Chapter 28
Dynamic Load Balancing in Software-Defined Networks Using Machine Learning

Kunal Rupani, Nikhil Punjabi, Mohnish Shamdasani and Sheetal Chaudhari

1 Introduction

In the SDN architecture, the data plane and the control plane are separated. The SDN architecture facilitates network virtualization. It is directly programmable, scalable and allows integration of services like load balancing, Firewall and Intrusion Detection System (IDS). The SDN architecture consists of three layers namely the application layer, the control layer and the data layer. The application layer consists of applications like load balancer, traffic monitoring, etc. The control layer is where the SDN controller resides. The data layer is responsible for forwarding the data from source to destination. In SDN, there has been limited research proposed on network load balancing. In previously proposed projects, the controller gets information from OpenFlow switches to analyze load on each link and accordingly modify the flow-tables by using a particular load balancing strategy. Since a routing plan which is dynamic and a static load balancing method is proposed in these strategies, these strategies fail to take advantage of SDN and do not make an efficient load balancing model. These algorithms work on SDN having multiple paths, but the strategy for routing is determined by considering the load condition of next-hop only while the property of global view in SDN is not leveraged. Therefore, such kind of strategies cannot determine the effective path in real time and hence they cannot achieve an ideal load balancing effect. To summarize, current research in load balancing in SDN reveals that the existing algorithms are too simple for a complex problem like load balancing and so they exhibit poor performance. It is also understood that in these algorithms data gathering is not done to the fullest due to which the desired accuracy is not achieved. The objective of the paper is to propose a system that load balances an SDN-based network in real time in order to make data transmission in SDN more efficient and reliable.

K. Rupani (✉) · N. Punjabi · M. Shamdasani · S. Chaudhari
Sardar Patel Institute of Technology, Mumbai, India
e-mail: krupani8@gmail.com

© Springer Nature Singapore Pte Ltd. 2020

S. Bhalla et al. (eds.), *Proceeding of International Conference on Computational Science and Applications*, Algorithms for Intelligent Systems,
https://doi.org/10.1007/978-981-15-0790-8_28

2 Related Work

In [2], the back-propagation artificial neural network is trained to find the integrated load condition of the network. Flow rules indicating the best path are then pushed into the switches by the controller. However, this technique lacks performance and does not consider node utilization as a part of load balancing a network. In [3], the algorithm can balance link load in a network quickly to resolve some congested path. Also minimized packet loss is achieved by changing paths of flows. But this technique consumes more time and does not consider node utilization. In [4], using the fuzzy synthetic evaluation model (FSEM) the paths can be dynamically adjusted by taking advantage of the global view of the network. However, this technique is not reliable as there is packet loss due to the time taken to detect link failure. In [5], as the load balancing algorithm keeps running, its performance improves over time but its initial performance is found to be low. The algorithm first finds the shortest path and then checks for link utilization. In [6], the fuzzy synthetic evaluation algorithm with dynamic weight (FSEADW) is used which supports dynamic weights to dynamically realize network status in real time to achieve better load balancing. However, this technique too ignores the overall network utilization as it does not consider node utilization.

3 Proposed System

The proposed system consists of two subsystems namely the simulation subsystem and the machine learning subsystem. The two subsystems are interconnected using the Django REST Framework using which path features are sent over from the simulation subsystem to the machine learning subsystem in JSON format using the HTTP protocol. Figure 1 shows the network architecture diagram for the proposed SDN load balancing system. The proposed system has minimal processor and memory requirements. Sending data in the form of JSON is again efficient as it requires less bandwidth.

3.1 Simulation

Mininet simulation software is used to simulate a DCN-based fat-tree topology network containing 8 hosts and then extended for network containing 16 hosts. The topology with 8 hosts is described. The hosts are labeled from h1 to h8. They are connected using switches which are arranged in the form of a tree structure. The topologies are created beforehand in python using the NetworkX library [7]. The Mininet simulator makes use of Floodlight Controller which acts as the SDN controller. The Mininet software provides support for python in the form of APIs. Using

