Benha Journal of Applied Sciences (BJAS) **print : ISSN 2356–9751**
Vol.(2) Issue(1) Oct.(2017), 91-97 **online : ISSN 2356–976x**

http:// bjas.bu.edu.eg

Performance Analysis of Applying Load Balancing Strategies on Different SDN Environments

M.I.Hamed, B.M.ElHalawany, M.M.Fouda and A.S.Tag Eldien
Electrical Engineering Dept., Faculty of Engineering, Shoubra, Benha Univ., Egypt
E-Mail: m.ibrahim@feng.bu.edu.eg

Abstract

Software-Defined Network (SDN) is considered a breakthrough to the global network. It plays an important role in performance improvement and network optimization. SDN is a new mechanism for managing and designing networks rather than the current traditional network system which does not afford more services and higher data rates; therefore, we analyze the effect of applying load balancing techniques and its importance in different SDN environments. In this paper, we propose a dynamic server load balancing technique in SDN architecture. Hence, we implement a server Connection-based load balancing technique and evaluate its performance with a static Round-robin and Random-based in both mininet emulation environment and Raspberry Pi OpenFlow-enabled switch using Ryu OpenFlow controller. The performance of the proposed algorithm is compared with Round-robin and random distribution of clients' requests. The results show that the proposed technique achieves more reliability and higher resource utilization than the Round-robin and Random-based load balancing strategies. In addition, the proposed scheme exhibits more scalability and low-cost characteristics.

Keywords: Software defined-network, Ryu controller, Load balancing, Open Flow, Raspberry Pi.

1.Introduction

In a traditional switch, packet forwarding which can be described as the "data plane" and high-level routing (the control plane) occur on the same device. Some of the drawbacks of the traditional networks are that the physical network devices such as switches, routers, and load balancers are vendor specific. Some of these devices are not compatible with the each other [1]; furthermore, it is not allowed to change their functionality. SDN solves most of the traditional network issues and limitations. SDN is a new technology which decouples the control plane from data plane based on virtualization concept. The data plane is still implemented in the switch itself but the control plane is implemented in software and a separate SDN controller makes the high-level routing decisions [2]. The switch and controller communicate with each other by means of the OpenFlow protocol [3]. Data plane is responsible for the processing and delivery of packets based on the state of the routers and endpoints (e.g., Internet Protocol (IP), Transmission Control Protocol (TCP), Ethernet, etc.), while the control plane determines how and where the packets are forwarded based on the state of the network devices. As a result, network's intelligence and state are logically centralized. SDN aims to make the network devices to be more software-based instead of hardware-based to improve the efficiency of the traditional network. SDN has a centralized controller which controls the traffic through the network. The most commonly used open-source SDN controllers are POX [4], Ryu [5], Trema [6], and OpenDayLight (ODL) [7]. In literature, Raspberry Pi is often used as an OpenFlow switch testbed because of its affordable price (only a few dollars) [8] while the other OpenFlow switches cost thousands of dollars [9]. In SDN, it is easy to program and adjust network rules and policies to manage network flows according to network requirements [10]. The SDN structure comprises of three layers as shown in Fig (1), which can be described as follows.

- Application layer: It is a layer at which applications and programs are installed that provides services to the infrastructure layer such as load balancing, firewall, and network monitoring.
- Control layer: It contains a centralized controller to control the traffic flows through the network and uses OpenFlow protocol to communicate with the infrastructure layer to monitor the overall view of the entire network.
- Infrastructure layer: It consists of both physical and virtual network devices such as switches, routers, and access points.

Currently, the load over the global network is very high and growing rapidly due to the continuously increasing user demands such as browsing, search engines, downloading files, and social networking; Therefore, a load balancer is required to distribute the requests across multiple resources such as servers and network links to enhance the overall network performance (e.g., reducing the latency and response time, increasing the throughput, and utilizing the available resources). In this paper, we analyze and investigate two SDN environments by applying load balancing applications based on software algorithms running on Ryu controller that allows flexibility to the network instead of dedicated expensive hardware load balancer devices. Ryu controller is chosen due to its performance, rapid development, and simple programmability in python. There are some advantages of using SDN technology such as:

- Directly programmable: SDN network is directly programmable because the control plane is decoupled from forwarding plane.
- Agile: It is easy to control the network traffic.
- Centrally controlled: As mentioned above, there is a centralized device called controller which provides the overall view of the network.
- Open standards-based: Most of the SDN controllers are not vendor specific; in addition, they are open-source which enables easy programming.

