Virtual Organized Provisioning in Distributed Cloud Environment with Open stack & SDN

Ruchil Shah[1], Prof. Hardik Upadhyay[2]
[1] *PG Student, IT Network & Security, GTU-PG School, Gandhinagar, India*
[2]*Assistant Professor, GPERI, Mehsana, India*

ABSTRACT

Now a days Network Environment tackle huge amount of traffic & serve thousands of clients. With the use of single server it is difficult to handle this huge load separately .so some simple solution is use multiple servers with load balancing acting as a front-end. SDN offers a cost-effective & flexible Approach in Implementing a Load Balancer. When we use SDN Approach it reduces the cost, offer flexibility in configuration, reduce time to deploy, provide automation without requiring the knowledge of any vendor specific Software-Hardware.
So main benefit of SDN Load Balancer is that it does not required any hardware that help as a Load balancer. Cost is reduced. Here design topology is consist of Open Daylight controller & switch .In data plane packet entry are kept via flow table. Here main goal is to reduce the response time & lowest the latency with the use of custom Load balancing algorithm.

Keyword: - *Cloud Computing, Openstack, Neutron, SDN, OpenDaylight Controller;*

1. INTRODUCTION

The Cloud Computing (CC) market has been increasing very significantly. Gartner Says by 2020 "Cloud Shift" Will Affect More Than $1 Trillion in IT Spending [1]. The National Institute of Standards and Technology (NIST) describes cloud computing as follows: "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models." [2]. The NIST describes of cloud computing contains of five key characteristics which includes on-demand, Broad Network Access, Resource Pooling, Rapid Elasticity, and Measured Service. Cloud computing can be deployed as private, public, and hybrid clouds services.

1.1 Cloud computing offers the following service models:

Software as a Service (SaaS) is a type of application that is available as a service to clients; it delivers software as a service over the Internet, removing the need to install and run the application on local computers in order to simplify the maintenance and support. The advantages of SaaS are lower cost, client familiarity with WWW, and web availability and reliability.
Platform as a Service (PaaS) model enables the deployment of applications without the cost and complexity of buying and managing the underlying hardware and software layers. A customer can deploy an application directly on the cloud infrastructure (without managing and controlling that infra-structure) using the programming languages and tools sup-ported by a provider. A customer has the control over its applications and hosting environment's configurations.
Infrastructure as a service (IaaS) model delivers users physical resources or virtual machines in terms of CPU, storage, load balancers or operating system. However, Some IaaS service providers provide disk image library and file-based storage. End users are charged on pay per use basis. Today most of cloud computing companies is able to deliver IaaS for end-users.
This paper is organized as follows: Section 2 introduces the Openstack architecture, Openstack Neutron details Section 3 gives the details of Software Defined Network, ODL Controller Section 4 describes the Custom Load

balancing Algorithm Section 5 describes Evaluation of Topology with various result analysis Section 6 Conclude the paper.

2. OPENSTACK Dashboard

2.1 Openstack

OpenStack was created during the around January 2010. Rackspace wanted to rewrite the infrastructure code running its Cloud servers offering, and considered open sourcing the existing Cloud files code. At the same time, Anso Labs (contracting for NASA) had published beta code for Nova, a Python-based "cloud computing fabric controller". Both efforts converged and formed the base for OpenStack. [3]

OpenStack is the most popular among the open source CMSs. The main goal of OpenStack is to produce the omnipresent open source cloud computing platform that will see the needs of private and public clouds regardless of size, by being easy to implement and hugely scalable. OpenStack has got various services developed under different projects. These projects are integrated to provide complete functionality of OpenStack.

2.2 Openstack Architecture

The Openstack task: Deliver a popular open source cloud computing framework in order to support all types of private or public cloud computing with variety system sizing but simple in deployment and high scalability [4]. Giving to the task above, OpenStack is a cloud computing controller which mange IT resources such as Storage, compute, and network in a data center. All of the Processes on IT resources are transported through a control panel which helps administrators control and interact with IT resources through a web based. OpenStack is opensource so it is free and its components are written in Python language. The logical architecture of OpenStack is shown in Fig. 1.It Describes five main components of OpenStack consist of Identity (Keystone), Compute (Nova), Image Service (Glance), Dashboard (Horizon), and Object Storage (Swift). OpenStack development is still in progress. OpenStack has full of cloud computing attributes that provide IaaS. Compute provide and manage instances for compute function (Virtual machines). Image Service keep image of instance, is used by Nova when deploying an instance. Object Storage provide storage function. Dashboard provides web based for administration of OpenStack. Identity provide authentication and authorization for all of the services in OpenStack [6].

