128	168	210	232	268	306	321

Figure (4.5): Server Average Throughput for static and dynamic Algorithms. vs. No. of Requests

4.4.2 The Adaptive Least Load Ratio (ALLR) Algorithm

This section aims to evaluate the performance of the ALLR load balancing algorithm compared with the same pattern of load balancing algorithms like least connection based and least bandwidth based load balancing algorithms running under the same conditions. Table 4.6 summarizes all of the parameters that have been involved in the evaluation procedure.

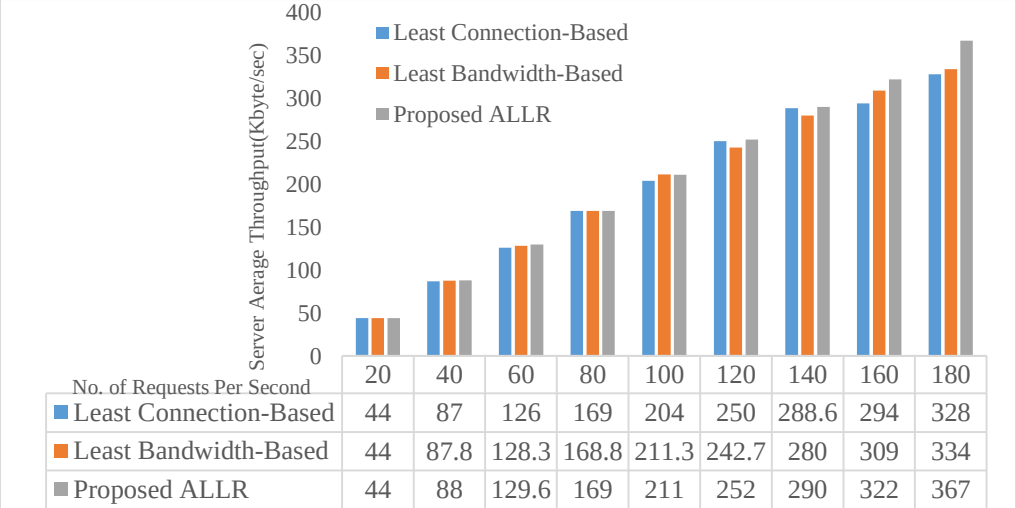
Table 4.6: Parameter Values of ALLR load balancing Algorithm

Parameter	***Value***
Benchmarking tool	httperf
No. of Apache HTTP web servers	3
No. of clients	9
Range of client requests	(0-180) request/second
SDN-based Platform Controller	POX
Virtual SDN Switch	OVS OpenFlow switch
Communication Port B/W POX & OVS	6633
HTTP web servers listening port	80
Virtual IP (VIP)	10.0.1.1
Loopback IP	127.0.0.1
	1- Least Connection-Based 2- Least Bandwidth-Based

Load balancing Algorithms invoked by POX	3-ALLR
send-buffer	4 Kbyte
receive-buffer	16 Kbyte

a) Server Average Throughput

The server Average throughput is the total payload over the entire session divided by the total time. Total time is calculated by taking the difference in timestamps between the first and last packet. Results in Figure (4.6) reveal that server average throughput of the three load balancing algorithms are barely distinguishable when the number of client requests is between 0 and 120 due to the server relaxation state. But, as the number of client requests starts to rise above 120 up to 140, 160, and 180, then the ALLR algorithm achieves a substantial improvement up to 7.25%.

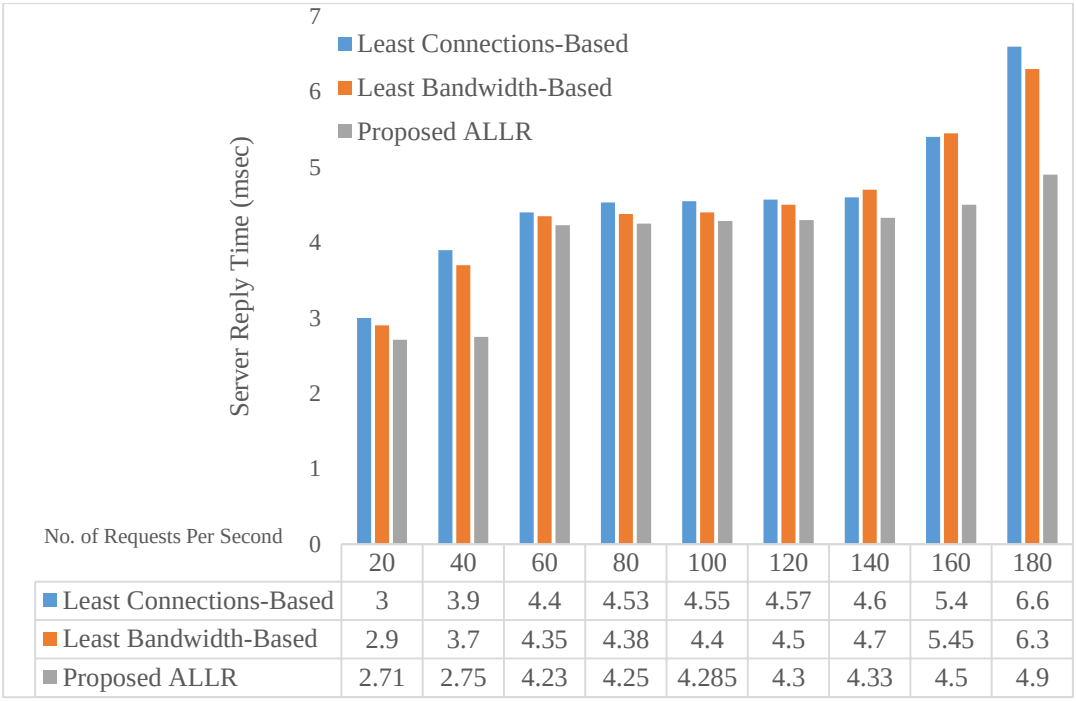


No. of Requests Per Second	20	40	60	80	100	120	140	160	180
Least Connection-Based	44	87	126	169	204	250	288.6	294	328
Least Bandwidth-Based	44	87.8	128.3	168.8	211.3	242.7	280	309	334
Proposed ALLR	44	88	129.6	169	211	252	290	322	367

Figure (4.6): Server Average Throughput of 3 Algorithms vs. No. of Requests per Sec

b) ***Server Reply Time***

The Server Reply Time, also referred to as server transfer time defined as the elapsed time between the first byte and the final byte of the response. Results in figure (4.7) reveal that as the number of client requests gradually rises from 0 to 120, then the server reply time of the three load balancing algorithms is nearby to each other. However, the situation changes as the number of client requests reaches 140 and above up to 160 and 180 (request/second) then ALLR algorithm improves the server reply time by reducing this metric to 19 % compared with dynamic least connection-based and dynamic least bandwidth –based algorithms.

