DOI: https://doi.org/10.33103/uot.ijccce.20.3.6

An Investigation of Using Traffic Load In SDN Based Load Balancing

Methaq Khamees Faraj[1], Ahmed Al-Saadi[2], Riyadh Jabbar Albahadili[3]
[1,2,3]*Department of Computer Engineering, University of Technology, Baghdad, Iraq*
120625@student.uotechnology.edu.iq, 120027@uotechnology.edu.iq, 120082@uotechnology.edu.iq

Abstract— *The number of devices connected to networks and the internet such as the Internet of Things, machine to machine, social media or speech traffic, etc., are rapidly increased that results in a huge amount of traffic. This leads to congestion that increases packet loss and reduces system performance. Therefore, a single server cannot handle this traffic and need to use some approaches to optimize network performance. The use of a load balancer to distribute network traffic among multiple servers could minimize the load on a single server, provide availability, scalability, and enhance network performance. A load balancer in a traditional network is a dedicated hardware device that is expensive, close vender, and non-programmable. A load balancer contains few algorithms that network engineers cannot change or create a new one. In contrast, Software Defined Network (SDN) that utilizes load balancer is programmable (hardware independent) and more agilely.*

The objective of this investigation is to implement the Least packet load algorithm, which is used in the traditional load balancer, using an SDN-controller Python Network Operating system (POX) in order to distribute load among servers. Moreover, it discusses some research opportunities that this work introduces to improve load balancing in SDN. This work is validated through extensive simulations and emulations that compare the proposed algorithm with four of the most widely cited schemes. The results indicate that the proposed algorithm improved network performance and achieve up to 21% increase to system throughput compared to other benchmark approaches.

Index Terms— *Software Defined Networking, OpenFlow, Load Balancing, POX.*

I. INTRODUCTION

With the growth of internet services, a progressive amount of transmission data causes more traffic among network devices. It leads to congestion and loss of information [1]. Traditional networks do not have centralized control, each element (routers, switches) in the network has its own control plane. Therefore, each network device forwards traffic according to its configuration like media access control learning (MAC) of forwarding tables and data planes [2].

Since traditional network devices are decentralized, a lot of administration work is required to manage and reconfigure devices separately which causes inconsistency in the network. With the advent of a new paradigm known as Software Defined Networking (SDN), the control plane is shifted from decentralized form as in traditional network devices to a centralized one. This new architecture simplifies equipment designed in data plane which leads to a decline in the cost of the switches manufacturing and reduces the necessary efforts in network management by enabling automation management via programmability [3].

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

SDN consists of three layers; infrastructure layer which contains network devices (routers and switches) that support SDN protocol (OpenFlow, NetFlow, sFlow [4]), control layer which represents network brain that is responsible for specifying the path of data packets and applying restrictions on them and lastly application layer where innovative applications are created like (load balancer, firewall … etc.). The Connection between the control layer and the data layer is called southbound an Application Program Interface (API) which done via a protocol like OpenFlow, while the application layer and control layer implemented by northbound API which provides abstract views of the network shown in Fig.1 [5]. A load balancer is a technique that distributes network traffic among multiple servers to maximize network resource utilization including decreased response time and increased bandwidth [3]. The traditional network load balancer is very expensive and closed by the vendor [6], in contrast, the software load balancer is open and cheaper than hardware but its efficiency depends on the installed server specification, in addition to operating system compatibility (update and change) [7].