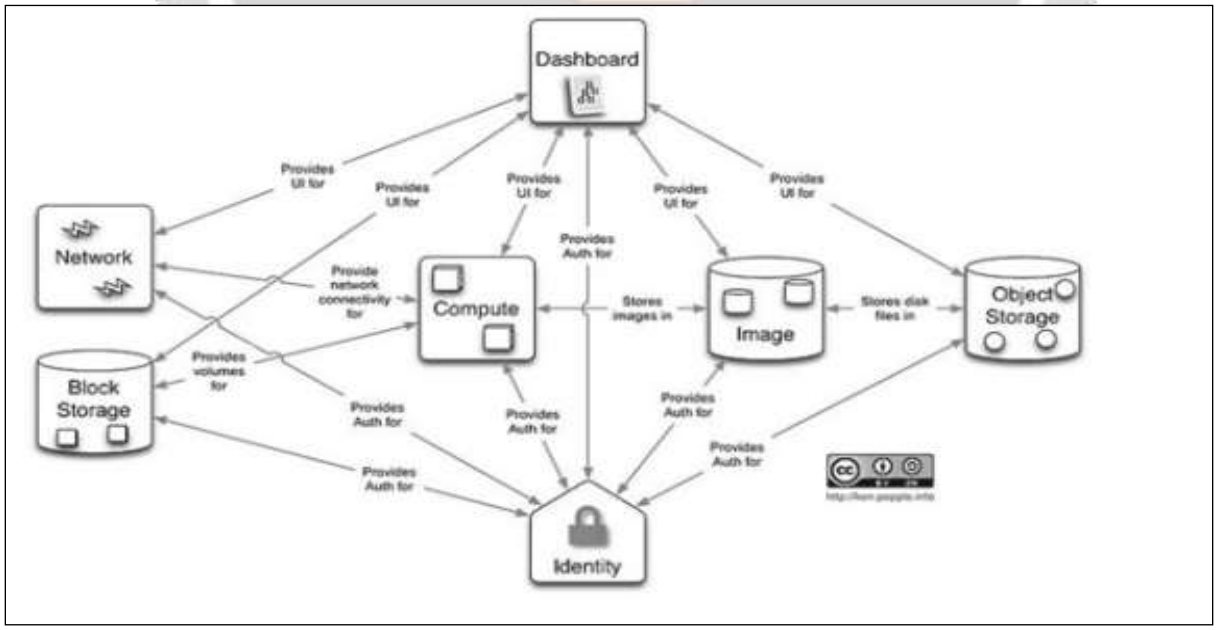

Fig-1 : Openstack Architecture

2.3 Openstack Neutron

Neutron is an OpenStack venture to give "organizing as an administration" between interfaces gadgets (e.g., vNICs) oversaw by other OpenStack administrations. It empower development modules (open and shut source) that present propelled arrange capacities like utilize L2-in-L3 burrowing to maintain a strategic distance from VLAN limits, give end-to-end QoS ensures, utilized observing conventions like Net Flow. Give anybody a chance to manufacture propelled organize administrations (open and shut source) that connect to OpenStack occupant systems like LBaaS, VPNaaS, FWaaS, IDaaS, SaaS, server farm interconnect-aaS. [4]

The Modular Layer 2 (ml2) module is a system permitting OpenStack Networking to all the while use the assortment of layer 2 organizing advancements found in complex true server farms. It right now works with the current OpenVSwitch, Linux Bridge, and hyperv L2 specialists, and is proposed to supplant and belittle the solid modules connected with those L2 operators. The ml2 structure is additionally expected to extraordinarily improve including support for new L2 organizing innovations, requiring a great deal less beginning and continuous exertion than would be required to include another solid center module.

