2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS) | 979-8-3315-3119-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/COMSNETS63942.2025.10885764

Dynamic Load Balancing on Switches of Software Defined Network Managed by OpenDayLight Controller

Manasa Kulkarni*, Bhargavi Goswami†, Muhammad Furqan†, Joy Paulose*
*CHRIST (Deemed to be University), Bangalore, India
†Queensland University of Technology, Brisbane, Australia

***Abstract*—In recent times, the world is becoming a global village where connectivity is a new norm irrespective of geographical location. Within corporate networks, huge setbacks are faced due to a lack of efficient resource management. Load balancing is inevitable to cater to a reliable, faster, and congestion-free communication experience for exponentially increasing online enterprises. Dynamic network resource management for high performance and low data transmission latency in a network is necessary. The major issue faced by the traditional network is that it relies on static hardware switches. Software-Defined Network approaches paved a way to overcome the limitations of traditional networks. This research proposes the Dynamic Load Balancing Algorithm for Software-Defined Networks to utilize network resources optimally. The major function of the proposed algorithm is to determine alternative paths and further distribute the incoming and outgoing network flows to achieve optimum network resource utilization with faster traffic flow completion. The experiment is performed with the OpenDayLight Controller on the Mininet simulator, which emulates the network with the novel scheme. The results prove that the proposed solution has accomplished the benchmarks of optimum throughput, reduced redundancy, and reduced flow completion time.**

***Index Terms*—Software Defined Networks, OpenDayLight Controller, Fat-Tree Topology, Dynamic Load Balancing, Optimum Resource Utilization, Traffic Management**

I. Introduction

A Keyword for faster technological advancement is software, and its integration with the networking industry leads to emerging technologies such as Software Defined Networks (SDN) and Network Function Virtualization (NFV) [1]. Software Defined Networks has emerged as a novel trend in the past few years that has led to the progression of centralized management of large computing network. Two base communication mechanisms supporting SDN are OpenFlow and Path Computing Element [2]. OpenFlow's Open Networking Foundation (ONF) standard protocol is used to separate the control plane from the switch and provide an interface for communication to the data plane in the controller [3].

The Load Balancing [4] is a way to optimise network performance to increase QoS. The scalable communication networks [5] demonstrated for other controllers like Ryu, FloodLight [6], Onos, Beacon, Mul and Pyretic [7] clearly states the requirement of efficient load balancing strategies.

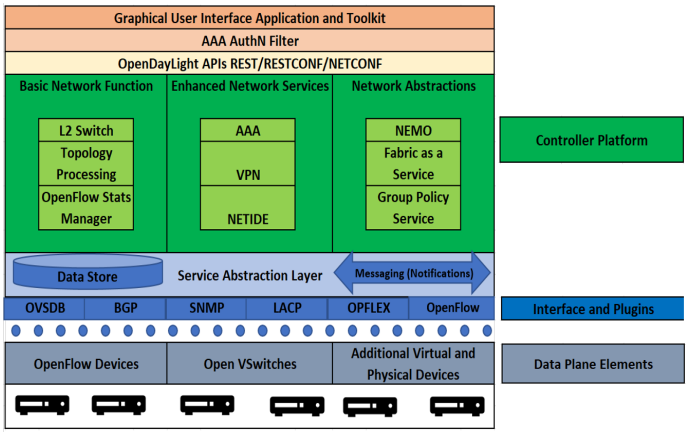

Fig. 1. Layers in SDN with OpenDayLight Controller

Furthermore, load balancing aids in the prediction of the traffic jam before occurrence. The load balancing improves SDN performance with the help of ODL controller and there has been a minimal study conducted in this domain [8], which prompted the authors to investigate further in this area.

The OpenDaylight Project is a modular, extensible, scalable and multi-protocol controller infrastructure built for SDN deployments [9]. Load balancing policy implementation is challenging in ODL based corporate networks. Figure 1 explains the layered architecture of ODL Controller. As we can see in the diagram, the top layer of Application Layer includes- Graphical User Interface Applications, AAA authentication Filter and OpenDayLight API's which is followed by the controller platform having three sections: Basic Network Function, Enhanced Network Services and Network Abstractions. ODL provides a model-driven service that allows users to write applications. Furthermore, it contains internal plugins such as- BGP, LACP, OPFLEX, and OpenFlow. It has data plane elements like: OpenFlow Devices, Physical Devices and Open Virtual Switches to transfer the data. All these features add a power pack for the load balancing concept.

