www.rspsciencehub.com Volume 02 Issue 09 September 2020

INTERNATIONAL RESEARCH JOURNAL ON ADVANCED SCIENCE HUB

e-ISSN : 2582 - 4376
Open Access

RSP SCIENCE HUB
(The Hub of Research Ideas)
Available online at www.rspsciencehub.com

Load balancing using openday light SDN controller: Case study

Prof.Vijaya Eligar [1]*, Dr.Nalini Iyer* [2]*, Nihal N.D* [3]*,Nikhil S.Hugar* [4]*, P.YashwantKumar* [5]*,M.N.Manjunath* [6]
[1]*Assistant Professor,School of Electronics and Communication, KLE Technological University, Hubballi , India*
[2]*Professor and HOS,School of Electronics and Communication, KLE Technological University, Hubballi , India*
[3]*Student,School of Electronics and Communication, KLE Technological University, Hubballi , India*
[4]*Student,School of Electronics and Communication, KLE Technological University, Hubballi , India*
[5]*Student,School of Electronics and Communication, KLE Technological University, Hubballi , India*
[6]*Student,School of Electronics and Communication, KLE Technological University, Hubballi , India*
vijayaeligar@kletech.ac.in[1]

Abstract

Traditional networking architectures have many limitations that need to be overcome to meet modern IT requirements. To overcome these limitations; Software Defined Networking (SDN) is taking place as the new networking approach. As traditional networking uses static switches, resource utilization is poor. There are packet loss and delay during switch breakdown. This paper proposes an implementation of a load balancing algorithm for an SDN based network to overcome the stated issues. To test the algorithm, a network is emulated using the Mininet and OpenDaylight platform (ODL) is used as an SDN controller. Python coding language is used to create fat-tree network topology and to write a load balancing algorithm. Finally, iPerf and Wireshark is used to test network performance. The network was tested before and after running the load balancing algorithm. The testing focused on some of the Quality of Service (QoS) parameters such as bandwidth and transfer rate in the fat-tree network. The algorithm increased bandwidth with at least more than 50\%, and improved network utilization

Keywords: SDN, ODL, Mininet, iPerf, Wireshark, QoS

1. Introduction

The modern-day IT industry adopts a traditional network architecture. Traditional network architecture uses separate switches and routers. In traditional networking the network and the data plane are one whole part and are not separate. Whereas in the Software Defined Networking (SDN) the whole network is divided into a data plane and control plane. Traditional network is a hardware based and Software Defined Networking (SDN) is a software-based networking which uses Application Programming Interfaces (API's) to directly connect to the applications, boosting network performance, security and making the network flexible. As the traditional network has many limitations there is a need for a notable change in the traditional networking. To overcome these limitations the Software Defined Networking (SDN) is stepping up as the new networking approach. In the complete network the role of the data plane is to transfer the data packets and the role of control plane which having its own intelligence acts as a manager, instructing the data planes throughout the network. In general, SDN provides the overall central view of the complete network.

There are two main protocols used by SDN to communicate with its Network Elements (NE) i.e. switches/routers. They are OpenFlow and open virtual switch database. OpenFlow protocol is an open standard protocol which allows the researchers to run their experiments, without the need of vendors to disclose the practical details of their network devices. OpenFlow is not to be misunderstood with SDN. Where SDN is the network architecture which divides the network plane and the data plane, the main job is to convey the messages from the control layer to the infrastructure layer.

Load Balancing provides an efficient way of distribution of traffic i.e. network and application traffic between different servers which are collectively known as clusters of servers. The main aim of load balancing is to prevent overloading and possible breaking down of a single server. Load Balancers are designed to address the three main problem domains, those are availability, performance and economy. Classic Load Balancer, Network Load Balancer and Application Load Balancer are the major load balancers. As the name suggests the application load balancer and the network load balancer are designed to increase the efficiency of the applications i.e. API's and the network.

