Traffic Based Load Balancer using Software Defined Networking (SDN)

Team: Netclaws

Brinda Raghunatha Bharadwaj *Chandana Boodanur Shivananda* *Deepashree Puttenhalli Vajrapp*
Pooja Sreedhar Garla *Sindhu Valasa Reddy* *Sourabh Kumar*

Department of Electrical Engineering
University of Southern California
Los Angeles, California

Abstract — **Network Load Balancing is a feature which distributes incoming network traffic among group of multiple servers within a cluster to avoid overloading any one host and improve performance. The algorithm, embedded functions, and network equipment used to implement and manage a load balancer has an impact on its cost. Software Defined Networking (SDN) offers a cost-effective and flexible approach in implementing a load balancer. Moving away from traditional hardware-based networking approach, this project implements traffic based load balancing with the help of software where, incoming traffic is distributed among the servers based on the loads already being served at each server. Here, execution time of each task queued at the server is considered before new request is sent to that server. The new traffic based load balancing approach in SDN reduces the cost, offers flexibility in configuration, reduces service time for each task, minimizes queuing delay and provides automation and facilitates building a network without requiring the knowledge of any vendor-specific software/hardware.**

***Keywords*—Load Balancer, Mininet, DETERLab, OpenFlow, POX, Python, Round Robin, Software Defined Networking (SDN), Traffic Based Load Balancing.**

I. Introduction

In a data center environment, the load balancer is an integral part of the network ecosystem. Load balancing distributes work among resources in such a way that no one resource should be overloaded, and each resource can have improved performance, depending on the load balancing algorithm.

Software Defined Networking (SDN) is a new approach to facilitate network management that has been gaining widespread attention [2]. Traditional networking involves buying the hardware and software together from a vendor and configuring them as per the requirements. In traditional networking, it is not possible to separate the data plane from the control plane. Therefore, the customer had to rely on the vendor for software bug fixes, additional features, and licenses. It is not possible for the customer to modify the vendor software code as per needs of the network. One of the major concern about hardware load balancing is cost for implementation and modifications. Additional costs are imposed when hiring professionals who are skilled in the vendor-specific configuration [5]. SDN tries to eliminate this vendor dependence and is a paradigm shift from closed vendor-specific networking to an open networking system. In SDN, a centralized controller deploys network flows in the bare-metal network switches. Therefore, the hardware for networking devices is available at a lower price and the software configuration can be modified as per needs of the customers. This generic piece of hardware can be programmed using an Application Programmable Interface (API)[1]. OpenFlow is a software protocol used in software configuration. OpenFlow enables the SDN controller to communicate with networking devices.

The project aims to implement a traffic based load balancer to the controller. Traffic based load balancer is advantageous over many load balancing algorithms available in the market today like round robin, weighted round robin, least cost based etc., Over burdening single server is eliminated, thus incoming traffic is directed to servers based on the required service time for already existing tasks at each server. Load balancer keeps the note of different task assigned to each server and the time required to service those tasks. Thus, the overall queuing delay and the time required for servicing new task is reduced. Hence, the performance of the system can be increased. By using SDN, the cost of having a dedicated vendor specific load balancer is reduced, vendor dependence is eliminated, and flexibility in terms of modification of the code is achieved.

The later of this paper is systematized as follow. Section II narrates about the related works in this field. Section III is the main backbone of the paper which talks about the research that is architecture, implementation, execution and the experimental results, Section IV set out conclusion.

II. Related Works

The numbers of papers published related to Load balancing based on SDN had an exponential increase recently, with most of them coming in the last decade.

The paper we referred[4], talk about the importance and advantages of SDN based load balancer compared with the traditional hardware based. Hardware based load balancer are highly application centric, it requires dedicated hardware for particular type of service. It is inflexible, non-programmable and non-centralized. More over hardware based load balancer are more expensive. Thus, the need for SDN based load balancer which is directly programmable, it has centralized management, responds quickly to changes, enables innovation, makes network more flexible and agile. SDN approach separates network control from forwarding plane where the control plane is directly programmable.