In this paper, to address the requirements of load balancing strategies in ODL controller, DLBA - Dynamic Load Balancing Algorithm is presented focusing on the requirements of the corporate networks. A systematic study is performed on

SDN dynamic load balancing on OpenDayLight Controller. The primary goal of this research is to develop a novel DLBA determining alternate path for parallel transmission to quickly complete the data flows. The solution proposes to distribute network loads efficiently to enhance the network performance for efficient resource utilisation. The mathematical model is presented for the designed DLBA load balancing scheme that dynamically distributes the flows in multi-path environment.

The key contributions of this research include: 1. The dynamic network management of resources is performed by the novel approach of Dynamic Load Balancing Algorithm (DLBA). 2. Proposes a load balancer for SDN-based data center networks which is demonstrated on OpenDayLight Controller. 3. The algorithm of DLBA distributes traffic. 4. It assists the data center operators for choosing second and third shortest path in the events of failures or congestion. The testing is focused in QoS parameters such as: Jitter, Throughput, Latency and Packet Loss before and after DLBA implementation. The experiment is performed to check the correctness of the working of the load balancing DLBA algorithm.

Paper is organised in the following manner. Section II describes the Related Work. Section III is formulating the design of the novel scheme of DLBA proposed. Section IV describes the tools and techniques used for the experiment. Section V describes the methodology to perform the experiment. Section VI discusses the scenarios followed by the discussions on the result in Section VII, Correlation Regression Analysis in Section VIII. Finally, paper concludes the research work which is followed by the future scope and references.

II. Related Work

This section presents the recent trends and approaches in Software Defined Networks and OpenDayLight controller by providing the groundwork of Dynamic Load Balancing scheme of DLBA.

In [10], load-balanced multi-path routing algorithm is proposed which can improvise the distribution of the traffic load and utilize the available bandwidth in the fat tree network. Dynamic load balancing is performed to improve the network performance and reduce network response time such as content caching and intelligent packet forwarding, in the contextof Content Delivery Network (CDN) workflows [11]. Nowadays, SDN significantly simplifies and improves the network management [12]. Author proposes a traffic-aware load balancing scheme for machine-to-machine (M2M) networks using software defined networking (SDN) [13] which is essential for load balancing.

In [14], authors propose a formalized method to describe regular topologies and a regular Topology Description Language (TPDL). This paper also proposes a novel topology-aware routing scheme: cRetor (controller-side Regular Topology Routing scheme). The experimental results show the conventional SDN network with Floodlight controller and DCell. It is believed that cRetor can exploit the potential of data center networks better and improve the efficiency. Even though this paper touches varied area of software-defined networking, there is a major gap on how to implement the concept with the controllers focusing upon aspects like load balancing.

The main motivation behind this study is to figure out the management of network resources for high performance and low latency data transmission. To fill those gaps, authors in this paper designed an algorithm namely Dynamic Load Balancing Algorithm (DLBA) as a solution. This study proposes the implementation of a dynamic load balancing algorithm DLBA with major contributions of a) finding closed loop alternative paths, b) distributing multi-path flow completion scheme for SDN in order and, c) to overcome issues and optimise network resource utilisation.

III. Mathematical Modelling

The aim of this research is to utilize network resources optimally and to reduce the congestion faced by the packets travelling towards destination. Ultimately, the purpose is to improve the throughput and reduce the latency by distributing the network traffic efficiently using Dynamic Load Balancing Algorithm (DLBA).

In the novel Dynamic Load Balancing Algorithm (DLBA) we propose a load balancing mechanism for SDN Architecture using total link cost matrix based variance from Shortest path suggested by Dijkstra's Algorithm [15]. After observing the flow of incoming and outgoing packet communication, alternate partially parallel path is identified using python libraries. After third iteration, data transmission is distributed into multiple paths leading to faster completion of data flow. The promising approach is described mathematically in this section.

