National Journal of Parallel and Soft Computing, Volume 01, Issue 01, March-2019

Server Load Balancing in Software Defined Networking

Arkar Soe Linn, Su Hlaing Win, Su Thawda Win
arkarsoelinn2015@gmail.com, suhlaingct1995@gmail.com, stdwthawda@gmail.com
University of Computer Studies, Mandalay, Myanmar

Abstract

Software Defined Networking (SDN) is the new emerging field in the era of information technology. The SDN is more flexible and programmable than the traditional network. The one of the important usages of SDN is the server load balancing strategy. The load on the servers is increasing as day by day with the internet usage. Therefore, there is a need to balance these load on servers in order to provide efficient services to end users without any delay. In our proposed system, Random, Round Robin and Weighted Round Robin load balancing strategies are implemented using an OpenFlow switch connected to a POX controller that are based on python in SDN.

1. Introduction

The load balancing method plays as the significant role in network. The load balancing network is divided into server and link load balancing methods. This system proposes the server load balancing method. In this system, the central load balancer is connected to each several target servers and hosts. This load balancer will receive the requests from multiple clients. Depending upon the various load balancing strategy, the purpose of the load balancer is to forward these incoming requests of the client to different servers. The SDN load balancer provides the facility to the programmers to build and design own load balancing strategy. Another strong point is that it does not require any separate hardware that behaves as a load balancer [5].

Software Defined Networking (SDN) provides a different approach in network design and management. It decouples the distributed control plane from the data plane and moves the control plane to the centralized controller. Thus, the controller has a complete view of the network topology plus the full control of network resources. Together with the controller's programmability, SDN offers efficient and flexible ways to deliver networking functions [4].

As the basic network device in data layer, OpenFlow enabled switch is used to implement data transmission function according to flow-tables allocated from controller [1]. Being as the "brain" of SDN, controller acquires application information from upper layer through the northbound interface. Flow-tables are generated in controller and allocated to OpenFlow switch through OpenFlow protocol. By acquiring network topology information, SDN controller provides the global network view for OpenFlow switch and implements the flexible network configuration and network management. As with traditional network, link redundancy technology in SDN can effectively solve the problems of network congestion and provide the robustness and stability for network. By evenly distributing traffic among multiple paths, load balance can be achieved in SDN.

The paper is organized as follows. Section 2 reviews the existing traditional load balancing schemes and introduces the overview of SDN and OpenFlow Protocol. Load balancing strategies are described in Section 3. Section 4 describes the architecture of proposed load balancer. Section 5 shows execution of the system. Section 6 covers experimental results and Section 7 contains conclusion and future work.

2. Related works

In the recent year, a single server is unable to handle all of the requests from the clients because of the large amount of traffic. So, it needs to balance the load traffic. The main purpose of load balancer is to distribute the load traffic of servers. The traditional load balancer are used to these problem but main problem with these load balancers are non-programmable, expensive hardware. Now a days SDN load balancer are used. Openflow dumb device can be converted into strong load balancer by creating SDN application (such as load balancer).

Kaur et al. proposed load balancing strategy that balance the load traffic in round robin algorithm. The capacity of server is not considered in this strategy as show in Figure 1. But the server capacity is necessary in reality because the capacity of server may vary from one server to the other. Because it couldn't happen that new server and old server have same capacity and speed.

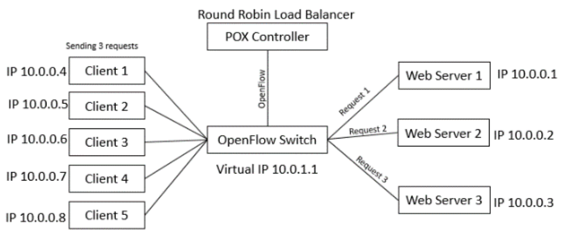


Figure 1. Round Robin Load Balancer

The proposed system solved this problem. In this paper, the Weighted Round Robin Load Balancing strategy is implemented. According to the different capacity of the servers, the different weighted are assigned to each server. The server with the highest weight handle more requests than the other servers.

