ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 10, Issue 4, pp: (24-32), Month: October - December 2022, Available at: www.researchpublish.com

DYNAMIC LOAD BALANCING OF NETWORK RESOURCES AND DATA TRANSMISSION IN SOFTWARE-DEFINED NETWORKS

Ahmad Idris Hammaadama[1], Babangida Albaba Abubakar[2]

Department of Computer Science, Umaru Musa Yar'adua University, Katsina Nigeria

DOI: https://doi.org/10.5281/zenodo.7261016

Published Date: 28-October-2022

Abstract: **Traditional networking architectures have a number of important flaws that must be solved in order to satisfy today's IT demands. Software Defined Networking (SDN) is emerging as a new networking solution to address these limitations. The implementation of a dynamic load balancing algorithm to efficiently distribute fat-tree network flows through multiple alternative paths between a single pair of hosts is described in this paper. The network was evaluated both before and after the load balancing algorithm was applied. The testing focused on some QoS parameters such as throughput, delay, latency, jitter, and packet loss between two servers in the fat-tree network. The performance of the proposed algorithm was evaluated through an existing result of [1], to evaluate the extent to which the maximum network resources utilization was achieved. The result obtained shows that the proposed solution was more promising in terms of minimizing end-to-end delay, latency, jitter, packet loss and maximizing throughput in which we observed a reduction in latency by 33% and packet loss by 32% while throughput was maximized by 41%.**

Keyword: **Software-Defined Networking, Fault-tolerance, Control Plane, Controller failure.**

1. INTRODUCTION

Fault tolerance is a critical property for the availability of computer networks that have been studied extensively since the 1950s. However, to explore and revise new networking solutions and technologies, this concept must be revisited from time to time [2]. In this context, the newly emerging paradigm of software-defined networks (SDNs) offers a ray of hope for a new networking architecture that is more flexible and adaptable.

SDN (Software-Defined Networking) is a new networking paradigm that has gained popularity in recent years. SDN decouples the control plane from the data plane [3] allowing network applications to be developed using high-level abstractions that are translated to network devices through a southbound interface [4]. In a basic SDN architecture, all of the switches in the network are controlled by a single centralized controller. It may not be ideal for the network to rely on a single controller because the controller is a single point of failure and a performance bottleneck [5]. Akyildiz *et al*. [6] also point out that it does not have enough control capabilities to manage the network with increasing traffic and scale. To enhance the scalability of the control plane for multi-domain SDN, several researchers propose deploying a logically centralized yet physically distributed multi-controller, such as HyperFlow [7], and Kandoo [8].

Every domain in a multi-domain network has its local controller, known as the domain controller to handle the switches and process flow requests from its domain, and domain controllers communicate with one another about their domain information to ensure a consistent network view [9]. Bari *et al*. [10] analyze that the controller will face a greater workload in handling flow requests as the number of underlying network devices and traffic increases, resulting in a higher probability of controller failure. A controller can also fail if the server or virtual machine that runs it results from a hardware failure or software crash, point Bari *et al*.

Some SDN protocols (e.g., OpenFlow 1.2) propose using a backup method to improve the multi-controller control plane's resiliency [9]. There are two roles for each controller: master and slave. The master controller of a switch is usually in charge of handling flow requests, while the slave controller is used as a backup. A role request message is sent by each controller to the switches informing them of its role. The domain controller acts as the master of the domain`s switches, but the switches can also connect to controllers from other domains and set them as slaves. If a switch's master controller fails, the switch will be reconnected to its predetermined slave controller, which will serve as a new master controller for the switch.

Ineffective Slave Controller Assignment can degrade network performance. It can even cause controller chain failure, resulting in the overall network's collapse. Research recently conducted by Hu *et al*. [1] introduced a Dynamic Slave Controller Assignment (DSCA) scheme that was able to prepare ahead for the failure of controllers to avoid network crashes. This work is novel and solid, and hence we intend to follow some of its approaches. However, we propose some modifications to the scheme, by implementing a Dynamic Load Balancing using Dijkstra`s algorithm that will be responsible for distributing traffic of incoming and outgoing network flows to achieve the best possible resource utilization of each of the links present in the network.