Web browsing utilizes a client-server model where a client requests data from a server and the server responds. Depending upon the type of service delivered, there can be millions of connections to the server at any time. In this case, it is necessary to ensure that the critical services like money transfer, online examinations, and business transactions are reliably delivered. A load balancer is a device which serves a key role in delivering these services and keeping the network infrastructure running. The load balancer attempts to distribute client requests among the cluster of web servers efficiently [1]. This allocation helps improve performance and reduce the possibility of a server being overloaded. Certain load balancers also provide failover capabilities. If a server fails, the load balancer provides fault tolerance by directing the requests to the remaining servers [2]. Several companies manufacture dedicated load balancers like F5 Networks, A10 Networks, Kemp technologies, and Barracuda Networks [3]. Different load balancing algorithms are available such as round robin, weighted round robin, server CPU utilization based, and IP-based hashing. A round robin load balancer distributes the client requests among the servers one-by-one. For example, the 1st client request is forwarded to server 1, 2nd request to server 2 and so on. But the problem with round robin is like the server capacity is not taken into account, single server may be overloaded with huge tasks. Thus, the importance for better algorithm comes to picture.

III. Research

Load balancing is the process of improving the performance of a parallel and distributed system through a redistribution of load among the processors or nodes. In a distributed network of computing hosts, the performance of the system can depend crucially on dividing up work effectively across the participating nodes. By distributing traffic efficiently so that individual servers are not overwhelmed by sudden fluctuations, it can save power and improve resource utilization[2].

Round-robin load balancing is one of the simplest methods for distributing client requests across a group of servers. Going down the list of servers in the group, the round-robin load balancer forwards a client request to each server in turn. When it reaches the end of the list, the load balancer loops back and goes down the list again. The main benefit of round-robin load balancing is that it is extremely simple to implement [5]. Using round robin load balancer method incoming packets or requests are distributed across the number of available switches in the network i.e. switch cluster. One can choose this load balancer, if all the switches present in the cluster have the same capability and handle equal amount of load. By considering this constrain, the round robin load balancing method is a simplest and effective method for load distribution. If the switches with equal capabilities are used for round robin load balancing this means that less capable switch gets flooded with the number of packets or requests even though it is not able to process that much number of requests. However, it does not always result in the most accurate or efficient distribution of traffic, because many round-robin load balancers assume that all servers are the same: currently up, currently handling the same load, and with the same storage and computing capacity.
The limitations of round robin load balancer are as follows -
1. Statically configured
2. Uneven load balancing

Load balancer use dedicated hardware. That hardware is expensive and inflexible. The control plane is decoupled from data plane in the Software Defined Networking (SDN) architecture, under which network intelligence and state are logically centralized and the underlying network infrastructure is abstracted from the applications [4]. By OpenFlow, researchers could use programmable features to design and experiment with new network technologies and architectures in real network. Users can define the logical network topology with software method, without concern for underlying network structure. By distributing traffic efficiently so that individual servers are not overwhelmed by sudden fluctuations, it can save power and improve resource utilization.

The OpenFlow standard enables an alternative approach where the commodity network switches divide traffic over the server replicas, based on packet-handling rules installed by a separate controller [7]. OpenFlow is first posed to separate the data plane and control plane from each other among traditional network equipment. By OpenFlow, researchers could use programmable features to design and experiment with new network technologies and architectures in real network. We will be using a POX controller in an SDN network. We wrote our own traffic based algorithm on POX controller.

A. *Architecture*

The server load balancing network model based on SDN is shown in Figure I. The architecture is implemented on Deterlab. DETERLab is a security and education-enhanced version of Emulab. DETERLab is a shared testbed providing a platform for research in cyber security and serving a broad user community, including academia, industry and government. Our Network consists of two clients, three server, a switch and a load balancer. Every time when the client requests for the task, the switch will forward it to the load balancer. Load balancer with the help of its algorithm decides and assigns the client task to the respective servers. The configuration, that is the algorithm which was pushed on the virtual switch by the controller became simple and independent of the vendor and operating system. Using the Python program running on the controller, the configuration could be changed to which ever algorithm required.