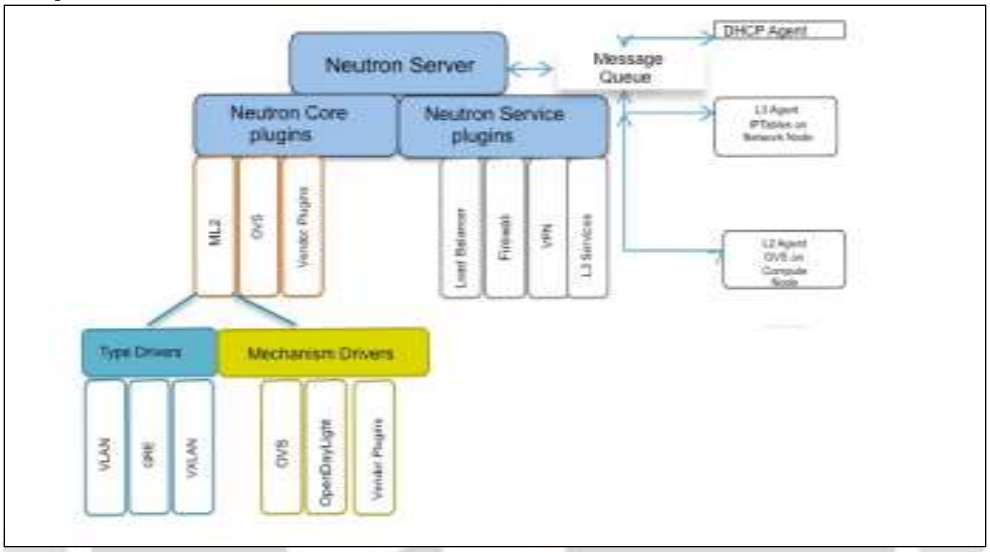

Fig-2: Logical Architecture of Openstack Neutron

OpenStack arrangement is in a private network. Instances are given IP address from DHCP agent of Neutron and are accessed within OpenStack using this private IP. When instances need to connect through public network, they use novel IP called floating IP. Each instance will have 2 IPs, private and floating. Neutron assigns the IP. It neither uses any DHCP service nor being assigned statically. It is Neutron L3 agent's responsibility to track the packets having floating IP to correct destination.

3. SOFTWARE DEFINED NETWORK

3.1 Software Defined Network Architecture

Software Defined Networks (SDN) is an amazing technology, and it can help save resources as well as improve resource utilization & power in networks according to Q. Y. Zuo et al. [8]. We know that in the current network infrastructure it depends on a vertical architecture, which are very complex itself. So a lack of flexibility has inappropriate effects. The SDN is a new Concept that breaks this vertical architecture of a network control plane and its data plane. SDN uses an application program that runs on top of the controller to monitor and manage the network in a centralized manner. The SDN controller make choices whether the packet is forwarded or dropped. It can also add a new flow entry in order to forward similar packets in the future. A typical load balancing technique includes using a dedicated load balancer to forward the client requests to different servers.

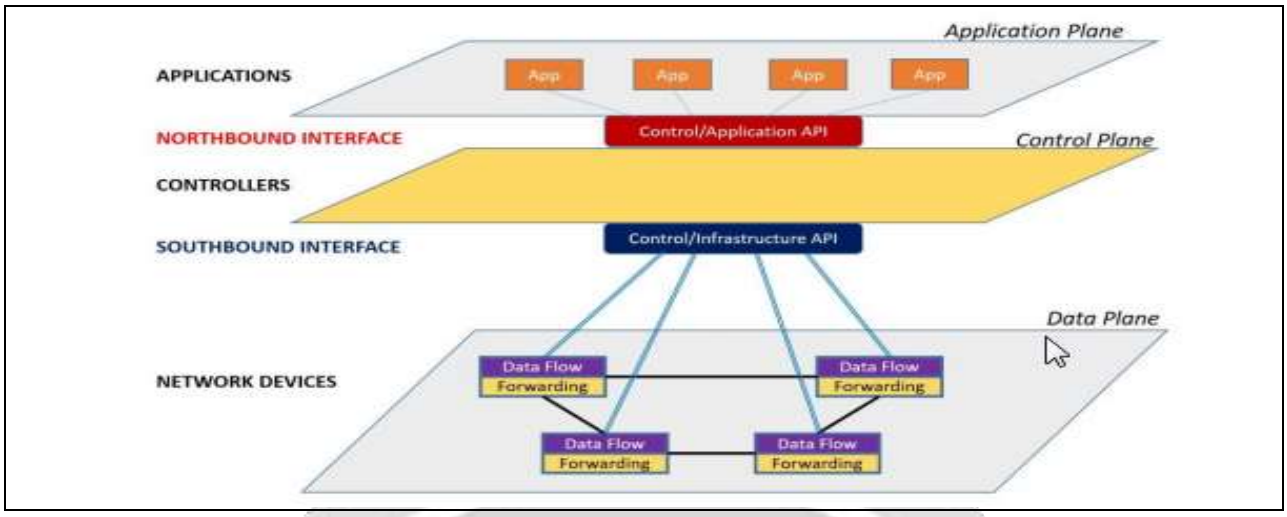

Fig-3 : Software Dedined Network Architecture

This technique requires dedicated hardware support, which is expensive and lacks flexibility due to the single point of failure in these load balancers. Any failure in the load balancers results in the inadequate distribution of traffic. To overcome these failures, SDN is used to describe the implementation of load balancing through parallel paths between the server and the client. The communication between the controller and load balancers is through Open Flow.

