Computer Systems Science & Engineering
DOI:10.32604/csse.2022.020947
Article

Adaptive Server Load Balancing in SDN Using PID Neural Network Controller

R. Malavika[1,*] and M. L. Valarmathi[2]

[1]Department of Information Technology, Government College of Technology, Coimbatore, 641013, India
[2]Department of Computer Sceince and Engineering, Dr. Mahalingham College of Engineering and Technology, Pollachi, 642003, India
*Corresponding Author: R. Malavika. Email: malavikkares21@yahoo.com
Received: 16 June 2021; Accepted: 23 August 2021

Abstract: Web service applications are increasing tremendously in support of high-level businesses. There must be a need of better server load balancing mechanism for improving the performance of web services in business. Though many load balancing methods exist, there is still a need for sophisticated load balancing mechanism for not letting the clients to get frustrated. In this work, the server with minimum response time and the server having less traffic volume were selected for the aimed server to process the forthcoming requests. The Servers are probed with adaptive control of time with two thresholds L and U to indicate the status of server load in terms of response time difference as low, medium and high load by the load balancing application. Fetching the real time responses of entire servers in the server farm is a key component of this intelligent Load balancing system. Many Load Balancing schemes are based on the graded thresholds, because the exact information about the network flux is difficult to obtain. Using two thresholds L and U, it is possible to indicate the load on particular server as low, medium or high depending on the Maximum response time difference of the servers present in the server farm which is below L, between L and U or above U respectively. However, the existing works of load balancing in the server farm incorporate fixed time to measure real time response time, which in general are not optimal for all traffic conditions. Therefore, an algorithm based on Proportional Integration and Derivative neural network controller was designed with two thresholds for tuning the timing to probe the server for near optimal performance. The emulation results has shown a significant gain in the performance by tuning the threshold time. In addition to that, tuning algorithm is implemented in conjunction with Load Balancing scheme which does not tune the fixed time slots.

Keywords: Software defined networks; PID neural network controller; closed loop control theory; server load balancing; server response time

1 Introduction

Achieving availability and responsiveness is turned out to be an important consideration in service provisioning model. The server load balancing is always used to distribute the user requests to redundant servers in the server farm. This farm provides a continuous service for ensuring availability and responsiveness of services to end users. The techniques that are employed in the server load balancer play a significant role in satisfying the end users. The Network Approach [1] implementation of Enterprise Content Delivery Network (ECDN) faces server load balancing because it is critically a great challenge in providing a service availability to the end user. In these days, reducing the heavy traffic network flux and alleviating the risk of single server have become targeted concepts for providing better services. For the sake of providing services, the hardware resources from ECDN are put into practice for a load balancer. However, the ECDN infrastructure is subjected to high cost, it involves more complexity and difficulty in managing the conditions. Software Defined Networking (SDN) [2,3] a New Networking paradigm uncouples the control plane from data plane. Further it enables the ECDN providers to implement their business logic as new network application and forward the request based on predefined policies [4]. In order to improve the web service responsiveness and network utilization, SDN offers the utmost benefits for most of the end users even with flexible investment cost. Additionally, the Central Controller of SDN maintains and monitors the networking elements and processes. The programmability and centralized control function has enabled the server load balancing in the software Defined Enterprise content Delivery Network (SD ECDN). Moreover, the SDN has allowed ECDN providers to be free from the governing networking devices and infrastructure so as to concentrate more on the business growth and network improvement. Load balancing module is implemented in SDN Controller. The controller in turn communicates to the whitebox switches using open standards protocol such as open flow [5,6]. And it would provide better composes of network resources. In general terms, balancing a Load in a network dictates many merits in terms of fault tolerance, high availability, scalable and secured server farm. The extensive literature on the topic of SDN based server load balancer using several factors namely reported load of server, least response time, up/down status of server, geographic locations of servers and network traffic flux are discussed.

The ECDN providers can entrust on SDN for allowing the load balancing as an essential functionality of network for the attainment of load balancing benefits. Besides, load balancing has lent a hand to amend the problem associated with the scalability of server farm by spreading out the user requests among all the servers. Load balancer which is implemented using SDN would handle the server and the failure from the web application by creating multiple abstract distribution trees. These innovative methods are used in order to withstand the failure. Load balancing solution has provided security for the server farm from different forms of attack by packet inspection and machine learning techniques. The inherent problem associated with the SDN based implementation of load balancing with single SDN Controller is the core point of its failure [7]. Many research works have been carried out in solving the bottlenecks associated with the single central SDN Controller by substituting it with many distributed controllers operating together as logically centralized controller. Thus, the controller load balancing in achieving the scalability of SDN controller [8] has also become a major research in SDN.