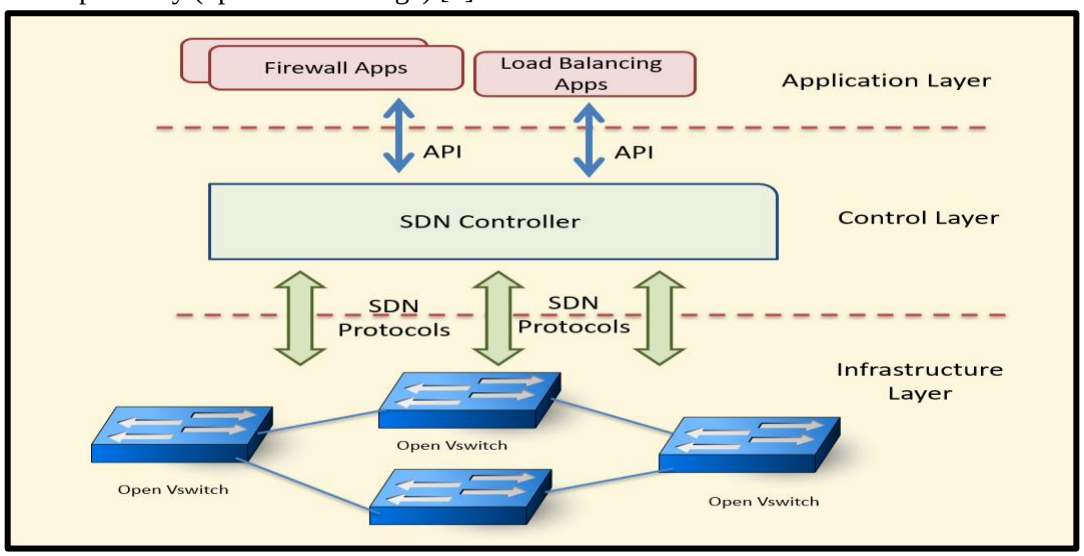

FIG1: SDN ARCHITECTURE [5]

This paper employed a real SDN controller (POX) [8] to control the network load by redirecting the client's requests to servers that currently receive the fewest packets. Evaluation is done through emulation and the results are compared with a number of the most popular algorithms used in the load balancer. The results show that the proposed algorithm is more efficient especially in heavy load of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) traffic.

The rest of the paper is organized as follows: Section II highlights the related work; section Ⅲ explains SDN-based Least Packet Load Balancer; section Ⅳ discusses performance evaluation; while section Ⅴ consists of the conclusion and future work.

II. RELATED WORK

In 2008 at Stanford University, McKeown et al. proposed the first and foremost SDN standard known as OpenFlow [9]. OpenFlow protocol acts as a vital role in SDN architecture; it allows innovation in the network, and it has one or more flow tables that used to redirect data forwarding in switches and routers [10].

Flow tables in each OpenFlow switches have flow entries; each entry determines how the incoming packets manipulate and forward to the desired destination as shown in Fig.2

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

[11]. Load balancing in SDN based networks can be categorized based on the algorithm utilized in distributing traffic. The simplest algorithm is a Random load balancing strategy [5], it forwards traffic among servers randomly without considering any Quality of Service (QoS) parameters. This strategy has some limitations in which one server may be overloaded.

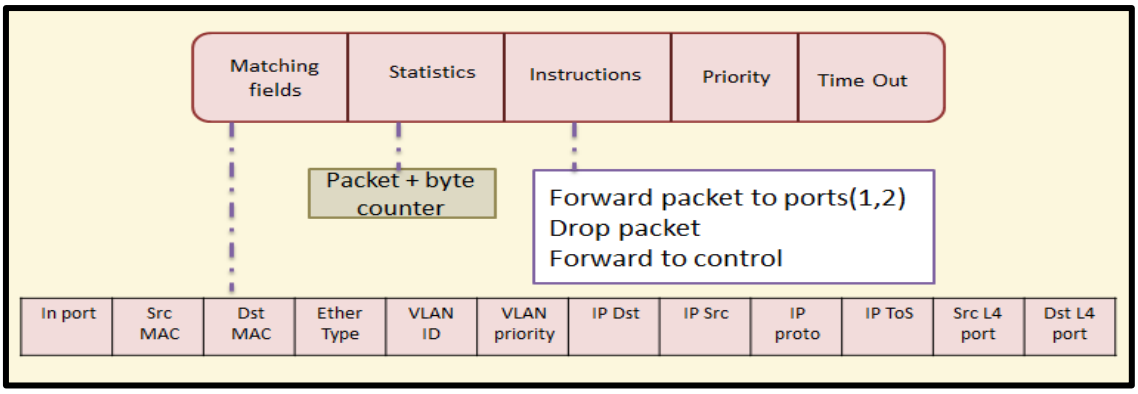

FIG. 2. FLOW ENTRY [11]

The second approach is to employ a round-robin algorithm to distribute the traffic evenly among nodes. This strategy is one of the most popular algorithms used in load balancing due to its simplicity [12-14]. This algorithm doesn't consider any QoS parameter on the servers that may lead to forwarding traffic through poor connections and low bandwidth links.

