Iraqi Journal of Information and Communications Technology(IJICT)
Vol. 4, Issue 2, August 2021
ISSN:2222-758X

EMULATION OF THE POX CONTROLLER AS A LOAD BALANCER

Sally A. Al-Hasnawi [1], **Mahmood K. Ibrahem** [2]
[1,2] College of Information Engineering, Al-Nahrain University, Baghdad, Iraq
{sally.karim, mahmoodkhalel}@coie-nahrain.edu.iq [1,2]
Received:20/11/2020, Accepted:22/4/2021

***Abstract*- By its nature, the server assessed in this study experiences bottlenecking, making load balancing an essential service within the network, particularly for users who increase the load. This paper investigates load-balancing related to software-defined networks (SDNs). Also, three load-balancing algorithms are evaluated and analyzed: the random algorithm (used as a default in Mininet), building and implementing the code of the round-robin algorithm, and building and implementing the code of the weight round-robin algorithm (which depends on two types of servers-namely, python and ruby servers). All evaluations were based on the tools used (i. e. , Ali tool, siege tool, and apache tool). The considered parts constitute three servers, one client, one pox controller, and one open virtual switch. The proposed topology needs to be connected to the pox controller-the OpenFlow protocol was utilized. The results show that the random algorithm performed better than the round-robin algorithm, specifically processing time and response time. Also, the round-robin algorithm had a better processing time and response time than the weight round-robin algorithm.**

keywords: **Software-defined network (SDN), Load balancing, random algorithm, Mininet, Round-robin algorithm, Weight round-robin algorithm, OpenFlow.**

I. INTRODUCTION

The software-defined network (SDN) is one of the most up-to-date computer networking paradigms today. This paradigm bolsters programmable interfaces by providing an active and suitable technique for organizing network traffic control [1]. The SDN differs from traditional networks. For example, whereas traditional networks involve tight coupling between network devices and the controller (Fig. 1), the SDN can decouple the control plane from the data plane [2]. The router and switch in an SDN can decide where to route newly arrived packages. The controller is isolated from network devices in the SDN to completely centralize the control. Unlike traditional networks, the SDN makes it easy to alter aspects of the network and check for and debug any problems that emerge. The SDN also includes advanced protocols, new functions, and fewer errors than traditional networks. Modifications occur only in the controller, where the edits are transferred to network devices. Also, the SDN is very flexible and can enlarge networks. A network's servers are often crammed with load due to increased user demand. Therefore, load balancing is considered vital for an SDN. The load must be balanced to improve the network's service ensure that quality of service (QoS) requirements are met. Load management uses the current infrastructure to balance the load, thus improving the QoS and satisfying customers [3]. Two main types of algorithms: static and dynamic, are relevant to this kind of environment. Static load balancing employs the round-robin algorithm, random algorithm, and weight round-robin algorithm [4]. Round Robin Algorithm distributes the requests in the following manner as it gives the first request to the first server, then the next request to the second server, and so on, it gives a request to the last server. If a request comes after that it returns to the first server to gives it this request and so the process continues. The random algorithm depends on the distribution of requests between servers in a random manner. Weight Round-Robin algorithm relies on its distribution method for the requests on the weight on each server. The server on which the weight setting is high will take more requests [4]. Dynamic load-balancing algorithms include Central Queue Algorithm and the

ISSN:2222-758X

Local Queue Algorithm. They differ from static algorithms in that the workload is distributed among the processors at runtime. The master assigns new processes to the slaves based on the new information collected. Unlike static algorithms, dynamic algorithms allocate processes dynamically when one of the processors becomes under-loaded. Instead, they are buffered in the main host's queue and allocated dynamically upon requests from remote hosts [4]. Static load-balancing algorithms are more stable than dynamic algorithms, and it is also easier to predict static algorithms' behavior. Accordingly, the current paper aims to evaluate static load-balancing algorithms, bearing in mind that the planned topology includes a Pox controller, a virtual switch, and three servers.

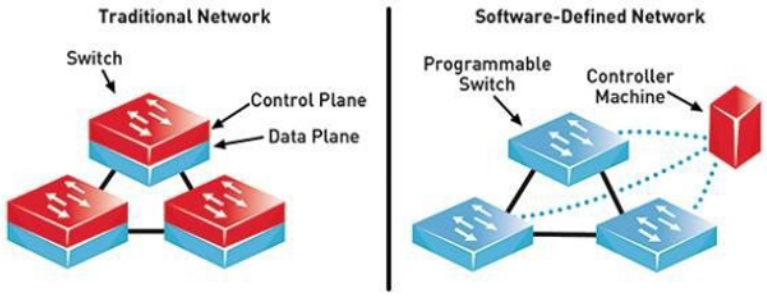

Figure 1: Comparison between SDN and traditional networks [2]