In order to increase the bandwidth, throughput optimization and include redundancy, network load balancing is performed. The network traffic is spread throughout several servers and this is achieved by network load balancing. This load balancing provides the ability to balance network sessions between various applications to provide equal amount of bandwidth between different LAN users. Link load balancer is usually involved in load balancing. Link balancer is a appliance which provides in-bound and out-bound to and from multiple internet links. They are placed between the firewall and the gateway router. There are different load balancing algorithms such as Round Robin, Least connections, weighted least connections, and much more. The purpose of this research is to introduce the dynamic load balancing algorithm of Nayan Seth in SDN-based network to test and examine the possibilities of accomplish better results.

2. Literature Review

2.1 Limitations of traditional networks

Today's network must scale to accommodate increased workloads with greater agility, while also keeping costs at a minimum. Traditional approach has significant limitations such as:

- Complexity: As the data communication is increasing rapidly day by day and with this there brings a need to add new devices. Whenever a new network element is added in the network, rewiring of the network is required. This also requires management of the network. Managing these big network frameworks becomes difficult. This makes scaling of network very difficult. [1-4]
- Static Environment: Since implementation of network wide policies in a conventional network is time-consuming and complicated, due to this the network will not be touched. This dormant feature of the network environment makes it very challenging for companies to gain from the rising convenience, such as introducing new apps or web services, stifling modernization and shackling business development.
- Vendor lock-in: Companies are often locked with only one provider. Due to lack of protocols for configuring equipment among different network device producers. This limits the possibility of tailoring the network to satisfy the individual business needs.[1]

The traditional is shown in fig.1.

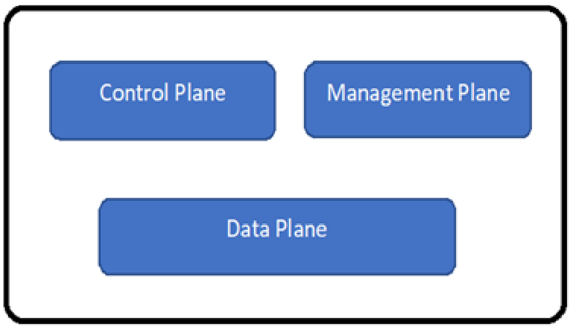


Fig.1. Traditional network

2.2 Software-Defined Networking

SDN is a developing network technology which overcomes the limitations of a traditional network. It is a significant change in the traditional network by separating the control and the data plane as

shown in fig.2. SDN is a type of network architecture that allows the network to controlled centrally [2]. SDN has created a very bright future with the help of this technology to surmount the need for reliable, secure, flexible and well managed next-generation networks. SDN allows network behaviour to be configured centrally through open API software applications. It provides various facilities like multi-vendor operability.
An abstraction of SDN-technology infrastructure and software form the basic technologies and equipment that provide network access physical communication. Instead of management interfaces closely linked to hardware as in conventional network, apps communicate with the network through API's. It enables users to create network aware apps, intelligently monitor network conditions, and automatically adjust the network configuration as needed.[3]

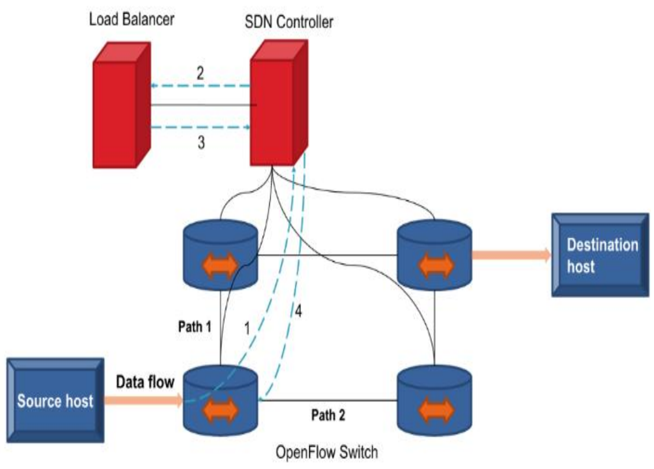
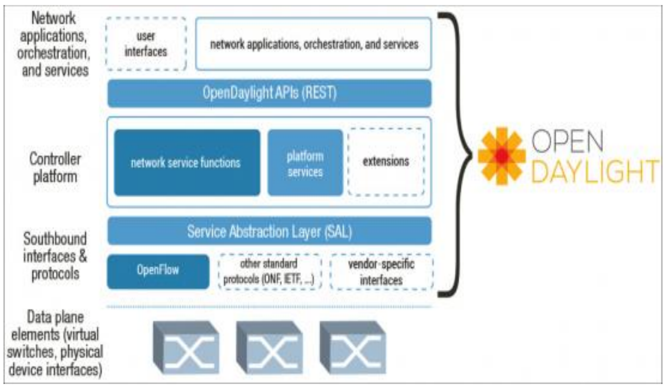