This paper focuses on how to improve the system using PID Neural Network Controller by the extension of LBBSRT [9] and SD-WLB [10], which works with both least server response time and traffic volume. In the study, the system was named as LBBNNC (Load balancing based on neural network controller). Both LBBSRT and SD-WLB had reviewed the server farm in a defined uniform time slot and it was found that it was not suitable for the inconsistent traffic. So, a system was designed based on PID Neural Network Controller with two threshold values or set points namely L and U to mark the status of the sever load as Low, Medium and high based on the maximum response time difference (MRTD) among servers in the farm which was below L, between L and U and above U respectively to change the probing time for measuring the MRTD of the server pool. However, the systems that were reviewed in the literature had incorporated the fixed time in fetching the response time. But in general, it is not ideal for all traffic conditions. The optimization engine in LBBNNC had tuned the probing time adaptively based on the thresholds that were associated with the load balancing algorithm for near optimal performance

Additionally, the specified approach LBBNNC was compared with Round robin [11] LBBSR, SD-WL, LBBNNC had exhibited higher network throughput, lessen response time and optimized open flow's PACKET-IN message overhead when the traffic conditions would have been unpredictable. With the implementation of closed loop control theory with two set points, LBBNNC had changed the probing time adaptively to measure the server load imbalance through maximum response time difference among servers in the server farm.

The particulars of the research and the important findings of it are provided in a detailed way.

1. Closed loop theory with two threshold values for branding the system load as low, medium and high grounded on the load balancer in SDN based network which optimizes the probing time to measure the server response time and the port traffic of switch for calculating the server load imbalance was proposed and the system had performed better than Round Robin, LBBSRT and SD-WLB.

2. Emulation of the experiment in Mininet [12] and the functioning of the system with other load balancing solutions in SDN based networks were carried out with respect to mean response time, memory usage and CPU usage.

Section 2 describes the related works. Section 3 shares a limelight on to the contribution which concentrates on adaptively change the probing time to measure the response time of all servers with two threshold values of MRTD. Thereby, the selecting server with minimum response time and minimum traffic volume on the port of the switch to manage all traffic conditions by considering the imbalance of the server load to balance the user requests among servers in server farm. Section 4 presents the experimental results acquired by Round Robin, LBBSRT, SD-WLB and our system LBBNNC. Section 5 brings about the document with conclusion and works in future.

2 Related Works

Maglev [13] a reliable and fast Layer 3 load balancer was intended to work on Linux server. It was designed to meet connection persistence by tracking the connections with reliable hashing in order to reduce the error on connection-based protocols. Despite of its merits, design and implementation techniques were extremely multifaceted and burdensome. Duet [14] was a cross layer 3 cloud scale load balancer system that used the custom hardware switches in datacenter with an integrated load balancing software function. The results had attained improvement in flexibility and availability of CONGA [15] a forwarding plane load balancing mechanism. CONGA had used the network flux cognizance maintained at the edge switches and it was implemented in the traditional hardware. Storage limitation was a problem to CONGA. HULA [16] a load balancing system was implemented in forwarding plane. Later it was prevailed over the limitation of CONGA by allowing every switch in the network to note down the congestion. That congestion helped in identifying the most appropriate path to a target through the neighboring switch. HULA was intended for the programmable switches without requiring the traditional hardware. Compared to CONGA, HULA is adaptable to the failures of network link but getting path utilization information is a burdensome task. The implementation of a distributed layer 4 load balancer was to create scale out web services that was operated on the commodity hardware and Ananta [17] was utilised to meet the requirements of multitenant cloud environment. Ananta provided back responses to the client bypassing the load balancing mechanism to increase the throughput and to reduce latency. Although Ananta had performed well, it would not be used by the public because of its proprietorship.

FDALB [18] had managed the dynamics of network traffic by reducing flow finish time with the information about flow sizes distribution. Plug-n-Serve [19] system had effectively balanced (load) the web traffic using OpenFlow by capturing the current state of network. It was suited for the networks that were built with inexpensive product hardware and at the same time, it was not suitable for structured networks such as data centers. The proposed system was used to load balance in SDN based networks using round robin strategy availing the agility of SDN. In general terms, SDN data planes are inexpensive and product-based silicon devices. They were uncoupled from the control plane using OpenFlow protocol to communicate with the forwarding plane. Later, it was allowed to design and develop round robin plan for selecting the server to achieve load balancing. SDN Controller independently had set up every flow in per flow-based routing. SDN POX Controller was used to develop the load balancer module, but round robin-based algorithm had faced its difficulty with the inconsistent network traffic that is usual in real time cases. Finally, the SDN POX Controller had taken the advantage of programmability and agility of SDN and calculated the response time of the server by dispatching. Pseudo ARP request and reply packets were collected to and from of all the servers in the server farm. It figured out the most suitable server for the request processing. This approach leverages the flexibility of SDN for selecting the server based on response time. The response time was measured by SDN Controller by probing the server with fixed interval of time. Web load balancing scheme had utilized both the server response time and the real time response time. the server had used the port traffic volume of the switch to calculate minimum value of weighted average of real time responses of all servers. The port traffic volume of the switch was used for selecting the suitable server to forward the request. Both LBBSRT and SD-WLB had worked well when the traffic was consistent *i.e.*, the frequency distribution of traffic was the same for all the time slots defined by the fixed time interval. But, when the traffic was not consistent, both LBBSRT and SD-WLB showed poor performance. The LBBSRT and SD-WLB were handled by extending the closed loop control theory. In a such way, the probing time was adaptively changed based on two threshold values for maximum response time. To select the server with minimum weighted average of response time of all servers, the port traffic static of the switch was used in the server farm. Then the user request was processed by the chosen web server.