2. RESEARCH OBJECTIVES

To achieve the aim of this research the following objectives are outlined as to:

I. Replicate and implement the DSCA scheme.

II. Enhance the DSCA scheme by adding a Dynamic Load Balancing module.

III. Perform a simulation on the proposed scheme to measure its performance.

IV. Test and validate the proposed algorithm’s functionality.

3. REVIEW OF RELATED LITERATURE

Software-Defined Network offers a ray of hope for a new networking architecture that is flexible and adaptable. Every day, a community of researchers works to develop fault-tolerant techniques to achieve a high-reliability control plane. Various literatures related to the topic will be discussed in this section.

In the early stages of SDN design, the entire network is managed by a single controller. To handle large-scale networks, some researchers propose using multi-controllers, which can be divided into two aspects: flat architecture (e.g., HyperFlow [7], Onix [11]), and hierarchical architecture (e.g., Kandoo [8]). The motive of a multi-controller is to improve the scalability of the SDN while also increasing the control plane's reliability. Based on this proposal, several works investigate the robust control plane together with multi-controller.

To characterize the reliability of SDN Y. Hu *et al.* [12] establish a new metric, called the expected percentage of control path loss. Meanwhile, the Reliability-aware Controller is shown to be an NP-hard issue, with numerous placement algorithms being investigated to tackle it.

To address the single path between switch and controller failure, Muller *et al.* [13] propose Survivor, a controller placement strategy. To avoid controller overload, Survivor correctly investigates the path diversity problem and considers capacity awareness when deploying controllers. It does not, however, take into account the situation of a multi-controller failure.

Song *et al.* [14] investigate the reliability of SDN from the perspective of the control path. They primarily focus on the reliability challenges in a control path network between the control layer and the data layer, and they develop control path reliability algorithms as well as a novel control message classification and prioritization system to improve SDN's stability.

Gonzalez *et al.* [15] propose a new mechanism that takes into account SDN controller consistency and fault tolerance. Its goal is to bring the performance of the SDN Master-Slave controller as close to that of a single controller as possible, regardless of whether the controller fails. This mechanism performs consistency and correction checks by introducing a simple replication scheme, which only affects network performance during a few slots.

Liang *et al.* [16] propose a scalable and crash-tolerant load balancing architecture that can dynamically shift load across multiple controllers using switch migration. This architecture also eliminates the problem of a single point of failure by allowing controller failover without the need for switch disconnection. The OpenDaylight controller is used to implement the prototype system.

Xie *et al.* [17] investigate minimal fault-tolerant coverage of controllers in the data centre environment. They approach the problem from three angles: minimal coverage, minimal fault-tolerant coverage, and minimal controller communication overhead. As a result, efficient algorithms are created to achieve those goals. It does not, however, take into account the network's dynamic traffic.

The above works are primarily concerned with determining how to deploy multiple controllers to increase reliability and avoid failure, but they are static and inflexible.

4. METHODOLOGY AND ALGORITHM DESIGN

The architecture designed for this system is a key component of this research, which is concerned with all aspects of the design and implementation of the proposed solution. The proposed architecture can be seen in fig. 1 below:

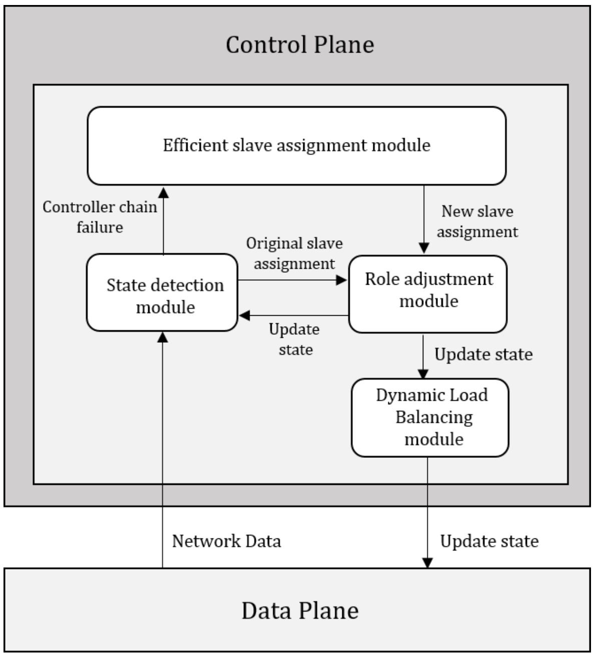

Fig. 1: Architecture of the proposed system

The system is divided into four components, as shown in figure 10: a state detection module, an effective slave assignment module, a role adjustment module and a dynamic load balancing module. The system takes network data as input, including topology connections and switch flow request rates.

The state detection module, as the first module, checks whether there are controller failures. Further, it has two output results: if the existing failed controllers cause controller chain failure, they will be labelled and delivered to an efficient slave assignment module. Otherwise, they're sent to the role adjustment module, which will carry out the original slave assignment.

The efficient slave assignment module is the core part of EDSCA, and it will complete the elastic slave assignment according to the controller failure condition and generate a new slave assignment for switches.

Based on the types of slave assignment, the role adjustment module changes the roles (e.g. master and slave) of some controllers (new or original). Consequently, the adjustment results are respectively delivered to the state detection module and dynamic load balancing module to ensure efficient network resource utilization and then finally update network states to the data plane, including the load information of controllers and connection relationships between switches and controllers.

Based on the four modules design and the corresponding feedback process, EDSCA can effectively distribute the traffic of incoming and outgoing network flows to achieve the best possible resource utilization of each of the links present in the network.

Flowchart of Dynamic Load Balancing Algorithm

As explained formerly, the task of the proposed algorithm is to distribute the traffic of upcoming and incoming network flows to achieve the best possible resource utilization of each of the links present in a network. To achieve such an aim, it is necessary to keep track of the current state of the network. Fig. 2 illustrates the steps of the load balancing algorithm.

The algorithm's first phase is to gather operational information about the topology and its devices. IP addresses, MAC addresses, ports, connections, and so on.

The next step is to use Dijkstra's algorithm to find route information. The goal is to narrow the search to a small segment of the Fat-Tree topology and find the shortest paths between the source and destination hosts.

The flows are formed based on the least transmission costs of the links at the given time after the transmission costs of the links have been calculated.

The best path is chosen based on the cost, and static flows are pushed into each switch in the current best path. As a result, every switch along the chosen path will have the required flow entries to carry out communication between the two end points.

Finally, the program maintains a dynamic state by updating the information every minute.

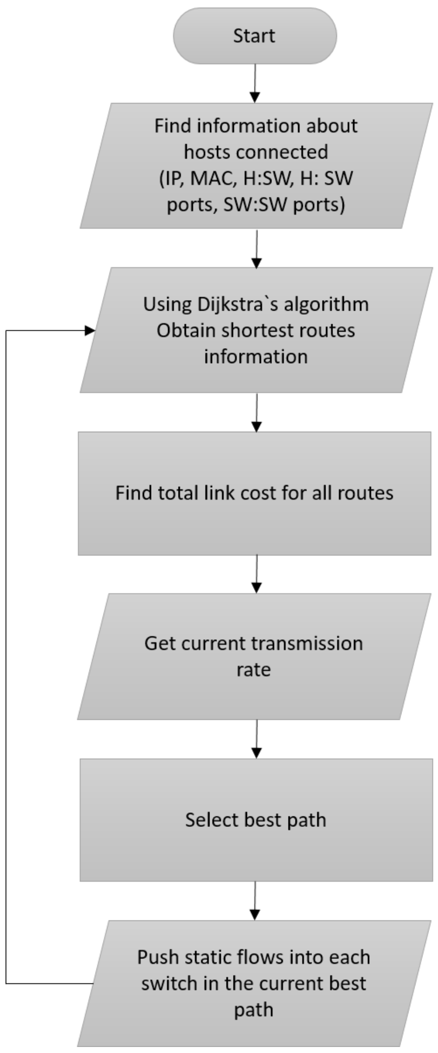
Start
Find information about hosts connected (IP, MAC, H:SW, H: SW ports, SW:SW ports)
Using Dijkstra`s algorithm Obtain shortest routes information
Find total link cost for all routes
Get current transmission rate
Select best path
Push static flows into each switch in the current best path