3.2 Open Daylight Controller Architecture

The first section of Software Defined Networks is the SDN controller called the regularity framework. The controller describes the SDN's worldview. In this research, an Open Day Light (ODL) controller is used. [21]

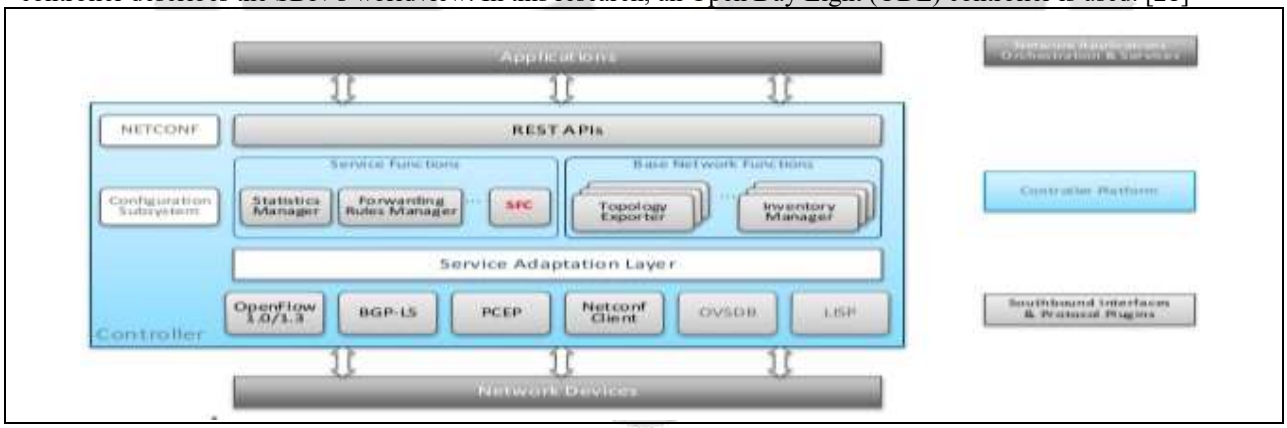

Fig4.Open Day Light Controller Architecture

It is the dependable section for intent connection with every programmable component of the system, giving a bound together Perspective of the system. The ODL controller is a northbound Application Peripheral Interface (API) used by various applications. These applications use the controller to collect data about the network, run algorithms to lead analytics, and use the ODL controller to create new rules throughout the network as researched by J. Turner et al. [5]. In Figure 4, the architecture of the OpenDaylight controller is given. This architecture is divided into two APIs namely Southbound and Northbound. The northbound deals with the application side, which helps the ODL to function in the preferred way including load balancer, firewall or any other application. There are numerous controllers such as the OpenDaylight controller, Floodlight, Griffon etc.

To construct the SDN load balancer, a controller mechanism running on a regularity framework, for instance ODL is required. The forwarder's components that are important have a separate control arrangement with the programming interface in OpenFlow. A complete physical foundation that connects all the components over a protected channel for SDN is needed, which is complex & costly. To manage this matter, this research utilizes a particular test system. Mininet is a device to mirror the Software Defined Networks that permit the fast prototyping of a large virtual foundation system with the utilization of a standout PC. It allows the use of virtual models of versatile systems in light of programming. For example, OpenFlow, which utilizes a primitive virtualization operating system with these primitives, associates and modifies models for Software Defined Networks in an abrupt manner.

3.3 Mininet: A virtual Network

Mininet is more compatible than emulators as it runs the prototype code of the application, unlike in emulators, which run on trial code. This supports in troubleshooting and running before implementing in production. Mininet switches are encoded using the OpenFlow protocol; this helps in testing and varying the code to requirement. It creates a realistic virtual network that runs on a real kernel, switch, and application code that runs on any Virtual Machine (VM), cloud, or native with a single command. Mininet is also an effective way to develop, share, and Experiment with OpenFlow and Software-Defined Networking systems as referred from the OpenFlow website. [20]

4. Custom Load balancing Algorithm

The algorithm for the SDN load balancer is easy as every packet has a certain hash allocated to it. The main rule directing in the algorithm is knowing the hash value and source IP in the table. When a packet comes to the load balancer, it identifies the hash value, and if it is already a registered hash, the load balancer matches it to the table and tells where the packet needs to be sent. When the load balancer comes across a new packet, the controller regulates whether the received packet is on the entrance interface or outlet interface. The flow table directing the packet flow; entries must be made whenever a new packet comes to the load balancer. The decision of whether the