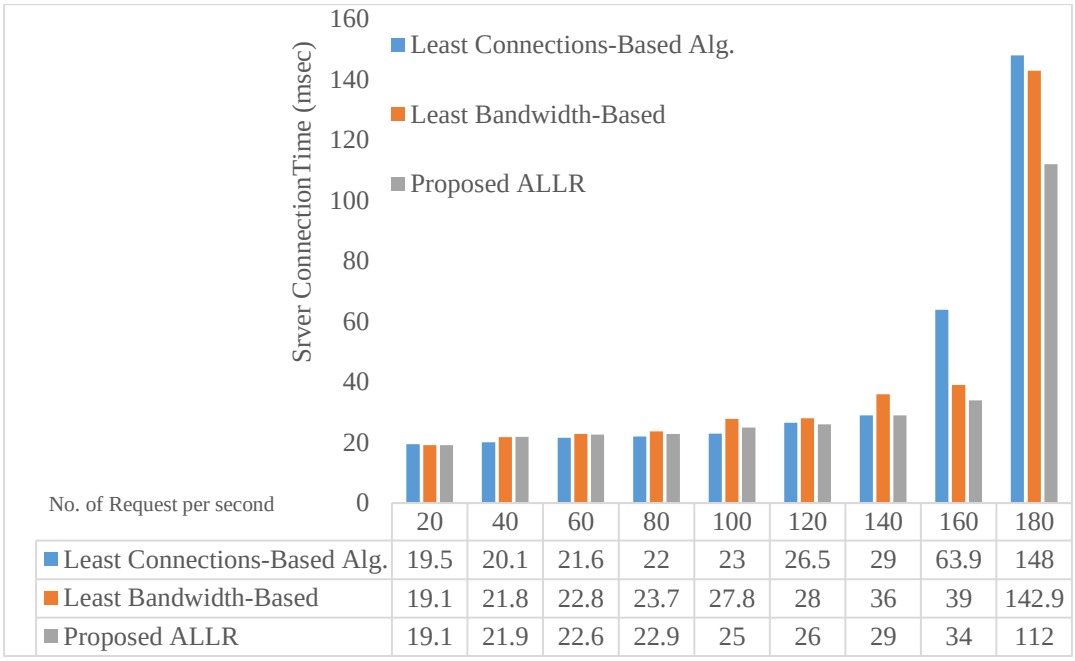


	20	40	60	80	100	120	140	160	180
Least Connections-Based	3	3.9	4.4	4.53	4.55	4.57	4.6	5.4	6.6
Least Bandwidth-Based	2.9	3.7	4.35	4.38	4.4	4.5	4.7	5.45	6.3
Proposed ALLR	2.71	2.75	4.23	4.25	4.285	4.3	4.33	4.5	4.9

Figure (4.7): Server Reply Time of 3 algorithms vs. No. of Requests per sec

c) ***Server Connection Time***

Server connection time that is defined as the session length between the client and server. Figure (4.8) clearly confirms that the ALLR algorithm regularly reduces server connection time to 16.38% compared with least connection-based and least bandwidth-based algorithms.



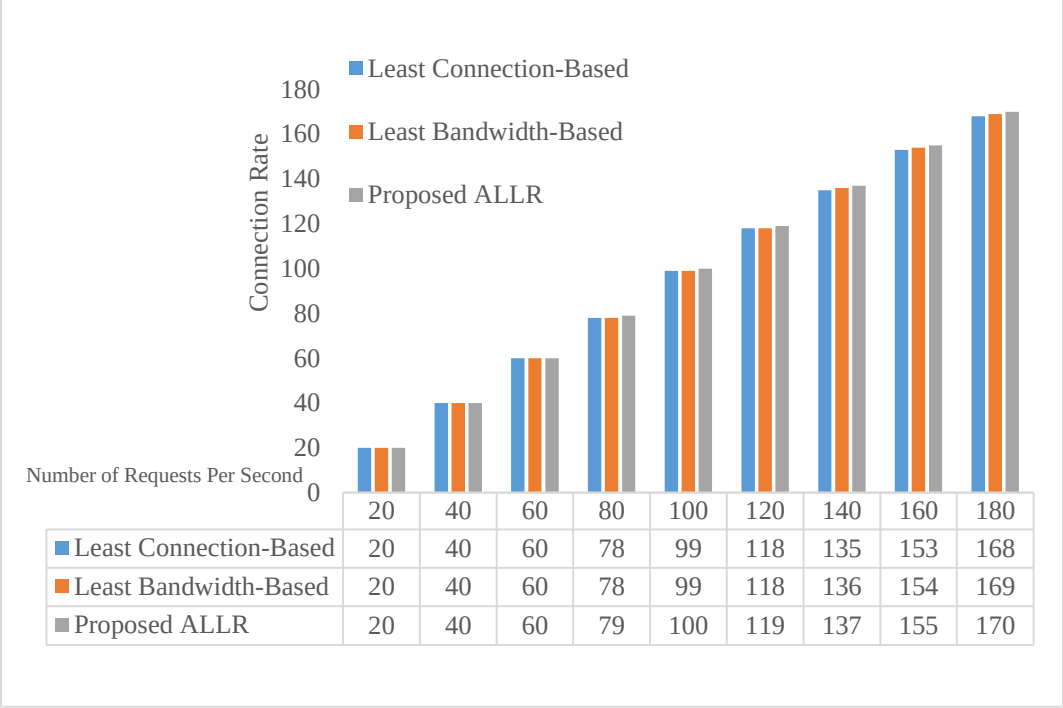
No. of Request per second	20	40	60	80	100	120	140	160	180
Least Connections-Based Alg.	19.5	20.1	21.6	22	23	26.5	29	63.9	148
Least Bandwidth-Based	19.1	21.8	22.8	23.7	27.8	28	36	39	142.9
Proposed ALLR	19.1	21.9	22.6	22.9	25	26	29	34	112

Figure (4.8): Server Connection Time of 3 algorithm vs. No. of Requests per Sec

d) ***Server Connection Rate***

The server connection rate denotes the newly accepted requests per client. Generally, the number of client requests is greater than the connection rate. Results obtained from figure (4.9) shows that during the first six values of client request 20, 40, 60, 80, 100, 120 (requests/second) the connection rate is not affected but when the

number of client requests jump above 120 (request/second), the connection rate begins to decline due to the growing number of dropped packets. However, the ALLR improves the connection rate up to 1.2% compared with dynamic least connection-based and dynamic least bandwidth-based.