2.2. SDN Overview

Software Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding plane and it is programmable. By centralizing the control plane, it expanded the possibility of network intelligence by having complete network visibility. The network infrastructure can be smartly utilized and performance of network have great opportunity for optimization [4].

In SDN, the network devices only implement the data plane. They accept instructions from the SDN controller through the OpenFlow and other southbound protocols for data forwarding. This reduces complexity of the network devices as forwarding devices no longer required to understand and implement the control plane. Figure 2 shows the SDN network architecture.

SDN tries to improve the current networks. Bellow there is a list of the main advantages of the SDN.

- It becomes easier to program the applications since the abstractions provided by the control platform and/or the network programming languages can be shared.
- All applications can take advantage of the same network information, leading to more consistent and effective policy decisions while re-using control plane software modules.
- All applications can take actions (i.e., reconfigure forwarding devices) from any part of the network. There is no need to devise a precise strategy about the location of the new functionality.
- The integration of different applications becomes more straightforward. For instance, load balancing and routing applications can be combined sequentially, with load balancing decisions having precedence over routing policies.

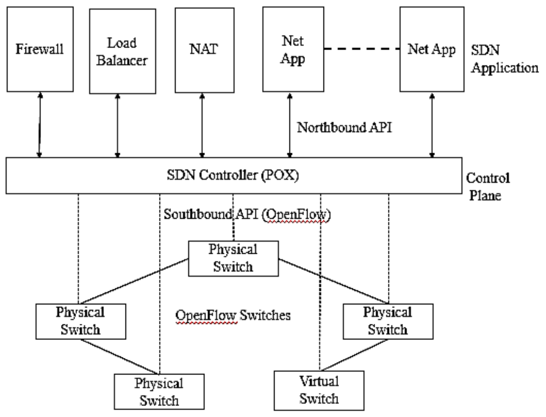


Figure 2. The architecture of SDN

2.3. OpenFlow Protocol

The OpenFlow protocol is the popular southbound protocol used for the communications between the controllers and the network elements. The OpenFlow is also the first standard comm-unication protocol defined between the control layer and the infrastructure layer in SDN architecture [3, 2]. It manages the switches in the network and allows an external entity like the controller to manipulate the flow of packets through the network. Openflow was designed as a tool focused on network research.

The Openflow architecture consists of three basic concept. (1) The network is built up by Openflow-compliant switches that compose the data plane; (2) the control plane consists of one or more Openflow controllers; (3) a secure control channel connects the switches with the control plane. All switched have tables showing the ingress and egress paths of a packet for that switch. Openflow makes use of this property and makes these tables accessible by the controller. An Openflow switch will receive its flow table entries and deletion from the controller through a secure channel.

When a new packet arrives to an Openflow switch, it will look into the flow table to find a match. If there is no match in the table, the packet will be sent to the controller. The controller processes the packet and marks the packet with an action like:

- Add a new flow for similar incoming packets
- Drop similar packets
- Tag with a queue ID

2.4. The SDN Controllers

The controller or network operating system is the heart of the SDN, which is responsible for controlling and managing all the OpenFlow switches [2]. Some of the SDN controllers used widely in academia and industry [3] are summarized as follows:

NOX is an open source development platform for C++ based Software-Defined Networking (SDN) control applications. It is developed by Nicria.

Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks and based on Beacon Controller.

Ryu is originated from NTT in Japan. Ryu is based python and it is simple and easy to use.

OpenDaylight and floodlight is based JAVA and has two major technical characteristics. One is the use of OSGi architecture and the other is the introduction of SAL.

POX is an open source SDN Controller whose modules are implemented in python language. In the proposed system, POX Controller is used to balance the load traffic of servers.