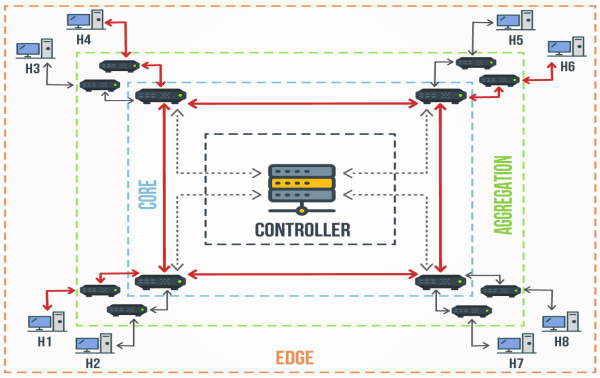

Fig. 2. Path Determination, Mutation and Operational Validity Analysis

For determining the several alternative paths that can be followed, statistical characteristics path mutations have to be analysed, to validate that alternative multiple paths are operational. As shown in the Figure 2, the controller is connected to the core switches, which can be an inter-switch sub-network or intra-switch single network. The aggregation of multi-path identification and validation is performed in this section. The mean square is calculated between the divided multi-path

alternatives using the formula in equation (1) where, PS_b is "inter", the path difference between switches and PS_w is "intra", the path difference within the switch. To determine the difference between the two paths, the degree of freedom is considered represented by df_b and df_w and the Load balancing Factor LB_F is the factor indicating the difference between the original path followed and the alternative paths available to share the load.

$$LB_F = \frac{PS_b}{PS_w} = LB_F(df_b, df_w) \tag{1}$$

$A_1, A_2, ...A_k$ represents different parts in the form of a factor, let n_i be the time under observation at Level A_i. For traffic samples at level A_i, are represented by $T_{i1}, T_{i2}, ..., T_{in}$. Therefore,

$$\overline{T} = \frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{n} {}_iT_{ij}, i = 1, 2, ...k$$

$$N = \sum_{i=1}^{k} n_i \tag{2}$$

Where $\overline{T}$ represents the midpoint of traffic values and the number of groups is k, the observation times is N in equation (2).

If sum of squares for SS_t total, SS_b between switches and SS_w within the switches represented by the equation (3), (4) and (5).

$$SS_t = \sum_{i=1}^{n} \sum_{j=1}^{k} (T_{ij} - \overline{T})^2 \tag{3}$$

$$SS_b = \sum_{j=1}^{k} n_j (\overline{T}_j - \overline{T})^2 \tag{4}$$

$$SS_w = \sum_{i=1}^{n} \sum_{j=1}^{k} (T_{ij} - \overline{T})^2 \tag{5}$$

In the final step, for the computation of LB_F, the derivation of PS_b and PS_w is done by substituting the value of square sum between switches and within switches lead us to equations (6) and (7).

$$PS_b = \frac{SS_b}{df_b}$$

$$= \frac{n_1(\overline{T}_1 - \overline{T})^2 + n_2(\overline{T}_2 - \overline{T})^2 + .. + n_k(\overline{T}_k - \overline{T})^2}{N - 1}$$

$$PS_b = \frac{\sum_{j=1}^{k} n_j (\overline{T}_j - \overline{T})^2}{N - 1} \tag{6}$$

$$PS_w = \frac{SS_w}{df_w}$$

$$= \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2 + ... + (n_k - 1)S_k^2}{N - k}$$

$$PS_w = \frac{\sum_{i=1}^{n} \sum_{j=1}^{k} (T_{ij} - \overline{T})^2}{N - k} \tag{7}$$

The derived values of SS_b and SS_w after dividing with the degree of freedom, resulted in the numeric values in sequel was provided to PS_b intra-switch and PS_w inter-switch communication. The final value of LB_F is obtained by taking a ratio of intra-switch and inter-switch communication that resulted in significant difference in time of completion of flows. Finally, the computed values are compared with the alternative paths existing between source and destination other than the Shortest path provided by Dijkstra's algorithm. The closest value is selected and followed by the flows to distribute and balance the load.

IV. Methodology

The scalability of the ODL controller needs to be checked and experimented along with mininet and OpenFlow switches. The topology is designed and generated using Python. The experimental topology supports networking between 8 servers and 10 switches.

A. Algorithm Description

The DLBA - Dynamic Load Balancing Algorithm is designed to accomplish the dynamic load balancing in the system. The algorithm distributes the incoming and outgoing traffic in order to achieve the best possible link resource utilization. Major steps involved in the load balancing procedure is provided in Algorithm 1. The notations used in Algorithm 1 is provided in Table I.

1. Load the DLBA Load Balancing Model.
2. Obtain the shortest route information using Dijkstra's algorithm [16] [17].
3. Find the total link cost for all the routes.
4. Find the current transmission rate.
5. Select the first, second and third best paths.
6. Push the flows into each switch in the selected paths.
After step 6, the step 2 is executed again to obtain shortest route.