Another popular approach in load balancing is weighted round-robin [15-17], in this algorithm some servers can get a higher share of the total traffic. This algorithm has efficiently used heterogeneous servers, different link quality, or based on security restrictions but this algorithm needs to set the weight manually while in the real-world scenario network environment may change during the runtime. Least connection load balancing strategy directs flow to the server which has a minimum number of active transactions [18-20]. This algorithm distributes workload equally among servers and requires robust servers when the number of clients and traffic ramp-up. The least bandwidth algorithm distributes traffic dynamically in which traffic forwards to the server with the lowest network traffic consumption [21]. Its gathers bandwidth information about each server in load balancing and take decision based on gathered information, thus, it's more accurate but harder to implement. Another approach is the Internet Protocol Hash Load balancing (IP hash) [22]. This algorithm checks the incoming packet and matches with Internet Protocol (IP) in the controller's log, if it is matched then updates the flow table otherwise it is redirected to a new server. This algorithm adopts the IP of traffic and does not take in it is a consideration load of servers, so it may result in unfair distribution.

Moreover, An SDN-aided mechanism for web load balancing based on server statistics (SD-WLD) [23] algorithm chooses the best server based on switch port traffic, which is counted in the number of received bytes, and response time of the server. despite this work enhances the throughput in lower response time, but it cannot be used in redirect traffic based on the service port since it does not handle packet header. Another algorithm is Session Initiation Protocol (SIP) load balancing based on SDN [24]. This approach uses SDN to offer a new architecture for SIP networks which is easy to configure and change. SIP networks consist of agents that make requests and servers which handle those requests. The algorithm distributes traffic among servers based on the least number of requests as a

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

server load. this work does not take in its consideration the size of the requested data, so it may make an unfair load distribution.

This work handles the limitation in previous methods by parsing the packet header and calculating the load on each server based on destination IP. SDN-based Least Packet Load Balancer (SLPLB) strategy is used efficiently in heterogeneous and homogeneous servers because it considers the current load on the server. This approach provides a research basis for many other applications like Denial of service, firewall, or forwarding packets based on some specific criteria. This approach parses the packet header which enables the network engineer to develop policies based on the information acquired from the packet header.

III. SDN-BASED LEAST PACKET LOAD BALANCER

SDN load balancer is a software-based that can be programmed based on network requirements. The efficiency of the load balancing is based on the technique that used to satisfy the network performance enhancement via resource utilization, minimize response time and reducing the overload [25].

In this work, an investigation that utilizes traffic in load balancing to develop an algorithm called SDN-based least packets load balancer (SLPLB). In this algorithm, a server is selected based on the number of packets forwarded to that server. The server selecting procedure is calculated by analyzing the statistics of each stream on a switch port; the number of packets on each port is obtained as:

$$Min\_L(S_d) = Min_1^n(S_i) \qquad 1 \leq i < n,\ 1 \leq d < n \qquad (1)$$

Where S is a pool of available servers for the client to use and the total number of servers are n. Min_1^n is the function of calculating load for incoming packet to switch and i is index of available server where $1 \leq i < n$. $Min\_L(S_d)$ is a minimum load of selected server, d is the index of least load selected server.

The OpenFlow connection starts by performing an initial handshake between the controller and switches, after that the controller finds out about the existence of the switches. The controller discovers how the switches are interconnected in topology via the link layer discovery protocol (LLDP) packet. The controller sends an OPFT-FLOW-MOD message that order switches to resend any received LLDP to the controller. After that, a PACKET-OUT message containing LLDP as payload will be sent to the switches by the controller. Each switch resends PACKET-OUT to all ports but the incoming one. When a switch receives the LLDP packet, it replies to the controller with an OPFT-PACKET-IN message. The topology is discovered through this mechanism.

After the POX controller discovers the topology (switches, links, and hosts), the controller will initiate the traffic in a random fashion because there is no history of the load on servers at startup and they all have the same load. After that, to collect load statistics on each port POX sends a "Flow- statics -request" to the switch every 14 seconds (this number is selected empirically). OpenFlow switch responds to the controller via Flow-statics-received. Flow-statics-received has been modified to obtain the total number of packets (load) that each server process. Then, the load balancer selects the server with the lowest number of packets and the controller prepares the appropriate flow rules by OPFT-FLOW-MOD message for the server specified in the switch to send the upcoming traffic as shown in the Fig. (3 and 4).

