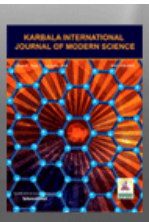

Karbala International Journal of Modern Science

Volume 7 | Issue 1 Article 6

A Proposed Adaptive Least Load Ratio Algorithm to Improve Resources Management in Software Defined Network OpenFlow Environment

haeeder munther noman mr.
college of information technology/university of babylon, hydermu79@gmail.com

Mahdi Nsaif Jasim Dr.
University of Information Technology and Communications, mahdinsaif@uoitc.edu.iq

Recommended Citation

noman, haeeder munther mr. and Jasim, Mahdi Nsaif Dr. (2021) "A Proposed Adaptive Least Load Ratio Algorithm to Improve Resources Management in Software Defined Network OpenFlow Environment," *Karbala International Journal of Modern Science*: Vol. 7 : Iss. 1 , Article 6.
Available at: https://doi.org/10.33640/2405-609X.2255

A Proposed Adaptive Least Load Ratio Algorithm to Improve Resources Management in Software Defined Network OpenFlow Environment

Abstract

So far, the existing load-balancing schemes (static and dynamic) have not taken over a concise and reasonable mechanism for systematic isolation of the resource counters monitored in the OpenFlow switch periodically. To address the aforementioned issue, the adaptive least load ratio algorithm was proposed and followed a straightforward procedure to generate the best possible decisions based on significant motivation characteristics identified in dynamic load balancing schemes like Least Connection-Based and Least Bandwidth-Based. the study was carried out using the HTTPerf tool as it provided a flexible facility for the generation of various HTTP workloads to evaluate server performance. Results revealed that, while Software defined network was under the influence of client requests load ranging from 0 up to 180 (req/sec), the proposed algorithm confirmed a faster server reply time up to 13.37%, server connection time up to 16.3% and average network throughput was improved up to 8%. Moreover, server CPU utilization time and average queue length were reduced to 2.5% and 14%. respectively. accordingly, and based on what the proposed algorithm has achieved in terms of quantitative performance parameters, it could be adopted in SDN-based data centers.

Keywords

ALLR, SDN-Based platform controller POX, SDN, Open Flow, mininet, HTTPerf

1. Introduction

As network-oriented services such as customer management system, domain name and e-mail expand, then a tremendous demand for server clusters is ultimately needed. Recently, considerable attention has been driven to manage network traffic usefully [1]. The rapid development in internet technology has been represented as a challenging task for the server cluster in large internet service providers [2]. The growth of users as well as network bandwidth have resulted in the server needing to process many access requests in a short period of time. In certain circumstances, the server lacks the ability to handle users' requests in a well-defined timing manner, eventually leading to a great opportunity for an increment in the waiting period accompanied with a reduction in the quality of service (QoS). To improve the server's performance, industries have been motivated towards adopting a variety of actions, like the increment of CPU processing speed, the development of server cache capability, the utilization of high-speed disk array, and the construction of the server cluster [3]. When the server cluster receives a request, a new challenge begins to emerge which stems from selecting the appropriate server to respond for user requests. Practically, in situation where the access requests being not assigned in a reasonable way, then load imbalance state between servers in the same cluster occurs. Accordingly, the load balancing concept is recommended to allocate the incoming traffic load among a group of servers to obtain a significant improvement regarding request response time, throughput, and fault tolerance, thereby attempting to solve the load imbalance problems among servers [4]. The traditional load balancing techniques employ variety of expensive hardware devices without precise traffic load control. This can lead to making these load balancers unsuitable for large-scale applications, costly, and hard to change [5].

2. Related work

Various algorithms for load balancing were proposed for solving the server overloading problem [6]. An early example was based on a single load balancing parameter found to be insufficient for selecting the best server to handle the client's requests [7]. Reference [8] proposed a flexible and cost-effective SDN load balancer methodology, which allowed network development in the absence of supplier information. As reported by [9], conventional load balancers were inflexible to be changed or modified since they were locked by vendors and their non-programmable design, which led to the lack of the network administrators to build their own algorithms.