Mininet emulator [11] is used to emulate the entire network structure including hosts, OpenFlow switches, and controllers. Mininet is an SDN emulator tool written in python via script file or Command Line Interface (CLI) commands. It allows creating virtual network components such as controllers, OpenFlow switches, virtual links, and hosts on a single machine; it supports several SDN topologies and is installed under Linux or Ubuntu operating system. Ryu [5] is considered one of the top most five controllers in terms of its usage, utilization, and deployment [12]. There are many components predefined in Ryu which can be modified, extended, and composed for creating new network management and control applications due to network requirements. Ryu supports OpenFlow protocol 1.0, 1.2, 1.3, 1.4, 1.5 versions for managing network devices [13]. All of the code is freely available under the Apache 2.0 license. Ryu is fully written in python. There are some limitations of mininet. One of these limitations is that the Central Processing Unit (CPU) resources are shared among the virtual components. All mininet hosts share the host file system and Path Identifier (PID) space; therefore, the performance is quietly affected by these limitations. In addition to using mininet, A real-time testbed is also implemented for our SDN network using Raspberry Pi [14] as a low-cost OpenFlow-enabled switch and compare its performance with the results obtained by using mininet.

Load balancing can be implemented for different situations including distributing traffic across multiple paths or distributing the clients' requests across the servers [15]. This improves the overall network performance by optimally utilizing the available resources which lead to a reduction in both latency and response time and an improvement in the network throughput. Distributing the clients' requests across the available servers avoids network's congestion and overload. The traditional load balancers are very expensive hardware devices; in addition, they are vendor specific (i.e., not open source) in contrast to using software load balancer application with SDN controller.

Bindhu et al. [16] implemented a congestion control and a dynamic load balancing algorithm by taking into account the weights of edges and nodes to find the shortest path between the nodes. The drawback of this work is that it is suitable only for small networks. Hu et al. [17] implemented a controller load balancing architecture for OpenFlow networks which partitions the control traffic across multiple controllers. One of these controllers called "super controller", responsible for partitioning the control traffic and distribute them across the available controllers in the network. Li et al. [18] introduced a load balancing routing algorithm in the Fat-Tree network to distribute the traffic across multiple paths equally. The drawback of this algorithm is that it takes a long time to be executed which causes more delay in the network.

In this paper, we propose a load balancing scheme for distributing client's requests across the available servers at which the next request is forwarded to the least-loaded server. We evaluate our load balancing scheme with Random-based and Round-robin and we also analyze their effects on the overall network performance. The results are compared based on mininet emulation environment and Raspberry Pi OpenFlow switch testbed.

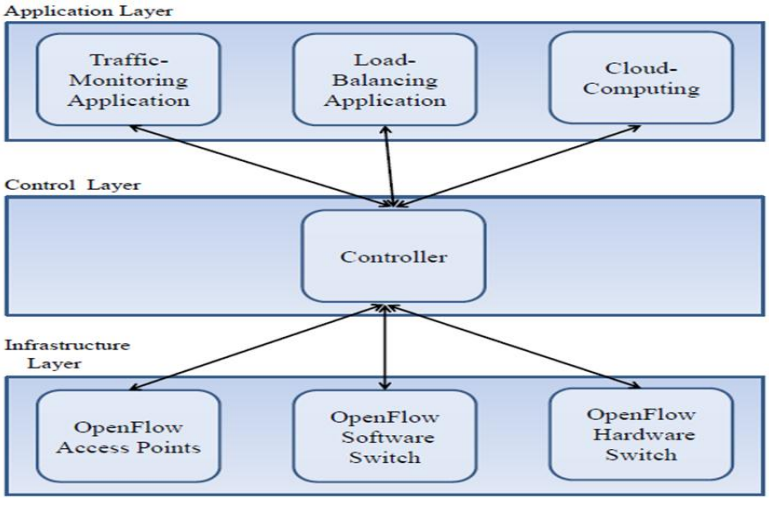

Fig (1) Three layers of SDN structure.