II. Related Work

Presented in [4] is the performance analysis of various load balancing algorithms based on different parameters, considering two typical load balancing approaches: static and dynamic. The analysis indicates that static and dynamic algorithms can both have advantages and weaknesses over each other. Deciding which type of algorithm to implement will be based on the type of parallel applications to solve. Little work has been on the subject of emulation of the pox controller as a load balancer. In [5], a two-level balancing solution related to the SDN networks is proposed, consisting of the usual balancing between servers and load balancing among network devices. The path is used to transmit data by the usual path load balancing protocol, producing much crowding on a single node. This kind of problem is solved by balancing the path load with the SDN's use alongside the distribution of the workload by adopting various parameters, [1] to correct the errors in the existing SDN controllers and networks they control and present a method of load balancing with the use of multiple servers. With this concept, the combination of a controller as a load balancer, together with widely-available architecture, is constructed mainly to develop the process marking's uptime structure. [6] Proposed an efficient load balancing technique by considering different parameters to maintain the load efficiently using Open Flow Switches connected to an ONOS controller. The flow requests are changed into a queuing model, managing the traffic propagation delay and controller capacity as two key factors impacting the multi-controller deployment. In [7], load balance is prepared using paths to connect load balancing by piecing together various physical links with several virtual links for transferring the required data. The source and the destination node, interrelated by a path, consist of various links and switches to join

those nodes. The current paper proposes to create a novel process of simulation to evaluate the algorithms of static load balancing employing a pox controller using three servers to distribute the load between them. In [8] , the authors propose a distributed load balancing algorithm using a weighted round-robin (WRR) mechanism. An analytical model of WRR is proposed to represent the task of the load balancing algorithm. Throughput and delay parameters are used to test the performance of the proposed algorithm. The results show that the WRR algorithm increases the availability of bandwidth and data transfer size. This approach decreases latency in SDN, their work based on an open-daylight controller.

III. THE CONTROLLER

At first, the control plane defines the regulation, which defines where the flow regulations on network instruments and SDN switches are attained to apply them on the data plane [9]. The control plane components are the software's device management and a theoretically integrated interface in the SDN network. With that, to handle several services, various frameworks shall be accessible in the controller. The North Bound Interface (NBI) is the accurate representation of the interface between the controller and the applications. The South Bound Interface (SBI) and networking equipment are regarded as the key parts of the data plane, playing the adapter's role between the networking devices and the controller. It works as a secure channel inside the switch that enables the flow's power to enter the switches flow tables. The procedure of administrating the whole process is mutually proactively or reactively completed in response to the existing packages. Transferring topology and flow is the main function of the controller to be applied in this area. Preparing organized queries for external ports related to packets messages is done by the Link Discovery Module. All through the shape of a communications packet, these test messages are returned to assist the controller in designing the network topology. The topology's director controls the topology itself, as the best paths among network nodes are defined by the decision module's paths. These paths design process enables the path construction to develop the various security policies and the Quality of Services (QoS). Along with queue administrators, the different information administrators are considered other key elements of the controller required to handle and process numerous outgoing and incoming queues. One of the major modules directly needed to connect with the data plane's flow tables is the Flow Manager (FM), relying on the southbound interface to make the required arrangements. Maintains that SDN controllers are attained in various settings, forms, and programs. There are several examples such as the following: firstly, Network Operating System (NOX), which is $C++$ -based and Python-based; secondly, Python Network Operating System (POX), which is Python-based; and thirdly, Beacon and Floodlight, both of which are Java-based [10]. Due to the increase in switches, a minimal latent change is shown in POX, where POX requires additional physical resources to accomplish its tasks effectively [11]. The Beacon controller is regarded as a system for releasing an app related to robust Java. Likewise, the Floodlight controller is a Java-based free-flow system that stems from the Beacon framework's origins. This controller is considered the most innovative free-flow controller, and it is marked by its simplicity to configure and its excellent performance. According to [12], OpenFlow, an open, standards-based communications protocol, is labelled with an example of device-based SDN. OpenFlow enables access to a network switch or router's forwarding plane over the network, which simplifies sophisticated traffic management, particularly for cloud and virtualized environments. The Open Networking Foundation (ONF) manages and standardizes the OpenFlow protocol, and it consists of SDN technologies related to the promotion of all together [13]. A major theme

of OpenFlow's aims involves separating the control plane from the data plane. In [14] and [15], the authors indicated that there are also several well-known efforts, such as having the control plane separated from the data plane and standardizing information exchange between the control and data planes, which are done by the ForCES proposed by IETF [16]. Fig. 2 illustrates the POX Controller.

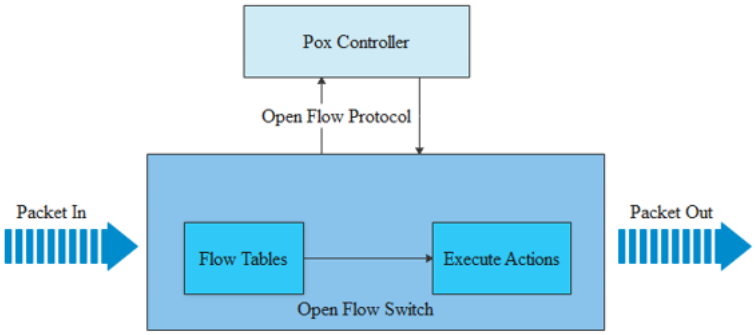

Figure 2: The pox controller [16]