SDN Load Balancing provides the programmers the ability to design and develop their own load balancing techniques. The Load balancing system is typically classified into two; Static and Dynamic. Static scheme distributes the load without taking into account the capability of nodes, such as RAM, server processing power or bandwidth, and links [10]. The static scheme came with several advantages such as the appropriateness for homogenous servers, less overhead, and the easiness implementation. However, this scheme is scalable and unable to reflect changes in dynamic attributes [11]. For instance, if one server receives numerous tasks, then there is a chance where a new task happens to be forwarded to the same server after a period of time independently on server capacity or task size. On the other hand, the dynamic scheme distributes the load according to the current status of the network nodes [12]. The Dynamic scheme ensures that the load balancing system tests the server's load ability and links at run time. However, the dynamic scheme neglects the type of user request and the size with the capability of using one algorithm for all of the different services. In fact, the static scheme is basically utilized for a small-sized network [13], so the performance is better as compared with that of dynamic load balancing. However, when the webserver is being adopted for large areas having requests dynamically generated, there is a possibility that the performance of static scheme can decline. Reference [14], suggested a dynamic scheme namely least connections relying on counting live connections for each server. Furtherly, the least bandwidth based scheme was proposed by [15], which provided the ability to access the data traffic transferred to the servers.

3. Contribution

The Contributions of this paper include:

- Load balancing has proved to be a very important research field [16] and various strategies using SDN technology have been suggested. Therefore, it is necessary to focus on developing an algorithm

https://doi.org/10.33640/2405-609X.2255

that has the ability to maximize throughput, reduce response time with an adaptive nature in load balancing, agility, and less downtime.

- The main objective is structured around the implementation of three innovative modules. The first module is responsible for the collection of the SDN OpenFlow switch statistics data records. The second module reads the statistics from the first module, leaves out the flow table, flow entry statistics, and carries out the process of gathering the port counters that provided by the OpenFlow protocol, and forwards them periodically every 5 seconds to the third module. The third module represents the load balancing module by acting as the main building block, after receiving server link bandwidth consumption during a particular timing period from the second module. Besides, it carries out the computations related to the selection of the least loaded server among a pool of servers.
- Proving the effectiveness of the proposed scheme through the evaluation of performance metrics against traditional schemes.

4. Proposed Architecture and Design

Mininet simulates a set of Linux end-hosts, switches, routers, and links. It utilizes the lightweight virtualization identified as the capacity of an operating system to be directly mounted on the hardware of the computer so that the device appears similar to a complete network. Mininet is a supportive tool for the open-source SDN community due to the fact, that it is frequently used as a simulation, verification, and resource testing platform. Being hosted on GitHub, mininet enables us to develop custom topologies and build hosts, switches, and controllers [17].

To calculate the performance of the server, it is important to utilize certain tools running on client machines for generating specific workloads. HTTPerf is a software tool which is employed to evaluate the hypertext transfer protocol (HTTP) web server performance [18]. HTTPerf tool comes up with several advantages such as the ability to support HTTP 1.1 protocol, preserve and generate server overload, and the extensibility towards statistics collectors and workload generators. In this work, various parameters are utilized for the simulation listed in Table 1, and suggested to run in SDN-Based environment that represented in Fig. 1 The majority of parameters are constant except the one related to the subjected load which is represented by the number of client requests per second. This varies gradually from 0 to 180 (req/sec) in order to explore their influence on the performance of the load balancing schemes.

Table 1
SDN-Based Platform Design Parameters.

No.	Item	Assigned Value
1	Host operating system	Linux (Ubuntu) ver 14.04
2	Programming language	Python Ver. 2.7.6
3	Network emulator	Mininet Ver. 2.2.1
4	Virtualization software	VMWare workstation
5	Simulation tool	HTTPerf
6	Total no. of connections	100
7	Max no. of requests per second in each connection	180 (req/sec)
8	Min no. of requests per second in each connection	0 (req/sec)
9	No. of HTTP web servers (3)	HTTP Server 1 -10.0.0.1 HTTP Server 2 - 10.0.0.2 HTTP Server 3 - 10.0.0.3
10	No. of HTTP clients (9)	Client 4 - 10.0.0.4 Client 5 - 10.0.0.5 Client 6 - 10.0.0.6 Client 7 - 10.0.0.7 Client 8 - 10.0.0.8 Client 9 - 10.0.0.9 Client 10 - 10.0.0.10 Client 11 - 10.0.0.11 Client 12 - 10.0.0.12
11	SDN controller platform	POX
12	Virtual SDN switch	OpenFlow switch
13	POX Controller to OpenFlow switch communication port	6633
14	HTTP web servers listening port	80
15	Virtual IP	10.0.1.1
16	POX Controller IP	127.0.0.1
17	Amount of time to wait for a server to respond	1.0 sec
18	Load balancing policies	Dynamic Least Connection Dynamic Least Bandwidth Proposed
19	No. of iterations	9
20	Link latency	No