2.Material and methods

In this section, we discuss three load balancing techniques for the system model shown in Fig (2), at which multiple servers are connected to the controller through OpenFlow switch. All servers provide the same service to the clients. The controller has a list of IP addresses assigned to the servers statically. The clients can access servers by using virtual IP address which represents the service's IP; the load balancer distributes the requests among servers. In other words, from the client's perspective, all servers are regarded as a single server.

When a client requests the virtual IP, the request is forwarded to the controller through the OpenFlow switch to decide which server the request is forwarded to. The controller selects the server according to the load balancing application; then, the controller modifies the packet header including destination Medium Access Control (MAC) address and destination IP of the selected server and sends these rules to the OpenFlow switch again. The switch forwards the request to the port assigned to the selected server; then, the selected server replies to the client's request through the OpenFlow switch. Every packet must be forwarded to the controller to decide which server will handle the client's request. The load balancing algorithms implemented in this paper are:

- Random-based: In this case, one server is selected randomly from the available servers.
- Round-robin: The requests are distributed among servers in a sequential manner. In other words, the chosen server is always the server of the next round in the network; therefore, all servers almost handle the same number of requests
- Server-based: This algorithm selects the server least number of concurrently active TCP connections of the server machines. Netstat command is used to verify the number of server's active connections.

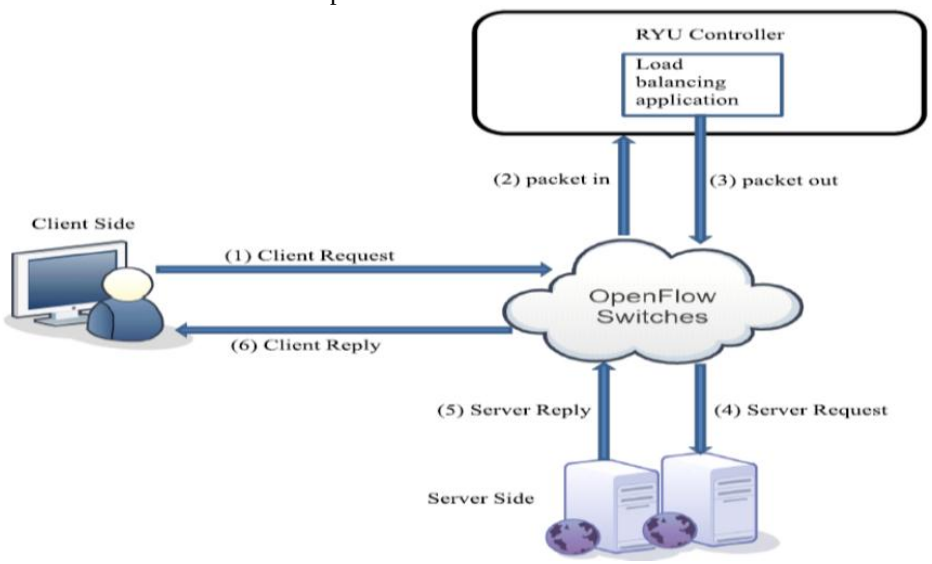

Fig (2) Load balancing algorithm operation steps.3.Results and discussion

In our environment, the network components composed of: OpenFlow switch, OpenFlow controller, load balancing application, hosts, and servers. In this experiment, Open vSwitch software is used in case of using Raspberry Pi and OVSK switch in case of using mininet; Ryu OpenFlow controller is used for controlling and managing the network flows with the aid of the load balancing application. There are three hosts used in this experiment as follows: one client generates Hypertext Transfer Protocol (HTTP) traffic and two HTTP servers reply to the clients' requests as shown in Fig (3) .

