International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 04, April-2021

Software Defined Network based Dynamic Load Balancing Algorithm using Floodlight in Hierarchical Networks

Kejela Abdi Sitota
[1]Computer Science,
Institute of Technology
[2]Wollega University, Nekemte 395, Ethiopia

***Abstract*— To meet a customer's business and technical goals for a corporate network design, our world is using a network topology consisting of many interrelated components. Network design experts have developed the hierarchical network design model to help us develop a topology in discrete layers. Because of unprecedented increasing demand of massive data volume, network management and configuration in today's network become a highly complex and challenging process. To relief from operational expenses and configuration complexity of these individual network resources, Software Defined Networking have a great role in simplifying and improving network management with high flexibility. Accessing network resources timely and efficiently are key requirement for the members of network community. Hierarchical Network might face imbalanced traffic load as few links undergo congestions while the remaining of links are underutilized.**

The research would like to study the Network traffic management techniques in hierarchical structured network. SDN-based dynamic load management is proposed for optimizing link utilization in a hierarchical network. The performance is compared to the static load management algorithm in term of throughput and delay.

Keywords—Hierarchical network, Load balancing, Software Defined Networking,

1. INTRODUCTION

Today, business and technical network requirements include enhancing performance and realizing broader connectivity. Companies have to meet more and more industry-specific security regulations and there is a growing demand for mobility. In order to comply with all of these criteria, networking protocols have evolved significantly over the last few decades. Software-Defined Networking (SDN)" is a term of the programmable networks paradigm. In short, SDN refers to the ability of software applications to program individual network devices dynamically and therefore control the behavior of the network as a whole. However, the networks architectures have been unchanged, increasing the complexity and hindering its configuration. In order to adapt to the new needs, new network paradigms such as Software Defined Networking (SDN), cognitive networks or automatic networks are emerging fast due the interests of carriers and ISPs. The first things to do is to analyze which are the problems of the existing networks since they have become a barrier to creating new, innovative services, and an even larger barrier to the continued growth of the Internet. Analyzing the data plane in the ongoing networks architecture we find well defined layers for different purposes, making them autonomous. For instance the physical layer (fibre,copper, radio) is independent of the link layer, the transport layer or the application layer, what allows the evolution of each of them independently. Thanks to the fact of dividing the problem in tractable pieces, networks have been able to evolve, increasing many magnitude changes in terms of speed, scale or diversity of uses. On the other hand, there is the control plane which, unlike the data plane, has no abstraction layers at all. Within the control plane there are several protocols to configure the network elements, such as Multiprotocol Label Switching or NETCONF.

Protocols tend to be defined in isolation, however, with each solving a specific problem and without the benefit of any fundamental abstractions. This has resulted in one of the primary limitations of today's networks: complexity. For example, to add or move any device, IT must touch multiple switches, routers, firewalls, Web authentication portals, etc. and update ACLs, VLANs, Quality of Services (QoS), and other protocol-based mechanisms using device-level management tools. In addition, network topology, vendor switch model, and software version all must be taken into account. Due to this complexity, today's networks are relatively static as IT seeks to minimize the risk of service disruption.

In order to increase available bandwidth, maximize throughput, and add redundancy; network load balancing must be used. Network load balancing is the ability to balance traffic across multiple Internet connections. This capability balances network sessions like Web, email, etc. over multiple connections in order to spread out the amount of bandwidth used by each LAN user, thus increasing the total amount of bandwidth available.

2. REVIEW OF RELATED WORKS

Several papers deal with the application of SDN in communication networks. The main technical features of SDN are described in [5]. Different models have been proposed to optimize network flow, resolve the network congestion problem by changing paths of flows during flow transmissions and achieve load balancing among different links. Simple approaches include common computer algorithm such as round-robin, random, least traffic, least latency.

Richard Wang, Dana Butnariu, and Jennifer from the Rexford Princeton have explored the possibility of running load balancing in the SDN based network. Their approach was based on creating as few wildcard rules as possible to insert into their switches. That enabled them to load balance proactively without involving the controller. However, they must make wildcard rules for the whole IP range and then split the IP ranges across multiple servers. The drawback of this approach is that the traffic from the whole IP range is often not uniform. They therefore had to implement an updating algorithm that changed the sizes of the IP range slices over time, which struggles with sudden changes in incoming traffic [17].

[18] Has proposed, a fabric topology to provide a flexible data-center network. With SDN controllers, and has the potential to address critical issues such as bandwidth utilization and network capacity in datacenter networks. In addition, the proposed network fabric consists of several key components: SDN controller, commoditized switches, and host machines. All switches are connected to form a switch pool. Each switch can connect to a certain number of switches based on performance requirements.

As a centralized manager, an SDN controller connects to the switch pool and collect network statistics at run-time. Host machines connect to switches via their Ethernet interfaces. The fabric topology is a critical part of proposed networks. Majority physical devices in such networks are low-cost and commoditized switches. There is no explicit hierarchy in the proposed network.