3 Proposed Load Balancing Methodology Using PID Neural Network Controller

The architecture of SDN is proposed in Fig. 1. and some of its applications are also mentioned. The important functionality in the network approach [20] implementation of Enterprise content delivery networks was load balancing. In order to Provide Scalability and availability, ECDN providers had used redundant surrogate servers at the geographical location that were close to the clients, but it would incur heavy price to the infrastructure providers. So, a system was designed which used the SDN innovation to implement a SD-ECDN based on network approach to enhance the user satisfaction by increasing the availability of servers.

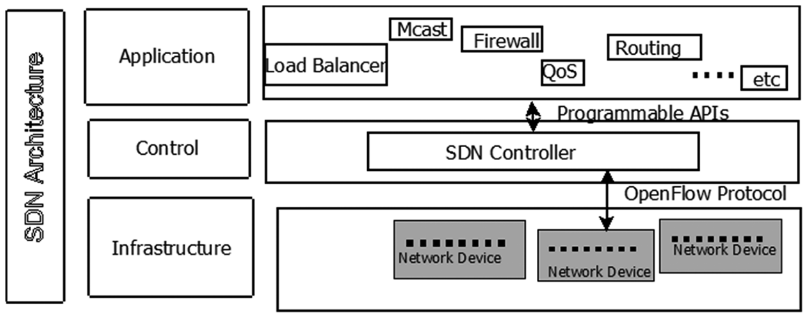

Figure 1: Innovation of SDN

LBBNNC was implemented using component-based framework for SDN namely RYU Controller [21]. Once the system got started, the web users had requested their service by locating the service *via* virtual load balancer IP configured by the ECDN provider. Once the request was directed to the networking (simple dumb device), which was not able to hold any intelligence because all the intelligence was extracted from the forwarding plane and it was encoded in the central SDN Controller. The Dumb Switch did not know how to handle the traffic flow so it would ask the controller on how to process the request. Based on the controller receiving the switch's request *via* OpenFlow's, the PACKET_IN message got processed. They process according to the rule enforced by the LBBNNC algorithm was implemented in RYU Controller. Normally, the job of LBBNNC algorithm is to choose the factors namely minimum response time and switch's port traffic volume. The proposed framework worked on how the controller retrieved the above-mentioned factors in variable time slots as defined by the probing time. The time was produced by the engine which was optimized in choosing the process of best server if the traffic condition was not consistent.

The proposed LBBNNC is comprised of four modules namely Server load imbalance monitor, traffic volume collector, probe interval producing engine and server chooser. "Server load imbalance monitor" checks for the load imbalance using maximum response time difference among servers in server farm and the MRTD value is being given to the "probe interval producing engine" to define the variable time slots for probing the server farm using two graded threshold values namely L and U. The "traffic volume collector", collects the volume of traffic in every port of all the switches in the network. Both the minimum response time and volume of traffic is given to the Server Chooser module for selecting the suitable server to process the user request. The modules in LBBNNC are depicted in the Fig. 2.

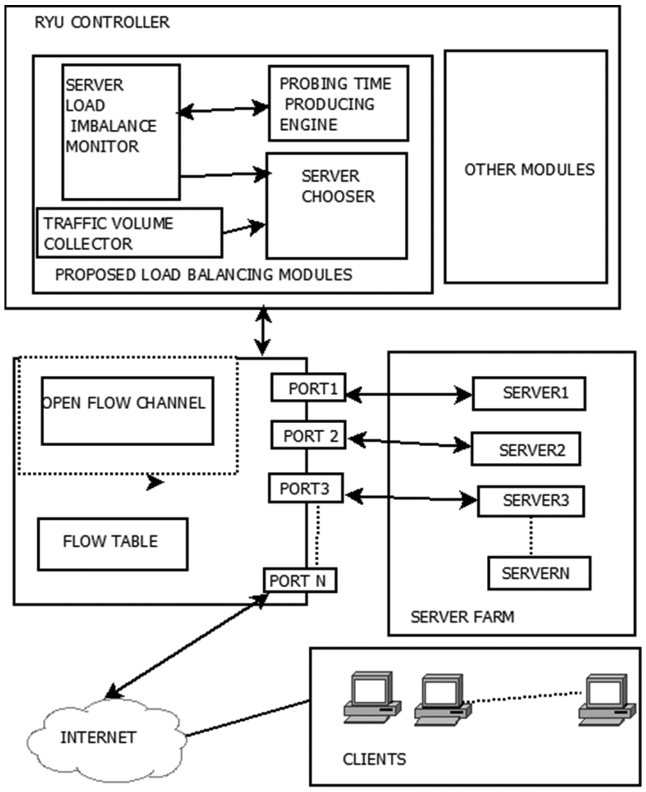

Figure 2: Modules of LBBNNC

3.1 Server Load Imbalance Monitor

The server load imbalance was calculated by the module using maximum response time difference of every server in the server farm. It works as defined by the Algorithm 1.