	20	40	60	80	100	120	140	160	180
Least Connection-Based	20	40	60	78	99	118	135	153	168
Least Bandwidth-Based	20	40	60	78	99	118	136	154	169
Proposed ALLR	20	40	60	79	100	119	137	155	170

Figure (4.9): Server Connection Rate of 3 Algorithms vs. No. of Requests per Sec

e) Server CPU Utilization

Server CPU utilization is the parameter responsible for measuring server load and the host usage of processing resources. Values in figure (4.10) reveals that ALLR algorithm enhances the CPU utilization to 2.5% without specifying any distinct limit of client

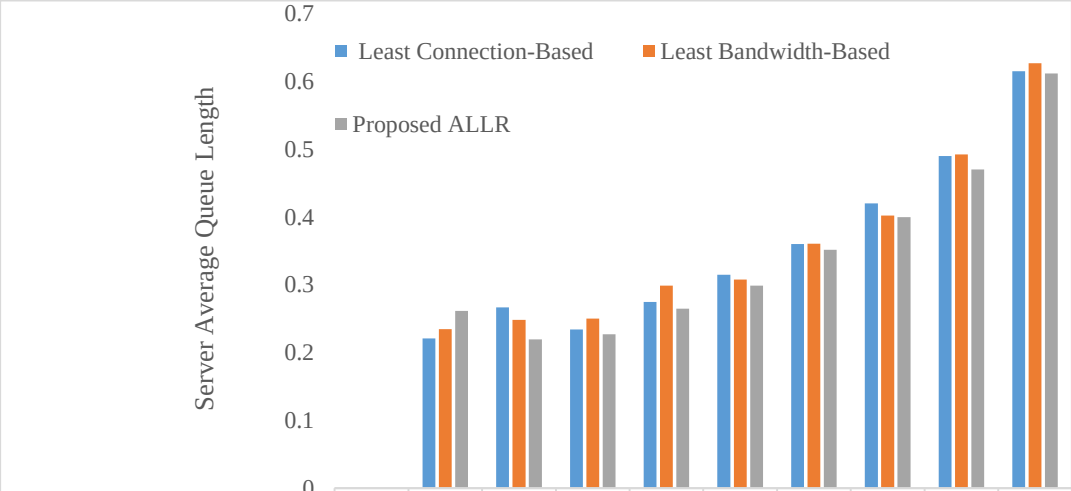
requests per second compared with least connection-based and least bandwidth –based algorithms.

	20	40	60	80	100	120	140	160	180
Least Connections-Based	18.10%	18.50%	19%	21.60%	24%	27%	30%	33%	38%
Least Bandwidth-Based	19.00%	19.90%	20%	23%	23.60%	26.50%	29%	33%	38.50%
Proposed ALLR	17.80%	18%	18.50%	21.00%	23%	26%	29%	32%	38%

Figure (4.10): Server CPU Utilization of 3 Algorithm vs. No. of Requests per Sec

f) Server Average Queue Length

The server average queue length cares about the expected time delay due to the waiting time a packet spends in a queue. Typically, servers are expected to create queues to process the incoming requests. Results from figure (4.11) demonstrates that the ALLR algorithm achieves a progress regarding the Server average queue length by declining this metric to 14% compared with least connection-based and least bandwidth –based algorithms.



No. of Requsts Per Second	20	40	60	80	100	120	140	160	180
Least Connection-Based	0.221	0.2669	0.234	0.275	0.315	0.36	0.42	0.49	0.615
Least Bandwidth-Based	0.2345	0.2484	0.25	0.2987	0.308	0.3605	0.4024	0.4925	0.627
Proposed ALLR	0.2615	0.2195	0.2269	0.265	0.2987	0.3515	0.4	0.47	0.612

Figure (4.11): Server Average Queue Length of 3 Algorithm vs. No. of Requests per Sec

4.4.3 The Hybrid-Based (HB) Algorithm

This section aims to evaluate the performance of the HB load balancing algorithm compared with a different pattern of load balancing algorithms like dynamic least connection based and static weighted round load balancing algorithms running under the same conditions. Table 4.7 summarizes all of the parameters that have been involved in the evaluation procedure.

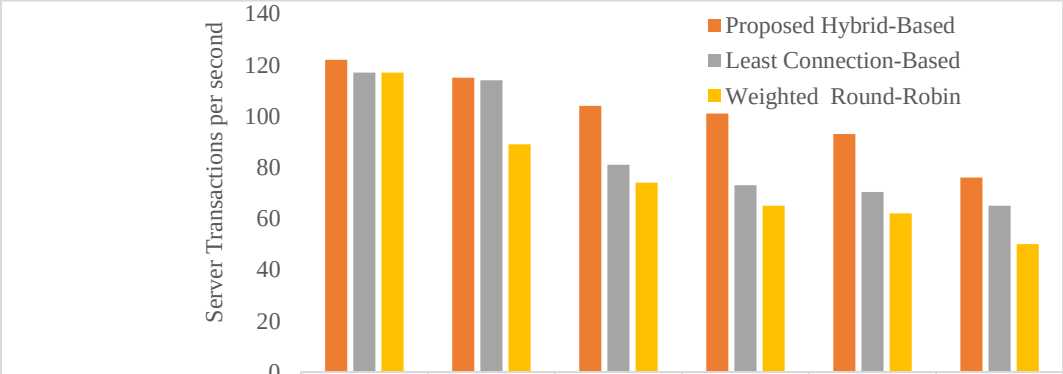
Table 4.7: Parameter Values of HB Load Balancing Algorithm

Parameter	***Value***
tool	OpenLoad
No. of Apache HTTP web servers	3
Range of Concurrent users	(1-350)
No. of HTTP clients	9
SDN-based Platform Controller	POX
Virtual SDN Switch	OVS- OpenFlow switch
Communication Port B/W POX & OVS	6633
HTTP web servers listening port	80
Virtual IP (VIP)	10.0.1.1
Loopback IP	127.0.0.1
Load balancing Algorithm invoked by POX	1- Weighted Round-Robin 2- Least Connection-Based 3- HB

a) Server Transactions per Second

The Server transactions per second is defined as the total number of processed transactions by a server in a given period. The load is represented by the simultaneous number of requests that attempt to access the servers' web applications. The findings from

figure (4.12) indicates that the HB algorithm improves the server transactions per second up to 22%.