Fig. 2. SDN

2.3 OpenFlow

It is considered as one of the first SDN standard. It originally defined the SDN communication protocol that permits the SDN controller, physically and virtually(hypervisor-based), to interact directly with the transmission level of network devices like switches and routers, so that it gets adapted to the continuous and ongoing business trends and essentials. Stanford University originally imagined and implemented this as a part of network research. Its main purpose was to allow experimental protocols on campus networks to be developed that were to be used for research and development. Prior to this universities have had to build their own research sites from scratch before that. [5-8]

Any system that wants to communicate with an SDN controller must be knowing the OpenFlow protocol to work in its domain. The SDN controller push converts into a switch/router flow table via this interface, enabling network administrators to track partition traffic, control flows for optimum performance and check for various different configurations and applications [4]. Open Networking Foundation, a non-profit organization was formed by a group of service providers to promote and standardize the use of OpenFlow in production networks. OpenDaylight controller is shown in fig.3.


Network applications, orchestration, and services
user interfaces
network applications, orchestration, and services
OpenDaylight APIs (REST)
Controller platform
network service functions
platform services
extensions
Service Abstraction Layer (SAL)
Southbound interfaces & protocols
OpenFlow
other standard protocols (ONF, IETF, ...)
vendor-specific interfaces
Data plane elements (virtual switches, physical device interfaces)
OPEN DAYLIGHT
Fig. 1. OpenDaylight

2.4 Network Load Balancing

This technique is very important in building high speed networks, ensuring efficiency of the network. The main necessity of the load balancing is to avoid congestion in the network and maintain efficient flow of data in the network is by distributing the traffic from the overloaded paths to the less loaded paths. Doing so, increases the network utiliza*tion and throughput.

2.5 Fat-Tree topology

Invented by Charles E. Leiserson in 1985. As the named suggests this network is in a form of tree, processors connected to the bottom layer. It is known by the fact that the number of links to its siblings is equal to the number of links up to its parent on the top level for each turn. It is three layer architecture. This topology includes the numerous paths between the hosts so that the path with the high bandwidth can be allocated. This multipath feature of the tree can be helps in traffic distribution among different network components.[5]. Fig.4 shows a fat-tree topology.

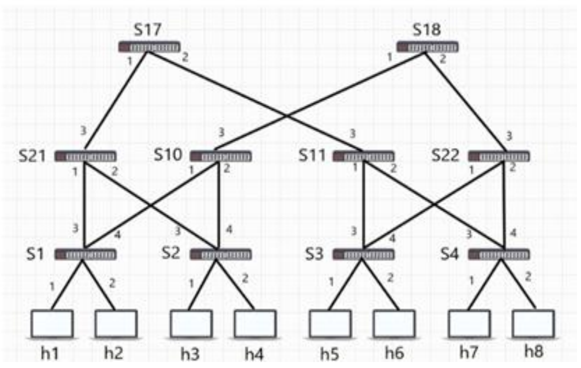


Fig. 2. Fat-tree topology

2.6 Network Load Balancing

The task of the Nayan Seth’s algorithm [9] is to distribute traffic of upcoming and incoming network flows in order to achieve the best possible resource utilization of each of the links present in a network. In order to achieve such aim, it is necessary to keep track of the current state of the network [7]. The flow chart of the implementation is shown in fig.5