[8] Proposed dynamic load balancer which dynamically shifts the load to the other shortest path when it is greater than the bandwidth of the link. By experimental analysis, the paper concluded that the proposed approach gives better results in terms of responses/sec and efficiency as compared with the existing Round-Robin load balancing algorithm.

[14] Implemented SDN load balancing using PyResonance controller. The controller is implemented with pyretic. Resonance implements an SDN control platform that uses event-driven network control. It preserves a Finite State Machine Model to define network policy.

[19] Has proposed a novel algorithm for purpose of load balancing for SDN-based datacenters. Mininet emulator was utilized for the purpose of emulating the proposed system, the suggested algorithm was added to the POX controller. To evaluate their algorithm, they have simulated a datacenter with a Fat-Tree topology(k=4).The algorithm was proposed to dynamically balance the load by means of re-routing utilizing the information at the SDN controller. The network performance was tested in term of throughput, loss, and received data size with and without applying the proposed algorithm. Results showed that the proposed algorithm outperforms the traditional load balancing scheme as follows; improves the throughput by a minimum of 21.9%, and increase the received data size by 20.8 %.

[20] Proposed Bandwidth based load balancing has developed to distribute the network requests among multiple users based on servers bandwidth consumption. The performance of the proposed approach has evaluated and compared with different load balancing schemes such as round robin and connection based under mininet emulation and Raspberry pi-based implementation. It has also pointed out a new open flow based solution for server's headache problem using SDN that could replace the traditional load balancer which is more dedicated and expensive hardware.

[9] Implemented software defined networking based load balancing using openflow to improve efficiency of load balancing in enterprise networks. The http request from different clients will be directed to different pre-defined http servers based on round robin scheduling.

3. ALGORITHM APPLIED TO SDN

The algorithm that performs load balancing in a hierarchical topology depending on the minimum transmission cost of links at the given time is proposed for balancing the load in this network. The REST API is used to collect operational information of the topology and devices as well as instantaneous traffic and port statistics. Since the topology presents a high multipath capability, the Dijkstra's algorithm is employed to find multiple paths of the same length and reduce the search to a small region of the topology. Among selected paths, the path with least load is selected and traffic flow is forwarded on that route. The new flows rules are therefore pushed to Open Virtual switches switch (OVSs) in order to update switch forwarding tables.

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 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 each switch 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. Finally, the program continues to update this information every minute there by making it dynamic. The performance of the algorithm is evaluated in a fat tree datacenter topology by collecting operational data of the links and switches.

Proposed algorithm for dynamic load balancing operation is described as follows:

```
Algorithm 1: Heuristic for Traffic Load Balancing
Input: T (Traffic Matrix), Network Topology, Link Capacity
Output: Minimizing maximum link utilization path allocation of flows in T
For all possible flow f in T do
    List all possible paths from $f_{src}$ to $f_{dst}$
    $list_p$ =Apply the Dijikstra's algorithm to find multiple paths of minimum length
    for all path p in $list_p$ do
        list $_{maximum\ link\ utilization}$[P] = maximum link utilization of p
    end for
    Pselected = listp[index of minimum in list maximum link utilization]
    Assign f to $p_{selected}$
    do for all link l in $p_{selected}$ update flow switch table
    end for
end for
```

4. SIMULATION TOOLS PROCESS

The goal of the proposed load balancing algorithm is to improve traffic management in the DCN when congestion occurs. To perform the load balancing experiment ubuntu operating system is used. SDN controller used is Floodlight controller and mininet is used to generate network topology. Mininet is connected to the floodlight controller through the ip address of controller and port 6653 and run our topology to generate the data center topology shown in figure 1, “Ping all” command is executed to check the reachability of all hosts in the defined network. Running network traffic is captured using wireshark and iperf is employed to test network performance. Python programming language implemented for developing Network topology and the algorithm. The simulated network topology consists of 2 core switches at core level, 4 distribution switches at distribution level, 4 access switch and 8 computers that can be viewed using REST API interface as shown in figure1. To perform load balancing test first, the network performance with normal packet transmission is observed first, and call this phase as ‘before load balancing (BB)’. Then apply the load balancing algorithm and call this phase as ‘after load balancing (AB)’. Again the network performance is observed to verify that the algorithm is capable of traffic management.

The experiment is conducted at Distribution level before load balancing and after load balancing. Also it is conducted at core level before load balancing and after load balancing. Finally the result obtained is analyzed. The result of the experiment is discussed in terms of Qos service measurement done.

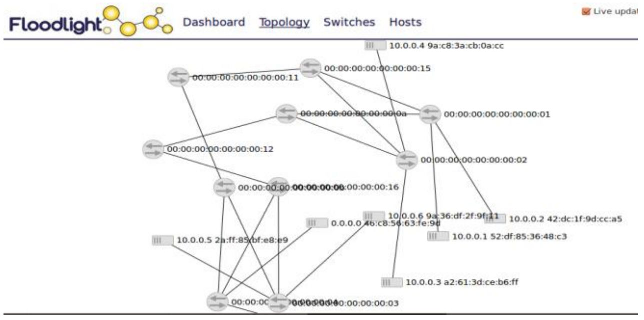


Figure 1: Floodlight REST API exhibits the Fat-tree generated by Mininet.