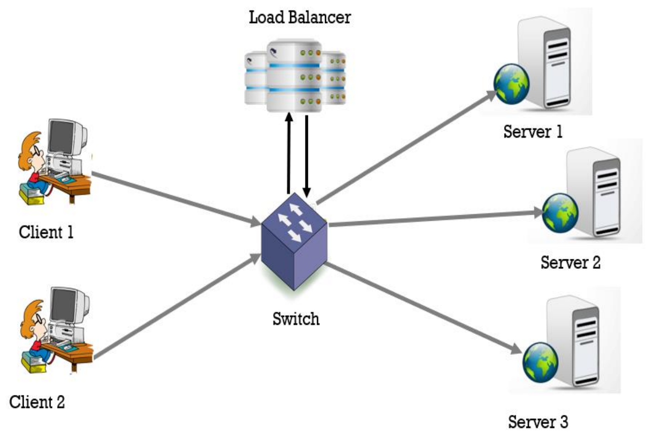


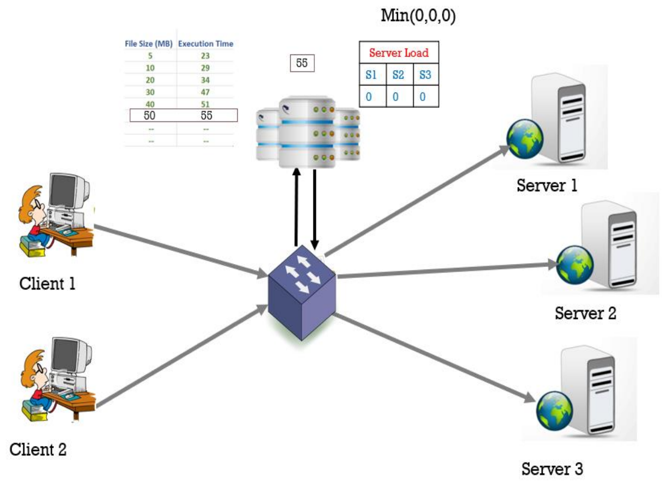
FIGURE I: Server Load Balancer network model based on SDN.

B. Implementation

The software-based load balancer was implemented using POX as follows. First, the virtual network topology was implemented in the mininet virtual environment, later using DETERLab. The test topology was configured to have a total of five hosts connected to the Open vSwitch. In which three were server and two were clients. The hosts are made to start with a dynamically assigned but easily readable virtual IP addresses. The Open vSwitch was configured to connect to the remote POX controller and communicate using the OpenFlow protocol. When the POX controller is started at the command line, it invokes the load balancer program. The controller thereby implements the traffic based functionality in the SDN. The controller installs flow table entries in the Open vSwitch according to the criteria defined in the round robin load balancer algorithm.

Second, the three hosts were configured as Servers. While the remaining hosts as clients. The virtual IP address is used by the clients to request tasks from the server. For every subsequent request made by the client to the virtual IP address, a different server would take up the task depending on the minimum server load criteria.

The load balancer algorithm works as follows - the Open vSwitch listens to all incoming packets from the client with a destination IP address as the virtual IP address. Controller already has a flow table which contains file size of the task and the execution time required by the servers to serve that task. The packet processing is flow-based and thus, the system matches the packet information with that of an existing flow table entry and obtains the execution time from that. Controller also maintains another table which dynamically keep tracks of load at each servers in terms of time required to serve all the queued tasks at particular servers. Initially, when all the server load is zero. The server is selected in a round robin fashion for every session. Later in the next iteration each server will be executing different tasks requiring different execution time. Next when the packet arrives it matches itself in the file size table and obtains the execution time and in server load table it sees for the server with least load, updates the table by adding its execution time to server's time which is least and informs the packet to move to that particular server. Next time when new packet arrives, same thing will be followed. Once the task is done at the server.It will update the controller about the work done status and the corresponding execution time of the server will be deducted to update the table for next process. The algorithm uses the OpenFlow libraries to listen and forward the packets while POX controller libraries are used map and modify the flow tables and also to take decision about sending the tasks to *server.*



File Size (MB)	Execution Time
5	23
10	29
20	34
30	47
40	51
50	55
--	--
--	--

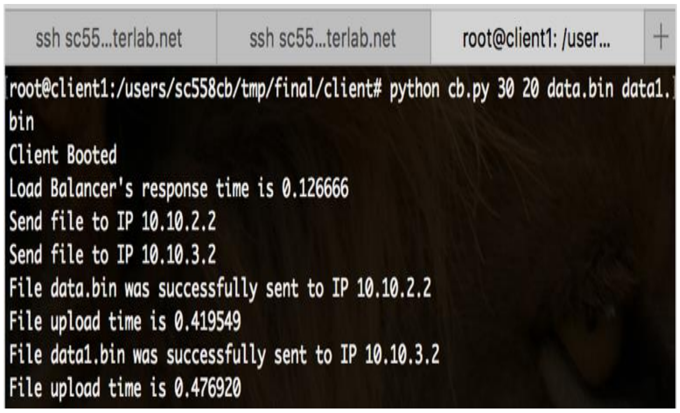
Server Load		
S1	S2	S3
0	0	0

FIGURE II: Server Load Balancer network model based on SDN with the flow table entries at the controller.