The network performance is measured in terms of throughput, number of errors, and response time for both algorithms. The response time refers to the time interval between generating HTTP request by a client and receiving the reply from the server. The simplest way to measure the request throughput is to send requests to the server at a fixed rate and to measure the rate at which replies arrive. The experiments are conducted using mininet installed on virtual machine (VMware) and a Raspberry Pi as OpenFlow switch. In our implementation, Ryu controller is used with OpenFlow protocol version 1.3; two laptop devices are used as a client and a controller respectively running Ubuntu 14.04 64-bit operating system and 3 Raspberry Pi kits model 2B V1.1 as two servers and OpenFlow switch running Linux operating system, called Debian-based Raspbian Kernel version 4.4. In this paper, Open vSwitch software is installed on one of the

Raspberry Pi kits to use it as a real-time low-cost OpenFlow switch. Open vSwitch is a multilayer software switch licensed under the open source Apache 2 license [19]. USB LAN dongles RD9700 model are used to extend the USB ports of Raspberry Pi OpenFlow switch to be connected to client and servers since Raspberry Pi has only one built-in Ethernet interface. The characteristics of the hardware devices used in our implementation are summarized in Table 1.

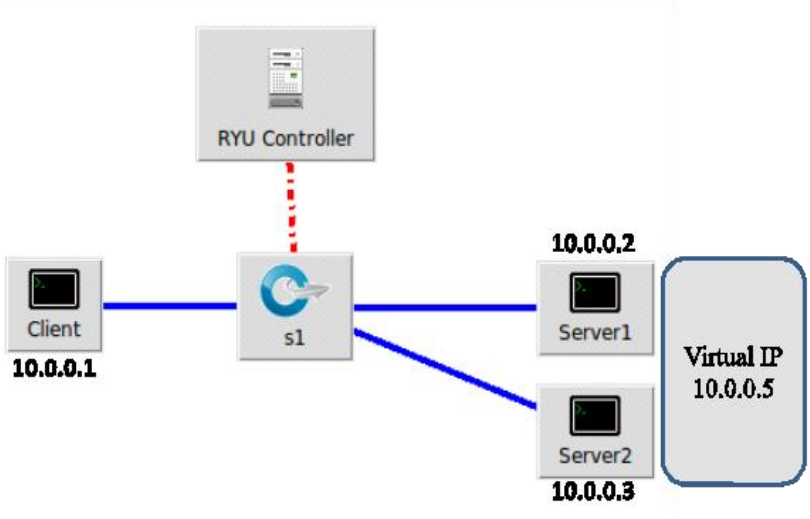

Fig (3) Network environment

Table (1) Hardware devices characteristics

Host name	Client	Controller	Raspberry Pi
CPU	Core i7 2.4GHz	Core i3 2.4Ghz	A 900MHz quad-core ARM Cortex-A7
Memory	8GB RAM	4GB RAM	1GB RAM
Additional Features			4 USB ports, one Ethernet port, and micro SD card slot

For our emulation environment, mininet version 2.2 is used to create 3 virtual hosts and one OpenFlow switch with the remote controller by using the following command:

```
$sudo mn --topo=single,3 --mac --
controller=remote --
switch=ovsk,Protocols=OpenFlow13
```

One of the three hosts is a client and the others are servers; the servers are running on port 80. In order to measure the web server performance, a software tool known as Httperf [20] is installed on the client. Httperf provides a flexible facility for generating various HTTP workloads. To measure the response time, requests are generated by using Openload tool [21] via the following command:

```
$sudo Openload http://virtual ip:80 (N)
```

where N is the number of concurrently clients requesting the web server. To make a host acts as a server running on port 80, the following command is executed on the host's CLI:

```
$sudo python -m SimpleHTTPServer 80&
```

In our experiment, a different number of connections are generated with different load sizes to request the virtual IP (service IP) with different request rates; therefore, HTTPerf tool is used to generate HTTP requests to the web server and then calculating throughput and number of errors using the following command on the client side:

```
$sudo Httperf --server="virtual ip" –num-
conns="number of connections" --rate="request
rate" --port=80
```

3. Results and discussion

In our scenario, the connection-based load balancing algorithm is compared with Random-based and Round-robin strategies in two environments: using mininet emulation environment with OVSK switch and Raspberry Pi low-cost OpenFlow switch. The network performance metrics are throughput, number of errors, and average response time of a web server.