Algorithm 1:

```
Input: Probe Time $T_{probe}$ received from Probing time producing engine
Output: Maximum Response Time Difference MRTD
1.while system starts do
2.     If current time % $T_{probe}$ == 0 then
3.        for each server do
4.    send a Pseudo ARP request to server and records the time $T_{send}$
5.    record the receiving time of ARP reply as $T_{receive}$
6.           calculate the response time by the formula $T_{send} - T_{receive}$
7.             store the response time in SQLite database as $T_{res}$
8.        end for
9.      end if
10.   calculate MRTD as $\max(T_{res}) - \min(T_{res})$ and send it to probing time producing engine
11. end while
```

After initializing, the system server load imbalance monitor sends pseudo ARP request to every server in the server farm with variable interval time slots defined by the probing time producing engine module and record the sending time $T_{send.}$ Once the reply from ARP received, this component computes the value for T_{res} *i.e.*, the response time using the formula $T_{send} - T_{receive}$ and the computed values are stored in the database. Besides computing the response time, this module also computes the Maximum response time difference as $\max(T_{res}) - \min(T_{res})$ and sends the MRTD value to probing time producing engine.

3.2 Traffic Volume Collector

The traffic volume that was flowing through the ports of the network switch were collected by the module and got stored in SQLite database with variable time slots as defined by the probing time producing engine. The port traffic value was extracted from OpenVswitches using OpenFlow message namely OFPPORTSTATSREQUEST. The implemented module based on SD-WLB is presented below.

Algorithm 2:

```
Input: Probe Time $T_{probe}$ received from Probing time producing engine
Output: Volume of traffic in switch's port
1.while system starts do
2.     If current time % $T_{probe}$ == 0 then
```

(Continued)

Algorithm 2 (continued)

```
3.          for each port of switch do
4.      prior_total_bytes = 0
5.      prior_total_seconds = 0
6.            Analyze request's response and get the total_bytes and total_seconds
7.             current_bytes = total_bytes-prior_total_bytes
8.             current_seconds = total_seconds-prior_total_seconds
9.             traffic_volume = (current_bytes * 8)/current_seconds
10.             prior_total_bytes = total_bytes
11.            prior_total_seconds = total_seconds
12.     store the port's traffic volume in SQLite database as traffic_volume
8.          end for
9.        end if
10.     send the Port's traffic volume to probing time producing engine
11.  end while
```

3.3 Probing Time Producing Engine

The working of this module is depicted in Algorithm 3. The network architect or designer would determine the performance levels acceptable for the system. The maximum response time difference (MRTD) among all the server systems in the server farm was viewed. MRTD was dictated by the two application delay tolerance thresholds L and U, which was dependent on the application running in the server farm. Certain unstructured computer networks are not maintained with sophisticated infrastructures such as campus networks where the traffic load cannot be predicted in advance. whenever the students have to submit the assignment, the network traffic and its volume may get elevated to some extent which cannot be resisted. In such cases, it is to be learned the application delay tolerance thresholds L and U in terms of maximum response time difference in an online manner. For that purpose, the system was modelled with neural network to dictate L and U values. The neural network PID Controller is shown in Fig. 3. The neural network model was constructed online from the feedback of the system. The neural controller was trained then for tracking of specific set points namely L and U by defining the cost function based on the neural network model [22]. MRTD attempted in dictating the server load imbalance and the probing time producing engine which took care of finding the optimal probing time that helped in achieving such performance level. The behavior of this module is as follows: the network architect or designer defines the expected behavior for load balancer system using measurable MRTD thresholds L and U (the set point SP1(L) and SP2(U)). The probe interval generator measures the actual value for that variable MRTD (the process variable, PV) and computes the error, e = SP − PV. Given that error, the Probe interval Generator must compute an actuation value that is optimal probe interval (controller output, CO) to provide input to the system that should minimize the error and bring PV to the value defined for SP with the formula.

$$p_{new=}\left({P_{old}}/{PV}\right) * sp2 \text{ if } \text{PV} > \text{SP2} \tag{1}$$

$$p_{new=}\left({PV}/{P_{old}}\right) * sp1 \text{ if } \text{PV} < \text{SP1}$$

$P_{new} = P_{old}$ if SP1 ≤ PV ≤ SP2

where P_{new} is the next probe interval window and P_{old} is the previous probe interval window. The actuation value CO in terms of probe interval window is given as input to the system that should minimize the error and bring PV to the value defined for SP.