C. Execution

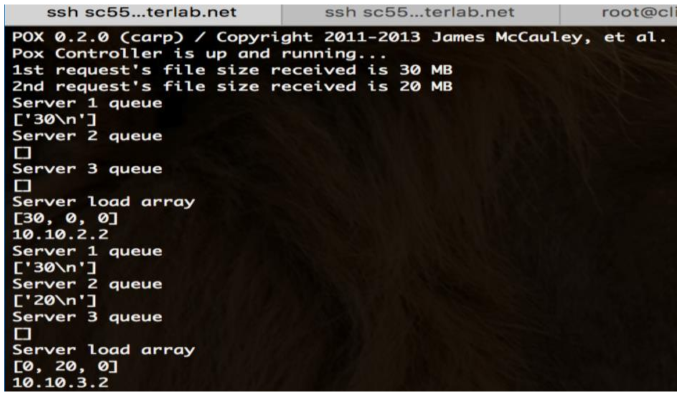
Two clients send requests simultaneously. The requests are directed to the load balancer. The first request that reached the load balancer contains the file size which is mapped to the execution time which is termed as the load on the server. The controller will run the load balancing algorithm and choose the server with least load. This IP is sent back to the client. The client then contacts the server with this new IP that it got from the controller and uploads the file. Once, the server is done with a task, it sends back the load balancer that he is done. Therefore, the controller decrements and updates the load on the respective server. And the cycle repeats. Thereby, ensuring the working of traffic based load balancing.

Initially, Client1 tells the load balancer that it wants to upload 2 requests of size 30 MB and 20 MB onto the server as in figure 1. The load balancer takes the 1st request which is 30 MB. At first, all the 3 servers are free. Thus, it schedules 30 MB to server1. The next request is the 20 MB file. Now, server2 and server3 are free. Thus, it schedules it to server 2 and the corresponding server's load gets queued up as seen in figure 2.



```
root@client1:/users/sc558cb/tmp/final/client# python cb.py 30 20 data.bin data1.
bin
Client Booted
Load Balancer's response time is 0.126666
Send file to IP 10.10.2.2
Send file to IP 10.10.3.2
File data.bin was successfully sent to IP 10.10.2.2
File upload time is 0.419549
File data1.bin was successfully sent to IP 10.10.3.2
File upload time is 0.476920
```

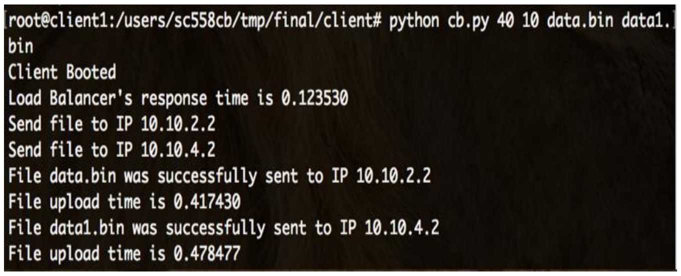
Figure 1



```
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
Pox Controller is up and running...
1st request's file size received is 30 MB
2nd request's file size received is 20 MB
Server 1 queue
['30\n']
Server 2 queue
[]
Server 3 queue
[]
Server load array
[30, 0, 0]
10.10.2.2
Server 1 queue
['30\n']
Server 2 queue
['20\n']
Server 3 queue
[]
Server load array
[0, 20, 0]
10.10.3.2
```