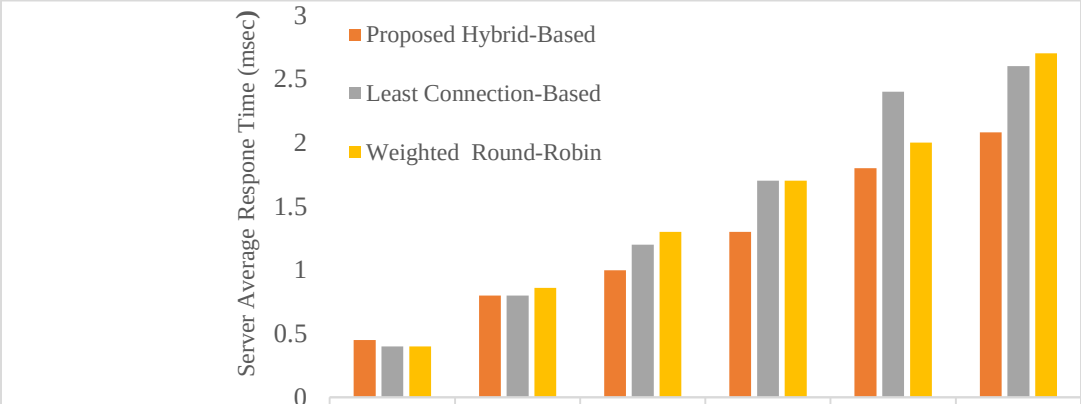


No. of Concurrent Users	50	100	150	200	300	350
Proposed Hybrid-Based	122	115	104	101	93	76
Least Connection-Based	117	114	81	73	70.38	65
Weighted Round-Robin	117	89	74	65	62	50

Figure (4.12): Server Transactions per Second of 3 algorithms vs. No. of Concurrent Users

b) Server Average Response Time

The server average response time reflects the required time to deliver requests result to the clients. Different variables may affect the response time, such as average thinking time, number of requests, number of users who accessed the system, and the bandwidth. However, figure (4.13) explains that the polylines fluctuate in a stable and minimal form during the application of the HB algorithm by reducing the server average response time to 17% compared with static weighted round-robin and dynamic least connection-based algorithms.

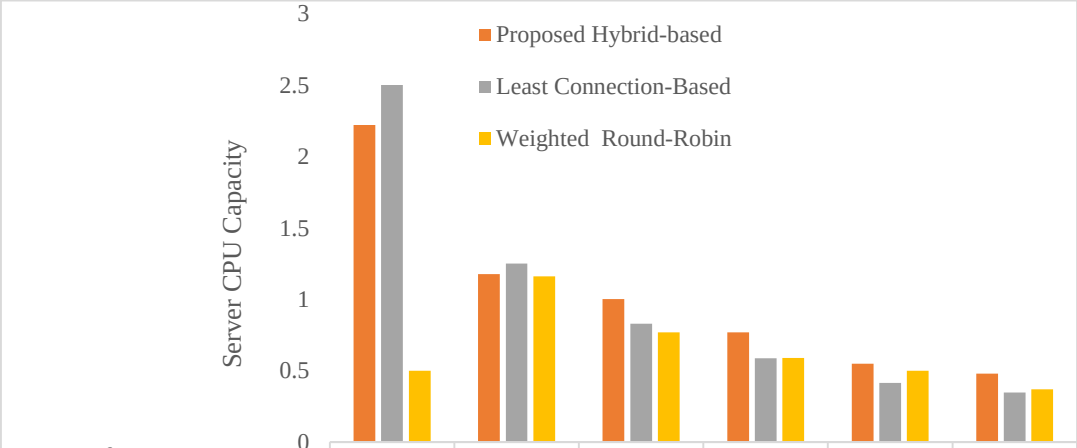


No. of Concurrent Users	50	100	150	200	300	350
Proposed Hybrid-Based	0.45	0.8	0.997	1.3	1.8	2.08
Least Connection-Based	0.4	0.8	1.2	1.7	2.4	2.6
Weighted Round-Robin	0.4	0.86	1.3	1.7	2	2.7

Figure (4.13): Server Average Response Time of 3Algorithms vs. No. of Concurrent Users

c) Server CPU Capacity

Server CPU capacity is defined as the process to calculate the number of resources needed to provide the desired level of services based on a given workload based on server average response time . Results from figure (4.14) unmistakably imply that the HB algorithm outperforms the dynamic connections-based and static weight round-robin algorithms in terms of server CPU capacity up to 5%.



No. of Concurremt Users	50	100	150	200	300	350
Proposed Hybrid-based	2.22	1.176	1.003	0.769	0.55	0.48
Least Connection-Based	2.5	1.25	0.83	0.588	0.416	0.348
Weighted Round-Robin	0.5	1.162	0.769	0.59	0.5	0.37

Figure (4.14): Server CPU Capacity of 3 Algorithms vs. No. of Concurrent Users

The HB load balancing algorithm overcomes static and dynamic load balancing algorithms issues depending on inheriting the distinctive characteristics and overcoming both strategies' limitations and drawbacks achieved by merging one or more static or dynamic load balancing algorithms. Accordingly, the above experimental results reveal the effectiveness of the HB load balancing algorithm. Counters are updated when every packet crosses that OpenFlow switch. Therefore, they must be restored in a streaming way. The ALLR and HB load balancing algorithms significantly leverage traffic engineering (TE) to monitor more networking parameters. Thus, they provide more particular view of network resource utilization by owning the ability to deliver individual statistical analysis related to server resources utilization. A significant performance improvement is detected due to the continuous evaluation feature of the traffic consumed between the servers that

allowed the server with the least load to respond to the next request. The server weights often refer to servers' actual capacities are not equally distributed. Hence, servers owned different characteristics and were closer to reality. Consequently, a significant improvement related to performance metrics is achieved when compared with algorithms such as weighted round robin, least connection based, and least bandwidth based. Regarding the least bandwidth-based algorithm, the server selects the server that consumes the least amount of traffic during the last 14 seconds. After the POX controller sends a Flow/statistics/request message to the OpenFlow switch, the latter reacts by sending a Flow/Statistics/Receive message representing the total number of processed packets. The load balancer selects the servers with the lowest number of processed packets. The controller carries out the preparation of the suitable flows through the utilization of the OPFT-FLOW-MOD message of the identified server to transmit the upcoming traffic. Finally, the balancer parses and stores the number of bytes that are transmitted to each server. The least connections-based algorithm forwards the network's request to the node with the least TCP connections (the node that servers the least number of client requests). However, in situations where a load of requests differs enormously, this approach is considered a good option for smooth distribution since all the long requests have no chance of being directed to a node. However, this algorithm performs appropriately when nodes of different processing capacities are available.