5. Proposed load balance algorithm

The framework structure consists of two parts. Firstly, the SDN console application modules and the server clusters that communicate with the console via OpenFlow switches. Secondly, the POX Console, one of the well-known SDN controllers written in Python, are used to implement SDN application modules. Besides, OpenFlow API version 1.0 is accommodated to communicate with the SDN controllers [19] and OpenFlow switches to implement the server pool segment. Three functional modules establish the

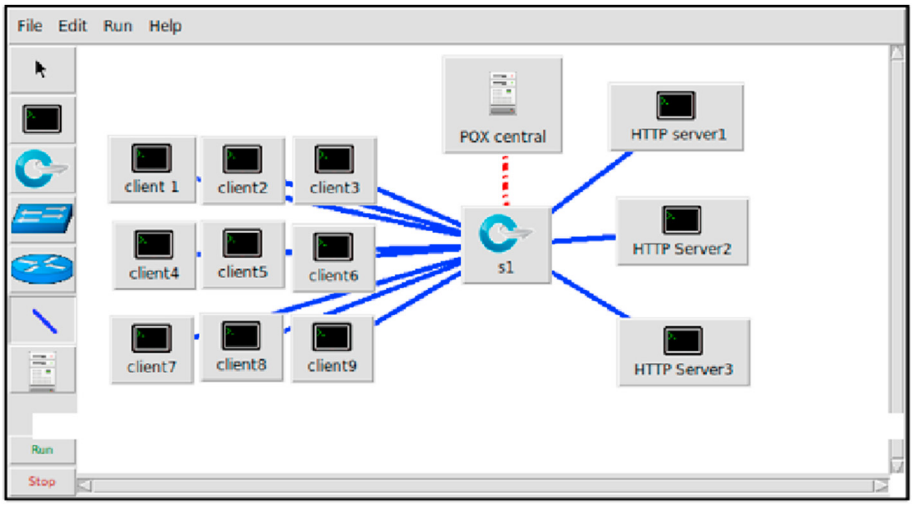

Fig. 1. SDN-Based platform network topology.

application module namely the CollectStatisicsData module (CSD), ServerMonitor module (SM), and the ALLRCompute module (ALLRCompute).

5.1. CollectStatisicsData Module (CSD)

By using this module, various types of statistics could be gathered. Firstly, there are statistics per table which is represented by the counters for this table, counters including matched packets, number of looked up packets, and the number of active flow entries. Secondly, there are individual flow entry statistics represented by the counter field in a flow entry. Finally, there are port statistics which in fact display the comparable flow entry statistics. This includes the number of packets and bytes sent and received via this port among others. Statistical data provided the basic network resources data which could be utilized for different types of functions depending on the implementation (see Table 2).

Table 2
CollectStatisticsData Algorithm.

Collect Statistics Data: Collection of OpenFlow switch statistics data algorithm	
Input:	- SDN-Based Platform Network topology
Output:	- Collects statistical data from OpenFlow switches after requesting them by the POX controller and report them to the ServerMonitor module periodically for further computations
Procedure CollectStatisicsData ( )	
Step 1	***While*** TRUE **do**
Step 2	***Collec***t Switch statistics data ( )
Step 3	***Update***, switch, switch port, flow table, flow records
	end while

5.2. ServerMonitor Module (SM)

The ability to determine the health of the server is a crucial part of the load balancing process). There is an urgent need to report server-related metrics such as bandwidth consumption to the controller during regular timing periods to perform the required computations which is related to the current server load. Without this information, the functionality of load balancing may operate inappropriately and send falsified requests for connection to different devices that are overloaded. Accurate and timely network resource statistics must therefore be implemented at different aggregation levels (such as flow, packet, and port) to rapidly adjust forwarding rules in response to server workload changes. Techniques such as NetFlow [20], sFlow [21], and JFlow [22], were utilized to collect full or filtered traffic statistics that mainly transfer confidential monitoring with significant overhead measurements to a central controller. Approaches like these may not be as effective solutions to be applied in SDN systems, such as large data center networks [23]. In this module, statistics of the first module are sent here. Accordingly, this module aggregates the counters related to ports including the number of bytes transmitted and received every 5 seconds and leave over the other statistics. Implementation of the predefined module involves invoking the executor at the interval set as described below by passing two parameters, the

Table 3
ServerMonitor Algorithm.