IV. Proposed Work of Load Balancing

The proposed aim of this paper is to evaluate the load-balancing of three algorithms and who's better than the other: random algorithm, round-robin algorithm, and weight round-robin algorithm. The evaluation process depends on response time and processing time. Response time is the total time it takes from when a user sends requests until the user gets the response. The response time will calculate depend on the following equation:

$$R_t = L_t + P_t \tag{1}$$

WhereR_t represents response time, L_t represents latency time, and Pt represents processing time. Latency time is part of the response time and is calculated from the time the message spends on the wire. Processing time part of the response time and is calculated from the time it takes from the processing of the algorithm's code at the controller plus the distribution processing between the servers. When processing time or latency change, response time change [17] of three different load-balancing algorithms: a round-robin algorithm, a random algorithm, and a weighted round-robin algorithm. First, the python servers [18] were used and python servers supporting (HyperText Transfer Protocol) HTTP 1.1 [19]. So the tools that support this type of HTTP and calculate response time were searched. The tool that was found is called the Ali tool [19] that supports HTTP1.1 and calculates response time. Ali tool is a tool that shows the chart of the response time for each request received, and this tool is created by Linux and gives correct results without any manipulation or interference by humans with these results. Then another tool was used to calculate the response time results. This tool called Siege was used to calculate the response time results. This tool supports HTTP1.1, so it was suitable for use with the Python servers. Since the three algorithms' response time results are similar, which is impossible, another tool was used called the Apache tool, not the Apache Server. The Apache tool supports HTTP1.0 , so it cannot be used with Python servers,

so another type of the server was used called WEBRICK Server [20] that uses the Ruby language and supports HTTP1.0. Apache tool was used to calculate the response time and processing time for the three algorithms.

Start

Send Request Stage:

Step-1:

- A request is sent to the Internet Protocol (IP) of the load balancer by the client.
- The IP load-balancer sends the Internet Control Message Protocol (ICMP) echo-request packet to all the servers (broadcast) by the controller to check if they are alive or not. Check Servers and Resolve Address Stage:

  Step-2:
- Alive server: The server sends the ARP server reply to the controller.
- Otherwise: the ICMP echo request packet is re-sent to the servers by the controller.

  Step-3:
- In a random algorithm, if the address of the server is resolved, the controller forwards the request to one of the servers that reply randomly (which is the default in Mininet).
- In a round-robin algorithm, if the server's address is resolved, it distributes the requests in the following manner as it gives the first request to the first server, the next request to the second server, and so on. Then, it gives a request to the last server and if a request comes after that it returns to the first server to gives it this request and so the process continues.
- In the weighted round-robin algorithm, if the server's address is resolved, a weight is assigned to each server based on criteria chosen by the site administrator. The higher the weight, the larger the proportion of client requests.
- Otherwise, the ICMP packet is in a queue until resolving the servers address the servers.

  Step-4:

  Evaluate Random and Round-Robin Algorithm using the Ali Tool
- First, Python servers that support HTTP1.1 were used.
- The Ali tool that supports HTTP1.1 was used with these servers to calculate the response time for each request received, and the results are shown in the form of a chart without any manipulation.

  Step-5:

  Evaluate Random and Round-Robin Algorithm using Siege Tool
- First, Python servers that support HTTP1.1 were used.
- A Siege tool was used that supports HTTP1.1, Which Sends requests to Python servers and evaluates the response time. And because the results looked similar, and this is impossible, a new tool called Apache was used.

  Step-6:

  Evaluate Random and Round-Robin Algorithm using Apache Tool
- Activate WEBRICK servers that support HTTP1.0 to activate the apache tool.

- Use Apache tool, which supports HTTP1.0, which sends requests to the WEBRICK servers and evaluates response time and processing time.

  Step-7:

  Evaluate weight Round-Robin Algorithm using Siege Tool
- First, Python servers that support HTTP1.1 were used.
- A Siege tool was used that supports HTTP1.1, Which Sends requests to Python servers and evaluates the response time. And because the results looked similar, and this is impossible, a new tool called Apache was used.

  Step-8:

  Evaluate weight Round-Robin Algorithm using Apache Tool
- Activate WEBRICK servers that support HTTP1.0 to activate the apache tool.
- Use Apache tool, which supports HTTP1.0, which sends requests to the WEBRICK servers and evaluates response time and processing time.

  End

V. Proposed Topology

One client (10.0.0.4), one open virtual switch, three servers (10.0.0.1, 10.0.0.2, and 10.0.0.3), and one pox controller are the suggested topology components, Fig. 3 illustrates the investigated topology.

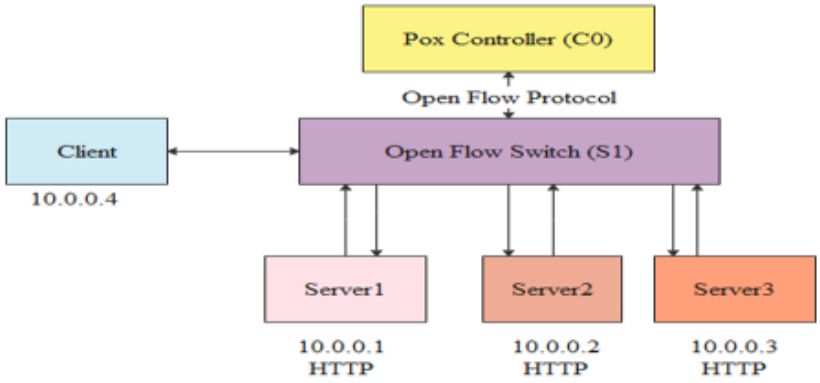

Figure 3: The investigated topology

VI. Implementation of The Proposed Work

A. *Random Algorithm*

1) Turn on the pox controller with the random algorithm load balancing. Activate the topology with one client, and three simple HTTP servers (python servers) support HTTP1.1.Use the Ali tool, which the tool that supports HTTP1.1 and calculates response time. Ali tool is a tool that shows the chart of the response time for each request received, and this tool is created by Linux and gives correct results without any manipulation or interference by humans with these

results. This tool sends 500 requests to python servers to evaluate response time to each received request. The chart shows the mean, that is, the average response time, which is 20ms. Fig. 4 the number of requests on the x-axis, while the response time is shown on the y-axis