4.4. Summary

This chapter accomplishes several tasks such as The POX controller performance evaluation according to several paradigms and metrics, the performance evaluation of static and dynamic load balancing running on SDN-based platform POX controller, and the proposition of novel load balancing algorithms, namely the ALLR and HB algorithm

5.1 Introduction

SDN is overgrowing. Therefore, it is necessary to put under consideration the challenges and obstacles. More than one solution has been suggested through this study to improve load balancing in the SDN-Based platform.

5.2 Conclusions

a) The allocated bandwidth is in the range of (100-300) Mbps to the adoption of queuing technology in managing the allocated bandwidth for medium applications, which is what the Apache servers provide in the proposed system. The effective bandwidth utilization in OVS OpenFlow switch rises directly with the allocated bandwidth until it reaches the maximum value at 300Mbps.

b) The size of packets is within 1500 bytes, which is defined as size of the maximum transmission unit (MTU) for the TCP protocol. The aim is to select the relation between the dropped packet and the packet size. Moreover, to select the best packet size to work with. Dropped packet fluctuates with packet size. But, packet with size of 512 and 1524 bytes records the minimum and maximum dropped packet respectively.

c) The average throughput and latency of the POX controller have been examined to fully understand the SDN technology. The rise in the number of OpenFlow switches causes a drop in the average throughput and a growth in the average latency. This behavior continues until recording the lowest average throughput and highest

average latency when the number of OpenFlow switches is equal to 64. Any rise in the number of OpenFlow switches leads to an increase in the flow installation time known as (path provisioning), and fewer rules are pushed from the POX controller to the OpenFlow switches. Accordingly, the recommendation is to avoid the use of 64 OpenFlow switches in SDN-based platform.

d) The performance for a set of static and dynamic load balancing algorithms is evaluated to identify algorithms that will be compared with ALLR and HB proposed algorithms. From our investigation regarding dynamic load balancing algorithms, the least bandwidth-based algorithm outperforms least connection based. Regarding static load balancing algorithm, weighted round robin outperforms round robin and random algorithm.

e) A dynamic server-based load balancing solution, namely the adaptive least loaded ratio (ALLR) load-balancing algorithm, is proposed and implemented. The ALLR algorithm is geared to provide a load balancing mechanism based on the computation of the least loaded server among a pool of n server members based on a specialized data-collection module to collect OVS OpenFlow switch ports statistics. According to the experimental results accomplished through this study, the ALLR algorithm exploits server resources and balances loads substantially compared with dynamic least connection-based and dynamic least bandwidth-based algorithms. The proposed ALLR algorithm undoubtedly confirms an improvement regarding server average throughput up to 7.25%,

server connection time up to 16%, and Server connection rate up to 1.2%. Moreover, the ALLR algorithm reduces server reply time, server CPU utilization, and server average queue length to 19%, 2.5%, and 14%.

f) The last objective is fulfilled through the proposition and implementation of a server based hybrid (HB) load balancing algorithm. HB algorithm utilizes the resources more efficiently as it relies on the adoption of advanced data collection module to collect OVS OpenFlow switch ports and flow entries statistics. The distinctive characteristics and overcoming the limitations existing in static weighted round-robin and dynamic least-connection-based load balancing algorithms. The experimental results reveal that the HB algorithm enhances the load balancing and request handling effect by increasing the server transactions per second up to 22%, CPU capacity up to 5%, and reducing average server response time to 17%.

5.3 Limitations

As a result of working within a virtual environment like mininet to simulate the SDN-based platform paradigms, the generated load by clients has been restricted to a certain threshold that belongs to the MaxClients parameter of the Apache HTTP servers. In contrast, the generated load in a real-time SDN may reach boundaries much higher than the virtual environment. This turns out the enhancement results of

the `proposed ALLR and HB load balancing algorithms to become too close to the results of the existing load balancing algorithms.

5.4 Future Research Directions

In the following, we present possible future research directions that may be conducted to extend the dissertation innovations.

a) Future work will address the scalability of the proposed load balancing algorithms achieved by the dynamic addition of the hosts to the existing pool when all pool members are being overloaded.

b) An important feature is to adopt multiple controllers with the proposed load balancing schemes (ALLR and HB). The aim is to avoid the single point of failure problem where additional controllers carry out the load balancing task when the master controller goes down.

c) The assignment of dynamic weights instead of static ones for the servers participating in the proposed load balancing schemes (ALLR and HB). Dynamic weights generally reflect actual server capabilities and further improve the performance of balancing sche

References

[1] B. A. A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surv. tutorials*, *vol. 16, no. 3, pp. 1617–1634, 2014.*

[2] V. Khatri, "*Analysis of OpenFlow protocol in local area networks*." 2013.

[3] D. J. Hemanth, "A Survey on Traffic Prediction and Classification in SDN," *Intell. Syst. Comput. Technol., vol. 37, p. 367, 2020*.

[4] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. *44, no. 2, pp. 87–98, 2014.*

[5] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun., vol. 67, pp. 1–10, 2015*.

[6] P. Berde, *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, *2014*

[7] S. Sezer, *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag., vol. 51, no. 7, pp. 36–43, 2013*.

[8] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *J. Comput. Networks Commun., vol. 2016, 2016.*

[9] S. Computing, "Case Studies in Secure Computing."

[10] N. S. Ghumman and R. Kaur, "Dynamic combination of improved max-min and ant colony algorithm for load balancing in cloud system," in *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–5, 2015.*

[11] J. S. Sabiya, "Weighted round-robin load balancing using software defined networking," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 6, no. 6, 2016.

[12] M. S. Sroya and V. Singh, "LDDWRR: Least Delay Dynamic Weighted Round-Robin Load Balancing in Software Defined Networking." *Int. J. Adv. Res. Comput. Sci., vol. 8, no. 5, 2017.*

[13] G. B. Haile and J. Zhang, "Dynamic Load Balancing Algorithm in SDN-based Data Center Networks."