3. Load Balancing Strategies

Random Strategy: One of the simplest algorithms and still very effective is the random load balancing. The switch is connected to the controller over a secured connection using SSL and uses port 6633 on the controller to exchange Open Flow packets. Once the packets start flowing into the controller via the switch the algorithm starts assigning the servers IP and Ethernet address to each connection. It uses a random function which returns a value that is in the range of the number of servers online.

Round-Robin Strategy: Round Robin is easy to implement and understand. The round robin policy uses a circular queue to decide where to send a request. It means that this method continuously rotates a list of services that are attached to it. When the virtual server receives a request, it assigns the connection to the first service in the list, and then moves that service to the bottom of the list.

Weighted Round-Robin Strategy: The Weighted Round Robin is similar to the Round Robin in a sense that the manner by which requests are assigned to the nodes is still cyclical, albeit with a twist. In this strategy, each server receives the request from the client based on criteria that are fixed by the site administrator. When setting up the load balancer, it needs to assign the "weights" to each node. In other words, a static weight is assigned to each server in Weighted Round Robin (WRR) policy. It is usually specified weights in proportion to actual capacities. For example, if server 1's capacity is 5 times more than server 2's, then it assigns a weight of 5 to server 1 and weight of 1 to server 2. In our system, servers' weight are assigned by using the capacity of the server, such as 1 for server 1, 3 for server 2 and 5 for server 3. The weighted round robin scheduling is better than the round robin scheduling when the processing capacity of servers are different.

4. Architecture of Load Balancer

In the architecture of load balancer, the load balancer is connected to the several target servers. The load balancer receive the requests from the client and redirect to the target servers based on load balancing algorithm that are configured by the administrator [6]. Figure 3. Show the architecture of load balancer. The load balancing system consists of Openflow switch connected to the POX controller and multiple server that are connected through the Openflow switch's ports. The static IP addresses are assigned to each server and the live servers IP are maintained in POX controller. The POX controller has a virtual IP address.

All of the clients' requests are redirect to the virtual IP address. When the request packets are dispatched to the virtual IP, information that are contained in the packet header, Openflow switch uses this information and contract this information with the stored information in flow entries of switch. If the flow table entries matches with the client's packet header information then based on the load balancing strategy, switch modifies the destination virtual IP address to the address of one of the servers and forward these packets to that particular server. If the flow table does not match with any header information, then the Openflow switch redirect these packets to the POX controller.

With the help of OpenFlow table, the controller inserts new flow entries to the switch's flow table. To implement load balancing application, python modules are written and that are executed by the POX controller.

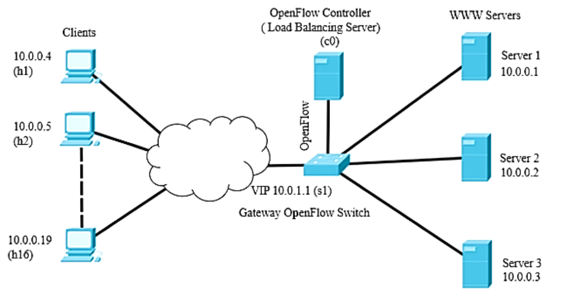


Figure 3. Load balancer architecture in SDN

5. Execution

The following software and tools were helpful in order to execute the functionality of the load balancer:

OracleVM – It is a powerful virtualization product for enterprise as well as home use. Not only is virtual box an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as open source software under the terms of the GNU General Public License (GPL) version 2.

Emulation Tool – Mininet is an emulation tool written in Python and C that allows to run a number of virtual hosts, controllers, switches, and links. It is widely used in open source network emulation environment. It uses container based virtualization to make a single system act as a complete network. It is a simple, robust and inexpensive network tool to develop and test Open Flow based applications. It is the Built-in component of Open vSwitch, and OpenFlow capable switch. Mininet can create a complex network topology for testing purposes, without configuring the physical networks. It can support custom topologies. It supports simple and extensible Python API for network creation and testing. Mininet combines the desirable features of simulators, test beds and emulators.