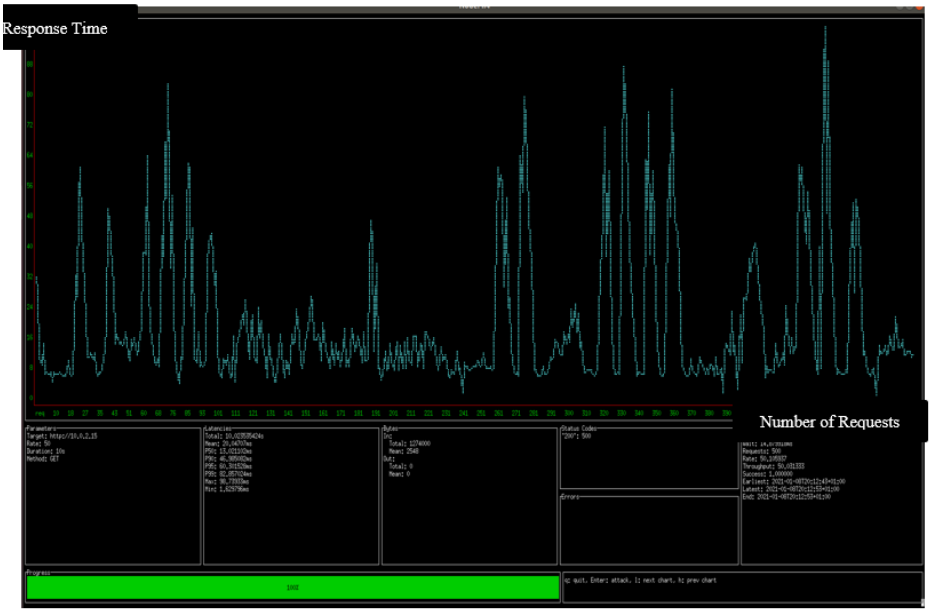

Figure 4: Number of requests vs. response time: ali tool, 500 requests, random algorithm

2) Turn on the pox controller with the random algorithm of load balancing. Activate the topology with one client and three simple HTTP servers (Python servers) supporting HTTP1.1. Use Siege tool (supporting HTTP1.1) , which sends 1000 requests. First, send one connection that is not concurrent, send two concurrent connections, then send three concurrent connections, send four concurrent connections, and finally send five connections concurrent to show the response time.
3) Turn on the pox controller with the random algorithm of load balancing. Activate the topology with one client and three WEBRICK servers that use the ruby language and support HTTP1.0. Use Apache tool (supporting HTTP1.0), which sends 1000 requests. First, send one connection that is not concurrent, send two concurrent connections, then send three concurrent connections, send four concurrent connections, and finally send five concurrent connections to show the response time and processing time.

B. Round-Robin Algorithm

1) Turn on the pox controller with the round-robin algorithm of load balancing. Activate the topology with one client and three simple HTTP servers (Python servers) supporting HTTP1.1. Use the Ali tool, which the tool that supports HTTP1.1 and calculates response time. Ali tool is a tool that shows the chart of the response time for each request received, and this tool is created by Linux and gives correct results without any manipulation or interference by

humans with these results. This tool sends 500 requests to the python servers to evaluate the response time for each request received. In this chart, we can see that the mean, which is the average response time (22ms), is more than in the random algorithm. Fig. 5 shows the x-axis, which is the number of requests, and the y-axis, which is the response time.

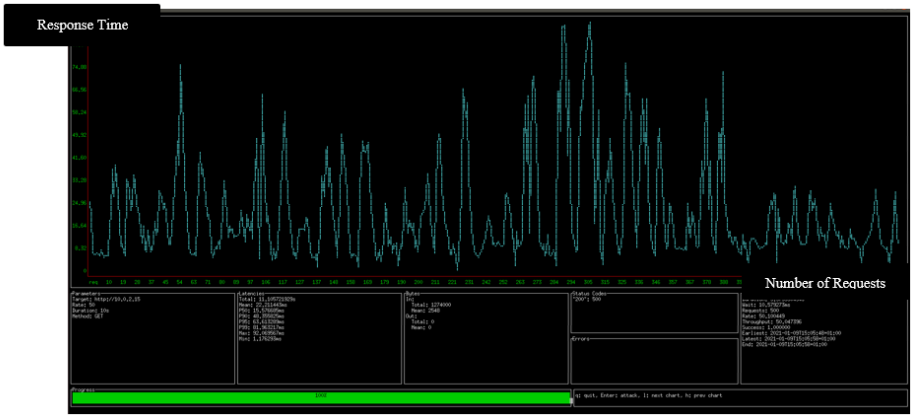

Figure 5: Number of requests vs. response time: ali tool, 500 requests, round-robin algorithm

2) Turn on the pox controller with the round-robin algorithm of load balancing. Activate the topology with one client and three simple HTTP servers (python servers) that support HTTP1.1. Use a sieging tool (support HTTP1.1) , which sends out 1000 requests. First, send one connection without the concurrence, then follow with two concurrent connections, then three concurrent connections, then four concurrent connections, and finally send five concurrent connections to record the response time.
3) Turn on the pox controller with the round-robin algorithm of load balancing. Activate the topology with one client and three WEBRICK servers that use the ruby language and support HTTP1.0. Use Apache tool (supporting HTTP1.0) , which sends 1000 requests. First, send one connection without concurrent, then send two connections concurrently. After that, send three connections concurrently, then send four connections concurrently, and finally send five connections concurrent to show the response time and processing time.