[14] D. Todorov, H. Valchanov, and V. Aleksieva, “Load Balancing model based on Machine Learning and Segment Routing in SDN,” in *2020 International Conference Automatics and Informatics (ICAI), pp. 1–4, 2020.*

[15] F. Cimorelli, et al., “A distributed load balancing algorithm for the control plane in software defined networking,” in *2016 24th Mediterranean Conference on Control and Automation (MED), pp. 1033–1040, 2016.*

[16] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, and A. Rezaee, “Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature,” *IEEE Access, vol. 6, pp. 14159–14178, 2018.*

[17] M. Priyadarsini, et al., “An adaptive load balancing scheme for software-defined network controllers,” *Comput. Networks, vol. 164, pp. 106918, 2019*.

[18] R. Gandhi, et al., "Duet: Cloud scale load balancing with hardware and software”. *ACM SIGCOMM Computer Communication Review, vol. 44, pp.27-38, 2014.*

[19] P. Vizarreta, et al. , “An empirical study of software reliability in SDN controllers,” in *2017 13th International Conference on Network and Service Management (CNSM), pp. 1–9, 2017.*

[20] S. Ejaz, Z. Iqbal, P. A. Shah, B. H. Bukhari, A. Ali, and F. Aadil, “Traffic load balancing using software defined networking (SDN) controller as virtualized network function,” *IEEE Access, vol. 7, pp. 46646–46658, 2019.*

[21] M.L. Chiang, H.S. Cheng, H.Y. Liu, and C.Y. Chiang, “SDN-based server clusters with dynamic load balancing and performance improvement,” *Cluster Comput., pp. 1–22, 2020.*

[22] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey", *Computer communications, vol.67, pp.1-10, 2015.*

[23] W. Chen, Z. Shang, X. Tian, and H. Li, "Dynamic server cluster load balancing in virtualization environment with OpenFlow", *International Journal of Distributed Sensor Networks, vol. 11, no.7, pp.531538, 2015.*

[24] S. K. Askar, “Adaptive load balancing scheme for data center networks using software defined network,” *Sci. J. Univ. Zakho, vol. 4, no. 2, pp. 275–286, 2016.*

[25] S. Raghul, T. Subashri, and K. R. Vimal, “Literature survey on traffic-based server load balancing using SDN and open flow,” in *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN), pp. 1–6, 2017.*

[26] K. Govindarajan and V. S. Kumar, “An intelligent load balancer for software defined networking (SDN) based cloud infrastructure,” in *2017 second international conference on electrical, Computer and Communication Technologies (ICECCT), pp. 1–6, 2017.*

[27] R. Abbasi, S. M. M. Gilani, A. Kabir, Q. Nawaz, and M. S. Riaz, “Load Balancing of SDN-enabled Wireless Network: Challenges, Technique and Evaluation,” *J. Inf. Commun. Technol. Robot. Appl., pp. 80–92, 2019*.

[28] M. K. Faraj, A. Al-Saadi, and R. J. Albahadili, “An Investigation of Using Traffic Load In SDN Based Load Balancing,” *IRAQI J. Comput. Commun. Control Syst. Eng., vol. 20, no. 3, 2020.*

[29] W. S. Prakash and P. Deepalakshmi, “DServ-LB: Dynamic server load balancing algorithm,” *Int. J. Commun. Syst.*, vol. 32, no. 1, 2019.

[30] A. M. Caulfield, E. S. Chung, M. K. Papamichael, D. C. Burger, and S. Alkalay, “Hardware implemented load balancing.” Google Patents, 26-Dec-2019.

[31] A. Bremler-Barr, D. Hay, I. Moyal, and L. Schiff, “Load balancing memcached traffic using software defined networking" . In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 1-9, IEEE, 2017

[32] H. M. Kavana, V. B. Kavya, B. Madhura, and N. Kamat, “Load balancing using SDN methodology,” *Int. J. Eng. Res. Technol.*, vol. 7, no. 5, pp. 206–208, 2018.

[33] O. Fundation, “Software-defined networking: The new norm for networks,” *ONF White Pap.*, pp. 1–12, 2012.

[34] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, “Software transactional networking: Concurrent and consistent policy composition,” *Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw.*, pp. 1–6, 2013.

[35] T. D. Nadeau and K. Gray, *SDN Software Defined Networks*, First Edit. O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. O’Reilly, 2013.

[36] P. Hui and T. Koponen, *Software Defined Networking*, vol. 2, no. 9. 2012.

[37] A.A Semenovykh and O.R. Laponina, "Comparative analysis of SDN controllers", *International Journal of Open Information Technologies*, vol.*6*, no.7, pp.50-56, 2018

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

[39] S.H. Yeganeh, A.Tootoonchian, Y. Ganjali, "On scalability of software-defined networking", *IEEE Communications Magazine*, vol.*51*, no.2, pp.136-141, 2013.

[40] E. Haleplidis, J. H. Salim, and D. Meyer, "Software-Defined Networking (SDN): Layers and Architecture Terminology," *Internet Res. Task Force*, pp. 1–35, 2015.

[41] L. Paraschis, "Software Innovations and Control Plane Evolution in the new SDN Transport Architectures," *Cisco Connect summit Toronto*, pp. 1–74, 2015.

[42] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *ACM Sigcomm Comput. Commun.*, vol. 44, no. 2, pp. 87–98, 2014.

[43] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," *Work. Hot Top. Networks, ACM*, pp. 1–6, 2010.

[44] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. - Conex. '12*, pp. 253–264, 2012.

[45] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A Network Virtualization Layer," *Network*, p. 15, 2009.

[46] M. Casado, N. McKeown, and S. Shenker, "From ethane to SDN and beyond." *ACM SIGCOMM Computer Communication Review*, vol.*49,no.*5, pp.92-95,2019.

[47] S. Shenker, M. Casado, T. Koponen, and N. McKeown, "The future of networking, and the past of protocols", *Open Networking Summit*, vol.*20*, pp.1-30, 2011.

[48] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Networks, vol. 71, pp. 1–30, 2014.*

[49] S. Clayman, L. Mamatas, and A. Galis, "Efficient management solutions for software-defined infrastructures," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, pp. 1291–1296, 2016.*

[50] M. El-Azzab, I. L. Bedhiaf, Y. Lemieux, and O. Cherkaoui, "Slices isolator for a virtualized OpenFlow node," in *2011 First International Symposium on Network Cloud Computing and Applications, pp. 121–126, 2011.*

[51] M. P. Fernandez, “Comparing OpenFlow controller paradigms scalability: Reactive and proactive,” *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA*, pp. 1009–1016, 2013.