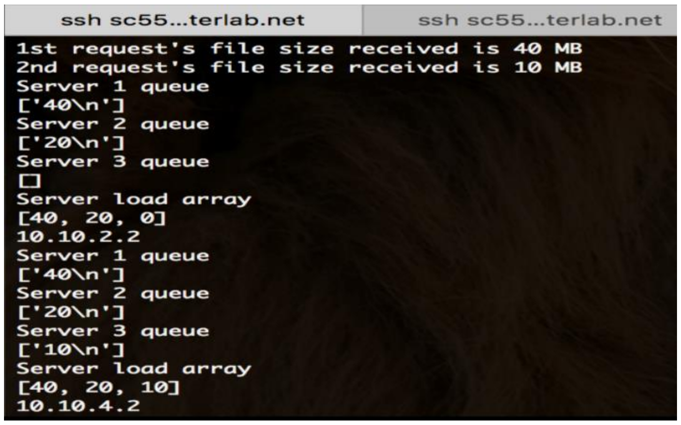
Figure 2

Immediately, the client sends 2 more requests simultaneously, it tells the load balancer it wants to upload a 40 MB and a 10 MB file as in figure 3. Now, server 1 has finished its previous job and is free, thus the server load list at the load balancer of server 1 has dequeued and decremented. Thus, 40 MB gets scheduled to server1. The next request which is the 10 MB file is scheduled to server 3 because it has the least load among all three as seen in figure 4.



```
root@client1:/users/sc558cb/tmp/final/client# python cb.py 40 10 data.bin data1.
bin
Client Booted
Load Balancer's response time is 0.123530
Send file to IP 10.10.2.2
Send file to IP 10.10.4.2
File data.bin was successfully sent to IP 10.10.2.2
File upload time is 0.417430
File data1.bin was successfully sent to IP 10.10.4.2
File upload time is 0.478477
```

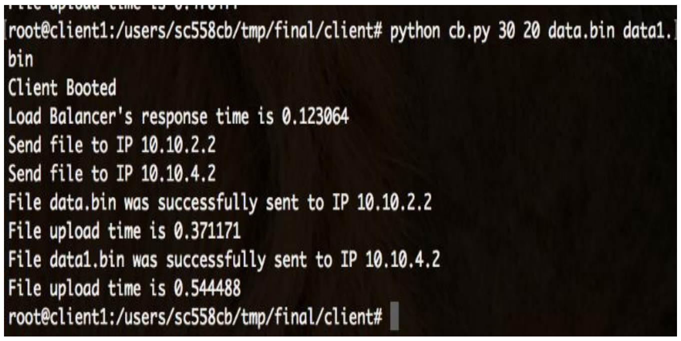
Figure 3



```
1st request's file size received is 40 MB
2nd request's file size received is 10 MB
Server 1 queue
['40\n']
Server 2 queue
['20\n']
Server 3 queue
[]
Server load array
[40, 20, 0]
10.10.2.2
Server 1 queue
['40\n']
Server 2 queue
['20\n']
Server 3 queue
['10\n']
Server load array
[40, 20, 10]
10.10.4.2
```

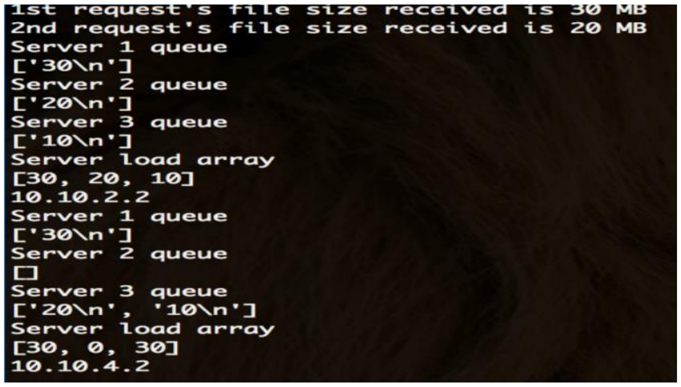
Figure 4

Similarly, the next two requests are a 30 MB and a 20 MB file which is scheduled to server 1 and server 2 according to the least load principle as explained prior which can be seen in figure 4 and 5.



```
root@client1:/users/sc558cb/tmp/final/client# python cb.py 30 20 data.bin data1.
bin
Client Booted
Load Balancer's response time is 0.123064
Send file to IP 10.10.2.2
Send file to IP 10.10.4.2
File data.bin was successfully sent to IP 10.10.2.2
File upload time is 0.371171
File data1.bin was successfully sent to IP 10.10.4.2
File upload time is 0.544488
root@client1:/users/sc558cb/tmp/final/client#
```

Figure 5



```
1st request's file size received is 30 MB
2nd request's file size received is 20 MB
Server 1 queue
['30\n']
Server 2 queue
['20\n']
Server 3 queue
['10\n']
Server load array
[30, 20, 10]
10.10.2.2
Server 1 queue
['30\n']
Server 2 queue
[]
Server 3 queue
['20\n', '10\n']
Server load array
[30, 0, 30]
10.10.4.2
```

Figure 6

D. *Experiment Results*

Let us look at the experimental results of this paper. We conducted the experiment for the round robin algorithm and traffic based algorithm on mininet. First performance we are talking about is server response time, the server response time is the time duration between a particular task getting added to the server's queue until the task has been completed by the server. From the figure we can see that for round robin approach server execution time is greater than that for traffic based approach. Round robin schedules the tasks irrespective of how big or tiny the tasks are leading to starvation problem for tiny jobs and thereby increasing the response time.