Controllers – In this proposed system, the POX controller is used for the central controller of entire network. The POX controller is a rewrite of the NOX controller and can be used on various platforms. Initially, POX was also published under the GPL, but has been available under the Apache Public License (APL) since November, 2013. POX is an open source development platform for Python-based software-defined networking (SDN) control applications, such as Open Flow SDN controllers. POX controller provides an efficient way to implement the Open Flow protocol which is the de facto communication protocol between the controllers and the switches. It can support the OpenFlow protocol version 1.0.

Testing Tool – In this experiment, OpenLoad testing tool is used to load balancing strategies. OpenLoad is an open source tool that can be tested based on parameter like Response Time and Number of Transactions that are executed per sec. The Weighted Round Robin and Round Robin load balancing strategies are compared in this paper.

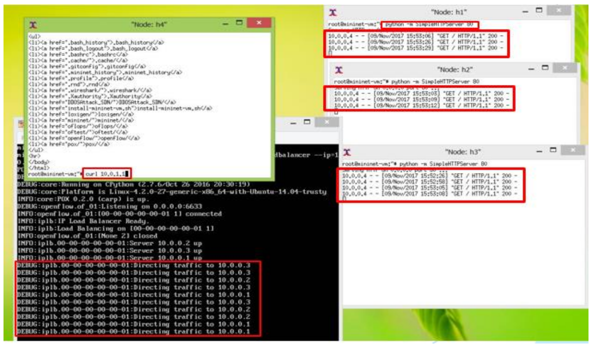
6. Experimental Setup and Results

Load balancing topology consist of 1 OpenFlow POX controller (c0), 1 OpenFlow switch (s1), 16 clients (h1,h2,….,h16) and Server 1, Server 2, Server 3 are used as simple 3 web servers. Load balancer consists of service IP (10.0.1.1) and clients will dispatch the requests to the service IP. The address that is specified to load balancer was service IP. It is responsibility of load balancer that the incoming request of the client is redirected to the server depending upon the strategies that are implemented in load balancer. Figure 4, 5 and 6 show the output after implementing three load balancing strategies in proposed software defined networking.

Moreover, the access frequencies of different clients are usually not the same in the real world. So we set up three different access frequencies, such as (1) 10 clients send a service request to the server continuously; (2) 13 clients send a request to the server continuously; and (3) 16 clients send a request to the server continuously. Figure 7, 8, and 9 illustrate the servers' response time for a period after the concurrent accesses under the three situations respectively. In the first case, 10 clients request the service simultaneously, the average server's response time of Round Robin and Weighted Round Robin are 0.233 s and 0.186 s respectively.

In the second case, 13 clients request the service simultaneously, the average server's response time of Round Robin and Weighted Round Robin are 0.317 s and 0.252 s respectively.

In the third case, the average server's response time of the two schemes, Round Robin and Weighted Round Robin are 0.404 s and 0.315 s respectively. In comparison of three case, the average server response times of Weighted Round Robin is smaller than the Round Robin. So Weighted Round Robin strategy is effective than Round Robin strategy.



```
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".gitconfig">.gitconfig</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".profile">.profile</a>
<li><a href=".rnd">.rnd</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".Xauthority">.Xauthority</a>
<li><a href="DDOSAttack_SDN/">DDOSAttack_SDN/</a>
<li><a href="install-mininet-vm.sh">install-mininet-vm.sh</a>
<li><a href="loxigen/">loxigen/</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
</ul>
<hr>
</body>
</html>
root@mininet-vm:~# curl 10.0.1.1
```