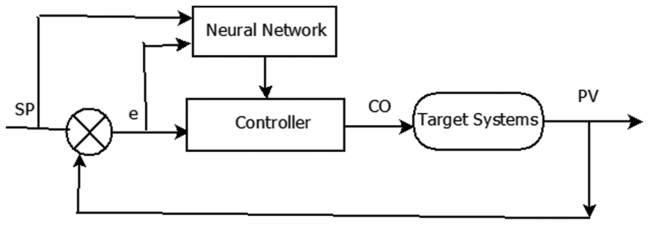

Figure 3: The neural network PID controller

The closed loop control theory [23–25] was used in the industrial automation field. The proposed Probe interval producing engine was designed as PID neural network controller that calculated their control decision as function of the error. The error was defined as the change between the actual measured variable being controlled. That was measured as MRTD and its desired value and thereby adjusting the time interval to probe response time dynamically using the feedback from the system until the desired value was achieved in the process variable (PV). The goal was achieved with PID Controller. It had controlled their decision as the sum of three functions of the error: one was relative to the current error, the other was relative to the sum of the past measured error and the last was proportional to the current derivative of the error. In that case controller output and the measured error are related by the Eq. (2).

$$co = k_p e + k_i \int_0^t e.dt + k_d \frac{de}{dt} + \text{bias} \tag{2}$$

The PID constants K_p, K_i and K_d had defined the weight of three components of the error which included request distribution pattern, request variability, server failure and link down. The neural network PID controller was implemented in probing time producing engine that generated the probe interval based on the information from the server performance monitor and calculated the maximum response time difference of the servers in the server farm dictating the server load imbalance. As the neural network PID Controller tries to cancel the error, it computes the optimal time to probe the server in server farm for calculating the real time response of all servers to proximate PV and SP.

Algorithm 3:

Input: Application delay tolerance as Global Threshold values in terms of maximum response time difference (MRTD) of every server in the server farm as L and U dictated by Neural network in PID Controller

Output: Probe Interval for next Iteration as θin terms of θ_{new}

(Continued)

Algorithm 3 (continued)

```
1.while system starts do
2.      θ = L
3.      If current time % θ == 0 then
4.      Obtain the recent response time of every server in server farm
5.      Calculate the MRTD with the formula Maximum (response time of all the servers) − Minimum
(response time of all the servers)
6.          If MRTD < L
7.  $\theta_{new} = \left( {MRTD}/{\theta} \right) * L$
8.          else if MRTD > U
9.              $\theta_{new} = \left( {MRTD}/{\theta} \right) * U$
10.          else
11.              $\theta_{new} = \theta$
12.      End if
12.      End if
13.      Set the $\theta = \theta_{new}$ as probe interval for next iteration
14. End While
```

3.4 Server Chooser

This module checks for each PACKET_IN message for new flow to be processed by the server or whether it is new user service request. If it is so, the module fetches the data from both server load imbalance and traffic volume collector module for calculating the metric $WANV_i$ to find the weighted average value of both minimum response time and port of the switch traffic volume as defined by SD_WLB using the formula

$$WANV_i = \alpha\left(\frac{ASRT_i - MINSRT_i}{MAXSRT_i - MINSRT_i}\right) + \beta\left(\frac{ASPT_i - MINSPT_i}{MAXSPT_i - MINSPT_i}\right) \tag{3}$$

where α and β are the variables which take the value between 0 and 1 and defined by the network administrator where $ASRT_i$ is the corresponding average server response time of i^{th} server for n recent response times of i^{th} server and $ASPT_i$ is the corresponding average switch's port traffic of i^{th} port of the switch for n recent values of traffic volumes of i^{th} port of the switch. The first term of Eq. (3) holds the standardized average response time value of i^{th} server and second term is the normalized form of port i traffic volume. $WANV_i$ is weighted average value of port traffic and server response time. The server selector component computes the values for (3) on the arrival of a new user service request for every server in the server farm. Finally, the server chooser picks the best server which is having minimum $WANV_i$. The Working principle of server chooser is shown in Algorithm 4.