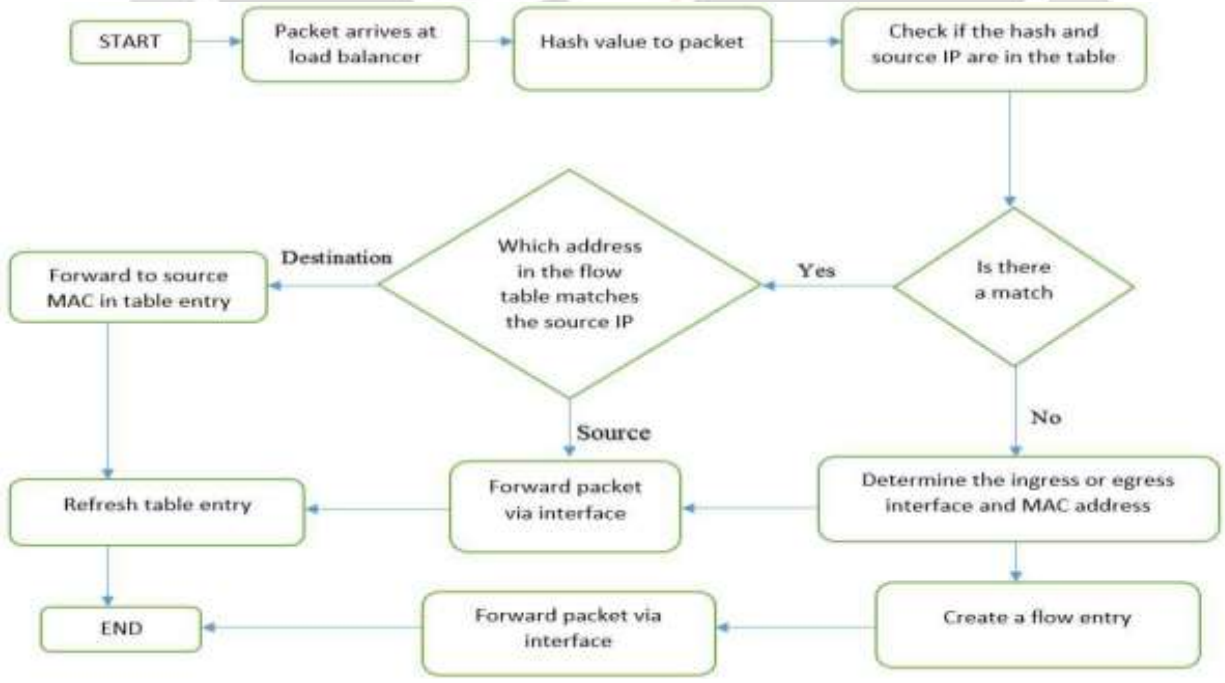

Fig-5.Custom Load Balancing Algorithm

Packet is coming from the entrance or outlet interfaces is made, and a fresh entry is created whenever a new packet arrives. In Algorithm, the flow chart explaining how the entries and decisions are made in the load balancer.

5. Evaluation

In this section, we describe the evaluation of Custom load balancer in an OpenStack environment. We evaluate the performance of our design by (i) comparing it with HAProxy, a popular open-source load balancer software, and (ii) measuring the outlays compulsory on the network controller.