ServerMonitor: Monitoring model algorithm	
Input:	- S, pool of servers {S0, S1, Sn-1} - Switch, OfPortNo: OpenFlow switch with port No. - n, time rate adopted to count OpenFlow switch Ports traffic information counters (n = 5 seconds)
Output:	- Bandwidth Consumption for each server link Reported to the POX controller periodically
Procedure ServerBandwidthConsumption ( )	
Step 1	***While*** TRUE ***do***
Step 2	***Call*** CollectStatisicsData ( )
Step 3	***Read*** Switch, Switch Port records
Step 4	***Count*** rx_bytes (switch, OfPortNo)
Step 5	***Count*** tx_bytes (switch, OfPortNo)
Step 6	***Count*** rx_packets (switch, OfPortNo)
Step 7	***Count*** tx_packets (switch, OfPortNo)
Step 8	***Count dropped packets*** (switch, OfPortNo)
Step 9	$B(Si) \leftarrow \frac{\sum rx\_bytes - tx\_bytes}{n}$
Step 10	***return*** B(Si)
Step 11	***Sleep*** (n units of time) ***end while***

switch [24] and OfPortNo [25] (Port number), which returns the average received and transmitted bytes counted every 5 seconds as presented in Table 3.

5.3. ALLRCompute module (ALLRCompute)

This section represents the main dynamic load balancing module that carries out the responsibility of measuring and distributing the incoming traffic to the best available server. As clients send various requests to the pool of servers providing a particular service, different servers basically have various workloads. The load balancing module mainly Relies on the least load ratio per server equation (Li=B(Si)/W(Si)), where (Bi) represents the server link bandwidth consumption reported periodically from ServerMonitor (SM) module and W(Si) is the static weight assigned by the administrator to server (Si). This weight reflects the capability of the server like server CPU and RAM. Server with a larger weight receives more load in comparison to the server with a smaller weight [26]. The list of IP servers is traversed only once, and the time consumed by this algorithm is O (n) (see Table 4).

5.4. Dynamic Least Connection-Based algorithm (L.C)

The requests received from the network are forwarded to the node with the least number of existing TCP connections. In situations where a load of requests differs enormously, this approach is considered a good option for smooth distribution since all the long requests have no chance of being directed to a node. However, the algorithm performs properly when nodes of different processing capacities are available [27].

Table 4
ALLRCompute Load Balancing Algorithm.

ALLRCompute: Adaptive Least Load Ratio (ALLR) load balancing Algorithm	
Input:	- L, an array of the load per server = (L0, L1,., Ln-1)
Output:	- Best server to handle the incoming request based on the Allocation of the Adaptive least load ratio server among a pool of servers
Step 1	***Call*** ServerMonitor (B(Si))
Step 2	***Assign*** static weight W(Si) for Si
Step 3	$L(Si) \leftarrow B(Si) / W(Si)$
Step 4	***n*** = number of servers
Step 5	***for*** m = 0 to n-1
Step 6	find minimum load sever
Step 7	***if*** server [m] < minimum load server Then
Step 8	***Exchange*** server [m], minimum load server
Step 9	***end*** **for**
Step 10	***return*** minimum load server

5.5. Dynamic Least Bandwidth-Based Algorithm (L.B)

The server with the lowest network traffic consumption during the last n second to respond to the next request is being selected [28]. The balancer parses and stores the number of bytes that are transmitted to each server.

6. Experimental setup and evaluation

All experiments were carried out using the POX controller, chosen due to free open source features. This feature in fact facilitates the addition and removal of restored units, and gives the experiments more flexibility and performance. Port 6633 is the default connection port for establishing communications between OpenFlow switch and POX controller [29]. In addition, the POX controller includes a list of IP addresses assigned to each server statically.

Mininet emulator carried out the creation of the virtual network topology, which is made up of three hosts acting as web servers holding the same configuration to provide the same web services to the clients. Nine hosts acted as clients attempting to access the servers using a virtual IP address representing the

service IP. If a particular client happened to request the virtual IP (service IP), then the request would be forwarded via the Open Flow switch to the POX controller to make the necessary computations and select the appropriate server to handle the request. The proposed algorithm mainly relies on three modules namely CollectStatisicsData, ServerMonitor, and ALLRCompute. The algorithm updates packet header including destination MAC and IP address of the selected server then redirects these rules to the OpenFlow switch, and forward the request to the port assigned for the selected server. Accordingly, selected server replies to the client's request through the OpenFlow switch again. An important issue is to utilize some tools running on any of the clients for generating specific workloads. HTTPerf is a software tool mounted on clients to measure the performance of the Web Server hypertext transfer protocol under stress. HTTPerf tool comes with many advantages such as the capability to maintain and produce server overload, the extensibility towards statistics collectors and workload generators, and the support of the HTTP 1.1 protocol. Accordingly, HTTPerf plays an important role in calculating parameters such as average throughput, server reply time, server connection length, and CPU time utilized by the client.