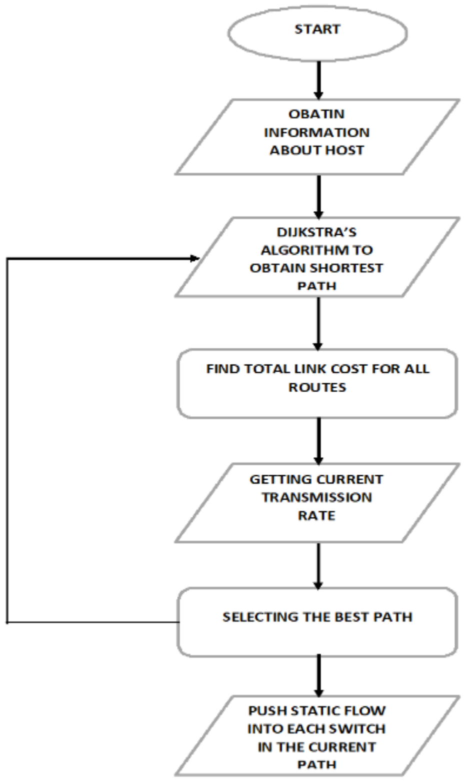

START
OBATIN INFORMATION ABOUT HOST
DIJKSTRA'S ALGORITHM TO OBTAIN SHORTEST PATH
FIND TOTAL LINK COST FOR ALL ROUTES
GETTING CURRENT TRANSMISSION RATE
SELECTING THE BEST PATH
PUSH STATIC FLOW INTO EACH SWITCH IN THE CURRENT PATH

Fig. 5.Implementation flow chart

The first step of the algorithm is to collect operational information of the topology and its devices. Such as IPs, MAC addresses, Ports, Connections, etc. Next step is to find route information based on Dijkstra's algorithm, the goal here is to narrow the search into a small segment of the FatTree topology and to find the shortest paths from source host to destination host. And then find total link cost for all these paths between the source and destination hosts. Once the transmission costs of the links are calculated, the flows are created depending on the minimum transmission cost of the links at the given time. Based on the cost, the best path is selected and static flows are pushed into eachs witch in the current best path. with that, every switch within the selected path will have the necessary flow entries to carry out the communication between the two end points. Hence, this program is dynamic in nature as it continue to update this information every minute. In this research a test-bed has been implemented under Linux, using Mininet software to emulate the network, the open-source[6] OpenDaylight platform (ODL) as SDN controller, and Python programming language to define the fat-tree topology and to write the load balancing algorithm program, and iPerf to test network performance. The fig.6 illustrate the design steps.

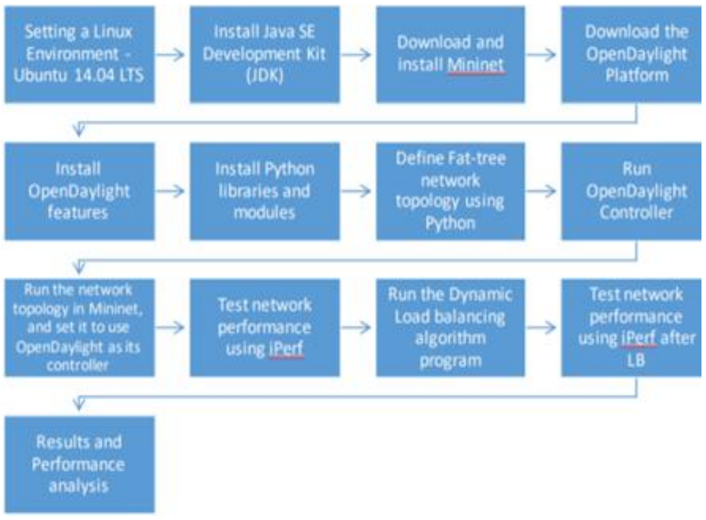

Setting a Linux Environment - Ubuntu 14.04 LTS
Install Java SE Development Kit (JDK)
Download and install Mininet
Download the OpenDaylight Platform
Install OpenDaylight features
Install Python libraries and modules
Define Fat-tree network topology using Python
Run OpenDaylight Controller
Run the network topology in Mininet, and set it to use OpenDaylight as its controller
Test network performance using iPerf
Run the Dynamic Load balancing algorithm program
Test network performance using iPerf after LB
Results and Performance analysis

Fig. 6. Design steps

Mininet:

Mininet is a network emulator that allows prototyping large networks on a single machine [6]. It runs a collection of end-hosts, switches, routers, and links on a singleLinux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. Mininet main advantages:

•Mininet is an open source project.