The Load Balancing Algorithm attempts to access route requests from clients to servers. It directs traffic and the major activity of DLBA is to send each request to the nearest server with a link load less than a predetermined threshold. The equation of link load computation is provided in Equation (8). If link load exceeds the threshold for all the servers, the algorithm will still choose the nearest server.

TABLE I
SYMBOL NOTATIONS

SYMBOL	DENOTION
a and b	Nodes that are required for the communication
L(a) and L(b)	Symbolises the weights
T	It stores the shortest path nodes
w(a,b)	Represents the cost between the nodes
$b^{'}$	Adjacent nodes
s	Server
selected	Selected server for the transmission
BalancedPaths	The shortest path selected
writeFlows	Final destination
KPaths	The shortest path choosen initially
LL	Link Load
Host	Hardware device connected to the network
server	shared resources to workstation
Process	Load Balancing Technique
$L(b^{'})$	Balanced weights
assignPaths	The path assigned before the load balancing

Algorithm 1 Load Balancing Algorithm

Input: Source, set of Servers
Output: BalancedPaths

1: Initialisation Process
2: server ← lookupServers;
3: LOOP Process
4: **for** all $b \neq a$ such that L(b) = w(a,b) **do**
5: L(a) = 0 and T = a
6: **while** $T \neq b$ **do**
find $b^{'} \in T, L(b^{'}) \leq L(b)$
7: **end while**
8: **end for**
9: **if** $NotEmpty(server\_list)$ **then**
10: **for** s in server **do**
11: KPaths ← selected.LL where,

$$\text{Link Load: (LL)} = \frac{Current_{Server->sw}^{Traffic}}{MAX_{Server->sw}^{Bandwidth}}$$

12: Host ← select KPaths
13: BalancedPaths ← assignPaths,
14: **end for**
15: writeFlows (BalancedPaths);
16: **end if**

$$\text{Link Load (LL)} = \frac{Current_{Server->sw}^{Traffic}}{MAX_{Server->sw}^{Bandwidth}} \quad (8)$$

where current traffic between server and switches is represented by $Current_{Server->sw}^{Traffic}$ and $MAX_{Server->sw}^{Bandwidth}$ is maximum bandwidth between server and switches.

The main goal is to find the alternative shortest path from the source to destination. Once the links are being calculated, the three best paths are selected depending on the minimum transmission cost and LL given in Equation (8). The selected best paths will have the flow entries to carry out the communication between the host and the destination. Finally, the algorithm updates the information every minute making the DLBA algorithm dynamic.

B. Classification of Network Scenarios

In this section, we describe the two scenarios used in the experiment to test DLBA effect on performance.

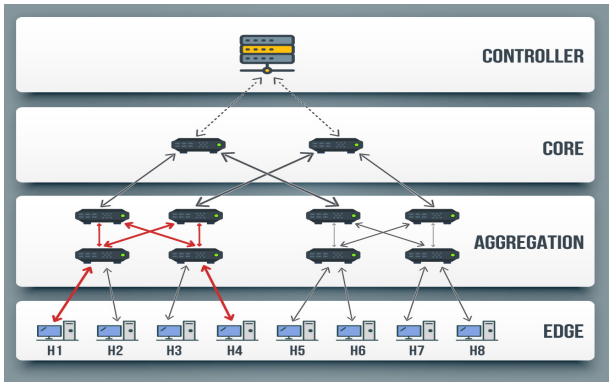

Fig. 3. Flow of packets traversing between Host 1 and Host 4

1) First Scenario: Trying to measure the performance at the Aggregation Layer: In this scenario, we can clearly see in the Figure 3 that, the servers h1 and h4 are selected to perform the load-balancing between them. The testing is focused on QoS parameters between the two servers in the fat-tree network. All these parameters are tested with 10 seconds for each test case.

2) Second Scenario: Trying to measure the performance at the Core layer: In this scenario, we can clearly see in the Figure 4 that, the servers h1 and h6 communicating with each other while balancing the load. The QoS parameters between the two servers in the fat-tree network.