5. SIMULATION RESULT

5.1. Performing load balancing at Distribution level

In this case the load balancing is done for the communication between hosts at a distribution level. The load balancing can run between any hosts in the defined network. For this experiment pc1 is choose as source. pc3 and pc4 is chosen as destination hosts. For the experiment the ping command is transmitted from pc1 to pc3 and pc4.The path taken by switch s1 to forward traffic from pc1 to pc3 and pc4 is captured by wireshark and also throughput and response time is tested. The test is done before load balancing (BB) and after load balancing (AB)

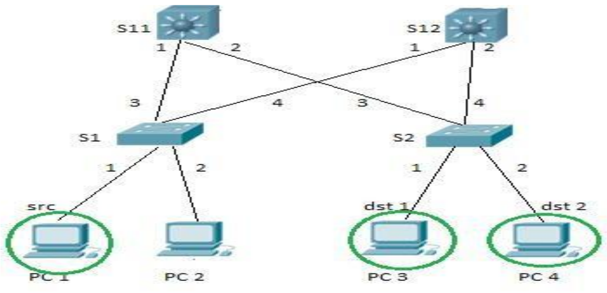
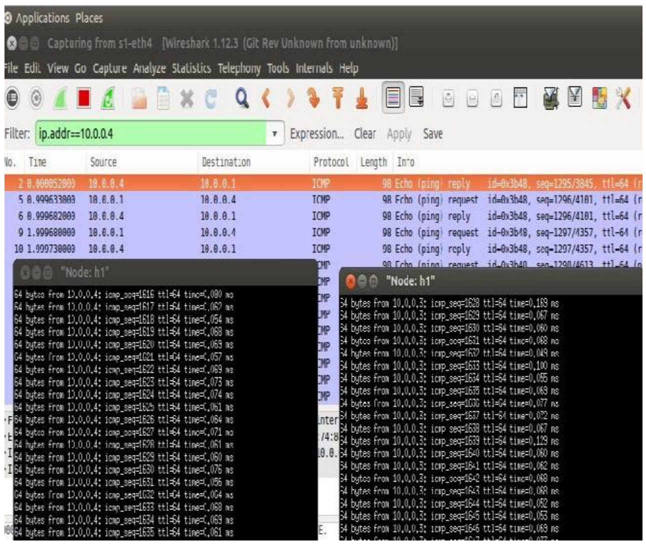


Figure 2: Distribution level test scenario

A. Distribution level testing before load balancing

Ping command is sent from pc1 to both pc3 and pc4 from the mininet terminal then after a wireshark is started to capture ICMP packets send from pc1 to pc3 and pc4. The result of wireshark capture show that both traffics is passing through s1-eth4 while there is no traffic on s1-eth3 which mean that there is no load balancing among the links.


Applications Places
Capturing from s1-eth4 [Wireshark 1.12.3 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save
"Node: h1"
"Node: h1"