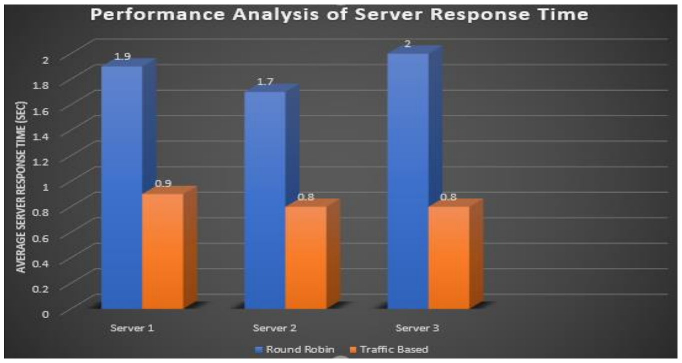


FIGURE III: Comparison of Server Response time for Round Robin and Traffic Based Load Balancer.

Second performance metric is the load balancer response time, which is the time taken by the load balancer to decide to which server the task must be allocated. We see that traffic based load balancer takes more time than round robin load balancer because in traffic based load balancer pre-processed value must be fetched and computed from a file but whereas for round robin load balancer the next server is calculated on the fly without involving any of those file computations. This can be visualized from figure II.

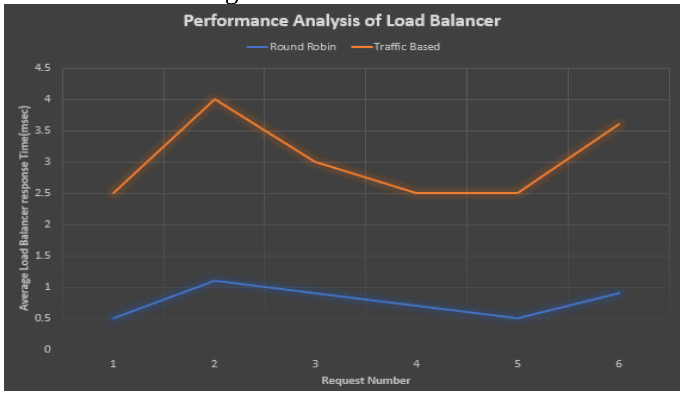


FIGURE IV: Comparison of Load Balancer Response time for Round Robin and Traffic Based Load Balancer.

Third performance metric we considered is the total response time which is the sum of the server response time and the load balancer response time or the time taken for the client request to be initiated and completed. We see that through load balancer response time for traffic based was greater than round robin. The overall response time for traffic based load balancer turns out to be lesser than round robin load balancer because the dominant factor is the queuing time or the server response time which is lower for the traffic based load balancer case than for round robin load balancer, thus reducing the overall time for traffic based load balancer scenario. The performance analysis for total response time can be see in figure III.

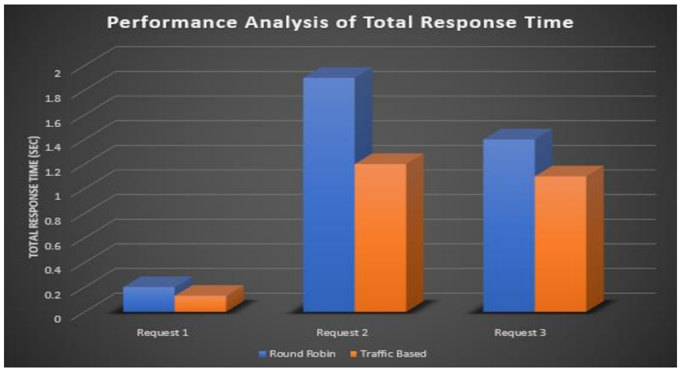


FIGURE III: Comparison of Total Response time for Round Robin and Traffic Based Load Balancer.

Fourth performance we measured was server utilization for two different approach. In round robin the load is not equally distributed among three servers. Hence server 1 has higher utilization compared to other two. Whereas in traffic based the load is almost equally distributed among the three servers. Hence all the three servers are equally busy. This could be seen in Figure IV.