Fig. 2: Dynamic Load Balancing Algorithm

Implementation Overview

In this study, we select OpenDaylight as the experimental controller and use Mininet as the test platform. OpenDayLight supports multiple versions of OpenFlow protocols. Both OpenDayLight and Mininet run on Linux Operating System precisely Ubuntu 18.04. Specifically, the configuration of the system includes AMD A4-9125 RADEON R3, 4 COMPUTE CORES 2C+2G 2.30 GHz and 8.00 GB RAM. Further, the proposed solution is loaded on OpenDayLight as an application. To define the fat-tree topology and write the load-balancing algorithm program, python was used, and iPerf to test network performance. The design steps are depicted in the diagram below.

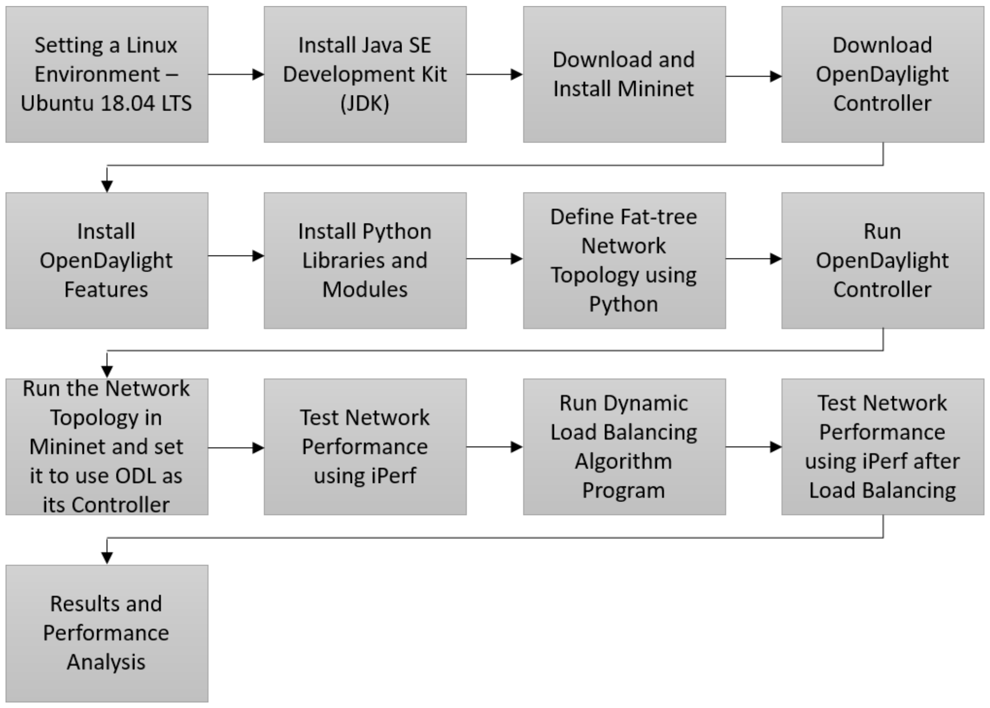
Setting a Linux Environment – Ubuntu 18.04 LTS
Install Java SE Development Kit (JDK)
Download and Install Mininet
Download OpenDaylight Controller
Install OpenDaylight Features
Install Python Libraries and Modules
Define Fat-tree Network Topology using Python
Run OpenDaylight Controller
Run the Network Topology in Mininet and set it to use ODL as its Controller
Test Network Performance using iPerf
Run Dynamic Load Balancing Algorithm Program
Test Network Performance using iPerf after Load Balancing
Results and Performance Analysis

Fig. 3: Implementation Steps

Simulation Scenario

Ten simulation cases were conducted for each test. For each case, the bandwidth of the edge and the capacity of a node were set to randomly be within the range shown in Table 1.