A. Performance of the emulated scenario using mininet:

HTTPerf tool is used to measure the throughput and the number of errors occurred in our network. As shown in Fig (4), the throughput in case of using our proposed algorithm is better than Random-based and Round-robin strategies. The number of errors that were encountered during a test using Connection-based technique is fewer than Random-

based and Round-robin as the request rate increased as shown in Fig (5) .

The Openload tool is also used to measure the average response time; the average response time for the proposed technique is lower than the Round-robin and random strategies (0.98, 1.068, and 1.23 ms respectively).

B. Performance of the Raspberry Pi scenario

The performance measurements for the hardware implementation using a Raspberry Pi as OpenFlow switch supports the results obtained in the emulation as shown in Fig (6) . The average response times for the proposed algorithm, Round-robin, and Random-based are 2.852, 3.013 and 3.022 ms respectively. Fig (7) shows the number of errors using Connection-based algorithm is less than Round-robin and Random-based as the request rate increased. Hence, the network performance is better using our proposed technique because it distributes traffic across available servers based on least number of server's active connections. In contrast, using Random algorithm, the controller may select only one server occasionally which causes server overload and other servers will be unloaded. We observed that there is a threshold point in the request rate in case of using both Raspberry Pi and mininet. After this point, the throughput decreased and the errors increased extremely