C. Weighted Round-Robin Algorithm

The weight for server one equals 1, the weight for server two equals 2, and the weight for server three equals 3. The percentage of weight for server one will be 16.66% cause (100/6=16.66%) , the percentage of weight for server two will be 33.32% cause (100/6=16.66%∗2=33.32%) , and the percentage of weight for server three will be 49.98% cause (100/6=16.66%∗3=49.98%). Fig. 6 below shows the weight distribution of the load balance for each server.

1) Turn on the pox controller with the weight round-robin algorithm of load balancing. Activate the topology with one client and three simple HTTP servers (python servers) that support HTTP1.1. Use a sieging tool (support HTTP1.1), which sends out 1000 requests. First, send one connection without the concurrence, then follow with two concurrent

connections, then three concurrent connections, then four concurrently connections, and finally send five concurrent connections to record the response time.

2) Turn on the pox controller with the weight round-robin algorithm of load balancing. Activate the topology with one client and three WEBRICK servers that use the ruby language and support HTTP1.0. Use Apache tool (supporting HTTP1.0) , which sends 1000 requests. First, send one connection without concurrent, then send two connections concurrently. After that, send three connections concurrently, then send four connections concurrent, and finally send five connections concurrent to show the response time and processing time.

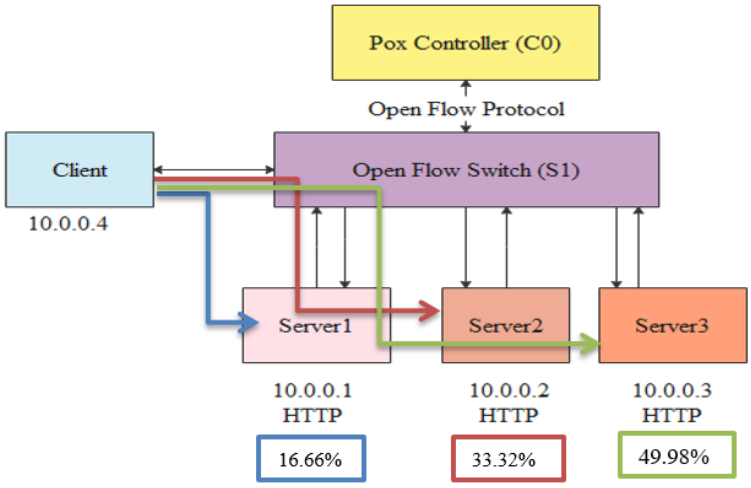

Figure 6: The investigated topology running the weight round-robin algorithm with weight distribution at server1 =16.66%, at server2=33.32%, at server3=49.98%

VII. Discussion

Software-Defined Network (SDN) is regarded as a new computer networking model where programmable interfaces are supported to present an appropriate way of adjusting the network traffic control. The vertically incorporated networking planes' elaboration is considered the key contribution. The new gap related to the SDN architecture, the control plane, and the data plane are used to upgrade the network's flexibility and manageability. Further, the controller decides where to direct the traffic. The data plane alongside the path directs the traffic to the next hop to the specific destination network. The SDN network uses the OpenFlow protocol to exchange data with the other elements network. Besides, the SDN integrates data flows, controls network policies, and network infrastructure. Accordingly, in this paper, evaluate and simulate three types of load balancing algorithms in the pox controller: random algorithm, the default in Mininet, building and implementing the code of the round-robin algorithm, building and implementing the code of the weighted round-robin algorithm. Compare the Random Algorithm results with Round Robin Algorithm and compare the results of the Round Robin algorithm with Weighted Round Robin Algorithm to find which one better than the other. The random algorithm is the default in the Mininet. Round Robin Algorithm and Weighted Round Robin Algorithm are building and used in this paper. In this work

to evaluate the algorithms, first use the Ali tool, which is a new load-testing tool for performing real-time analysis. Ali tool produces a chart that shows the response time for each request received. Ali's tool shows that in Random Algorithm, the average response time is 20ms, and in Round Robin Algorithm, the average response time is 22ms, which means that Round Robin Algorithm has a slower response time than the Random Algorithm. Then use the Siege tool, an HTTP load-testing utility that supports different concurrent requests with different URLs and supports HTTP1.1. Use a simple HTTP server that supports HTTP1.1. Siege tool measures response time. Siege tool first sends connection without concurrent gives response time 0.01sec, then sends 2 concurrent connections gives response time 0.01sec, then 3 concurrent connections gives response time 0.02sec, then 4 concurrent connections gives response time 0.02sec, and finally 5 concurrent connections gives response time 0.03 sec. The results show that Random Algorithm and Round Robin Algorithm and Weighted Round Robin Algorithm have the siege results, which is impossible, so decide to use another tool: the apache tool. The Apache tool is an open-source command-line program that works with any web server and supports HTTP1.0. The Python server, on the other hand, cannot be used because it supports the only HTTP1.1. Hence, we decided to use the WEBRICK server, which is written in the Ruby language and supports HTTP1.0. The Apache tool measures the response and processing times. First, it evaluates the response time in random and round-robin algorithms by sending the first connection without concurrent random response time 20ms and in round-robin 28ms. It then sends the second connection concurrent in random gives response time 23ms and in round-robin 32ms and then sends the third one concurrent in random gives response time 33ms and in round-robin 37ms. Subsequently, the fourth connection is sent concurrently in random gives response time 46ms and the value 52ms in the round-robin algorithm. The fifth connection is concurrent in random gives response time 66ms and 72ms in the round-robin algorithms, as shown in Fig. 7.