Table 1: Simulation Parameters - First Scenario (Aggregation Layer)

Parameter	***Setting***
Bandwidth on Edges (TCP)	1Gbps ~ 24Gbps
Bandwidth on Edges (UDP)	100Mbps ~ 800Mbps
Number of Nodes	8
Capacity of Nodes	1Gbps ~ 28Gbps
Number of Switches	4
Number of Edges	6
Controller	OpenDaylight Controller
Testing tool	iPerf
Monitoring Tool	Wireshark
Testing Environment	Mininet

Performance Measurement

In this scenario the severs h1 and h4 have been selected to perform the load balancing between them. Consider fig. 4 below.

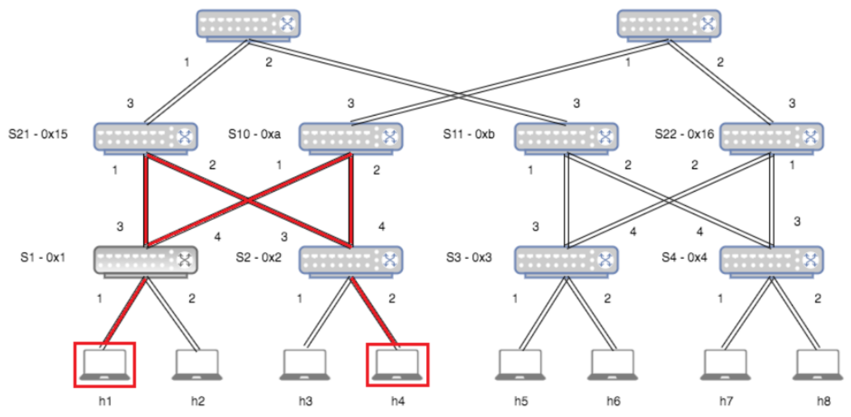

Fig. 4: The selected hosts and possible paths

The network was tested before and after running the proposed algorithm. The testing focused on some QoS parameters such as throughput, delay, jitter, latency, and packet loss between the two servers in the fat-tree network.

Delay and Latency have been measured by sending five Internet Control Message Protocol (ICMP) Echo Request packets to the destination host and calculating the time until ICMP Echo Reply was received at the source host.

Throughput, Jitter, and Packet Loss have been tested using iPerf, first by using TCP and then by using UDP.

5. RESULT

The network was tested ten times using DSCA and EDSCA (proposed method) methods to study their behaviors to find if there is any improvement. The obtained results are shown in the table 2 below.

Table 2: Test Results of the First Scenario.

Test No.	Schemes	TCP	UDP		
		Throughput (Gbits/sec)	Throughput (Mbits/sec)	Jitter (ms)	Packet Loss (%)
1	DSCA	7.12	661	0.031	14%
	EDSCA	12.1	825	0.016	13%
2	DSCA	9.46	706	0.038	21%
	EDSCA	13.2	832	0.022	15%
3	DSCA	9.83	597	0.034	33%
	EDSCA	13.4	835	0.013	16%
4	DSCA	9.09	646	0.016	27%
	EDSCA	12.9	829	0.015	12%
5	DSCA	7.08	565	0.064	23%
	EDSCA	12.5	833	0.019	13%
6	DSCA	10.5	631	0.024	23%
	EDSCA	13.3	832	0.017	11%
7	DSCA	9.83	755	0.068	27%
	EDSCA	13.6	823	0.020	12%
8	DSCA	9.53	657	0.019	17%
	EDSCA	12.8	832	0.013	14%
9	DSCA	9.48	823	0.048	21%
	EDSCA	13.4	880	0.017	15%
10	DSCA	10.7	829	0.062	24%
	EDSCA	13.8	885	0.021	14%

An average performance has been calculated to summarize the previous table, as shown in the table 3 below:

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 10, Issue 4, pp: (24-32), Month: October - December 2022, Available at: www.researchpublish.com

Table 3: Average Test Results

Schemes	TCP	UDP		
	Throughput (Gbits/sec)	Throughput (Mbits/sec)	Jitter (ms)	Packet Loss (%)
DSCA	9.26	687.0	0.040	23.00%
EDSCA	13.1	840.6	0.017	13.50%