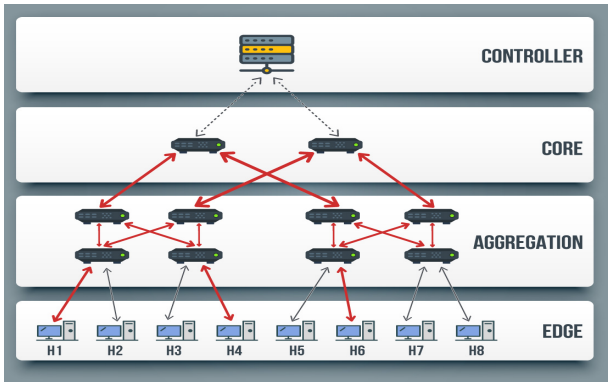

Fig. 4. Flow of packets traversing between Host 1 and Host 6

C. Tools and Techniques

Table II specifies the tools and its configuration used to perform the experiment and validate our novel Load Balancing Scheme. In order to perform the experiment, Mininet is used

as a simulator to provide all the topologies required for the experiment. The Ubuntu 14.04 LTS, has been configured on the VMWare Workstation 15 pro. Iperf is used for the data analysis and entire experiment is performed in OpenDayLight Beryllium. GNUPLOT is used for visualization of results and the topology is generated using Python. Finally, the experiment confirms the outcomes with Wireshark.

TABLE II
SPECIFIED TOOLS ALONG WITH THEIR CONFIGURATION

TOOLS	CONFIGURATION
Operating System	Ubuntu 20.04 LTS
Processor	2 CPUs
Memory	4 GB
Virtual Machine	VM Ware Workstation 15 Pro
Mininet	2.2.0
Iperf	Version 3
OpenDayLight	Beryllium (4th Release)
Gnuplot	5.2 (Stable)
Xterm	Xterm-351
Python	3.7
Wireshark	2.2.0

D. Implementation Overview

This research has established a test-bed which has been implemented under Linux platform. The flow of the implementation is as follows:

1. Setting the Linux Environment (Ubuntu 20.04 LTS).
2. Installing the Java Development Kit (JDK).
3. Downloading and installing the Mininet.
4. Downloading the OpenDayLight platform (stable Beryllium version is being installed for this experiment).
5. Installing the ODL features.
6. Implementing the Load Balancing Model of DLBA using Python libraries and modules.
7. Rebuilding and Run the ODL controller.
8. Running the topology in the Mininet.
9. Testing the network using iPerf.
10. Running the DLBA Dynamic Load Balancing algorithm in background.
11. Testing the performance using iPerf after the Load Balancing.
12. Testing the results performance.

V. RESULTS & DATA ANALYSIS

As we can see in the Table I that the network has shown the better performance in the first scenario after running the load balancing algorithm. The average network Throughput before load balancing was 156.16 Mbytes, and it became 517.11 MBytes after the load balancing.

A. Parameters to analyze the load balancing factor

Results are represented using graphs. Further, the data is analysed to check the correctness and determine whether the algorithm is yielding the desired result or not.

1) Jitter: The figure 5 depicts the calculated Jitter before and after load balancing in two scenarios. As shown in the graph 5, the highest and average peak is found between 60 and 90 seconds. The red and blue lines clearly demonstrate a significant increase in delay, particularly when communication begins. The reason for the increased initial Jitter prior to load balancing is that the congested queued packets are waiting to be transmitted. Once load balancing has been applied, a drastic reduction can be observed in Jitter which improves the overall performance of the network making it behave with stability.

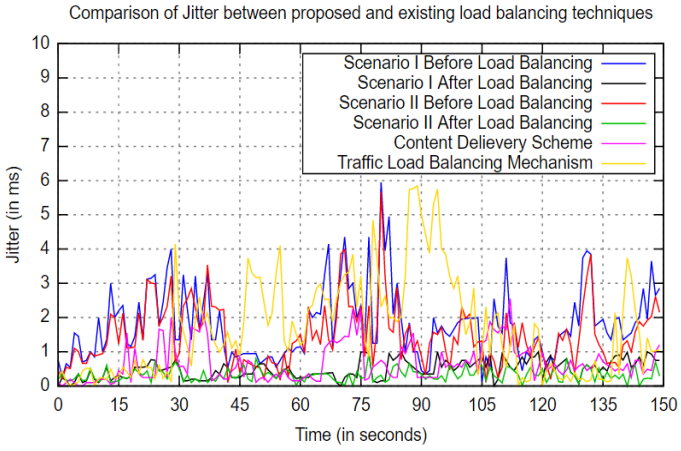

Fig. 5. Computed Jitter before and after load balancing

2) Packet Loss: Figure 6 represents the computed packet loss before and after the load balancing for two scenarios. In the absence of load balancing, it results in degraded network performance. The highest peak is found in between 80 to 140 seconds and this further leads to high level packet loss. On an average there is 57.47% and 53.81% packets lost before load balancing. The black and green lines represents the improvement in packet-loss after implementing load balancing. Overall, black and green lines show that the packet loss stays low and stable throughout the communication.