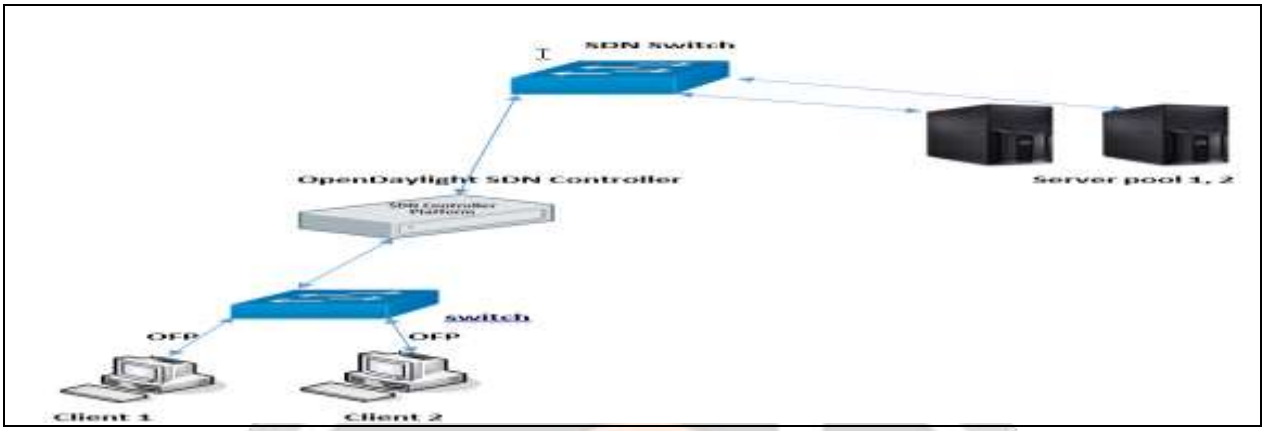

Fig-5 : Topology one

Load balancers Evaluation and the performance are based on different parameters such as latency and response time. The main Objective of this imitation is to prove that the SDN with an OpenFlow protocol on load balancers acts more efficiently. Three designs are studied to test the efficiency of the load balancers. The design topologies are than compared with design topologies with Round Robin Algorithm. The assessment of the topologies are based on response time and latency. The best strategy should be able to operate efficiently even with the increase in complications. Designs are created virtually on an Openstack Multi node server with all the components installed on the machine with the help of virtualization. This research mainly requires SDN supported switches to act as a forwarding plane and an SDN controller functions as a control plane. The first design topology is simple design in which there are equal number of clients and servers, here there are clients and two servers in the server pool. Figure 4 describes the design topology one in which two servers acting as server pool connected to SDN switch and the clients connected to a normal switch. These two clients should communicate with the servers in the server pool.

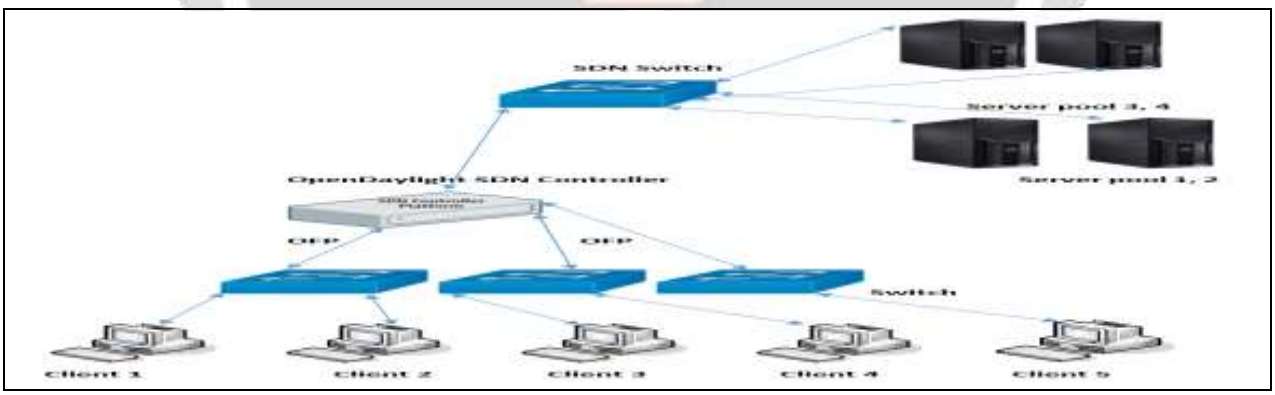

Fig-6 : Topology Two

The second outline topology is a more unpredictable plan where there are five clients, yet there is no same quantities of servers as in topology one. Figure 5 depicts outline two, in which there is an alternate quantities of servers and clients. Continuously, this is the situation where the idea of load adjusting jumps out at balance the expanded number of solicitations. In this outline, there are five clients however just four servers in the server pool to adjust those five clients. SDN switch is interfaces the controller and the servers.

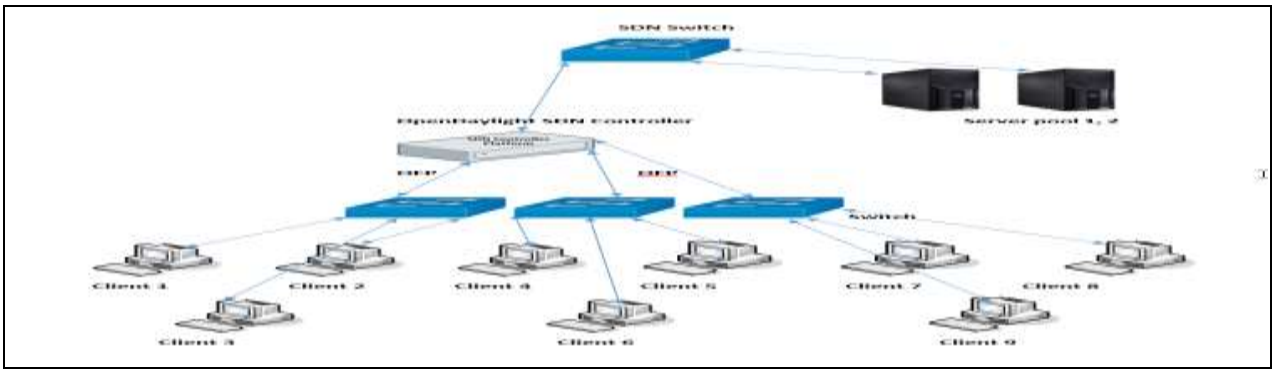

Fig-7 : Topology Three

The third plan is a truly complex outline where there are just two servers in the server pool however there are nine clients. Figure 6 depicts the entangled outline where there just two servers in the server pool which serves nine clients. This plan is especially same to the past two, however there is an expansion in the quantity of clients which is distinctive. Continuously Scenario, there will be not very many servers yet various clients. The conventional load balancers slack in these sorts of plan topologies where the reaction time and inactivity will increment for the customary load balancers.