7. Results and discussions

This section discusses the empirical findings concerning the impact of a gradual increase in client requests on the performance of a load balancing system. Results of five categories of metrics namely average server throughput, server reply time, server connection time, server CPU utilization, and server average queue length, are presented and explained to validate the assumptions considered when the dynamic load balancing mechanism is designed. In these experiments, a few numbers of requests are sent in each data-trace, this is due to the limit of buffer for sending and receiving data in situations where the HTTPerf tool was adopted [30]. It is important to observe that the differences in performance between the two dynamic load balancing schemes other than our proposed algorithm, namely Least Connection-Based and Least Bandwidth-Based seem to be very close (less than 1%) especially when the comparison was carried out in terms of performance metrics.

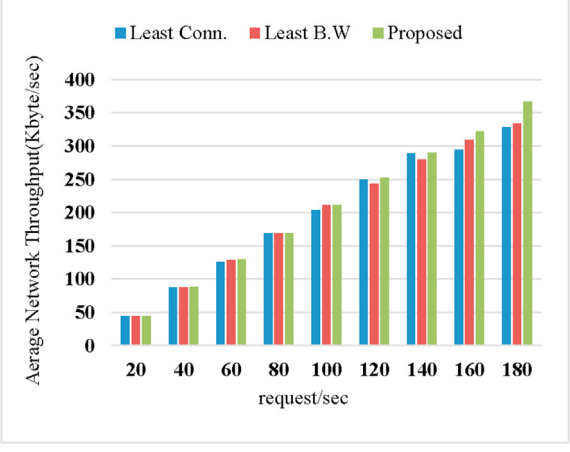

Fig. 2. Average Network Throughput Vs Request Rate.

7.1. Average Network Throughput

Average Network Throughput (*A.N.T*) could be evaluated through Equation (1) [31, 32].

$$\sum \frac{No.\ of\ Successful\ Packets\ *\ Average\ Packet\ Size}{Total\ time\ spent\ in\ delivering\ that\ amout\ of\ data} \quad (1)$$

According to the global view information collected by the POX controller basically provided by the OpenFlow protocol, results from Fig. 2 Demonstrated a close increment regarding average network throughput for the three load balancing schemes especially when the request rate was between 0 and 180. As the request rate started to rise above 120 to 140, 160, and 180, then our proposed scheme possessed a gradual increase of average network throughput achieving an improvement up to 7.25%.

7.2. Server Reply Time

Server Reply Time (*R.T*) [33], referred as transfer time, which is defined as the time between the first byte and the final byte of the response. Equation (2) illustrates the formula involved in the calculation of server reply (RT) [34].

$$Tfb + Tlb \quad (2)$$

whereas *Tfb* is the time of the first byte of the response, *Tlb* is the time of last byte of the response. In general, the reply time is less than the average response time [35], and is affected by client requests. Results from Fig. 3 revealed that as the request rate escalades gradually from 0 to 120, then the performance of the three load balancing schemes was again close to each other with a slight improvement recorded in favor of our

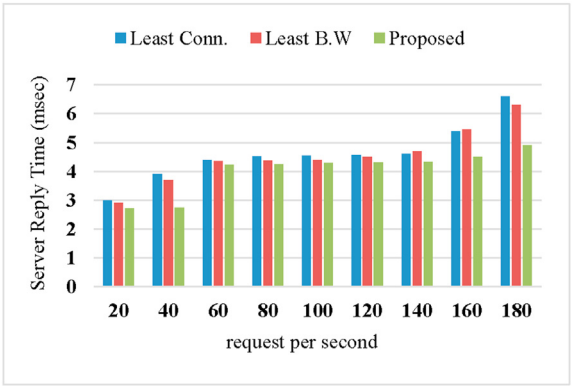

Fig. 3. Server Reply time Vs Request Rate.

proposed scheme. However, when the request rate reached 140 and beyond, it could be obviously observed that our proposed scheme points to clear progress in server reply time up to 13.37%.