Algorithm 4:

Input: Server response time and Switch's Port Traffic

Output: Best server to process user request

```
1.if controller receives a PACKET_IN message then
2.     If request is user service request then
3.         for each server do
4.     fetch n recent data for Response time from SQLite database;
5.     fetch n recent data for port traffic volume from SQLite database;
6.          calculateWANV$_i$ using the formula (3);
7.           select server with minimum WANV$_i$
8.         make an entry to the current flow in the switch with target address as chosen server address
9.         end for
10.        else
11.     send to other modules
12.     end if
13. end if
```

The LBBNNC works as follows: Initially a user request is directed to the virtual load balancer IP, but initially the switch does not possess flow entry, so it packs the user service request as PACKET_IN Message and send it to Ryu Controller to dictate on how to handle with this flow. LBBNNC was implemented in the controller that made the entry for suitable rules of the present flow traffic in the switch. The subsequent flows that match the rules are transmitted to the chosen server by the switch. The proposed system had its merits in handling the inconsistent traffic by calculating the Maximum Response Time Difference (MRTD) among all servers in the server farm. Using the MRTD value, the controller adjusts the probe interval bestowed by the formula (1) and the server with minimum weighted average normalized metric which is selected for processing the request till the next probe. The Optimization engine (Neural network PID Controller) in the probe interval generator computes the change between ideal load and server load and balances the overloaded server and under loaded server by choosing a server with WANV$_i$. While the traffic increases, the probe interval is proportionally reduced and while the traffic decreases, the probe interval is proportionally increased. When the traffic is high, the request is not queued up to the same server for a long time because of reduction in probe interval. The proposed algorithm possesses two advantages; one is the handling of inconsistent traffic condition with maximized throughput and the other is optimal probe message overhead.

4 Performance Evaluation

Experiment was done using Mininet emulator to match the network with 1 switch, 1 controller and 18 host machines. The process level virtualization was carried out to emulate switch and hosts, RYU Controller was used as remote controller in the experiment and OpenFlow Protocol was used as south bound communication protocol to communicate with controller and switch. The software which are used in the experiment are tabulated in Tab. 1.

Table 1: Software used in the implementation and experiment

Mininet	Network emulator	Mininet 2.2.1
Open V switch [26]	Virtual SDN switch	2.3.0
Ryu	SDN controller	4.30
Openflow	Southbound protocol	1.3
Ubuntu	Operating system	14.04
Python	Programming language	2.7.6
Web server	Apache2	2.4.7
Httperf	Web server benchmarking tool	0.9

In the experiment, a single topology was used with 1 switch (OpenVSwitch) and 18 nodes or hosts, out of which 3 nodes were configured as apache web servers. The servers had the running version 2.4.7 of the Apache http server. The Apache daemon was configured according to the report [27]. Each Server was configured to serve a python file with execution time as 0.3 seconds

Httperf version 0.9 was installed on client node. An instance of httperf running on host issues, the http request to apache web server to access python file. The aim of the experiments is to evaluate the load balancer performance which needs high throughput and minimum error rate. Tab. 2 depicts the important evaluation parameters.

Table 2: Important parameters in the simulation experiment

Usage	Parameter	Value
Evaluation network topology	Single topology defined by mininet	1 switch, 1 controller and 18 hosts No of servers: 03 No of clients: 04
Server capacity	Apache with multithreading capacity	Default values of Apache2 that comes with Httpd configuration
Implementation of closed loop control theory	Set point value used in probe interval producing engine	0.0025 s
Implementation of server selector component	α and β	0.5

The Load balancer had a virtual IP address defined in RYU Controller, and all the requests of the clients were sent to the specified virtual IP address by hiding the server IP address from clients. VIP address of load balancer might be publicly available to all clients. The load balancer with VIP would distribute the traffic among all servers using a load balancing algorithm based on closed loop control theory. Address mapping of the client to the server and the server to the client was indirectly carried out by the load balancer module through the destination port number in TCP or UDP segment header. The workload mix with the auto bench was designed from the demanded request rate from 10 per second to 100 request per second and the demanded requested rate was increased at the rate of 5 requests per second.

The experiment was executed with 50000 requests with timeout value as 20 s in Httperf for four different load balancing algorithms namely round robin, LBBSRT, SD-WLB and the proposed LBBNNC. From Fig. 4, it is evident that the algorithm LBBNNC has shown better performance for the achieved request rate, in spite of its greater number of concurrent connections. Fig. 5 depicts that the proposed LBBNNC algorithm has shown higher average response rate compared to the other algorithms. Fig. 6. shows the minimum number of errors in LBBNNC compared to other algorithms. The LBBNNC has provided better performance in terms of average response rate, achieved request rate, error and average response time as shown in the Fig. 7.