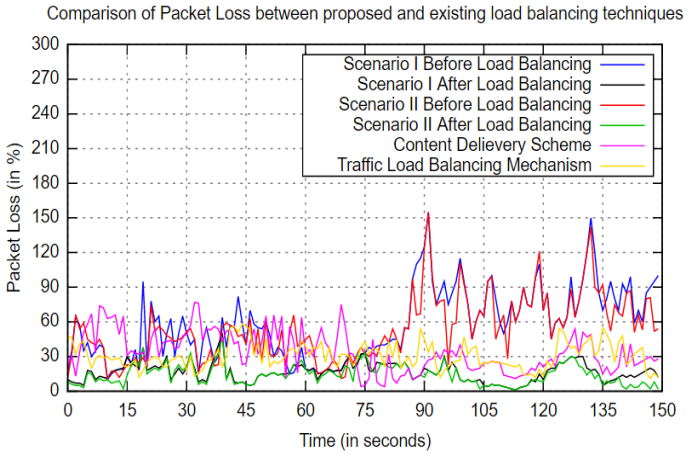

Fig. 6. Computed Packet Loss before and after load balancing

3) Latency: Figure 7 represents the computed Latency before and after the load balancing for the two scenarios. As we observe there is an average peak between 25 to 30 and 75 to 100 seconds with an average of 2.427 and 1.67 milliseconds latency observed before the load balancing. It can be observed that after the load balancing is implemented, latency is reduced by an average of 0.673 and 0.573 milliseconds and it remains low and stable. The low packet loss and the re-transmitted packet may contribute to the reduced latency resulting in improvement of overall network performance.

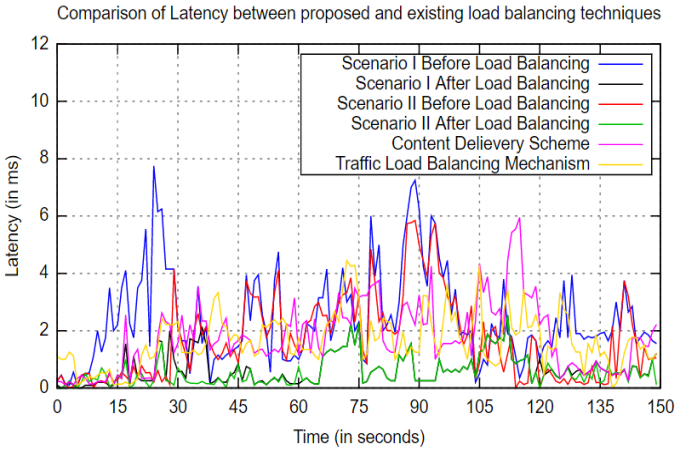

Fig. 7. Computed Latency before and after load balancing

4) Throughput: Figure 8 represents the computed Throughput before and after the load balancing for the two scenarios. The blue and red lines represent the obtained throughput before load balancing and the black and green lines which represent the throughput after load balancing. It can be observed that, the improved stable throughput of 517.11 and 651.233 MBytes is obtained after load balancing is reflected throughout network. This proves that there is a profound influence of load balancing over the network performance parameters.

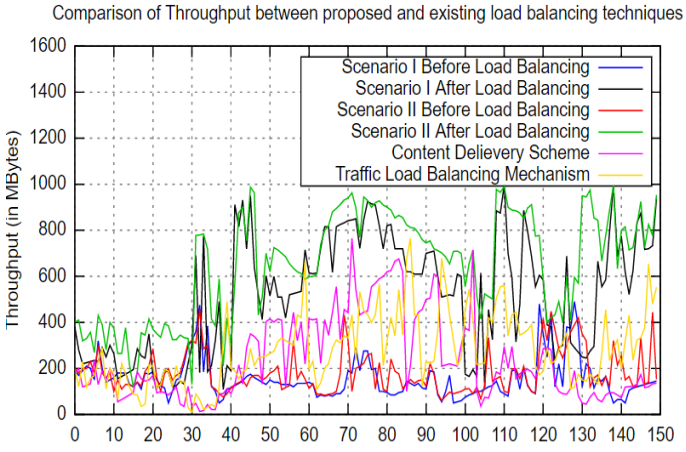

Fig. 8. Computed Throughput before and after load balancing

As provided in the Table III, the average values are computed for the network parameters before and after load balancing for the two different scenarios. It can be observed that there is an significant improvement which shows gain in performance in the second scenario after running the load balance DLBA Algorithm. The average network Throughput before load balancing was 182.97 MBytes, and it became 615.2 MBytes after the DLBA load balancing. The average delay has decreased by 6%. Jitter has decreased by 0.326 ms and the Packet Loss has decreased by 2.3%. Second scenario has shown greater performance under second layer of fat-tree topology, but as the network grows to larger extent.