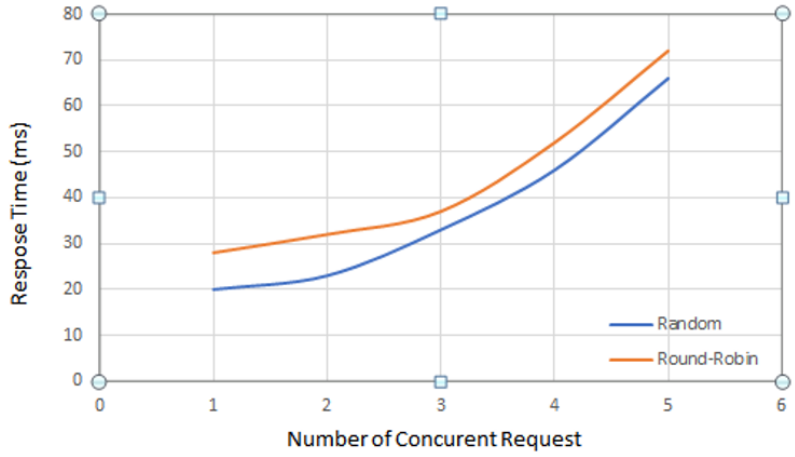

Figure 7: Measure the response time of the random algorithm versus the round-robin algorithm: apache tool, the total number of requests 1000

Afterward, the Apache tool evaluates the processing time in random and round-robin algorithms by sending the first connection without concurrent in random with the result 11ms and 20ms in the round-robin algorithm and then sending the

second connection concurrent in random with the result 17ms and 27ms in the round-robin algorithm, the third connection concurrent in random with the result 23ms and in round-robin 28ms, and then the fourth connection concurrent in random with the result 37ms and in round-robin 41ms, and then the fifth connection concurrent in random with the result 50ms and in round-robin 54ms, as shown in Fig. 8. We can conclude that the random algorithm is better than the round-robin algorithm because the code lines of the random algorithm are less than the round-robin algorithm for that when the CPU cycle processing line by line of the code of the random algorithm it takes less processing time also when the microcontroller of the controller multitasking this algorithm take less processing time than round-robin algorithm because the line of code less. Moreover, the mechanism of distribution requests between the servers of the random algorithm depends on distributing the requests randomly between servers without the need to take hard decisions to decide which server will receive the next request, which makes it less response time than the mechanism of the round-robin algorithm, which distributes the requests in sequential order of processes, coming back to the first when all processes have been given tasks. This requires more time to decide which server will receive the next request.

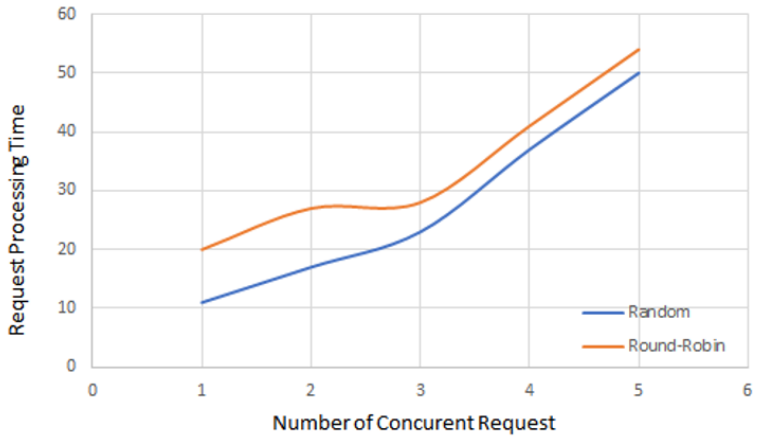

Figure 8: Measure the processing time of the random algorithm versus the round-robin algorithm: apache tool, the total number of requests 1000

This paper uses an apache tool to evaluate the response time in the round-robin algorithm and the weighted round-robin algorithm by first sending 1 connection (round-robin: 28ms, weighted round-robin algorithm: 36ms), then 2 connections concurrently (round-robin: 32ms, weighted round-robin: 36ms), then 3 connections concurrently (round-robin: 37ms, weighted round-robin: 44ms), then 4 connections concurrently (round-robin: 52ms, weighted round-robin: 55ms), and then 5 connections concurrently (round-robin: 72ms, weighted round-robin: 100ms), as shown in Fig. 9. Finally, the apache tool is used to evaluate the processing time in the round-robin algorithm and the weighted round-robin algorithm by first sending 1 connection (round-robin: 20ms, weighted round-robin: 21ms), then 2 connections concurrently (round-robin: 27ms, weighted round-robin: 28ms), then 3 connections concurrently (round-robin: 28ms, weighted round-robin: 32ms), then 4 connections concurrently (round-robin: 41ms, weighted round-robin: 45ms) , and then 5 connections concurrently

ISSN:2222-758X

(round-robin: 54ms, weighted round-robin: 82ms), as shown in Fig. 10. The results can conclude that the round-robin algorithm is better than the weighted round-robin algorithm because the round-robin algorithm has fewer code lines than the weighted round-robin algorithm, for that when the CPU cycle processing line by line of the code of the round-robin algorithm, it takes less processing time also when the microcontroller of the controller multitasking this algorithm take less processing time than weighted round-robin algorithm because the line of code less. Also, the mechanism of distributing the requests of the round-robin algorithm depends on passing out the requests in sequential order of processes coming back to the first one when all the processes have been given tasks. In contrast, in the weighted round-robin algorithm, a weight is assigned to each server based on the site administrator's criteria. The higher the weight, the larger the proportion of client requests, which causes overload on one server without the others, leading to increase processing time, especially when using the same type of hardware for all the servers because what's the benefit of fall the load on one server without the others and increase the response time and processing time. The weight round-robin algorithm will be a benefit in case use it with multicore.