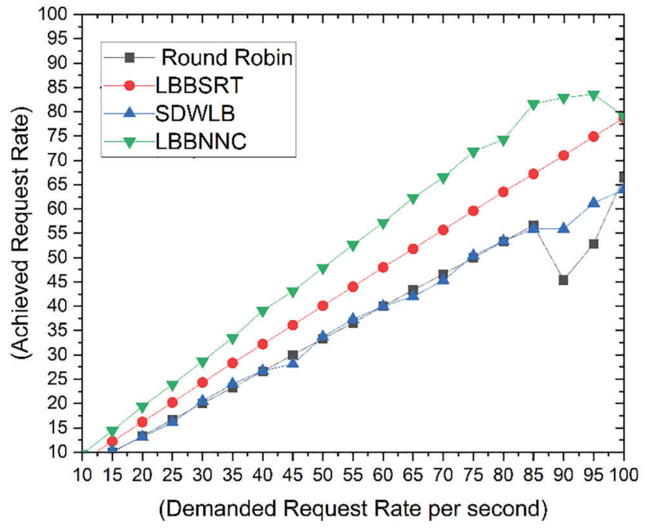

Figure 4: Achieved request rate

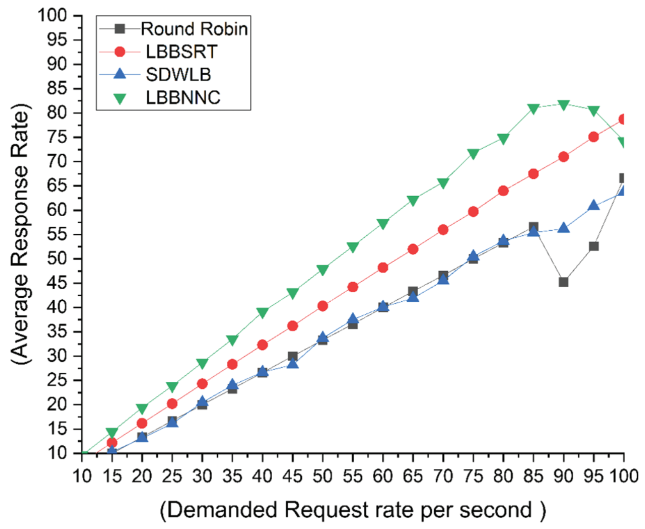

Figure 5: Average response rate

The achieved request rate and the average response rate of the four schemes (Round Robin, LBBSRT, SDWLB, and LBBNNC) were 35, 44, 36 and 51 respectively. Compared to Round Robin, LBBSRT and SDWLB, the proposed scheme had showed improvement in achieving request rate and response rate by 34%,16% and 42% respectively.

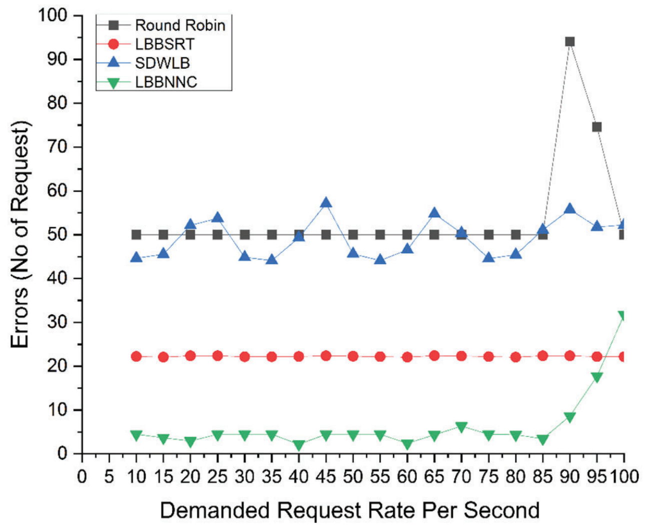

Figure 6: Request rate *vs*. errors

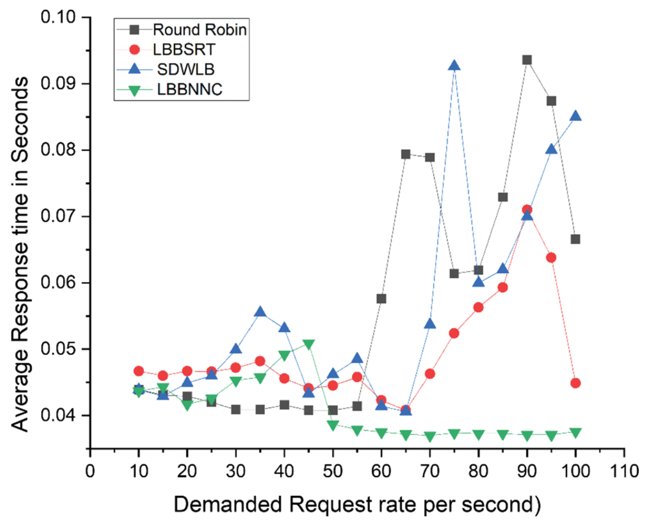

Figure 7: Average response time

The errors that were produced from the four schemes (Round Robin, LBBSRT, SDWLB, and LBBNNC) were 54, 22, 49 and 7 respectively. Compared to Round Robin, LBBSRT and SDWLB, the proposed scheme had showed improvement in minimizing the errors by 671%, 214%, 600% respectively.

The average response time of the server for four schemes (Round Robin, LBBSRT, SDWLB, and LBBNNC) were 0.057, 0.049, 0.056 and 0.041 s respectively. Compared to Round Robin, LBBSRT and SDWLB, the proposed scheme had showed improvement in minimizing the average response time by 39%, 20% and 37% respectively.