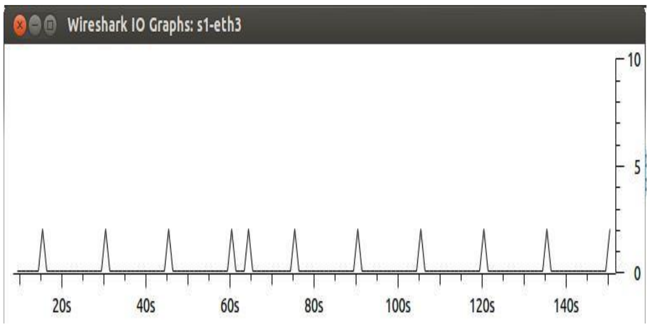
```
Filter: ip.addr==10.0.0.4

No.  Time         Source     Destination  Protocol  Length  Info
  2  0.000052000  10.0.0.4   10.0.0.1     ICMP      98 Echo (ping) reply    id=0x3b48, seq=1295/3845, ttl=64 (r
  5  0.999633000  10.0.0.1   10.0.0.4     ICMP      98 Echo (ping) request  id=0x3b48, seq=1296/4101, ttl=64 (r
  6  0.999682000  10.0.0.4   10.0.0.1     ICMP      98 Echo (ping) reply    id=0x3b48, seq=1296/4101, ttl=64 (r
  9  1.999680000  10.0.0.1   10.0.0.4     ICMP      98 Echo (ping) request  id=0x3b48, seq=1297/4357, ttl=64 (r
 10  1.999730000  10.0.0.4   10.0.0.1     ICMP      98 Echo (ping) reply    id=0x3b48, seq=1297/4357, ttl=64 (r
```

```
64 bytes from 10.0.0.4: icmp_seq=1616 ttl=64 time=0.080 ms
64 bytes from 10.0.0.4: icmp_seq=1617 ttl=64 time=0.067 ms
64 bytes from 10.0.0.4: icmp_seq=1618 ttl=64 time=0.054 ms
64 bytes from 10.0.0.4: icmp_seq=1619 ttl=64 time=0.068 ms
64 bytes from 10.0.0.4: icmp_seq=1620 ttl=64 time=0.069 ms
64 bytes from 10.0.0.4: icmp_seq=1621 ttl=64 time=0.057 ms
64 bytes from 10.0.0.4: icmp_seq=1622 ttl=64 time=0.069 ms
64 bytes from 10.0.0.4: icmp_seq=1623 ttl=64 time=0.073 ms
64 bytes from 10.0.0.4: icmp_seq=1624 ttl=64 time=0.074 ms
64 bytes from 10.0.0.4: icmp_seq=1625 ttl=64 time=0.061 ms
64 bytes from 10.0.0.4: icmp_seq=1626 ttl=64 time=0.064 ms
64 bytes from 10.0.0.4: icmp_seq=1627 ttl=64 time=0.071 ms
64 bytes from 10.0.0.4: icmp_seq=1628 ttl=64 time=0.061 ms
64 bytes from 10.0.0.4: icmp_seq=1629 ttl=64 time=0.060 ms
64 bytes from 10.0.0.4: icmp_seq=1630 ttl=64 time=0.076 ms
64 bytes from 10.0.0.4: icmp_seq=1631 ttl=64 time=0.056 ms
64 bytes from 10.0.0.4: icmp_seq=1632 ttl=64 time=0.064 ms
64 bytes from 10.0.0.4: icmp_seq=1633 ttl=64 time=0.068 ms
64 bytes from 10.0.0.4: icmp_seq=1634 ttl=64 time=0.069 ms
64 bytes from 10.0.0.4: icmp_seq=1635 ttl=64 time=0.061 ms
```

```
64 bytes from 10.0.0.3: icmp_seq=1628 ttl=64 time=0.169 ms
64 bytes from 10.0.0.3: icmp_seq=1629 ttl=64 time=0.067 ms
64 bytes from 10.0.0.3: icmp_seq=1630 ttl=64 time=0.060 ms
64 bytes from 10.0.0.3: icmp_seq=1631 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=1632 ttl=64 time=0.049 ms
64 bytes from 10.0.0.3: icmp_seq=1633 ttl=64 time=0.100 ms
64 bytes from 10.0.0.3: icmp_seq=1634 ttl=64 time=0.055 ms
64 bytes from 10.0.0.3: icmp_seq=1635 ttl=64 time=0.069 ms
64 bytes from 10.0.0.3: icmp_seq=1636 ttl=64 time=0.077 ms
64 bytes from 10.0.0.3: icmp_seq=1637 ttl=64 time=0.072 ms
64 bytes from 10.0.0.3: icmp_seq=1638 ttl=64 time=0.067 ms
64 bytes from 10.0.0.3: icmp_seq=1639 ttl=64 time=0.129 ms
64 bytes from 10.0.0.3: icmp_seq=1640 ttl=64 time=0.060 ms
64 bytes from 10.0.0.3: icmp_seq=1641 ttl=64 time=0.062 ms
64 bytes from 10.0.0.3: icmp_seq=1642 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=1643 ttl=64 time=0.068 ms
64 bytes from 10.0.0.3: icmp_seq=1644 ttl=64 time=0.052 ms
64 bytes from 10.0.0.3: icmp_seq=1645 ttl=64 time=0.053 ms
64 bytes from 10.0.0.3: icmp_seq=1646 ttl=64 time=0.069 ms
```

Figure 3: Wireshark capture traffic on s1-eth4 that go to pc4

Wireshark IO Graph output shows the throughput from pc1 to pc3 and pc4 are routed through s1-eth4 only, while no flow is found in the s1-eth3 as shown in the figure 4. The small traffic pics shown in s1-eth3 correspond to the LLDP (link layer discovery protocol) packets sent frequently from the rest API to collect statistics.



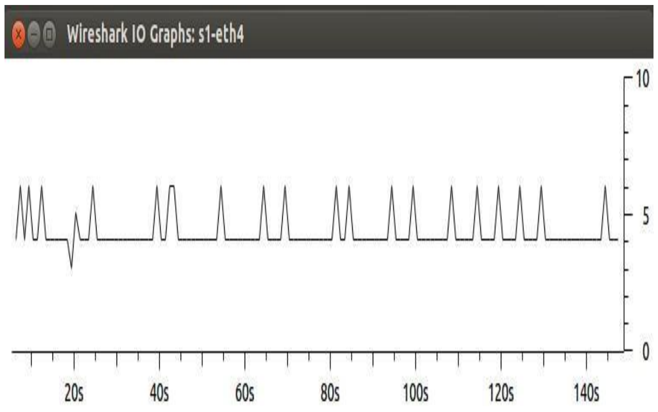


Figure 4: Wireshark io graph showing traffic on s1-eth4 and no traffic on s1-eth3

I. Quality of Service Testing at Distribution level before load balancing

The network was tested before running the load balancing algorithm. The testing focused on some of Quality of service (QoS) parameters such as Bandwidth, Transfer and response time. This is tested by using iperf and maintaining ping command between hosts.