Throughput

The simulation results of throughput are shown in fig. 5 and fig. 6. The results show that the proposed approach has better performance. The DSCA scheme experiences the most throughput setback with an average of 9.26 Gbps using TCP and 687 Mbps using UDP. However, EDSCA performed better with a throughput of 13.1 Gbps and 840.6 Mbps using TCP and UDP respectively. Furthermore, the proposed approach has a relatively smaller overhead because it leverages the REST API technology for obtaining topology information. The result suggests that the proposed approach (EDSCA) gives 41% better performance in comparison to the initial approach.

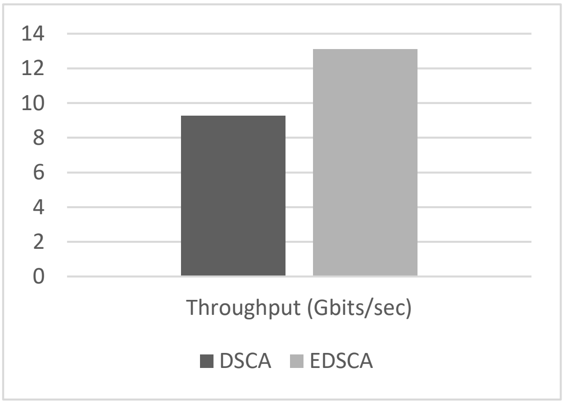

Fig. 5: Throughput – TCP

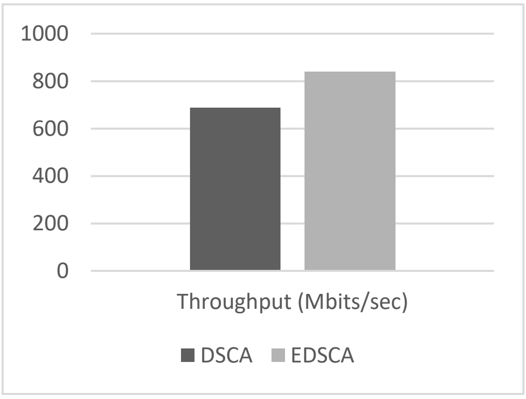

Fig. 6: Throughput - UDP

Jitter

Part of the objective of this work is to minimize the jitter effect during data transmission. The experimental results show that there has been a significant reduction of jitter in the proposed scheme than in the original work. From fig. 7 below, we can see that the proposed method recorded a Jitter of 0.017 ms while DSCA 0.040 ms thereby minimizing the jitter by 57% approximately.

Packet Loss

This section compares the packet loss result of the two methods. Fig. 8 below shows that there is significant decreased in packet loss in the proposed method with a packet loss of 20% for the DSCA and 13.5% for EDSCA. Therefore, the proposed method was able to reduce the packet loss by 32%.

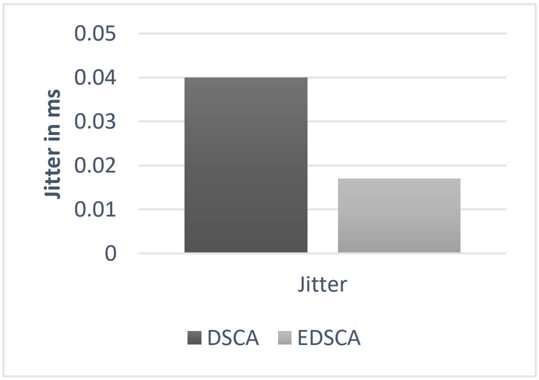

Fig. 7: Jitter

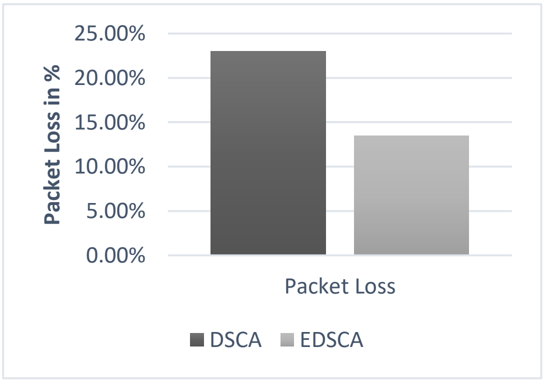

Fig. 8: Packet Loss

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 10, Issue 4, pp: (24-32), Month: October - December 2022, Available at: www.researchpublish.com