•Custom topologies can be created.

•Mininet runs real programs.

•Packet forwarding can be customized.

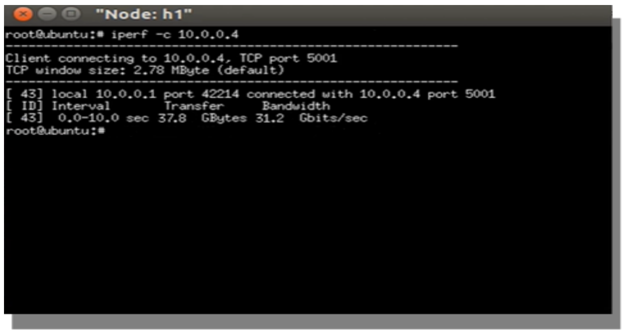
Compared to simulators, Mininet runs real, unmodified code including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code) and easily connects to real networks.

OpendayLight:

The OpenDaylight Project (ODL) [6]is a highly available, modular, extensible, scalable and multi-protocol controller infrastructure built for SDN deployments on modern heterogeneous multi-vendor networks. ODL provides a model-driven service abstraction platform that allows users to write apps that easily work across a wide variety of hardware and south-bound protocols. Furthermore, it contains internal plugins that add services and functionalities to the network. For example, it has dynamic plugins that allow to gather statistics as well as to obtain the topology of the network. [9,10]

iPerf:

iPerf is a commonly used network testing tool for measuring Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) bandwidth performance and the quality of a network link. By tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6), the user is able to perform a number of tests that provide an insight on the network's bandwidth availability, delay, jitter and data loss.iPerf is an open source software as shown in fig.7.It runs on various platforms including Linux, UNIX and Windows.



```
root@ubuntu:# iperf -c 10.0.0.4
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 2.78 MByte (default)
------------------------------------------------------------
[ 43] local 10.0.0.1 port 42214 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 43]  0.0-10.0 sec 37.8  GBytes 31.2  Gbits/sec
root@ubuntu:#
```

Fig. 3. iPerf h1 h4

Python:

In this research, Python has been used in mininet to define the Fat-tree topology, also it has been used to write the load balancing algorithm program. Python is an interpreted, object-oriented language suitable for many purposes. It has a clear, intuitive syntax, powerful high-level data structures, and a flexible dynamic type system. Python can be used interactively, in stand-alone scripts, for large programs, or as an extension language for existing applications. The language runs on Linux, Macintosh, and Windows machines. Python is easily extensible through modules written in C or C++, and can also be embedded in applications as a library. There are also a number of system specific extensions. A large library of standard modules written in Python also exists. Compared to C, Python programs are much shorter, and consequently much faster to write. In comparison with Perl, Python code is easier to read, write and maintain. Relative to TCL, Python is better suited for larger or more complicated programs.

Results

After selecting the best path and pushing the flow into that, we analyze the results using wireshark and iperf. We see that there is increase in both bandwidth and Transfer rate. The jitter is decreased. Hence, the expected results are obtained by following the procedure. Fig.8 shows the (ping) jitter from h1 to h4. And the transfer and bandwidth after load balancing is shown in fig.9