6. RESULT ANALYSIS.

Comparing the latency and response time with SDN custom load balancing and using Traditional load balancing.

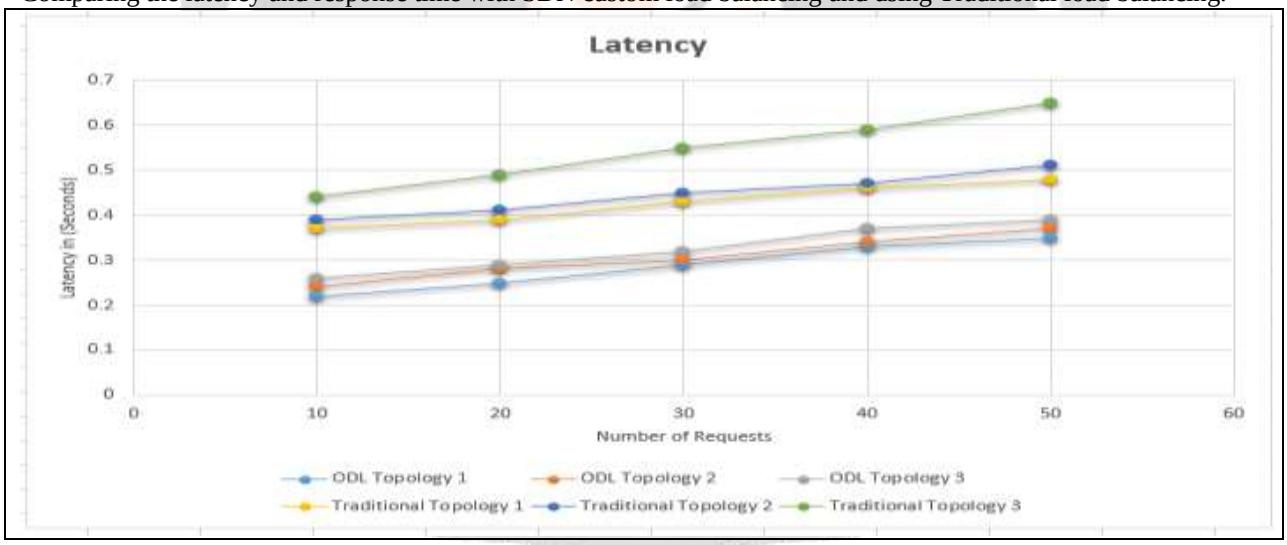

Fig-8 : Latency in ODL topology vs Traditional Topology Comparison

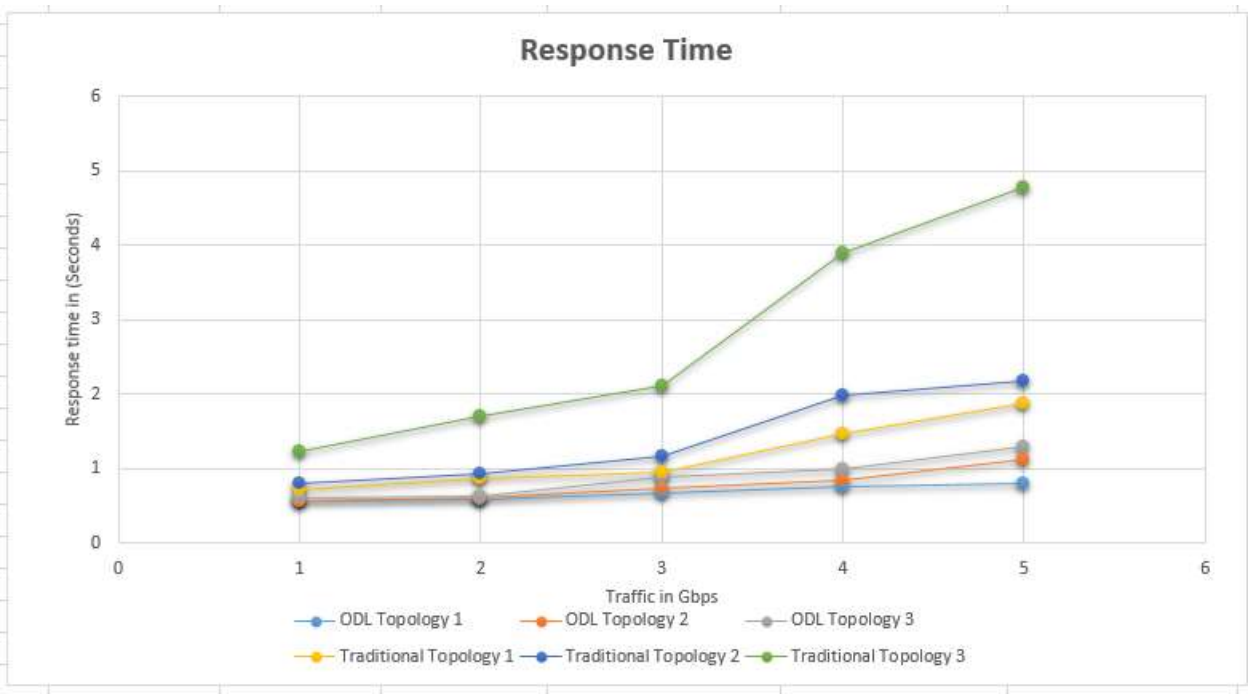

Fig-9 : Response time in ODL topology vs Traditional Topology Comparison

It can be seen that there is a decrease in latency and response time for the load balancers using custom load balancing. This reduction is mainly because the dropping off buffer size of the load balancer. The buffer size is reduced when OpenFlow is used. This move from Traditional load balancers to SDN custom load balancers using OpenFlow is effective as the latency time and response time for the system is decreased. This proves that the use of OpenFlow performs load balancing more efficiently & effectively.

6. Conclusion & Future Work

Cloud computing and open source cloud OpenStack today become a trend in technology and is one of the first choice of companies in the world. Custom Load balancing has been implemented as a set of modules inside an OpenDaylight based OpenFlow controller that provides network service for an OpenStack managed cloud through an LBaaS driver extension. Our evaluation of load balancing capabilities on a hardware test bed demonstrates that it is faster than a standard software load balancer widely used in the industry. The goal of this project, which is to design of an efficient load balancer using a SDN, is accomplished. The controller used is blocking free and handles multiple functions concurrently. It should also be able to act as a firewall. The future controller should be efficient and should be able to reconfigure the network within nanoseconds in case of a failure.

REFERENCES

[1] 'Gartner Report'http://www.gartner.com/newsroom/id/3384720
[2] "NISTPublication"http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-146.pdf