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

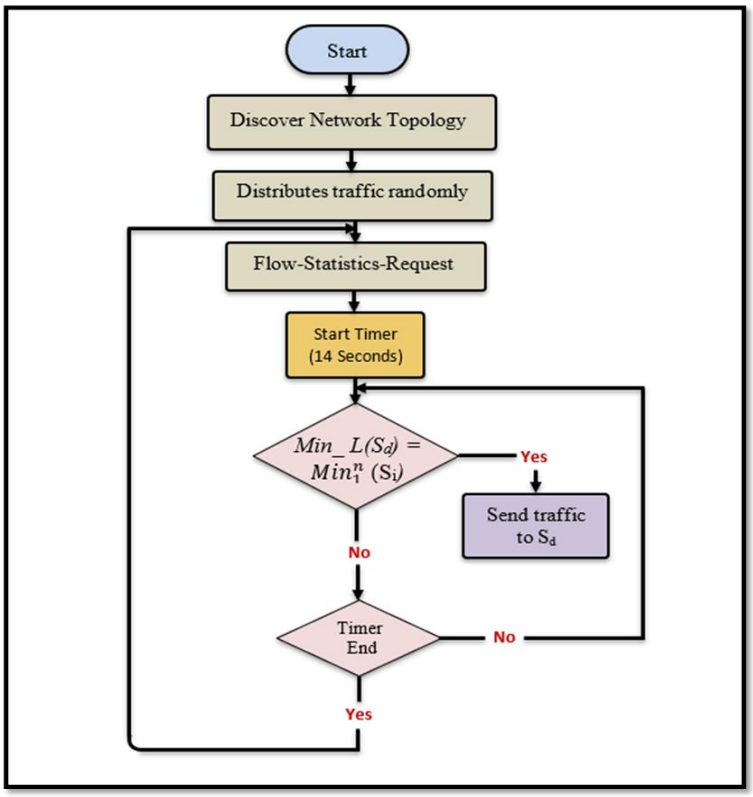

FIG. 3. SLPLB FLOWCHART

```
_____***** SLPLB_Algorithm******_____

//input
 Input: ServersIP with  --servers=ip1, ip2, ….
 server=[ip1,ip2,...]
//initialization
 Update_time = 14
 load[] = 0
 last_update_time = time.time()
//Process
 if time.time() - last_update_time > Update_time
   load[] := 0
   last_update_time = time.time()
 //collect load statictics on each server
   load =Flow-statics-received()
 else
   While (time.time() - last_update_time < Update_time)
   Do
      dest_server = SendTraffic(load)
//update flowtable in OpenFlow protocol to forward traffic
//to the selected server
   OPFT-FLOW-MOD(dest_server)
  END While
//Functions
 //Flow-statics-request and Flow-statics-received are two
 //events in POX.To obtain load statictics on each server
 //Flow-statics-received is reprogrammed.
 Function Flow-statics-received()
   if Dest_IPAddr in ServersIP
       load[Dest_IPAddr] := load[Dest_IPAddr] + Packet_count
    return load
 EndFunction
 Function SendTraffic(load)
// for all servers associated with controller
     bset_server=server[0]
     for X := 1 to Length(server)-1
          if load[bset_server] > load[server[x]]
            bset_server:=server[X]
          return bset_server
 EndFunction
```

FIG. 4. SLPLB ALGORITHM

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

IV. PERFORMANCE EVALUATION

In this section, SLPLB is evaluated using Mininet [26] tool emulator which is very widely employed in SDN research and POX controller. The proposed algorithm is compared with Random, Round-robin, Weighted Round-robin, least connection and Least bandwidth consuming in terms of throughput.

A. SIMULATION SETUP

Mininet is used to emulate the OvSwitch [27] that supports OpenFlow [10] protocol in order to evaluate the proposed SDN based load balancing algorithm. Mininet emulation is installed on oracle virtual box version 6.0.14 that is used to create network topology utilized in this paper. A real programmable SDN controller (POX) is used as a remote controller in which the proposed algorithm is deployed. The emulated network utilizes OpenFlow protocol using remote POX controller on port: 6633, three servers and a different number of clients implemented in three scenarios (6,15,30) to test the performance of the proposed method in light, medium and high load as shown in Fig. 5. Internet Performance Working group (Iperf) [28] tool is utilized to generate TCP and UDP traffic from clients to servers and then calculate the throughput on each server.