7.3. Server Connection Time

The next line in the output presented the server connection time which reflected the lifetime statistics for a successful connection [36]. The lifetime of a connection is defined as the time between a TCP connection initiation and connection termination. A connection is considered successful if it had at least one request that results in a reply from the server. Fig. 4 showed that our proposed scheme regularly improved server connection time demonstrating an obvious relative outperformance of up to 16.38%.

7.4. Server CPU utilization

CPU utilization [37], is exploited as the parameter responsible for measuring server load, denoted as *(U)* and determined through the use of Equation (3) [38].

$$\frac{BT}{OT} \tag{3}$$

Three factors need to be taken into consideration, *OT* (observation time) defined as the total server monitoring time, *BT* (Business time) defined as the total time where the server is being on during *OT*, and the total number of requests completed during *OT*.

Fig. 5 clearly showed that the higher the request rate to access the server, the higher the server CPU utilization reported for that server. proposed scheme improved this metric by approximately 2.5% in comparison with other schemes and at the same time avoided the degradation of other metrics.

7.5. Server Average Queue length (M/M/1)

This metric is vital [39], as it assists us to be aware about expected time delay due to the waiting time a packet spends in a queue. Normally, servers are expected to create queues to process requests directed to the servers. To get the length of this queue (Q), we used Equation (4) [40].

$$\frac{U}{1-U} \tag{4}$$

where U is the Server CPU utilization. After analyzing the results shown in Fig. 6, our proposed load balance scheme brought the attention that it reduced the server average queue length by 14%. Due to working within the virtual environment represented by the mininet program to simulate the SDN mechanism, the maximum number of users' requests was restricted to 180. Because if it exceeded that, all servers entered the saturation stage, which caused the queues to be filled, packets dropping, and the client re-sending. Hence, a saturated server will operate but with less performance. Therefore, results of the proposed algorithm

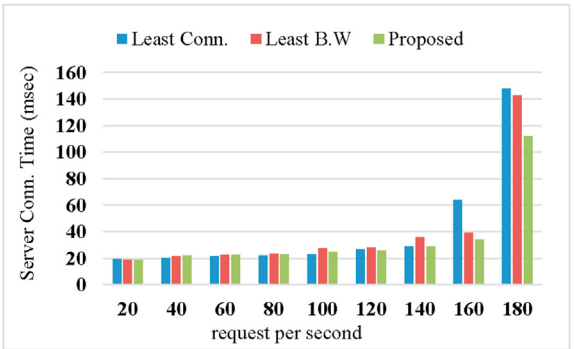

Fig. 4. Server Connection Time Vs Request Rate.

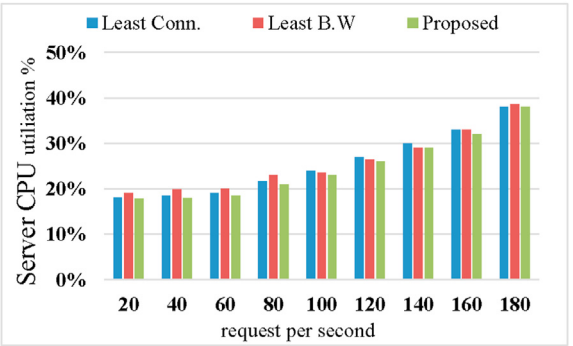

Fig. 5. Server CPU Utilization Vs Request Rate.

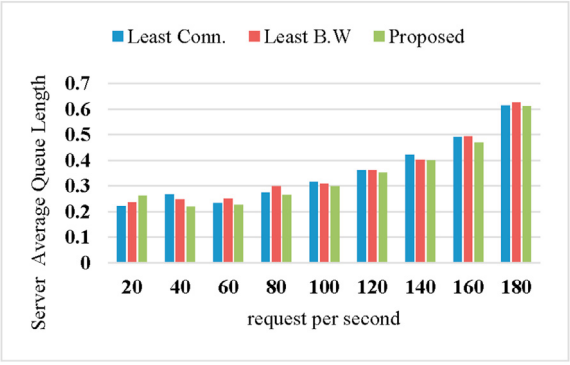

Fig. 6. Server Average Queue Length Vs Request Rate.

improvements were close to other dynamic algorithms. However, within the real working environment of the SDN-Based platform, the maximum number of user requests per second could reach higher limits like 5000 to 10000 before entering server saturation criteria. Based on that, a wide difference in the number of client requests between the real environment and the virtual environment lead to a large variation in the enhanced results of the proposed algorithm.