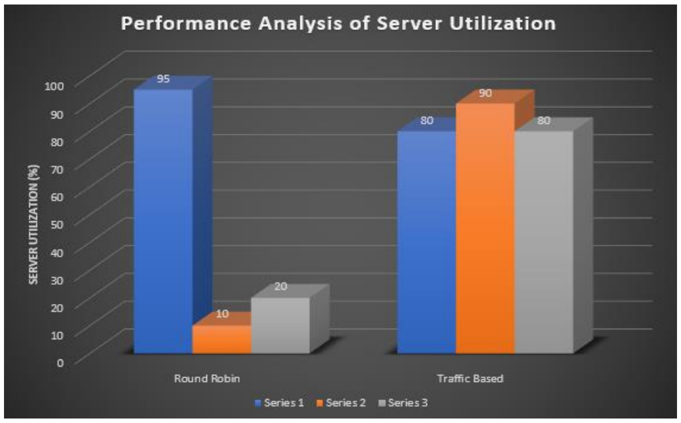


FIGURE IV: Comparison of Server Utilization for Round Robin and Traffic Based Load Balancer.

IV. CONCLUSION

The Traffic Based load balancer was successfully implemented in a POX SDN controller, and the software-based nature of a load balancer helped reduce the cost of implementation for users. The project is vendor independent because it relies on open standards. This provides flexibility in configuration and deployment by allowing the company to install the software on any white-box or OpenFlow supported device. Thus, significantly reducing the period of time required to deploy new services when compared to a traditional hardware-based approach. The team can publish this project in the upcoming ACM and IEEE conferences. For instance, the 13th International Conference on Network and Service Management; 2017 International Conference on Computer Systems, Electronics and Control (ICCSEC); 2017 4th International Conference on Networking, Systems and Security (NSysS).

Acknowledgment

We would like to thank Professor, Young Cho for his guidance and support throughout the project and also for his constructive comments during review sessions.

Reference

[1] Ghaffarinejad, Ashkan, and Violet R. Syrotiuk. "Load balancing in a campus network using software defined networking." In Research and Educational Experiment Workshop (GREE), 2014 Third GENI, pp. 75-76. IEEE, 2014.

[2] Zhou, Yuanhao, Li Ruan, Limin Xiao, and Rui Liu. "A method for load balancing based on software defined network." Advanced Science and Technology Letters 45 (2014): 43-48.

[3] Nunes, Bruno Astuto A., Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. "A survey of software-defined networking: Past, present, and future of programmable networks." IEEE Communications Surveys & Tutorials 16, no. 3 (2014): 1617- 1634.

[4] Kaur et al. "Round-robin based load balancing in software defined networking." In Computing for Sustainable Global Develop. (INDIACom), 2015 2nd Int. Conf. , pp. 2136-2139. IEEE, 2015.

[5] Kaur, Sukhveer, Japinder Singh, and Navtej Singh Ghumman. "Network programmability using POX controller." In ICCCS International Conference on Communication, Computing & Systems, IEEE, no. s 134, p. 138. 2014.

[6] Nanxi Kang, Zhenming Liu, Jennifer Rexford and David Walker. Optimizing the “One Big switch” Abstraction in software-defined networks. In the Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, December 2013.

[7] Yannan Hu, Wendong Wang, Xiangyang Gong, Xirong Que and Shiduan Cheng, BalanceFlow: Controller Load Balancing For OpenFlow Networks, State Key Laboratory of Networking and Switching Technology Beijing University of Posts and Telecommunications, Beijing, 100876, China, 2012

[8] A Load Balancing Method Based on SDNMao Qilin, Shen WeiKangNanjing Institute of Technology, Nanjing, Jiangsu, 211167, China

[9] Analysis of Issues with Load Balancing Algorithms in Hosted (Cloud) Environments Branko Radojević, Mario Zagar Croatian Academic and Research Network (CARNet), Zagreb, Croatia, Faculty of Electrical Engineering and Computing (FER), Zagreb. Croatia Branko.Radojevic@CARNet.hr, Mario.Zagar@fer.hr

[10] Eyal Cidon, Sean Choi, Sachin Katti, and Nick McKeown. AppSwitch: Application-layer Load Balancing within a Software Switch. In Proceedings of the First Asia-Pacific Workshop on Networking, August 03-04, 2017, 7 pages.