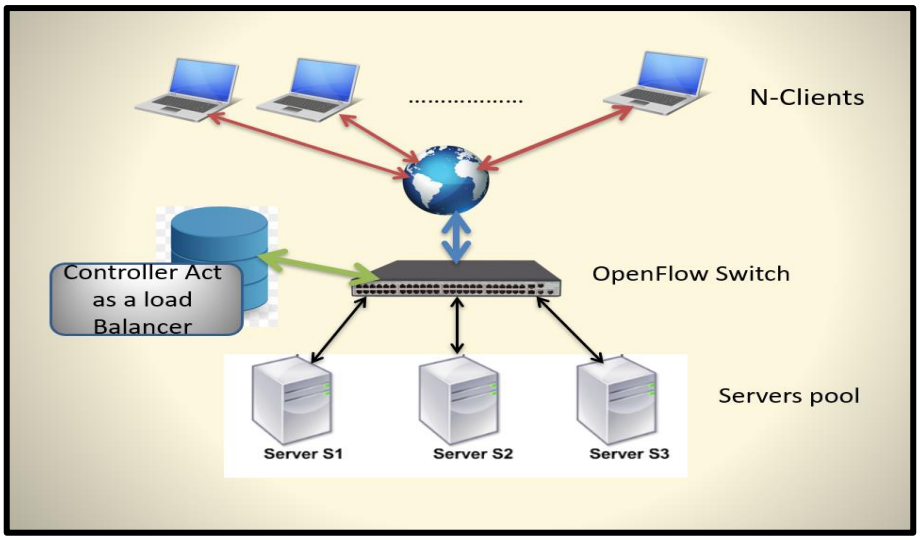

FIG. 5. NETWORK TOPOLOGY

B. EVALUATING AND VALIDATING RESULTS

The proposed algorithm in this paper is evaluated and compared with random, round-robin, weighted round-robin, least connections, least bandwidth which are one of the most cited algorithms in this field. The results are compared in terms of throughput under various amounts of loads for each algorithm. This paper utilized an Analysis of Variance Statistical (ANOVA) to validate our work similar to the work in [29]. ANOVA is a statistical tool used to verify that the compared algorithms are statistically different using F > Fcrit.

F, Fcrit, and P are parameters of ANOVA; where F is a comparison of the variation between sample means and the variation within sample means, Fcrit is the value extracted from analysis variance table, and P is the probability of the difference happened by chance. The acceptable value for P is less than 0.05. If F is greater than Fcrit then the null hypothesis is rejected at the 0.05 significance level and the throughput samples mean are significantly different [29]. Fisher's least significant difference (LSD) is used to show whether the proposed algorithm (SLPLB) achieves higher throughput. LSD value is calculated using Eq. (2).

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

$$LSD_{A,B} = t_{(0.05/2,DFW)}\sqrt{MSW.\left(\frac{1}{n_A}+\frac{1}{n_A}\right)} \quad (2)$$

Where t is critical value for degree of freedom associated with mean square variance (MSWithin) and n is a number of samples [30]. Table Ⅰ and Ⅱ show the ANOVA and LSD results for each scenario respectively.

TABLE Ⅰ ANOVA RESULT

Number of transmission nodes	ANOVA Test		
	F	F_{crit}	P < 0.05
6Hosts_TCP	22.92016	2.266062	1.22E-17
6Hosts_UDP	23.70836	2.266062	3.86E-18
15Hosts_TCP	13.37687	2.239486	5.9E-12
15Hosts_UDP	11.95593	2.239486	1.05E-10
30Hosts_TCP	49.71002	2.226649	3.3E-44
30Hosts_UDP	25.29412	2.226649	1.58E-23

TABLE Ⅱ LSD RESULT

Number of transmission nodes	Throughput Average for the Networks (Mbps)						LSD
	SLPLB	***Random***	***RR***	***WRR***	***LBW***	***LC***	
6Hosts_TCP	56.581	43.345	76.467	55.751	55.847	56.607	19.547
6Hosts_UDP	45.193	33.037	63.58	38.62	42.093	40.56	18.290
15Hosts_TCP	53.027	34.647	38.111	39.793	41.100	38.423	11.840
15Hosts_UDP	50.955	34.412	35.573	37.189	38.971	37.829	11.858
30Hosts_TCP	39.535	23.861	25.54	21.883	34.518	30.886	4.605
30Hosts_UDP	45.981	29.196	35.092	34.686	37.912	36.714	5.840