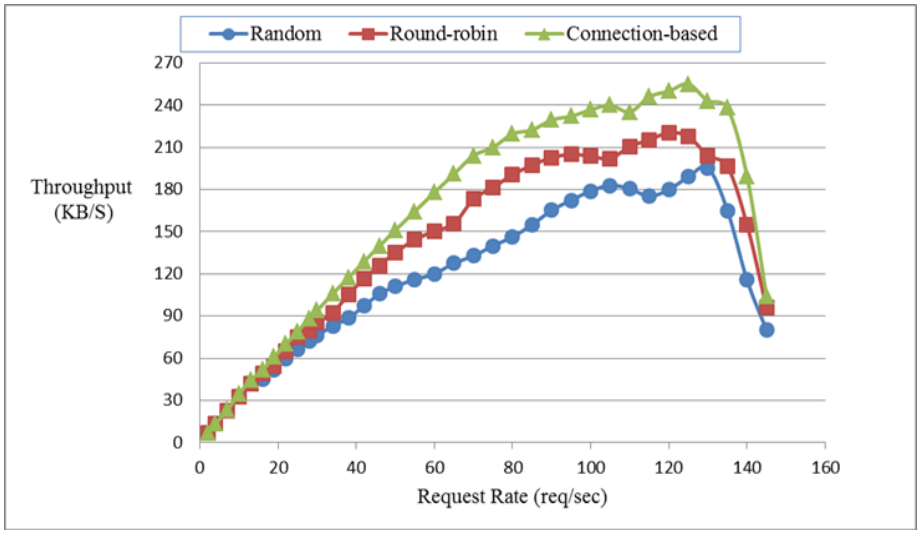

Fig (4) Request rate vs Throughput using mininet

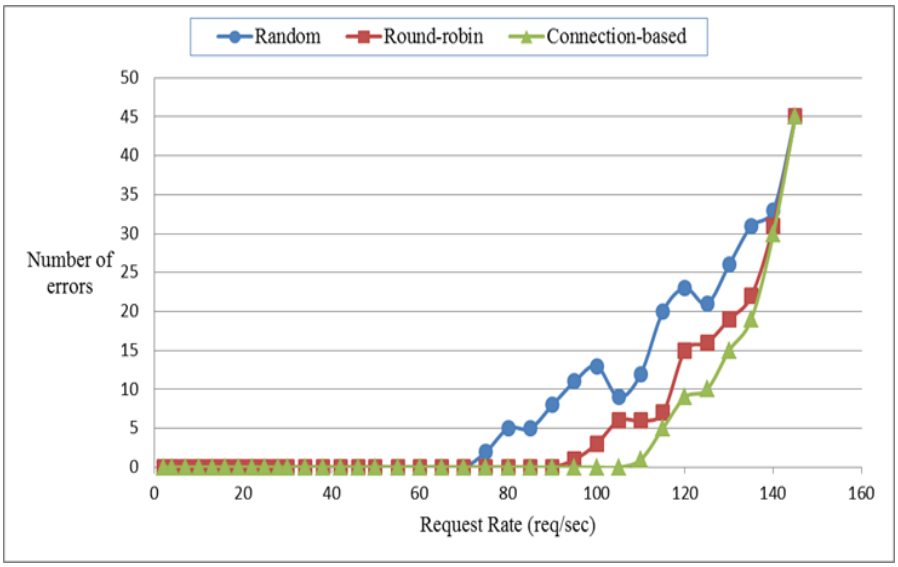

Fig (5) Request rate vs Number of errors using mininet

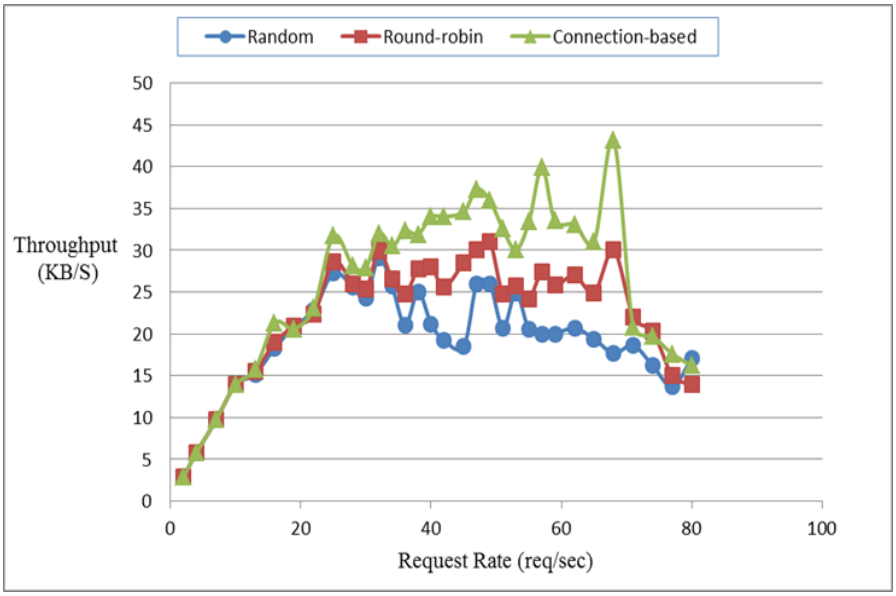

Fig (6) Request rate vs Throughput using Raspberry Pi

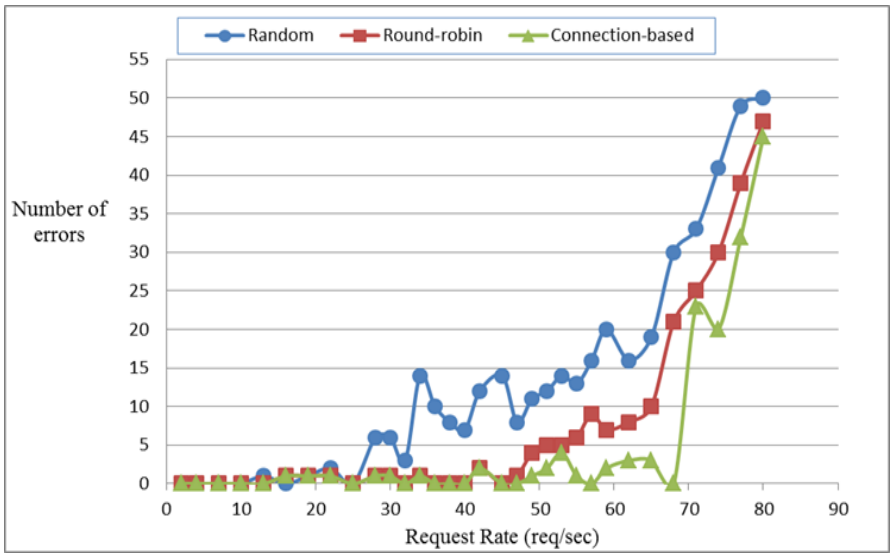

Fig (7) Request rate vs Number of errors using Raspberry Pi.

4. Conclusions

This paper presents an OpenFlow-based solution for servers' overload problem by implementing a server-based load balancing application in SDN environment instead of the traditional load balancer. In this paper, three load balancing techniques are compared and evaluated using mininet emulation tool and real-time Raspberry Pi testbed using Ryu controller. It is found that the proposed algorithm is better than the Round-robin and Random-based strategies in terms of throughput, number of errors, and response time.

5. Acknowledgment

This work has been partially supported by Scientific Research Fund at Benha University, Project "Evolution Toward 5G Cellular Wireless Networks: Reliability and Energy Efficiency Challenges".

References

[1] S.Kaur, J.Singh, K.Kumar, and N.S.Ghumman, "Round-Robin Based Load Balancing in Software Defined Networking," Computing for Sustainable Global Development (INDIACom), New Delhi, India,2015.

[2] D.Kreutz, F.M.V.Kreutz, P.E.Verssimo, C.E. Rothenberg, S.Azodolmolky, and S.Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14, 2015.

[3]"OpenFlow Protocol", available at, https://www.opennetworking.org/sdn-resources/openflow .

[4] "POX OpenFlow Controller", available at, https://OpenFlow.stanford.edu/display/ONL/POX+Wiki .

[5] "Ryu SDN Framework", available at, https://osrg.github.io/Ryu/