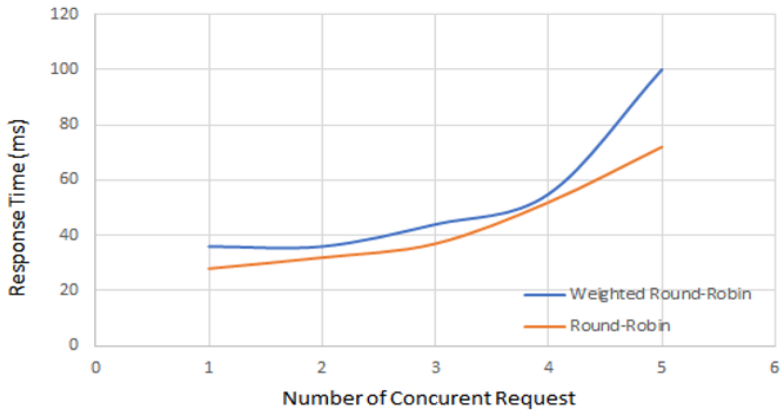

Figure 9: Measure the response time of the round-robin algorithm versus the weighted round-robin algorithm; apache tool, the total number of requests 1000

In [1, 4-8], use another type of controllers, such as the floodlight controller, ONOS, or Open daylight controller. The experiments related to demonstrating the survival time of a system are increased by using all of the resources with maximum utilization during a DDOS attack compared to the current balancing solution in SDN networks. Use multiple path strategies to improve the current rates of efficiency, reliability, and utilization by the fuzzy synthetic evaluation model. Work to increase up-time of the server when the prior controller fails to choose another controller. Evaluate the performance of the weight round-robin algorithm. This paper greatly addresses the dire need to use a pox controller to conduct a simulation of multiple servers' strategy (python, ruby servers) to evaluate response time and processing time, depending on multiple types of algorithms. Finally, the tests were performed on Ubuntu 16.04 virtual machine with 4.00 GB of RAM, with the processor 1.70 GHz, python 2 for the random algorithm, and python 3 for the other algorithms. Mininet 2.2.2 and pox 0.5.0 were also used.

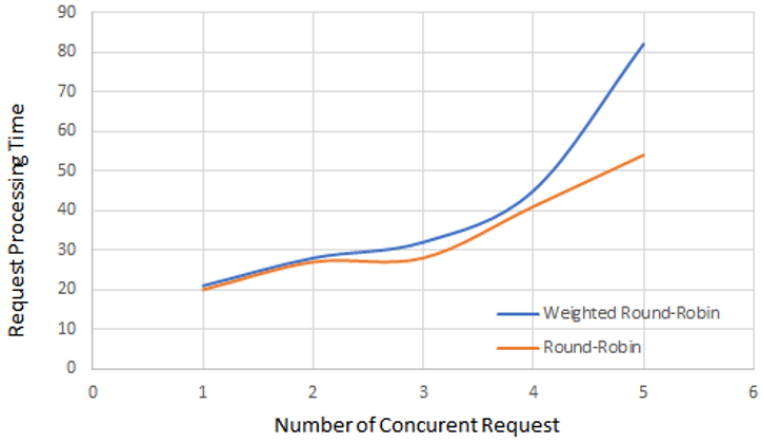

Figure 10: Measure the processing time of the round-robin versus the weighted round-robin algorithm; apache tool, the total number of requests 1000

VIII. Conclusion

1) This paper described using a random algorithm, round-robin algorithm, and weighted round-robin algorithm as load-balancing techniques for software-defined networks.
2) The code of the round-robin and weighted round-robin algorithms was building and implementing in this paper.
3) Evaluation using a simulation implemented in the Python programming language.
4) The simulation was undertaken using the Ali tool, which provides information about the response times for every request received in a chart to evaluate response times.
5) The Siege tool was used with a Python server. Because the siege tool shows the same response time results for the three algorithms, another tool that was used to evaluate is the Apache tool.
6) Apache tool didn't work in the python server due to its HTTP1.0 and python server support HTTP1.1. So a Ruby server was used to evaluate response times and processing times.
7) The siege tool is not accurate tool because it gives the same results of response time in the three algorithms, which is impossible.
8) According to the results, the random algorithm better than the round-robin algorithm, which is due to the less number of lines of code in the random algorithm, so when the CPU cycle processing line by line of the code of the random algorithm, it takes less processing time also when the microcontroller of the controller multitasking this algorithm take less processing time than round-robin algorithm because the line of code less. Additionally, the mechanism of Moreover, the mechanism of distribution requests between the servers of the random algorithm depends on distributing the requests randomly between servers without the need to take hard decisions to decide which server will receive the next request, which makes it less response time than the mechanism of the round-robin algorithm, which distribute the requests in sequential order of processes, coming back to the first when all processes have been given tasks. This requires more time to decide which server will receive the next request.