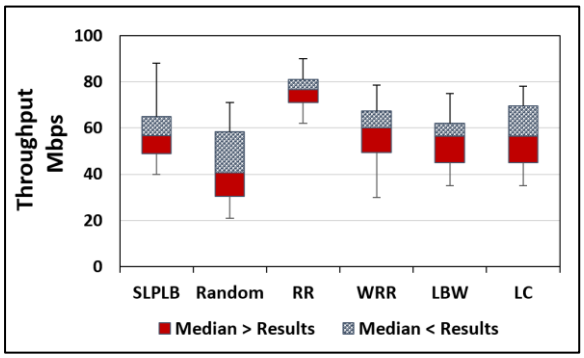

FIG. 6. THROUGHPUT AVERAGE OF 6 TCP CLIENTS

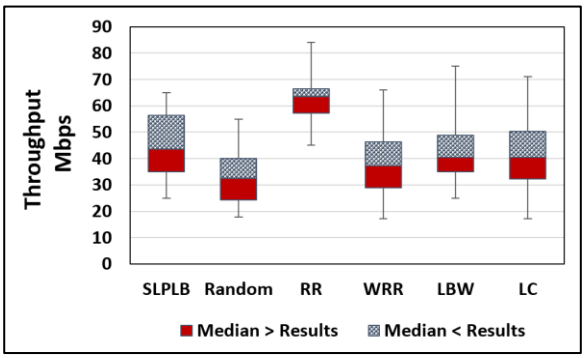

FIG. 7. THROUGHPUT AVERAGE OF 6 UDP CLIENTS

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

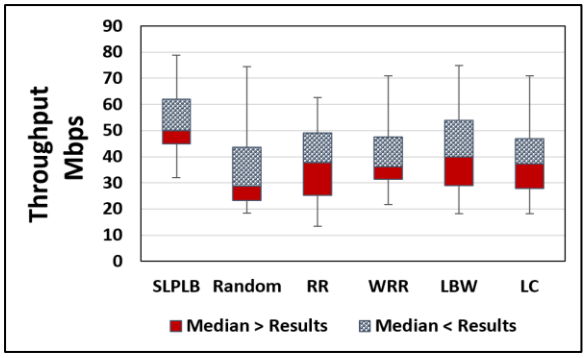

FIG. 8. THROUGHPUT AVERAGE OF 15 TCP CLIENTS

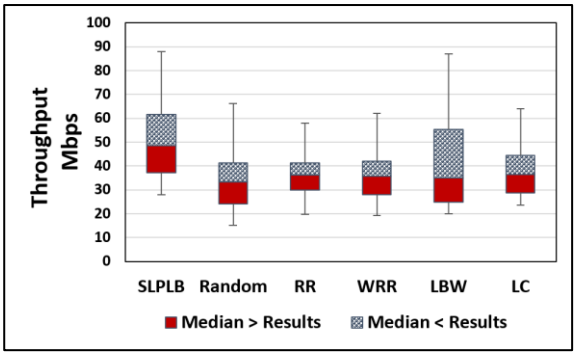

FIG. 9. THROUGHPUT AVERAGE OF 15 UDP CLIENTS

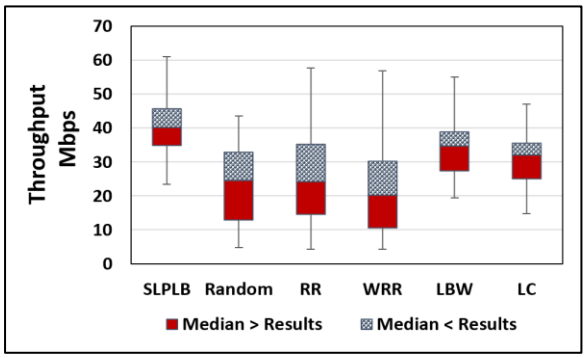

FIG. 10. THROUGHPUT AVERAGE OF 30 TCP CLIENTS

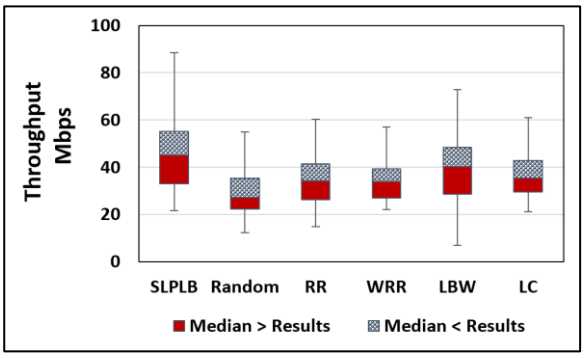

FIG. 11. THROUGHPUT AVERAGE OF 30 UDP CLIENTS

A number of different throughput results are generated in each scenario for each algorithm. The average value of these results is calculated for each algorithm (SLPLBavg, Randomavg, RRavg, WRRavg, LC_{avg} and LBW_{avg}). If the absolute value of (SLPLBavg – RRavg) is greater than the LSD value, then the two averages are statically different. LSD for SLPLB avg is compared with all of the benchmark algorithms to validate that the proposed algorithm is causing the statistical difference that ANOVA test shows. Fig. (6-11) show the results of the proposed algorithm compared with the benchmark and represented by Box and Whisker graph. In the Box and Whisker, Box is divided by the median so it is possible to show the average throughputs higher and lower than the median while Whisker represents the maximum and minimum values. In this graph, data is divided into four quartiles. The first quartile represents 25% of data which starts from the lower value and called Q1. Q2 which represents the second quartile has the percentage from 25.1 up to 50%, which is the median, while the third quartile above the median (Q3) has a percentage from 50.1% up to 75% and the last one (Q4) represents the highest quartile of data up to the maximum value. For example, figure 10 illustrates the SLPLB throughput results between 46 and 35 Mbps while the benchmark algorithms achieve throughput between 39 and 10 Mbps (with an increase of 21%). More than 50% of the median relatively throughput of SLPLB is greater than other benchmarks. Furthermore, this figure shows the proposed algorithm is more consistent since it is less varying while RR is less consistent with high load on the network. In fig. 7 illustrates the Round Robin algorithm works better than the proposed algorithm when the load on servers is not high.

V. CONCLUSION AND FUTURE WORK