TABLE III
THE STATISTICAL ANALYSIS OF THE PARAMETERS BEFORE AND AFTER THE DLBA IMPLEMENTATION

Scenario	Scenario I		Scenario II	
Parameters	Before	After	Before	After
Throughput(MBytes)	156.16	517.113	182.973	651.223
Packet Loss(%)	57.473	16.546	53.814	14.193
Latency(ms)	2.427	0.673	1.670	0.573
Jitter(ms)	1.803	0.486	1.477	0.467

VI. PREDICTION USING CORRELATION AND REGRESSION MODEL

A. Correlation

To determine the strength of linear association between the two variables, i.e., Before Load Balancing (BLB) and After Load Balancing (ALB) of DLBA Algorithm, we have used Pearson's correlation coefficient calculated using equation (9) where r is correlation coefficient, x_i is the value of x variable provided in parameters column, $\overline{x}$ is mean of x variable, y_i is the value of y variable for ALB concerning BLB for the scenario I or scenario II and, $\overline{y}$ is the mean of y variable.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{9}$$

The calculated r values indicate that the throughput increases after applying load balancing. A positive relationship is observed for Latency, Jitter, and Packet-loss parameters for Scenario I performance, but it differs for Scenario II. We can keep that once the packet-loss decreases, the throughput increases. The detailed records can be observed in Table IV. Overall, the results have the highest latency impact over the performance in both Scenario I and Scenario II.

TABLE IV
CORRELATION MODEL TO DETERMINE EFFECT OF DLBA LOAD BALANCING

Parameters	Correlation Coefficient (r)	
	Scenario I	Scenario II
Throughput	-0.164710296	-0.014812267
Latency	0.149807835	0.160056674
Jitter	0.037395709	0.077130892
Packet Loss	0.037395709	-0.142180127

TABLE V
REGRESSION MODEL TO PREDICT PERFORMANCE OF LOAD BALANCING ALGORITHM

f & p	Scenario I	Scenario II
Significance F	0.022263617	0.000019708
P_Latency	0.040452067	0.000016590
P_Jitter	0.071069313	0.163830694
P_Packet Loss	0.274574744	0.158268679

B. Regression

The purpose of this section is to predict the performance of communication networks based on the simulation results and further derive the relationship between a dependent (throughput) and independent variables (latency, jitter, and packet loss). In this section, we derive the multiple regression using equation (10) where Y_i is dependent variable throughput to determine the network's performance based on independent variables X_i, here, Latency, Jitter, and Packet Loss are the matrix under observation. Additionally, f is a function with 150 observations, and e_i is the error rate.

$$Y_i = f(X_i, \beta) + e_i \tag{10}$$

The obtained result of our regression model is provided in Table V which observes the impact of Latency, Jitter, and Packet Loss on the Throughput matrix. The most significant values of F indicating the probability of regression outcome are provided, along with the P-value indicating the impact of the stated parameters on the performance.

After observing the obtained values of F from our Regression Model, our analysis is highly significant for Scenario I and II after DLBA was applied. The P value of latency and jitter shows a powerful impact on the network's performance in both scenarios. Based on the derived regression model, we predict that the network's performance will be consistently improving in the future, and the proposed DLBA Load Balancing Model is trustworthy.

VII. CONCLUSION

The authors conclude by achieving enhanced network performance by implementing a profound load balancing technique named DLBA - Dynamic Load Balancing Algorithm over OpenDayLight controlled SDN - software-defined network. This research describes the implementation of the dynamic load balancing algorithm to distribute the flow of fat-tree networks efficiently. This network is tested before and after running the load-balanced algorithm(DBLA). Our focus was on various testing parameters such as; time delay, packet loss, throughput, and improved network utilization. This research creates a base of load balancing experimentation for the SDN research community, inspiring more researchers to contribute. With this research, load balancing can improve throughput up to 20 percent, reduce the packet loss to 4.8%, and reduce jitter and latency values four times. These results inspire the network industry to implement load balancing for every congested networking architecture to optimize resource utilization. The demand for networking has been increasing multi-fold in recent years.