Result obtained from testing Quality of service before load balancing at distribution level are described in the table below.

No.	Transfer(GBytes)	Bandwidth(GBits/sec)	Response time
1	10.5	9.0	0.32
2	12.01	11.0	0.474
3	12.5	10.95	0.323
4	11.35	9.51	0.465
5	11.55	10.01	0.67
6	12.93	11.23	0.487
7	11.1	9.93	0.526
8	10.3	9.1	0.323
Average	**11.53**	**10.09125**	**0.449**

Table 1: Result at distribution level before load balancing

B. Distribution level testing after load balancing

This time the load balancing algorithm written in python is run in the terminal where we run the floodlight controller. The test is to see whether it is able to find alternate best paths when the initial path from source to destinations got congested.

Ping command is done from pc1 to both pc3 and pc4 from the mininet terminal then after a wireshark is started to capture ICMP packets send from pc1 to pc3 and pc4. The result of wireshark capture show that traffic from pc1 to pc3 is passing through s1-eth3 while traffic from pc1 to pc4 is passing through s1-eth4 which mean that there is load balancing among the links.

This time Wireshark IO Graph output shows there is traffic on both interfaces of switch one which mean that load is balancing is done.

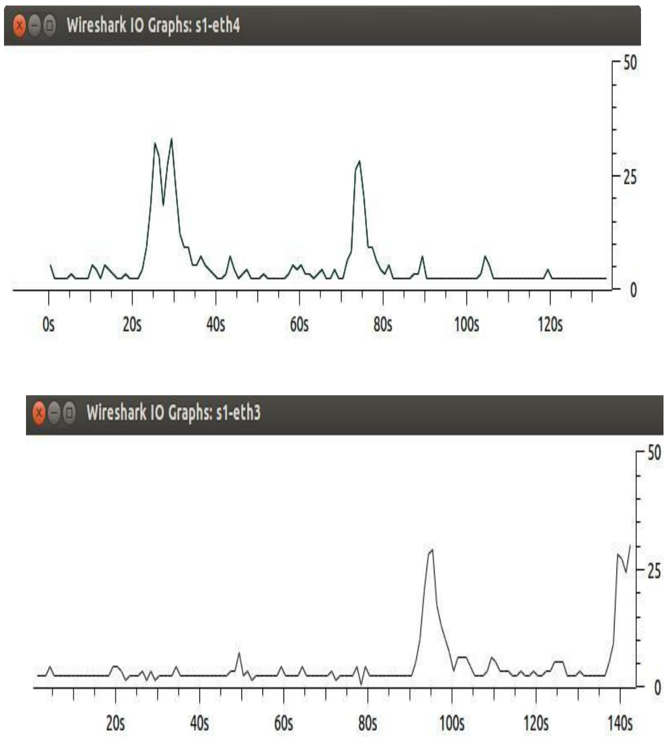
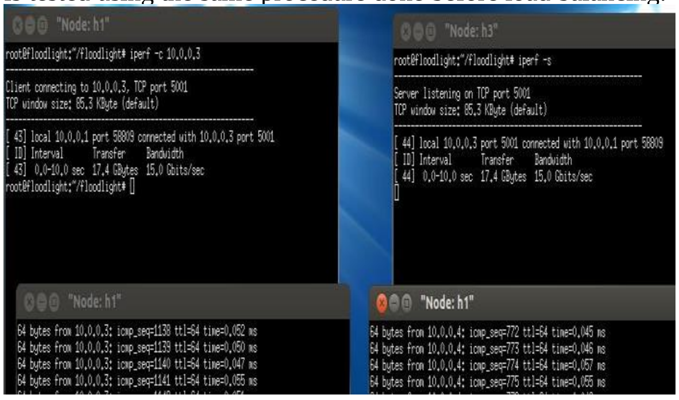


Figure 5: Wireshark io graph showing traffic on s1-eth4 and s1-eth3

II. Quality of Service Testing at distribute level After Load balancing

The network was tested again after running the load balancing algorithm. Bandwidth, Transfer and response time is tested using the same procedure done before load balancing.



```
root@floodlight:~/floodlight# iperf -c 10.0.0.3
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 43] local 10.0.0.1 port 58809 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 43]  0.0-10.0 sec  17.4 GBytes  15.0 Gbits/sec
root@floodlight:~/floodlight# []
```

```
root@floodlight:~/floodlight# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 44] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 58809
[ ID] Interval       Transfer     Bandwidth
[ 44]  0.0-10.0 sec  17.4 GBytes  15.0 Gbits/sec
[]
```

```
64 bytes from 10.0.0.3: icmp_seq=1138 ttl=64 time=0.052 ms
64 bytes from 10.0.0.3: icmp_seq=1139 ttl=64 time=0.050 ms
64 bytes from 10.0.0.3: icmp_seq=1140 ttl=64 time=0.047 ms
64 bytes from 10.0.0.3: icmp_seq=1141 ttl=64 time=0.055 ms
```

```
64 bytes from 10.0.0.4: icmp_seq=772 ttl=64 time=0.045 ms
64 bytes from 10.0.0.4: icmp_seq=773 ttl=64 time=0.046 ms
64 bytes from 10.0.0.4: icmp_seq=774 ttl=64 time=0.057 ms
64 bytes from 10.0.0.4: icmp_seq=775 ttl=64 time=0.055 ms
```

Figure 6: QoS test after load balancing at distribution level

Result obtained from testing Quality of service after load balancing at distribution level are described in the table below.