8. Conclusions

According to the results accomplished through this study, it can be concluded that our proposed scheme under SDN-Based platform is exploited server resources and balancing loads in a better way in comparison with another load balancing schemes. Obviously, an improvement is made regarding performance, the main reason was due to the continuous evaluation feature of the traffic consumed between the servers that allowed the server with the least load to respond to the next request. Moreover, the server weights referring to loads of servers were not equally distributed. Hence, servers owned different characteristics. Moreover, it can be concluded that when the request rate was above 180 (req/sec), the servers enter the saturation criteria, which leads to a sudden decrease in the transfer rate [Before the resources exhausted, the acceptance queue of servers is not full. Therefore, no packet losses are experienced despite of the queue delay effect. The queue is complete and several packets are dropped after saturation. This leads to client retransmissions which causes even more dropped packets. Thus, the saturated server operates less efficiently].

9. Future work

It is possible, as future work, to implement the proposed algorithm on any architecture in SDN-Based platform, such as linear, single, tree, and fat tree, then evaluating the performance through each architecture.

Acknowledgements

The authors would like to acknowledge the endless support that has been provided by the staff of the Software Department, College of Information Technology, University of Babylon. Furthermore, the scholarship provided by the Middle Technical University, Ministry of Higher Education & Scientific Research is gratefully appreciated.

References

[1] I.Z. Bholebawa, R.K. Jha, U.D. Dalal, Performance analysis of proposed network architecture: OpenFlow vs. traditional network, Int. J. Comput. Sci. Inf. Secur. 14 (2016) 30.
[2] A. Oussous, F.Z. Benjelloun, A.A. Lahcen, S. Belfkih, Big Data technologies: A survey, J. King Saud Univ. Inf. Sci. 30 (2018) 431–448.
[3] E. ernández-Orallo, J. Vila-Carbó, Web server performance analysis using histogram workload models, Computer Networks 53 (2009) 2727–2739.
[4] C. Chen-Xiao, X. Ya-Bin, Research on load balance method in SDN, Int. J. Grid Distrib. Comput. 9 (2016) 25–36.
[5] H.M. Kavana, V.B. Kavya, B. Madhura, N. Kamat, Load balancing using SDN methodology, Int. J. Eng. Res. Technol. 7 (2018) 206–208.
[6] N. Shahzad, G. Mujtaba, M. Elahi, Benefits, security and issues in software defined networking (SDN), NUST J. Eng. Sci. 8 (2016) 38–43.
[7] T.E. Emad, A.H. Morad, M.A. Abdala, Load balance in data center SDN networks, Int. J. Electr. Comput. Eng. 8 (2018) 3086. .
[8] M. Karaku, A. Durresi, Quality of service (QoS) in software defined networking (SDN): A survey, J. Netw. Comput. Appl. 80 (2017) 200–218.
[9] J.S. Sabiya, Weighted round-robin load balancing using software defined networking, Int. J. Adv. Res. Comput. Sci. Softw. Eng. 6 (2016) 621–625.
[10] M. Mehra, S. Maurya, N.K. Tiwari, Network Load balancing in Software Defined Network: A Survey, Int. J. Appl. Eng. Res. 14 (2019) 245–253.
[11] E.J. Ghomi, A.M. Rahmani, N.N. Qader, Load-balancing algorithms in cloud computing: A survey, Journal of Network and Computer Applications 88 (2017) 50–71.
[12] Y. Wang, X. Tao, Q. He, Y. Kuang, A dynamic load balancing method of cloud-center based on SDN, China Communications 13 (2016) 130–137.
[13] P. Beniwal, A. Garg, A comparative study of static and dynamic load balancing algorithms, Int. J. Adv. Res. Comput. Sci. Manag. Stud. 2 (2014) 1–7.

[14] M.E. Mustafa, A.M. Ibrahim, Load Balancing Algorithms Round-Robin (RR), Least-Connection and Least Loaded Efficiency 1 (2017) 25−29.

[15] A.A. Neghabi, N.J. Navimipour, M. Hosseinzadeh, A. Rezaee, Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature, IEEE Access 6 (2018) 14159−14178.

[16] A. Shukla, S. Kumar, H. Singh, Load balancing approaches for web servers: A survey of recent trends, International Journal of Engineering 31 (2018) 242−248.

[17] D.Y. Lee, A Study on the Flow Analysis on the Software-Defined Networks through Simulation Environment Establishment, J. Korea Inst. Information, Electron. Commun. Technol. 13 (2020) 88−93.