[52] W. Braun and M. Menth, “Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices,” *Futur. Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[53] K. Dhamecha and B. Trivedi, “Sdn issues-a survey,” *Int. J. Comput. Appl.*, *vol. 73, no. 18, 2013.*

[54] S. Badotra and S. N. Panda, “Software-defined networking: A novel approach to networks,” in *Handbook of Computer Networks and Cyber Security*, Springer, *pp. 313–339, 2020.*

[55] S. Alalmaei, Y. Elkhatib, M. Bezahaf, M. Broadbent, and N. Race, “SDN Heading North: Towards a Declarative Intent-based Northbound Interface,” in *2020 16th International Conference on Network and Service Management (CNSM), pp. 1–5, 2020.*

[56] G. Parulkar, “SDN fundamentals: motivation, architecture, and benefits,” *CSI Trans. ICT, vol. 8, no. 1, pp. 7–9, 2020.*

[57] M. R. Adrian, D. H. Kurniawan, A. Faza, J. Maulina, and M. R. Shihab, “A Brief Look at Software Defined Network (SDN) Implementation: Gaining Benefits and Coping with the Challenges at a Telecommunication Company,” in *IOP Conference Series: Materials Science and Engineering*, *vol. 879, no. 1, p. 12046, 2020.*

[58] A. Ghaffari, “Performance issues and solutions in SDN-based data center: a survey,” *J. Supercomput., pp. 1–49, 2020*.

[59] P. Rygielski, M. Seliuchenko, S. Kounev, and M. Klymash, “Performance Analysis of SDN Switches with Hardware and Software Flow Tables.,” *2016.*

[60] P. Isaia, and L. Guan, “Performance benchmarking of SDN experimental platforms". In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 116-120, IEEE, 2016.

[61] J. W. ZHAOGANG SHU, JIAXIANG LIN, SHIYONG WANG, DI LI, SEUNGMIN RHO, CHANGCAI YANG1, "Traffic Engineering in Software-Defined Networking: Measurement and Management," 2016.

[62] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. K. Khan, “Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art,” *IEEE Commun. Surv. Tutorials*, v*ol. 19, no. 1, pp. 303–324, 2016.*

[63] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks", *Computer Networks*, vol.*71*, pp.1-30,2014.

[64] B.Görkemli, S. Tatlıcıoğlu, A.M. Tekalp, S. Civanlar, and E. Lokman," Dynamic control plane for SDN at scale", *IEEE Journal on Selected Areas in Communications*, vol.*36*,no.12, pp.2688-2701,2018.

[65] T. E. Ali, A. H. Morad, and M. A. Abdala, "Load balance in data center SDN networks," *Int. J. Electr. Comput. Eng., vol. 8, no. 5, p. 3086, 2018.*

[66] M. Fatica, M. and G. Ruetsch, "*CUDA Fortran for scientists and engineers*", Elsevier, *2014.*

[67] S. Harsh, "Evaluation of Multiple Controller Software Defined Networks." *2016.*

[68] A. Saker Ahmad, and A. Mohammad, "A Study of OpenFlow Protocol and POX Controller in Software Defined Networks(SDN) Using Mininet", *Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series* Vol. 14, no.4, pp.9142, 2019

[69] A. A. A. Osman, "Service based load balance mechanism using Software-Defined Networks." University of Malaya, *2017.*

[70] D. Kreutz, F. M. V Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Member, and S. Uhlig, "Software-Defined Networking : A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[71] T. A. Assegie, "Software defined network emulation with OpenFlow protocol," *Int J Adv Appl Sci ISSN, vol. 2252, no. 8814, p. 8814, 2020.*

[72] S. Azodolmolky, ""Software Defined Networking with OpenFlow", 2013.

[73] X. Sun, T. S. E. Ng, and G. Wang, "Software-Defined Flow Table Pipeline," in *2015 IEEE International Conference on Cloud Engineering*, pp. 335-340,2015.

[74] X. Shi *et al.*, "An OpenFlow-Based Load Balancing Strategy in SDN," *C. Mater. Contin.*, v*ol. 62, no. 1, pp. 385–398, 2020.*

[75] V. W. Protocol, "OpenFlow Switch Specification 1.5," *Open Netw. Found. ONF*, vol. 0, pp. 1–205, 2013.

[76] M.A. Beiruti, and Y. Ganjali, "Load migration in distributed sdn controllers. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium"*, pp. 1-9, IEEE, 2020.

[77] L. Zhu, et al., " SDN controllers: Benchmarking & performance evaluation", *arXiv preprint arXiv:1902.04491,* 2019.

[78] M. Zaher, , S. Molnár, *" Enhancing of micro flow transfer in SDN-based data center network, IEEE International Conference on Communications (ICC)*, pp. 1-6, IEEE, 2019.

[79] M. Dusi, R. Bifulco, F. Gringoli, and F. Schneider, "*Reactive logic in software-defined networking: Measuring flow-table requirements*. In Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International pp. 340-345, IEEE, 2014.

[80] P. Lin, et al., " *Seamless interworking of SDN and IP*. In ACM SIGCOMM computer communication review , vol. 43, no. 4, pp. 475-476. ACM, 2013.

[81] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Comput., vol. 17, no. 2, pp. 20–27, 2012.*

[82] M. Qilin and S. Weikang, "A load balancing method based on SDN," in *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation, pp. 18–21, 2015.*

[83] L.-D. Chou, Y.-T. Yang, Y.-M. Hong, J.-K. Hu, and B. Jean, "A genetic-based load balancing algorithm in openflow network," in *Advanced technologies, embedded and multimedia for human-centric computing*, Springer, *pp. 411–417, 2014.*

[84] A. A. Abdelltif, E. Ahmed, A. T. Fong, A. Gani, and M. Imran, "SDN-based load balancing service for cloud servers," *IEEE Commun. Mag., vol. 56, no. 8, pp. 106–111, 2018.*

[85] G. Lakhani and A. Kothari, "Fault Administration by Load Balancing in Distributed SDN Controller: A Review," *Wirel. Pers. Commun., vol. 114, no. 4, pp. 3507–3539, 2020.*

[86] G. Huang, "Achieving Effective Resource Management in Distributed SDN Controller Architectures," *2020.*

[87] S. Mostafavi, V. Hakami, and F. Paydar, "Performance Evaluation of Software-Defined Networking Controllers: A Comparative Study," *Comput. Knowl. Eng.*, vol. 2, no. 2, pp. 63–73, *2020*.