No.	Transfer (GBytes)	Bandwidth (GBits/sec)	Response time
1	17.4	15.0	0.131
2	18.5	16.2	0.165
3	17.0	15.1	0.09
4	16.4	15.0	0.10
5	18.1	16.01	0.147
6	18.2	16.3	0.14
7	16.5	14.0	0.121
8	16.8	14.2	0.122
Average	**17.3625**	**15.22625**	**0.145**

Table 2: Result at distribution level after load balancing

C. Discussion of distribution level test result

The network performance test results before load balancing and after load balancing have the difference in terms of response time and throughput. Average of transfer rate, bandwidth and response time before load balancing is 11.53, 10.09 and 0.449 respectively. While average of transfer rate, bandwidth and response time after load balancing is 17.36, 15.23 and 0.144 respectively. The network performance is improved after load balancing which result in transfer rate improved with 50% and bandwidth improved with 50 %. Response time also improved by average from 0.449 to 0.144.

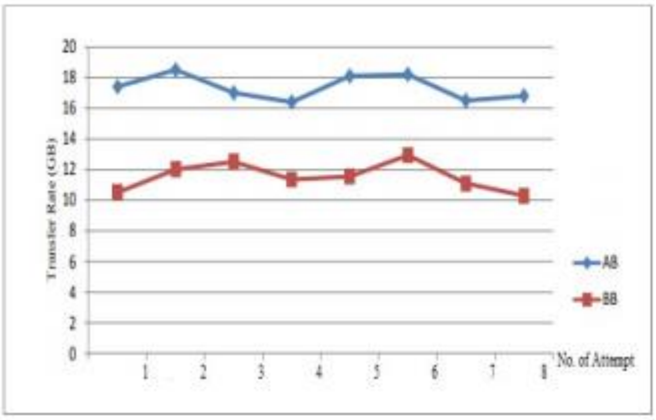


Figure 7: Transfer rate comparison at distribution level

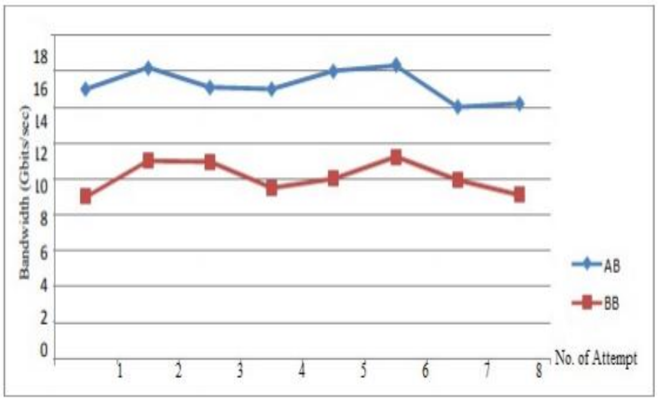


Figure 8: Bandwidth comparison at distribution level

5.2 Performing Load balancing at core level

In this case the load balancing is done for the communication between hosts at a core level. For this experiment pc1 is choose as source. Pc5 and pc6 is chosen as destination hosts. For the experiment the ping command is transmitted from pc1 to pc5 and pc6. The test is done before load balancing (BB) and after load balancing (AB).

A. Core level testing before load balancing

Ping command is done from pc1 to both pc5 and pc6 from the mininet terminal then after a wireshark is started to capture ICMP packets send from pc1 to pc5 and pc6. The result of wireshark capture show that both traffics is passing through the same interface while there is idle interface which show that there is no load balancing among the links.

I. Quality of service testing at core level before load balance

The testing focused on some of Quality of service (QoS) parameters such as Bandwidth, Transfer and response time. This is tested by using iperf and maintaining ping command between hosts.

Result obtained from testing Quality of service before load balancing at core level are described in the table below

No.	Transfer(GBytes)	Bandwidth(GBits/sec)	Response time
1	9.80	8.41	0.642
2	8.79	7.2	0.367
3	9.32	8.1	0.834
4	8.3	7.0	0.365
5	8.5	7.1	0.323
6	9.54	8.32	0.552
7	9.12	8.20	0.456
8	8.23	7.2	0.587
Average	**8.95**	**7.69**	**0.516**

Table 3: Result at core level before load balancing

B. Core level testing after load balancing

Ping command is done from pc1 to both pc5 and pc6 from the mininet terminal then after a wireshark is started to capture ICMP packets send from pc1 to pc5 and pc6. The result of wireshark capture show that traffics is passing through different interface as shown in the figure below.