5 Conclusion

SDN paves the way for better arrangement of networks and provides the way to improve the load balancing solution. The proposed scheme had proved to be better than Round Robin, LBBSRT and

SDWLB by adaptively adjusting the time window to probe the server farm for the extraction of the maximum response time difference. It was due to control the system using closed loop control theory so as to achieve minimum error rate and maximum average response rate of user requests. From the study, it was implied that further tuning of the algorithm with different values of L and U with varied online neural network model was required to be done for the enhancement of the performance in terms of average response time of user requests. Single point of failure was the bottleneck that was associated with the proposed algorithm. In order to eliminate the problem, it is intended to use multiple controllers instead of single controller and to address the issues pertinent to the synchronization of multiple controllers to have more resilient load balanced SDN networks. In future, the same load balancing can be enriched for cloud computing framework.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that there is no conflict of interest regarding the publication of the paper.

References

[1] R. Buyya, A. M. K. Pathan, J. Broberg and Z. Tari, "A case for peering of content delivery networks," *IEEE Distributed Systems Online*, vol. 7, no. 10, pp. 1–3, 2006.

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

[3] O. N. Foundation, "Software-defined networking: The new norm for networks," 2012. [Online]. Available: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/.

[4] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[5] A. Lara, A. Kolasani and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2013.

[6] N. Mc. Keown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson *et al.,* "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[7] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient SDN architecture," in *Proc. DRCN, IEEE*, Paris, France, pp. 145–151, 2016.

[8] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Computer Networks*, vol. 112, no. 7, pp. 279–293, 2017.

[9] H. Zhong, Y. Fang and J. Cui, "Reprint of LBBSRT: An efficient SDN load balancing scheme based on server response time," *Future Generation Computer Systems*, vol. 80, no. 1, pp. 409–416, 2018.

[10] K. Soleimanzadeh, M. Ahmadi and M. Nassiri, "An SDN-aided mechanism for web load balancing based on server statistics," *ETRI Journal*, vol. 41, no. 2, pp. 197–206, 2019.

[11] S. Kaur, K. Kumar, J. Singh and N. S. Ghumman, "Round-robin based load balancing in software defined Networking," in *Proc. INDIA Com*, Delhi, India, pp. 2136–2139, 2015.

[12] K. Kaur, J. Singh and N. S. Ghumman, "Mininet as software defined networking testing platform," in *Proc. ICCCS*, Punjab, India, pp. 139–142, 2014.

[13] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov *et al.,* "A fast and reliable software network load balancer," in *Proc. NSDI*, California, USA, pp. 523–535, 2016.

[14] R. Gandhi, H. Harry Liu, Y. Charlie Hu, G. Lu, J. Padhye *et al.,* "Duet: Cloud scale load balancing with hardware and software," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 27–38, 2014.

[15] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu *et al.,* "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. SIGCOMM, ACM*, Chicago Illinois USA, pp. 503–514, 2014.

[16] N. Katta, M. Hira, C. Kim, A. Sivaraman and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proc. SOSR '16*, Santa Clara CA, USA, pp. 1–12, 2016.

[17] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg *et al.,* "Ananta: Cloud scale load balancing," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 207–218, 2013.

[18] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu *et al.,* "Fdalb: Flow distribution aware load balancing for datacenter networks," in *Proc. IWQoS*, Beijing, China, pp. 1–2, 2016.

[19] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown and R. Johari, "Plug-n-Serve: Load-balancing web traffic using OpenFlow," *ACM Sigcomm Demo*, vol. 4, no. 5, pages 6, 2009.

[20] B. M. Moreno, C. P. Salvador, M. E. Domingo, I. A. Pena and V. R. Extremera, "On content delivery network implementation," *Computer Communications*, vol. 29, no. 12, pp. 2396–2412, 2006.

[21] RYU project team, Copyright 2014, **Project Ryu**," [Online]. Available: https://osrg.github.io/ryu-book/en/html.

[22] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[23] W. J. Rugh, *Linear System Theory (2nd Ed.)*. USA: Prentice-Hall, Inc, 1996.

[24] S. Maheswaran, S. Sathesh, G. Saran and B. Vivek, "Automated coconut tree climber," in *Proc. 2017 I2C2, IEEE*, Tamil Nadu, India, pp. 1–6, 2017.

[25] S. Maheswaran, B. Vivek, P. Sivaranjani, S. Sathesh and K. Pon Vignesh, "Development of machine learning based grain classification and sorting with machine vision approach for eco-friendly environment," *Journal of Green Engineering*, vol. 10, no. 3, pp. 526–543, 2020.

[26] D. Mosberger and T. Jin, "HTTP ref—a tool for measuring web server performance," *ACM Sigmetrics Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[27] J. T. J. Midgley, "Autobench an HTTP benchmarking suite," Copyright (C) 2001–2003, [Online]. Available: https://github.com/menavaur/Autobench.