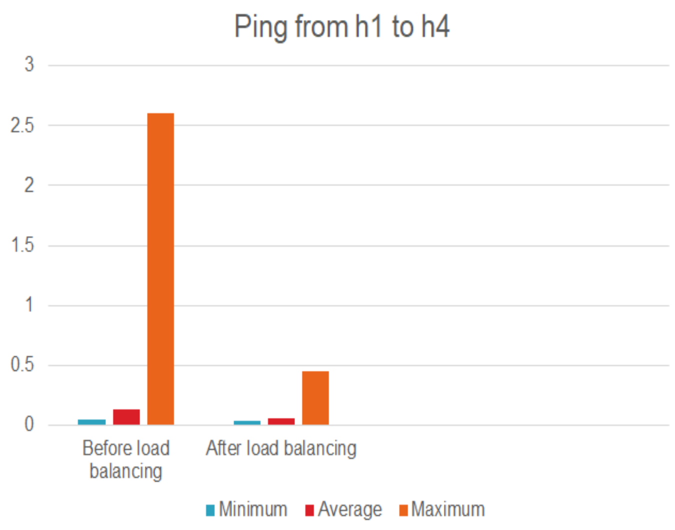


Fig. 4. Jitter result

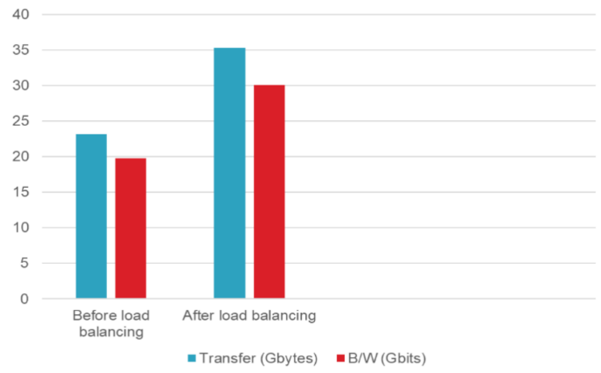


Fig. 5. Transfer and B/W result

Conclusions

This research describes the implementation of Nayan Seth's dynamic load balancing algorithm to efficiently distribute flows for fat-tree networks through multiple alternative paths between a single pair of hosts. The network was tested before and after running the load balancing algorithm. The testing focused on some of QoS parameters such as throughput, delay, and packet loss between two servers in the fat-tree network. The results showed that the network performance has increased after running the load balancing algorithm program, the algorithm was able to increase throughput, and improve network utilization. However, in large networks it increased packet loss and jitter.

In future work, next suggestions are planned: The first suggestion is to investigate the performances of the dynamic load balancing program on a different popular SDN controller, such as Research Floodlight, Beacon, NOX/POX, etc. and compare the results. The second suggestion is to investigate the performances of different topologies of different sizes, other than the fat-tree topology. To test if there are any other limitations with the algorithm. And finally, is to extend the algorithm to traditional networks, or hybrid networks with both OpenFlow and regular switches.

References

[1]. https://carriersolutionsgroup.com/how-is-sdn-different-than-a-traditional-network/

[2]. Badotra, Sumit. (2017). A Review Paper on Software Defined Networking. International Journal of Advanced Computer Research.

[3]. https://www.ciena.com/insights/what-is/What-Is-SDN.html

[4]. https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/

[5]. https://clusterdesign.org/fat-trees/

[6]. Lindinkosi Zulu, Patrice Umenne, Kingsley Ogudo. A "Emulating Software Defined Network Using Mininet and OpenDaylightController Hosted on Amazon Web Services Cloud Platform to Demonstrate a Realistic Programmable Network", 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC),Electronic ISBN: 978-1-5386-6477-3

[7]. Diego Kreutz , Fernando M. V. Ramos , Paulo Esteves Veríssimo , Christian Esteve Rothenberg , Siamak Azodol, "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE (Volume: 103 , Issue: 1 , Jan. 2015) ISSN (Online) 1558-2256.

[8]. Aniket Pramanik, Rishikesh, Vikash Nagar, Satyam Dwivedi, Biplav Choudhury , "A Study on Topology in Computer Network", 2014 7th International Conference on Intelligent Computation Technology and Automation, Electronic ISBN: 978-1-4799-6636-3

[9]. Nayan Seth. April, 2016. SDN Load Balancing. Retrieved from

[10]. https://github.com/nayanseth/sdnloadbalancing