This paper investigated and implemented the SDN-based Least Packet Load Balancing method in the SDN load balancer through three scenarios. SLPLB parses the incoming packet in each switch to estimate the current load of serve. As a result, this permits the load balancing algorithm to dynamically select the server. The suggested SDN based algorithm outperforms the benchmark algorithms especially

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

when the load is high on the network. The suggested model provides the foundation for future research on load balancing by parsing the traffic header and redirect traffic to specify services like email, web, and video conference based on service port in the packet header. Another future research direction for this work is to develop this algorithm to avoid distributed denial of service through IP filtration. Another future research is to implement a test bed with a multi-controller system instead of a single controller to avoid single point failure in the network.

REFERENCES

[1] J. Saisagar, D. Kothari, R. Kothari, and V. Chakravarthy, "SDN enabled packet-based load-balancing (PLB) technique in data center networks," ARPN J. Eng. Appl. Sci., vol. 12, no. 16, pp. 4762–4768, 2017.

[2] P. Goransson and C. Black and T. Culver, "How SDN Works," in Software Defined Networks a comprehensive approach, Boston: Morgan Kaufmann, 2017, Ch. 4, pp. 61–79.

[3] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy and Challenges," IEEE Communication. Survey. Tutorials, vol.20, issue 1, 2017.

[4] P. Phaal, "sFlow Specification Version 5", July 2004, accessed on 22 DEC 2019, [Online]. Avail- able: http://sflow.org/sflow version 5.txt.

[5] W. Prakash, "DServ - LB: Dynamic server load balancing algorithm," International Journal of Communication Systems - Wiley Online Library, Vol. 32, issue 1,2019

[6] U. Zakia, H. Yedder, "Dynamic Load Balancing in SDN-Based Data Center Networks," 18th IEEE annual information technology, electronics and mobile communication conference (IEMCON), Vancouver, BC, Canada, 2017.

[7] "What is load balancing," accessed on DEC 2019, [online]https://www.citrix.com/glossary/load-balancing.html

[8] "Installing POX — POX Manual Current documentation," Accessed on Dec 2019, [online] Available https://noxrepo.github.io/pox-doc/html/

[9] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks," Comput. Commun Rev., vol. 38, no. 2, pp. 69–74, 2008.