[88] M. T. Islam, N. Islam, and M. Al Refat, "Node to node performance evaluation through RYU SDN controller," *Wirel. Pers. Commun., pp. 1–16, 2020.*

[89] D. Lunagariya and B. Goswami, "A Comparative Performance Analysis of Stellar SDN Controllers using Emulators," in *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), pp. 1–9, 2021*.

[90] Y. Li, X. Guo, X. Pang, B. Peng, X. Li, and P. Zhang, "Performance Analysis of Floodlight and Ryu SDN Controllers under Mininet Simulator," in *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops), pp. 85–90, 2020*.

[91] C. D. Cajas and D. O[94]. Budanov, "SDN Applications and Plugins in the OpenDaylight Controller," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp. 9–13, 2020.*

[92] P. Vizarreta *et al.*, "Assessing the maturity of sdn controllers with software reliability growth models," *IEEE Trans. Netw. Serv. Manag., vol. 15, no. 3, pp. 1090–1104, 2018.*

[93] K. P. S. S. Rohith and A. Anand, "Analytical Study of different Load balancing algorithms," *Int. J. Adv. Stud. Comput. Sci. Eng., vol. 7, no. 1, pp. 21–26, 2018.*

[94] U. Zakia and H. Ben Yedder, "Dynamic load balancing in SDN-based data center networks," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 242–247, 2017.*

[95] S. Hamadah, "A survey: a comprehensive study of static, dynamic and hybrid load balancing algorithms," *Int. J. Comput. Sci. Inf. Technol. Secur. (IJCSITS), ISSN, pp. 2249–9555, 2017.*

[96] B. Pfaff, et al., "The *Design and Implementation of Open vSwitch*. In *NSDI,* pp. 117-130, 2015.

[97] D.R. Mariscal Basilio, A.E. Salazar Rueda, " *Diseño e implementación de una red overlay, con switches SDN basado en hardware de bajo costo, para el FCI-011 temonet-fase II* (Doctoral dissertation, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera de Ingeniería en Networking y Telecomunicaciones)*., 2020.*

[98] D. Kunda, S. Chihana, and M. Sinyinda, "Web server performance of apache and nginx: A systematic literature review," *Comput. Eng. Intell. Syst., vol. 8, no. 2, pp. 43–52, 2017.*

[99] S. Bekman and E. Cholet, *Practical mod_perl*. " O'Reilly Media, Inc.," *2003.*

[100] R. Bowen and K. Coar, *Apache Cookbook: Solutions and Examples for Apache Administration*. " O'Reilly Media, Inc.," *2007.*

[101] S. Narayanan, "A Study of Load Balancing Algorithms in Software Defined Network." *2020.*

[102] B. Haile and J. Zhang, "Dynamic Load Balancing Algorithm in SDN-based Data Center Networks." *2021*.

[103] R. Pandey, “Comparing VMware Fusion, Oracle VirtualBox, Parallels Desktop implemented as Type-2 hypervisors. [110] S. E. Quincozes *et al.*, “Survey and Comparison of SDN Controllers for Teleprotection and Control Power Systems.,” *2019.*

[104] M. Hasan, H. Dahshan, Abdelwanees, and E. Elmoghazy, " SDN Mininet Emulator Benchmarking and Result Analysis", In *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)* pp. 355-360, IEEE, 2020.

[105] A. Botta, W. de Donato, A. Dainotti, S. Avallone, and A. Pescapé, “D-ITG § VERSION § Manual,” *2019.*

جمهورية العراق- وزارة التعليم
العالي والبحث العلمي
جامعة بابل
كلية تكنولوجيا المعلومات
قسم البرامجيات

نهج تخصيص الموارد المحسن للشبكات المعرفة برمجيا

اطروحة مقدمة الى مجلس كلية تكنولوجيا المعلومات للدراسات العليا بجامعة بابل
كجزء من متطلبات درجة الدكتوراة فلسفة في تكنولوجيا المعلومات/ برامجيات

من قبل

حيدر منذر نعمان مدحت

باشراف

ا.م.د مهدي نصيف جاسم علوان

2021 A.D **1443 A.H**

الخلاصة

اتاح ظهور الشبكات المعرفة بالبرمجيات (SDN) الفرصة لمديرين الانظمة لامتلاك تقنية إدارة حركة المرور التي تضمنت ميزة التكلفة المنخفضة ، وشكل التشغيل المرن ، والاستخدام الأمثل للموارد. قدمت SDN حلاً غير مكلف وقابل للتطوير وواعد لقيود موازن الأحمال التقليدية من خلال تمكين البرمجة والتحكم المركزي. يتمثل التحدي الرئيسي الذي تواجهه خوارزميات موازنة الحمل الحالية في الافتقار إلى القدرة على تمكين هندسة المرور (TE) من تجميع وتوليد إحصائيات استخدام النطاق الترددي لتحليل حركة المرور ومراقبتها ، وهي مهمة صعبة أخرى إذا كانت المراقبة المطلوبة دقيقة. يهدف العمل الذي تم إجراؤه إلى إنتاج نموذج لتحسين استخدام موارد الخادم من خلال تطوير خوارزميات جديدة لموازنة الحمل مثل موازنة الحمل الأقل تكيفًا (ALLR) وخوارزمية موازنة الحمل الهجينة (HB) . تم اختبار نموذج خوارزميات موازنة الحمل المقترحة باستخدام أنواع مختلفة من أدوات قياس الأداء الشبكي مثل httperf و OpenLoad.النتائج تكشف ان خوارزمية ALLR تحسن متوسط انتاجية الخادم إلى ٧,٢٥٪ ، ووقت اتصال الخادم إلى ١٦٪ ، ومعدل اتصال الخادم إلى ١,٢٪. علاوة على ذلك ، تقلل خوارزمية ALLR من وقت رد الخادم ، واستخدام وحدة المعالجة المركزية للخادم ، ومتوسط طول قائمة انتظار الخادم إلى ١٩٪ ، و ٢,٥٪ ، و ١٤٪ مقارنة بخوارزميات موازنة الحمل الديناميكي الاخرى مثل المستند الى الاتصال الاقل (LCB) و المستند الى النطاق الترددي الاقل(LBB). خوارزمية HB تعزز موازنة التحميل ومعالجة الطلب عن طريق زيادة معاملات الخادم في الثانية الى ٢٢٪ ، وسعة وحدة المعالجة المركزية الى إلى ٥٪ وتقليل متوسط وقت استجابة الخادم إلى ١٧٪ مقارنة بخوارزمية جولة-روبن الموزونة (WRR) و خوارزمية موازنة الحمل المستندة إلى الاتصال الأقل(LCB).