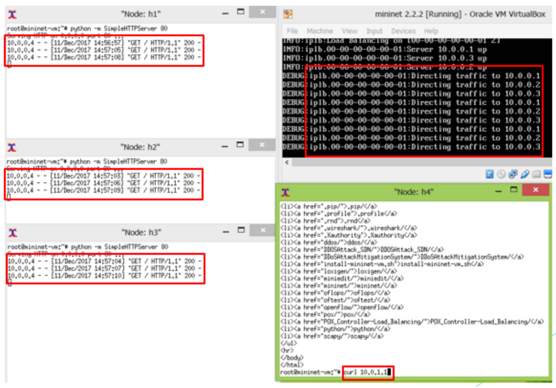
```
dbalancer --ip=1
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:30:19)
DEBUG:core:Platform is Linux-4.2.0-27-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:iplb:IP Load Balancer Ready.
INFO:iplb:Load Balancing on [00-00-00-00-00-01 1]
INFO:openflow.of_01:[None 2] closed
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.2 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.3 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.1 up
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
```

```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [09/Nov/2017 15:53:06] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:53:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:53:29] "GET / HTTP/1.1" 200 -
```

```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [09/Nov/2017 15:53:03] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:53:09] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:53:12] "GET / HTTP/1.1" 200 -
```

```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [09/Nov/2017 15:52:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:52:58] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:53:05] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [09/Nov/2017 15:53:08] "GET / HTTP/1.1" 200 -
```

Figure 4. Random load balancing



```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [11/Dec/2017 14:56:57] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [11/Dec/2017 14:57:05] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [11/Dec/2017 14:57:08] "GET / HTTP/1.1" 200 -
```

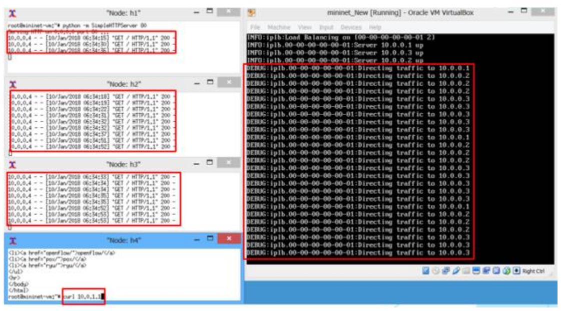
```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [11/Dec/2017 14:57:03] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [11/Dec/2017 14:57:06] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [11/Dec/2017 14:57:09] "GET / HTTP/1.1" 200 -
```

```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [11/Dec/2017 14:57:04] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [11/Dec/2017 14:57:07] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [11/Dec/2017 14:57:10] "GET / HTTP/1.1" 200 -
```

```
INFO:iplb:Load Balancing on [00-00-00-00-00-01 2]
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.1 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.3 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.2 up
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
```

```
<li><a href=".pip/">.pip/</a>
<li><a href=".profile">.profile</a>
<li><a href=".rnd">.rnd</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".Xauthority">.Xauthority</a>
<li><a href="ddos/">ddos/</a>
<li><a href="DDOSAttack_SDN/">DDOSAttack_SDN/</a>
<li><a href="DDoSAttackMitigationSystem/">DDoSAttackMitigationSystem/</a>
<li><a href="install-mininet-vm.sh">install-mininet-vm.sh</a>
<li><a href="loxigen/">loxigen/</a>
<li><a href="miniedit/">miniedit/</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
<li><a href="POX_Controller-Load_Balancing/">POX_Controller-Load_Balancing/</a>
<li><a href="python/">python/</a>
<li><a href="scapy/">scapy/</a>
</ul>
<hr>
</body>
</html>
root@mininet-vm:~# curl 10.0.1.1
```

Figure 5. Round robin load balancing



```
root@mininet-vm:~# python -m SimpleHTTPServer 80
10.0.0.4 - - [10/Jan/2018 06:34:15] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:30] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:36] "GET / HTTP/1.1" 200 -
```

```
0.0.0.4 - - [10/Jan/2018 06:34:18] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:19] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:22] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:31] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:32] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:32] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:37] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:51] "GET / HTTP/1.1" 200 -
0.0.0.4 - - [10/Jan/2018 06:34:52] "GET / HTTP/1.1" 200 -
```

```
10.0.0.4 - - [10/Jan/2018 06:34:33] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:34] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:34] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:35] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:35] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:52] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:53] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [10/Jan/2018 06:34:53] "GET / HTTP/1.1" 200 -
```

```
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
<li><a href="ryu/">ryu/</a>
</ul>
<hr>
</body>
</html>
root@mininet-vm:~# curl 10.0.1.1
```

```
INFO:iplb:Load Balancing on [00-00-00-00-00-01 2]
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.1 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.3 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.2 up
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.3
```