[10] ONF, "OpenFlow Switch Specification 1.4.0," accessed on DEC 2019, [online] https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf

[11] N. Joshi and D. Gupta, "A Comparative Study on Load Balancing Algorithms in Software Defined Networking," in International Conference on Ubiquitous Communications and Network Computing UBICNET, India, 2019.

[12] S. Kaur, K. Kumar, J. Singh, and N. Ghumman, "Round-robin based load balancing in Software Defined Networking," in 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2015.

[13] H. Uppal and D. Brandon, "OpenFlow Based Load Balancing," 2010, [online] available https://courses.cs.washington.edu/courses/cse561/10sp/project_files/cse561_openflow_project_report.pdf

[14] M. Koerner and O. Kao, "Multiple service load-balancing with OpenFlow," in 2012 IEEE 13th International Conference on High Performance Switching and Routing, Belgrade, Serbia,2012

[15] S. Vyakaranal and J. Naragund, "Weighted Round-Robin Load Balancing Algorithm for Software-Defined Network, "Emerging Research in Electronics, Computer Science and Technology (ICERECT), Lectural notes in Electrical Engineering 545, 2019. [online] https://doi.org/10.1007/978-981-13-5802-9_35

[16] G. Tiwari, V. Chakaravarthy, and A. Rai, "Dynamic load balancing in software defined networking," Int. J. Eng. Adv. Technol., vol. 8, no. 5, pp. 2706–2712, 2019.

[17] J. Singh, "Weighted Round-Robin Load Balancing Using Software Defined Networking," in International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), vol. 6, no. 6, pp. 621–625, 2016.

[18] K. Kaur, S. Kaur, and V. Gupta, "Flow statistics-based load balancing in OpenFlow," in International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, 2016.

[19] M. Elgili, "Load Balancing Algorithms Round-Robin (RR), Least-Connection and Least Loaded Efficiency," GESJ: Computer Science and Telecommunications, issue 51, no. 1, pp25-29, 2017.

[20] Y. Shengsheng, Y. Lihui, L.Song, and Z. Jingli, "A variable weighted least-connection algorithm for multimedia transmission," Journal of Shanghai University, vol 7, no. 3, pp 256-260, 2003.

[21] L. Padilha and D. Batista, "Effectiveness of Implementing Load Balancing via SDN," In Proceedings of the 37th Brazilian Symposium on Computer Networks and Distributed Systems, (pp. 249-256). Porto Alegre: SBC. 2019, [online] doi:10.5753/sbrc_estendido.2019.7796.

[22] P. Suwandika, M. Nugroho, M. Abdurahman, "Increasing SDN Network performance using load balancing schema on webserver," in IEEE 2018 6th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia,2018

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

[23] K. Soleimanzadeh, M. Ahmadi, M. Nassiri, "SD-WLB: An SDN-aided mechanism for web load balancing based on server statistics," ETRI Journal, vol. 41, issue 2, pp. 179-206, 2019.
[24] A. Montazerolghaem, "SIP Server Load balancing Based on SDN," 2019 available [online] https://arxiv.org/ftp/arxiv/papers/1908/1908.04047.pdf
[25] M. Albowarab, N. Zakaria, Z. Abidin" Load Balancing Algorithms in Software Defined Network," International Journal of Technology and Engineering (IJRTE), vol.7, issue-6S5,2019.
[26] Mininet: An Instant Virtual Network on your Laptop (or other PC) – Mininet, Accessed 14 Dec 2019 [online] available http://mininet.org/
[27] Production Quality, Multilayer Open Virtual Switch, Accessed on 21 JUN 2020, [online] available https://www.openvswitch.org/
[28] Iperf, "iPerf 2 user documentation, "Accessed on 4 April 2020, [online] available https://iperf.fr/iperf-doc.php#doc
[29] A. Al-Saadi, R. Setchi, Y. Hicks, and S. M. Allen, "Multi-rate medium access protocol based on reinforcement learning," Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern., vol. 2014-Janua, no. January, pp. 2875–2880, 2014.
[30] R. Lyman, T. Longecker, "An introduction to statistical methods and data analysis," 6th ed, USA, Thomson Learning Academic Resource Center, pp.463,2010.

Received 19 February 2020; Accepted 28 April 2020

 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352