[18] M. Hamed, B. ElHalawany, M. Fouda, A.T. Eldien, Performance analysis of applying load balancing strategies on different SDN environments, Benha J. Appl. Sci.(BJSA) 2 (2017) 91−97.

[19] T.E. Ali, A.H. Morad, M.A. Abdala, Load balance in data center SDN networks, Int. J. Electr. Comput. Eng. 8 (2018) 3086.

[20] P. Krishnan, J.S. Najeem, A review of security threats and mitigation solutions for SDN stack, Int. J. Pure Appl. Math 115 (2017) 93−99.

[21] M. Afaq, S.U. Rehman, W.C. Song, A framework for classification and visualization of elephant flows in sdn-based networks, Procedia Computer Science 65 (2015) 672−681.

[22] M. Usman, A.C. Risdianto, J. Han, J. Kim, Interactive visualization of SDN-enabled multisite cloud playgrounds leveraging smartx multiview visibility framework, Comput. J. 62 (2019) 838−854.

[23] T. Chen, X. Gao, G. Chen, The features, hardware, and architectures of data center networks: A survey, J. Parallel Distrib. Comput. 96 (2016) 45−74.

[24] W. Li, W. Meng, L.F. Kwok, A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures, Journal of Network and Computer Applications 68 (2016) 126−139.

[25] J. Xia, Z. Cai, G. Hu, M. Xu, An active defense solution for ARP spoofing in OpenFlow network, Chinese J. Electron. 28 (2019) 172−178.

[26] G. Singh, K. Kaur, An improved weighted least connection scheduling algorithm for load balancing in web cluster systems, Int. Res. J. Eng. Technol. 5 (2018) 1950−1955.

[27] A.M. Alakeel, A guide to dynamic load balancing in distributed computer systems, Int. J. Comput. Sci. Inf. Secur. 10 (2010) 153−160.

[28] P. Zhu, J. Zhang, Load balancing algorithm for web server based on weighted minimal connections, J. Web Syst. Appl. 1 (2017) 1−8.

[29] W.H. Muragaa, K. Seman, M.F. Marhusin, A POX Con-troller Module to Prepare a List of Flow Header Information Extracted from SDN Traffic, Int. J. Comput. Syst. Eng. 11 (2017) 1305−1308.

[30] M. de la Cruz, J. Labrador, D. Dinawanao, WebSurge: A Profile-based Stress Testing Tool with Distributed User Agents for Web Applications, J. Comput. Innov. Eng. Appl. 1 (2016) 28−37.

[31] S.J. Jung, Y.M. Bae, W. Soh, Web performance anal-ysis of open source server virtualization techniques, International Journal of Multimedia and Ubiquitous Engineering 6 (2011) 45−51.

[32] D. Mosberger, T. Jin, httperf a tool for measuring web server performance, ACM SIGMET-RICS Performance Evaluation Review 26 (1998) 31−37.

[33] A.A. Abdelltif, E. Ahmed, A.T. Fong, A. Gani, M. Imran, SDN-based load balancing service for cloud servers, IEEE Communications Magazine 56 (2018) 106−111.

[34] M. Arman, Analisa Kinerja Web Server E-learning Menggunakan Apache Benchmark dan Httperf, Jurnal Integrasi 8 (2016) 93−100.

[35] M. Paliwal, D. Shrimankar, O. Tembhurne, Controllers in SDN: A review report, IEEE Access 6 (2018) 36256−36270.

[36] R. Hashemian, D. Krishnamurthy, M. Arlitt, Web workload generation challenges−an empirical investigation, Software: Practice and Experience 42 (2012) 629−647.

[37] X. Ye, G. Cheng, X. Luo, Maximizing SDN control resource utilization via switch migration, Computer Networks 126 (2017) 69−80.

[38] K. Sood, S. Yu, Y. Xiang, Performance analysis of software-defined network switch using M/Geo/1 model, IEEE Communications Letters 20 (2016) 2522−2525.

[39] F.R. Cruz, R.C. Quinino, L.L. Ho, Bayesian estimation of traffic intensity based on queue length in a multi-server M/M/s queue, vol. 46, 2017, pp. 7319−7331.

[40] T.C. Yen, C.S. Su, An SDN-based cloud computing architecture and its mathematical model, in: 2014 International Conference on Information Science, Electronics and Electrical Engineering, vol. 3, 2014, pp. 1728−1731.