[6] "Trema", available at, https://trema.github.io/trema/.

[7] "The OpenDaylight platform", available at, https://www.opendaylight.org/.

[8] "Raspberry Pi price", available at, https://www.raspberrypi.org/blog/price-cut-raspberry-pi-model-b-nowonly-25/ .

[9] "HP OpenFlow switch price", available at, https://www.opennetworking.org/sdn-openflow-products?start=20.

[10] "SDN", available at, http://www.cse.wustl.edu/%7Ejain/cse570-13/ftp/sdn/index.html

[11] "Mininet", available at, http://mininet.org/.

[12] R.Khondoker, A.Zaalouk, R.Marx, and K.Bayarou, "Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers," World Congress on Computer Applications and Information Systems (WCCAIS), Hammamet, Tunisia, 2014.

[13]"Osrg/Ryu",available at, https://github.com/osrg/ Ryu.

[14]"Raspberry Pi", available at, https://www.raspberrypi.org.

[15] H. Uppal and D. Brandon, 2010. "OpenFlow Based Load Balancing," University of Washington, USA.

[16] M. Bindhu and G. P. Ramesh "Load Balancing and Congestion Control in Software Defined Networking using the Extended Johnson Algorithm for Data Centre," International Journal of Applied Engineering Research (IJAER), vol. 10, no. 17, pp. 12911, 2015.

[17] Y.Hu, W.Wang, X.Gong, X.Que, and S.Cheng, 2012 . "BalanceFlow: Controller load balancing for OpenFlow networks.", Cloud Computing and Intelligent Systems (CCIS), IEEE 2nd International Conference on IEEE, Hangzhou, China.

[18] Y.Li and D.Pan, "OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support," IEEE International Conference on Communications (ICC-2013), Florida, USA, 2013.

[19] "Open vSwitch/ovs", available at, https://github.com/Open vSwitch/ovs.

[20]"Httperf", available at, https://github.com/httperf/httperf.

[21] "OpenLoad", available at, http://openwebload.sourceforge.net/