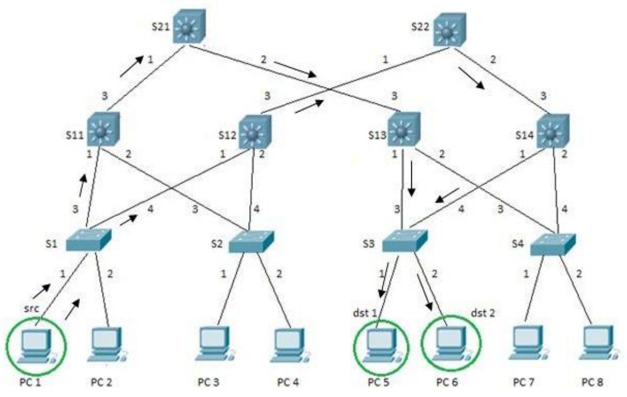


Figure: 9 Traffic paths after load balancing

II. Quality of service testing at core level after load balancing

The network was tested again after running the load balancing algorithm. Bandwidth, Transfer and response time obtained from the test is described below.

No.	Transfer(GBytes)	Bandwidth(GBits/sec)	Response time
1	11.85	10.81	0.435
2	12.91	9.8	0.23
3	11.76	9.7	0.461
4	11.95	9.2	0.411
5	10.96	11.83	0.552
6	12.75	10.45	0.514
7	12.55	10.30	0.487
8	10.90	9.75	0.423
Average	**11.95**	**10.23**	**0.439**

Table 4: Result at core level after load balancing

C. Discussion of core level test result

The network performance test result before load balancing and after load balancing has the difference in terms of response time and throughput. Average of transfer rate, bandwidth and response time before load balancing show on table is 8.95, 7.69 and 0.516 respectively. While average of transfer rate, bandwidth and response time after load balancing is 11.95, 10.23 and 0.439 respectively.

The network performance is improved after load balancing which result in transfer rate improved by with 33% and bandwidth improved with 33 %. The average response time improved from 0.516 to 0.439

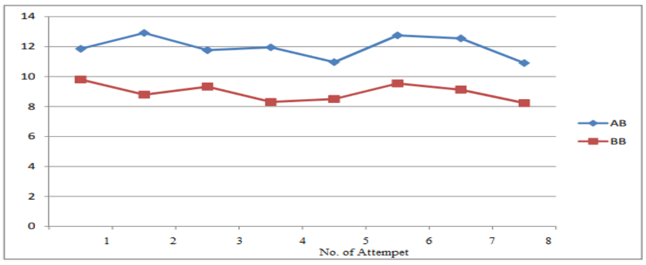


Figure 10: Transfer rate comparison at core level

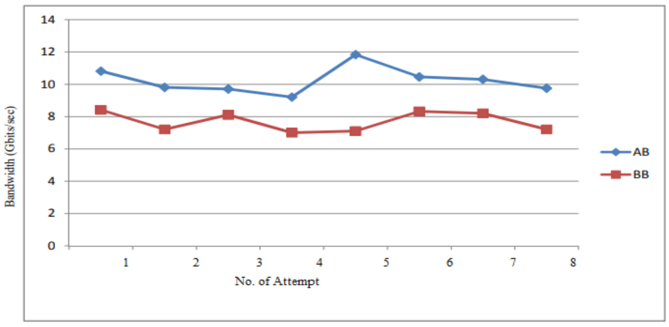


Figure 11: Bandwidth comparisons at core level

6. CONCLUSION

Floodlight controller based dynamic load balancing is implemented in data center network which has two alternative paths from source to destination. The network performance is tested before and after implementing load balancing to test quality of service in selected type of data center network.

The result of performance test done at distribution and core level before and after implementing load balancing is analyzed. The network performance shows improvement after implementing load balancing in terms of quality of service test done.

This thesis describes the implementation of a dynamic load balancing algorithm to distribute the different traffic flows carried by a network through the different parallel paths between source and destination. The implementation has been conducted over a Software Defined Network, trying to explore the capabilities that this new paradigm of networking brings to us.

SDN provides a view of each of the elements of the network as well as control over them. Thus, allows a dynamic control of the actions to be done in each possible situation. Regarding the optimization of resources, such control over the network adds a bunch of new functionalities, such as the dynamic routing proposed in this project.

Nowadays, OpenFlow is the most popular protocol for the southbound interface (i.e. to communicate the controller with the network elements), nonetheless, it presents some limitations. One of them is the monitoring of the network, which is involved in this project. OpenFlow, being a control-plane protocol has great difficult accurately determining data plane measurements. The way to do this with OpenFlow entails making periodic flow stats requests to the switches on the network. The problem is that stats are never truly accurate.