[3] "Openstack Details" https://www.openstack.org
[4] Performance Analysis of Network Virtualization in Cloud Computing Infrastructures on OpenStack by VONhab Van, Le Minh Chi, Nguyen Quoc Long, Dac-NhoungLe.doi 10.1007/978981-10-0419-3_12, Springer 2016
[5] Shaoka Zhao et al. 'Deployment and Performance Evaluation of Virtual Network based on OpenStack'. In: (2013).
[6] Rahul Bhatnagar, Suyash Raizada and Pramod Saxena. 'ISSUE IN CLOUD-COMPUTING'.

[7] 9. F. Callegati, W. Cerroni, C. Contoli and G. Santandrea (2014), Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case, 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), pp. 132 – 137.

[8] Q. Y. Zuo, M. Chen, G. S. Zhao, C.Y. Xing, G. M. Zhang, and P. C. Jiang, "OpenFlow based SDN technologies," Ruanjian Xuebao/Journal of Software, vol. 5, pp. 1078-1097, 2013.

[9] "New Features for Amazon EC2: Elastic Load Balancing, Auto Scaling, and Amazon CloudWatch," http://aws.amazon.com/blogs/aws/new-aws-load-balancing-automaticscaling-and-cloud-monitoring-services

[10] R. Gandhi et al., "DUET: Cloud Scale Load Balancing with Hardware and Software," in ACM SIGCOMM 2014.

[11] "Nmon," http://nmon.sourceforge.net/.

[12] "HA Proxy Load Balancer," http://haproxy.lwt.eu.

[13] "NetScalar VPX Virtual Appliance," http://www.citrix.com.

[14] "F5 Load Balancers," https://f5.com/glossary/load-balancer

[15] "A10 Networks AX Series," http://www.a10networks.com

[16] "F5 BIG-IP," http://www.f5.com

[17] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," Paper presented at ACM SIGCOMM '2007, Conf., August 2014, Kyoto, Japan.

[18] The OpenFlow Switch Specification. http://OpenFlow Switch.org

[19] "Auto Scaling in the Amazon Cloud," http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html

[20] "Mininet" http://mininet.org/sample-workflow/

[21] "OpenDayLight Controller" https://www.opendaylight.org/