Figure 6. Weighted round robin load balancing

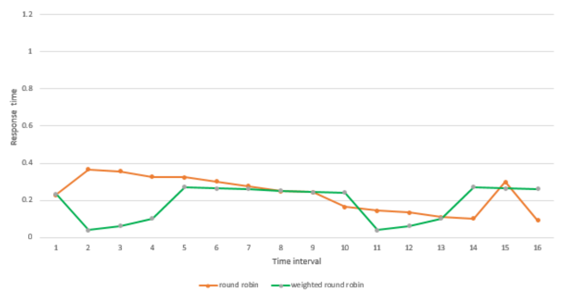


Figure 7. The server's response time of 10 clients simultaneously send requests

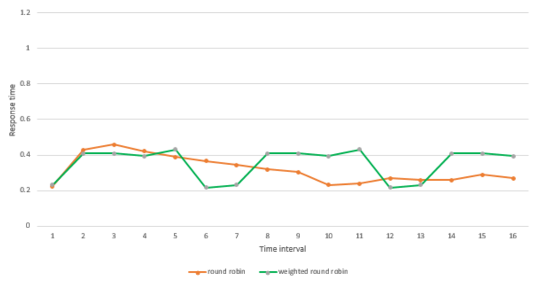


Figure 8. The server's response time of 13 clients simultaneously send requests

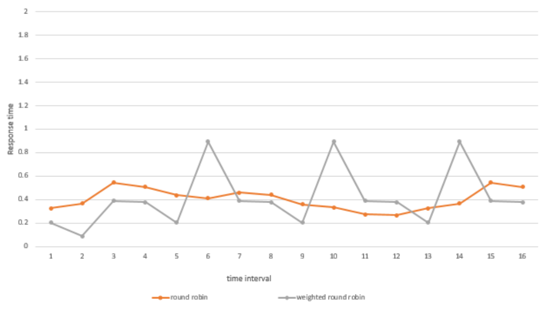


Figure 9. The server's response time of 16 clients simultaneously send requests

7. Conclusion and Future Work

Traditional networking has relied upon distributed control logic which limits its agility. The routers and switches need to keep itself updated by periodically refreshing its flow table and network map by communicating with surrounding devices. In order to solve the problems of lower efficiency and higher deployments cost of load balancing in the traditional networks, this paper proposes a Weighted Round Robin load balancing under the SDN architecture.

Short coming of our task is that only POX controller was used to test our code. Any other controllers don't be taken into account. The servers load balancing based on the servers' response times can be configured on the POX controller for a further extension.

References

[1] Cui Chen-xixo, Xu Ya-bin, Research on Load Balacne Method in SDN: International Journal of Grid and Distributed Computing, Vol 9. No. 1(2016), pp. 25-36.

[2] C. Rotsos, N. Sarrar, S. Uhlig, et al., OFLOPS: An open framework for OpenFlow switch evaluation, in: Passive and Active Measurement, Springer, Berlin, Heidelberg, 2012, pp. 85–95.

[3] D.B. Hoang, M. Pham, On software-defined networking and the design of SDN controllers, in: 2015 6th International Conference on the Network of the Future, (NOF), IEEE, 2015, pp. 1–3.

[4] Gaurav, Server and Network Load Balancing in SDN Content Delivery Data Center Network: Bachelor of Engineering in Computer Science, M.D.U, Rohtak India, June 2010.

[5] Hong Zhong, Yaming Fang, Jie Cui, LBBSRT: An efficient SDN load balancing scheme based on server response time: Future Generation Computer Systems 68(2017) pp. 183-190.

[6] Marti Boada Navarro, Dyanmic Load Balancing in Software Defined Networks: 10th semester, Network and Distributed System, June, 2014.