By the time they have been processed on the switch, sent over the network, and then processed by the controller they will be out of date. This is one of the problems sho 4wed in the emulation section. In addition to that, the stats requests generate a large amount of traffic (even though the traffic between controller and the switches is carried by a dedicated channel, it should be reduces as much as possible), and increase the computational cost in the controller to process these messages. It is because of that in the implementation the stats are updated once per second, which leads to a delay in the rerouting process. As a conclusion about this point, data-plane measurements are best handled by third party applications, thus improving the results of the implementation.

In relation to the load balancing, the proposed algorithm works well for non-priority traffic, with the objective of reroute the traffic flows which are carrying less traffic. Based on that principle, it has been shown how it is possible to reduce congestion and simultaneously enhance network resource utilization.

7. FUTURE WORK

Even though the implementation has been designed to be adapted to hierarchical topology, a further analysis with different topologies of different sizes should be made so could be guaranteed that it has no limitations on that aspect. A deepest analysis with a larger number of flows, with different kinds of traffic patterns is also needed.

Testing this floodlight controller based dynamic load balancing for different type of network structure to test performance of this proposed method to trouble if it has some drawback. Comparing floodlight controller based dynamic load balancing with other software defined controller based load balancing to test performance.

8. REFERENCES

[1] Martí Boada Navarro. "Dynamic Load Balancing in Software-Defined Networks". Aalborg University, Department of Electronic Systems, Fredrik Bajers Vej 7B, DK-9220 Aalborg. June 2014.

[2] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in Proc. Inter. Conf. on Advanced Information Networking and Applications, Perth, Australia, April 2010 pp. 551-556.

[3] H. Long, Y. Shen, M. Guo and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," in Proc. Inter. Conf. on Advanced Information Networking and Applications, Barcelona, Spain, Mar 2013, pp. 290-297.

[4] Y. L. Lan, K. Wang, and Y. H. Hsu, "Dynamic load-balanced path optimization in SDN- based data center networks," in Proc. IEEE Inter. Conf. on Comm. Sys, on Networks and Digital Signal Processing, Prague, Czech Republic, July 2016, pp. 1-6.

[5] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," J. Network and Computer Applications, vol. 67, pp. 1-25, May 2016..

[6] Brian Underdahl and Gary Kinghorn. "Software Defined Networking for Dummies", Cisco Special Edition, John Wiley & Sons, Inc., Hoboken, New Jersey, 2015.

[7] Y. Li and D. Pan, "OpenFlow based load balancing for Fat-Tree networks with multipath support," in Proc. IEEE Inter. Conf. on Comm., Budapest, Hungary, June 2013, pp. 1-5.

[8] Smriti Bhandarkar and Kotla Amjath Khan. "Load Balancing in Software- defined Network (SDN) Based on Traffic Volume". Advances in Computer Science and Information Technology (ACSIT), Krishi Sanskriti Publications, Volume 2, Number 7; April – June, 2015.

[9] Senthil Ganesh N and Ranjani S. "Dynamic Load Balancing using Software Defined Networks". International Journal of Computer Applications (0975 –8887), International Conference on Current Trends in Advanced Computing(ICCTAC-2015), May 2015.

[10] Project Floodlight, Floodlight. (2012). from http://floodlight.openflowhub.org/

[11] Mininet: An Instant Virtual Network on Your Laptop (or Other PC) [online]. Available:http://www.mininet.org/.

[12] iperf - The TCP, UDP and SCTP network bandwidth measurement tool [online]. Available:https://www. iperf.fr/.

[13] Guido van Rossum. "An Introduction to Python for UNIX/C Programmers". Proceedings of the NLUUG najaarsconferentie. Amsterdam, Netherlands, 1993.

[14] Yuanhao Zhou, Li Ruan, Limin Xiao and Rui Liu. "A Method for Load Balancing based on Software-Defined Network". Advanced Science and Technology Letters, Vol.45 (CCA 2014), pp.43-48, 2014.

[15] http://mininet.org/overview/

[16] (2012). Inter-datacenter wan with centralized te using sdn and openflow. White paper, Google, Inc.

[17] Richard Wang, J. R., Dana Butnariu. (2011). Openflow-based server load balancing gone wild. Princeton University.

[18] L. Chen, M. Qiu, J. Xiong, "An sdn-based fabric for flexible data-center networks" in Cyber Security and Cloud Computing (CSCloud) 2015 IEEE 2nd International Conference on, IEEE, pp. 121-126, 2015.

[19] F. S. Fizi, S. Askar, "A novel load balancing algorithm for software defined network based datacenters", Proc. IEEE Int. Conf. Broadband Commun. Next Generat. Netw. Multimedia Appl. (CoBCom), pp. 1-6, Sep. 2016.

[20] M.I. Hamed, B.M. ElHalawany, M.M. Fouda, A.S. Tag Eldien, "A New Approach for Server- based Load Balancing Using Software-Defined Networking", Proceedings of the 2017 IEEE International Conference on Intelligent Computing and Information Systems (ICICIS 2017), December 5–7, 2017.