VIII. FUTURE SCOPE

Researchers plan to investigate the dynamic load balancing on other stellar SDN controllers such as; Floodlight, Ryu, and Beacon and compare and contrast their features, results, and limitations. We plan to investigate various topologies other than fat-tree topology and understand if there is any performance variation after implementing our load balancing algorithm of DLBA. Finally, extending the investigation to compare our SDN-based load balancing model with the load balancing mechanisms of traditional networks.

REFERENCES

[1] W.-L. Chin, H.-A. Ko, N.-W. Chen, P.-W. Chen, and T. Jiang, "Securing nfv/sdn iot using vnfs over a compute-intensive hardware resource in nfvi," *IEEE Network*, vol. 37, no. 6, pp. 248–254, 2023.

[2] E. Kosmatos, R. Casellas, K. Nikolaou, L. Nadal, D. Uzunidis, C. Matrakidis, J. M. Fabrega, M. S. Moreolo, and A. Stavdas, "Sdn-enabled path computation element for autonomous multi-band optical transport networks," *Journal of Optical Communications and Networking*, vol. 15, no. 11, pp. F48–F62, 2023.

[3] S. Asadollahi, B. Goswami, A. S. Raoufy, and H. G. J. Domingos, "Scalability of software defined network on floodlight controller using ofnet," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, 2017, pp. 1–5.

[4] M. D. Tache, O. Păscuțoiu, and E. Borcoci, "Optimization algorithms in sdn: Routing, load balancing, and delay optimization," *Applied Sciences*, vol. 14, no. 14, p. 5967, 2024.

[5] D. Lunagariya and B. Goswami, "A comparative performance analysis of stellar sdn controllers using emulators," in *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, 2021, pp. 1–9.

[6] G. B. Khan, Mohammed and S. Asadollahi, "Data visualization of sdn during load balancing experiment using floodlight controller," in *Data Visualization*. Springer, Singapore, 2020, pp. 161–179.

[7] M. Kulkarni, B. Goswami, and J. Paulose, "Experimenting with scalability of software defined networks using pyretic and frenetic," in *International Conference on Computing Science, Communication and Security*. Springer, 2021, pp. 168–192.

[8] A. Shirvar and B. Goswami, "Performance comparison of software-defined network controllers," in *2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, 2021, pp. 1–13.

[9] S. Badotra and S. N. Panda, "Evaluation and comparison of opendaylight and open networking operating system in software-defined networking," *Cluster Computing*, vol. 23, no. 2, pp. 1281–1291, 2020.

[10] T. Badageri, B. Hamdaoui, and R. Langar, "Load-balanced multipath routing through software-defined networking," in *2024 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2024, pp. 1068–1073.

[11] H. Yang, H. Pan, and L. Ma, "A review on software defined content delivery network: a novel combination of cdn and sdn," *IEEE Access*, vol. 11, pp. 43 822–43 843, 2023.

[12] U. Zakia and H. B. Yedder, "Dynamic load balancing in sdn-based data center networks," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2017, pp. 242–247.

[13] P. Baniya, P. Nand, and B. Bhushan, "Dynamic load balancing schemes for software-defined networking (sdn)," in *International Conference on Intelligence Science*. Springer, 2023, pp. 493–506.

[14] Z. Jia, Y. Sun, Q. Liu, S. Dai, and C. Liu, "cretor: An sdn-based routing scheme for data centers with regular topologies," *IEEE Access*, vol. 8, pp. 116 866–116 880, 2020.

[15] A. Choudhary, S. S. Kang, and S. Singla, "Multi-objective dijkstra algorithm for enhancing qos in sdn through balanced routing," in *2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)*, vol. 1. IEEE, 2024, pp. 1169–1175.

[16] J.-R. Jiang, W. Yahya, and M. T. Ananta, "Load balancing and multicasting using the extended dijkstra's algorithm in software defined networking," in *ICS*, 2014.
[17] A. BinSahaq, T. Sheltami, A. Mahmoud, and N. Nasser, "Fast and efficient algorithm for delay-sensitive qos provisioning in sdn networks," *Wireless Networks*, vol. 30, no. 5, pp. 4607–4628, 2024.