Table 4: Latency and Delay Test Results

Schemes	Min	Avg	Max	Mdev	Delay (ms)
DSCA	0.0412	0.1183	1.2244	0.2731	0.3262
EDSCA	0.0426	0.0796	0.3468	0.101	0.1372

Latency

Fig. 9 below shows the simulation result of the Latency of both schemes. From the simulation results, we can see the proposed approach has less end-to-end latency than the original DSCA. The average latency experienced on the fat-tree topology for DSCA and EDSCA was 0.1183 and 0.0796 milliseconds respectively. In conclusion, the proposed method was able to reduce to latency by approximately 33%.

Delay

From fig. 10 below, the simulation results show that the average delay in the proposed scheme has been significantly decreased when compared with the original work with an average of 0.3262 ms and 0.1372 ms for DSCA and the EDSCA method respectively. Therefore, the proposed solution was able to minimize the delay in the transmission of network flows by 58% approximately.

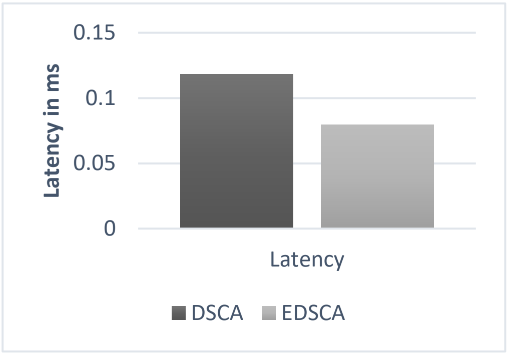

Fig. 9: Average Latency

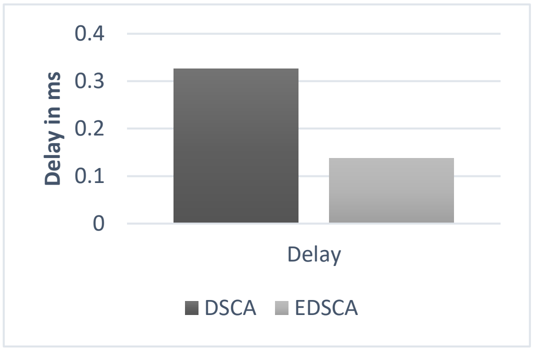

Fig. 10: Delay

6. CONCLUSION

The implementation of a dynamic load balancing algorithm to efficiently distribute fat-tree network flows through multiple alternative paths between a single pair of hosts is described in this study. The network was evaluated both before and after the load balancing algorithm was applied. The focus of this research work is on some QoS parameters such as throughput, delay, latency, jitter, and packet loss between two servers in the fat-tree network. The performance of the enhanced algorithm was evaluated through the existing result of [1], to evaluate the extent to which the maximum network resources utilization was achieved. The results obtained shows that EDSCA was more promising in terms of minimizing end-to-end delay, latency, jitter, packet loss and maximizing throughput.

The results showed that the network performance has increased after running the load balancing algorithm program, the algorithm was able to minimize delay, latency, packet loss, jitter and maximize throughput, thereby improving network utilization.

Recommendation for Future Research

Although the objectives of this research work were achieved, the following suggestions are made:

1. The first suggestion is to investigate the performances of the dynamic load balancing program on a different popular SDN controller, such as Floodlight, Beacon, NOX/POX, RYU etc. and compare the results.

2. 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.

Limitation of the Study

The limitation experienced during this research work is that the connectivity between a node and a host was limited to only 800 Mbps in UDP and 20 Gbps in TCP. This may not necessarily be the case in a Data Center with redundant high capacity links between hosts.