9) The results also showed that the round-robin algorithm is better than the weighted round-robin algorithm because the round-robin algorithm has fewer code lines than the weighted round-robin algorithm, for that when the CPU cycle processing line by line of the code of the round-robin algorithm, it takes less processing time also when the microcontroller of the controller multitasking this algorithm take less processing time than weighted round-robin algorithm because the line of code less. Also, the mechanism of distributing the requests of the round-robin algorithm depends on passing out the requests in sequential order of processes coming back to the first one when all the processes have been given tasks. In contrast, in the weighted round-robin algorithm, a weight is assigned to each server based on the site administrator's criteria. The higher the weight, the larger the proportion of client requests, which causes overload on one server without the others, leading to increase processing time, especially when using the same type of hardware for all the servers because what's the benefit of fall the load on one server without the others and increase the response time and processing time. The weight round-robin algorithm will be a benefit in case use it with multicore.

REFERENCES

[1] A. Rao, S. Auti, A. Koul, and G. Sabnis, "High Availability and Load Balancing in SDN Controllers" , International Journal of Trend in Research and Development, Vol. 3 (2), pp. 2394-9333, 2016.
[2] P. Cisar, D. Erlenvajn, and S. Maravic Cisar, "Implementation of Software-Defined Networks Using Open-Source Environment" , Tehnicki vjesnik, Vol. 25 (1), pp. 222-230, 2018.
[3] Mohammad Riyaz Belgaum, Shahrulniza Musa, Muhammad Mansoor Aalam, and Mazliham Mohd, "A Systematic Review of Load Balancing Techniques in Software-Defined Networking" , IEEE Access 7,Vol. 8, 2020.
[4] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma, "Performance Analysis of Load Balancing Algorithms" , World Academy of Science, Engineering and Technology International Journal of Civil and Environmental Engineering Vol. 2 (2) , 2008.
[5] Mikhail Belyaev, Svetlana Gaivoronski, "Towards Load Balancing in SDN-Networks During DDoS-attacks" , in the 2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC) , Moscow, Russia, 2014.
[6] Guoyanli, Xinqiang Wang, AND Zhigang Zhang, "SDN-Based Load Balancing Scheme for Multi-Controller Deployment" , IEEE Access 7, Vol. 7, pp. 39612-39622, 2019.
[7] V. Deeban Chakravarthy, B.Amutha, "Path Based Load Balancing for Data Center Networks Using SDN" , International Journal of Electrical and Computer Engineering (IJECE), Vol. 9 (4), pp. 3279-3285, 2019.
[8] Shashidhara B. Vyakaranal and Jayalaxmi G. Naragund, "Weighted Round-Robin Load Balancing Algorithm for Software-Defined Network" , Emerging Research in Electronics, Computer Science and Technology, Vol. 545, pp. 375-387, 2019.
[9] V. Varadharajan, K. Karmakra, U. Tupakula, and M. Hitchens, "A Policy-Based Security Architecture for Software-Defined Networks" , IEEE Transactions on Information Forensics and Security, Vol. 14 (4), pp.897-912, 2018.
[10] D .Kreutz, F. M. V. Ramos, P.Verissimo, Fellow, C. Esteve Rothenberg, Member, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey" , Proceedings of the IEEE, Vol. 103(1), pp. 14-76, 2014.
[11] Liehuang Zhu,, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani, Fellow, "Sdn Controllers: Benchmarking & Performance Evaluation", Available: http://arXiv.org/pdf/1902.04491.pdf, 2019.
[12] N.McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Cott Shenker, Jonathan Turner, "OpenFlow: Enabling Innovation in Campus Networks" , ACM SIGCOMM Computer Communication Review, Vol. 38(2), pp. 69-74, 2008.
[13] A. Neeraja, Dr. N. Chandra Sekhar Reddy, Mukund, "Improving Network Management with Software Defined Networking" , International Journal of Science and Research (IJSR), Vol. 3 (7), pp. 1-4, 2012.
[14] Ali Akbar Neghabi1, Nima Jafari Navimipour, Mehdi Hosseinzadeh, Ali Rezaee, "Nature-Inspired Meta-Heuristic Algorithms for Solving The Load Balancing Problem in The Software-Defined Network" , International Journal of Communication Systems, Vol. 32 (4), pp. 3875, 2017.
[15] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework" , Apr. 2004, RFC 3746. [Online] Available: http://www.cisco.com/en/US/solutions/collateral/ ns341/ns525/ns537/ns705/ns827/white-paper-c11-481360.pdf.
[16] W. Xia, Y. Wen, C. Heng Foh, D. Niyato, and H. XieXia, "A Survey on Software-Defined Networking" , IEEE Communications Surveys and Tutorials, Vol. 17 (1), pp. 27-51, 2014.
[17] Zahid Iqbal, "Latency and Processing Time and Response Time" , 2014. [Online] Available: http://www.google.com/amp/s/testingsorttricks.wordpress.com /2015/06/10/latency-and-processing-time-and-response-time/amp/.
[18] Kishna, "Python 2 vs Python 3: Key Differences" , 2021, [Online] Available: https://www.guru99.com/python-2-vs-python-3.html.
[19] Chandan Kumar, Adnan Rehan, "How to Perform Web Server Performance Benchmark" , 2015, [Online] Available: https://www.google.com/amp/s/geekflare.com/web-performance-benchmark/amp/.
[20] Drew Mclellan, Rachel Andrew, "Let's Build a Simple HTTP Server" , 2019, [Online] Available: https://noti.st/esther/c9GuGJ/slides%20The%20server%20code.