REFERENCES

[1] Hu, T., Yi, P., Guo, Z., Lan, J., & Hu, Y. (2019). Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks. Future Generation Computer Systems, 95, 681–693. https://doi.org/10.1016/j.future.2019.01.010

[2] Lee, P. A., & Anderson, T. (1990). System Structure and Dependability. In Fault Tolerance: Principles and Practice (pp. 11–49). Springer Vienna. https://doi.org/10.1007/978-3-7091-8990-0_2

[3] Da Rocha Fonseca, P. C., & Mota, E. S. (2017). A Survey on Fault Management in Software-Defined Networks. IEEE Communications Surveys and Tutorials, 19(4), 2284–2321. https://doi.org/10.1109/COMST.2017.2719862

[4] Aly, W. H. F. (2017). A Novel Fault Tolerance Mechanism for Software Defined Networking. Proceedings - UKSim-AMSS 11th European Modelling Symposium on Computer Modelling and Simulation, EMS 2017, 0, 233–239. https://doi.org/10.1109/EMS.2017.47

[5] Sidki, L., Ben-Shimol, Y., & Sadovski, A. (2017). Fault tolerant mechanisms for SDN controllers. 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2016, 173–178. https://doi.org/10.1109/NFV-SDN.2016.7919494

[6] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2016). Research challenges for traffic engineering in software defined networks. IEEE Network, 30(3), 52–58. https://doi.org/10.1109/MNET.2016.7474344

[7] D. Dotan, R. Y. P. (2005). HyperFlow: An integrated visual query and dataflow language for end-user information analysis. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'05(in: Proc.), 27–34.

[8] Hassas Yeganeh, S., & Ganjali, Y. (2012). Kandoo: A framework for efficient and scalable offloading of control applications. HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks, 19–24. https://doi.org/10.1145/2342441.2342446

[9] Hu, T., Guo, Z., Baker, T., & Lan, J. (2017). Multi-controller Based Software-Defined Networking : A Survey.

[10] Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., & Boutaba, R. (2013). Dynamic controller provisioning in software defined networks. 2013 9th International Conference on Network and Service Management, CNSM 2013 and Its Three Collocated Workshops - ICQT 2013, SVM 2013 and SETM 2013, i, 18–25. https://doi.org/10.1109/CNSM.2013.6727805

[11] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks, Comput. Netw. Int. J. Comput. Telecommun. *Comput. Netw. Int. J. Comput. Telecommun. Netw.*, *71 (3)*(3), 1–30. https://doi.org/10.1109/MNET.2016.7474344

[12] Hu, Y., Wang, W., Gong, X., Que, X., & Cheng, S. (2014). On reliability-optimized controller placement for Software-Defined Networks. China Communications, 11(2), 38–54. https://doi.org/10.1109/CC.2014.6821736

[13] Muller, L. F., Oliveira, R. R., Luizelli, M. C., Gaspary, L. P., & Barcellos, M. P. (2014). Survivor: An enhanced controller placement strategy for improving SDN survivability. 2014 IEEE Global Communications Conference, GLOBECOM 2014, 1909–1915. https://doi.org/10.1109/GLOCOM.2014.7037087

[14] Song, S., Park, H., Choi, B., Choi, T., & Zhu, H. (2017). for Enhancing Software-Defined Network (SDN) Reliability. 14(2), 302–316.

[15] Gonzalez, A. J., Nencioni, G., Helvik, B. E., & Kamisiński, A. (2016). A fault-tolerant and consistent SDN controller. 2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings. https://doi.org/10.1109/GLOCOM.2016.7841496

[16] Liang, C., Kawashima, R., & Matsuo, H. (2015). Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers. Proceedings - 2014 2nd International Symposium on Computing and Networking, CANDAR 2014, December 2014, 171–177. https://doi.org/10.1109/CANDAR.2014.108

[17] Xie, J., Guo, D., Zhu, X., Ren, B., & Chen, H. (2020). Minimal Fault-Tolerant Coverage of Controllers in IaaS Datacenters. IEEE Transactions on Services Computing, 13(6), 1128–1141. https://doi.org/10.1109/TSC.2017.2753260