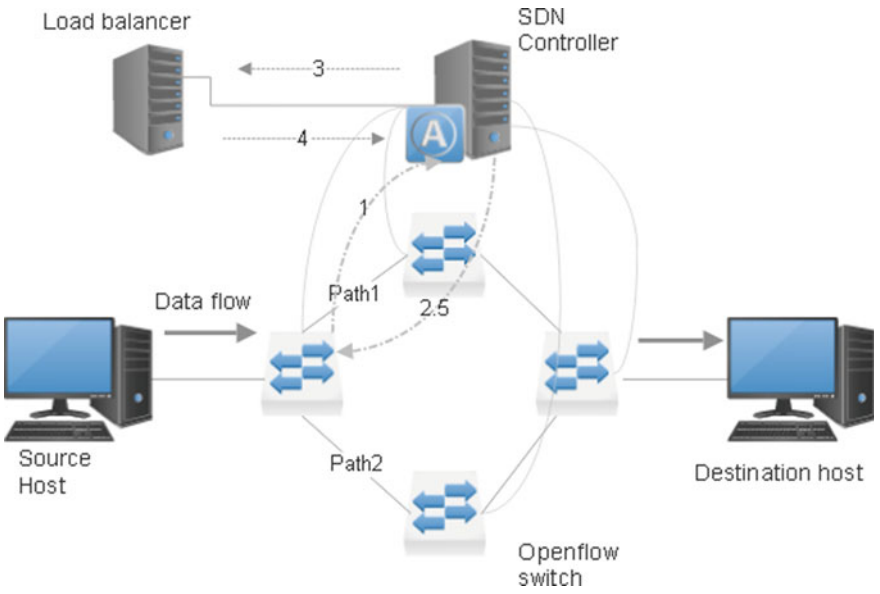


Fig. 1 Network architecture for proposed system

these predefined APIs, path features are extracted for the machine learning subsystem. Wireshark [8] is used to monitor the traffic while Mininet is running in parallel.

3.2 Artificial Neural Network

This subsystem is used to find the minimum loaded path between a source node and destination node in real time. Sequential model from keras library is used to train the dataset obtained from the simulation subsystem. The structure of the back-propagation artificial neural network is shown in Fig. 2. The predictors, i.e., the inputs are BW Ratio, latency, packet loss rate, the number of hops and node utilization from source to destination. These inputs form the input layer. Figure 2 shows that the hidden layer has three neurons. This number is varied from three to eleven using the formula given in (1) specified in [2] and the most accurate model is chosen. In (1), N is the number of neurons in the hidden layer, the number of neurons in the input layer is denoted by m, the number of neurons in the output layer is denoted by n and a is a constant between 1 and 10. The output layer gives the integrated load on the input path depending upon the path features. The activation function used is the popular ReLU activation function as it overcomes the problem of gradient descent. The learning rate is also adjusted to get an accurate model.

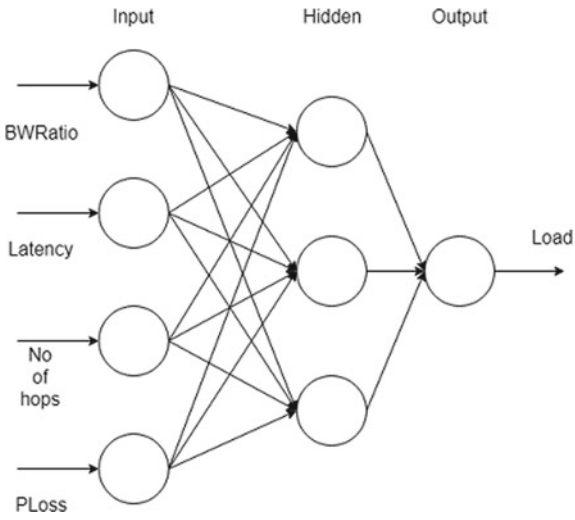


Fig. 2 Artificial neural network structure with three neurons in the hidden layer

The number of epochs value is set to 100 so that the mean squared error at the end of training is very close to zero. Also, initially, random weights are assigned. Setting the network parameters correctly is a must to get the desired accuracy of the model.

$$N^2 = m + n + a \tag{1}$$

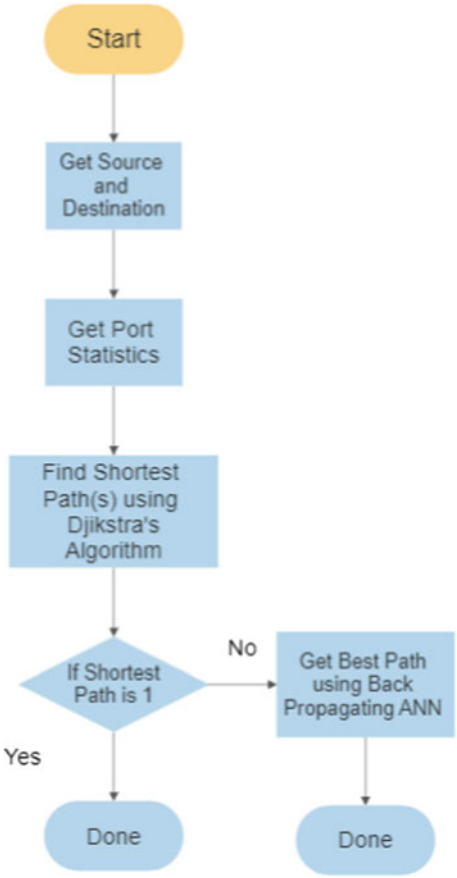
4 Methodology

SDN controller has the property of global view of network that it possesses at any stage of the simulation. All the paths from one node to every other node are found by updating the topology information of the network. By leveraging the property of global view in SDN, the effective load condition of every path can be determined easily. The System Flow Diagram is shown in Fig. 3.

4.1 Network Simulation

Network simulation is first done for data collection and then for final testing. For testing the system, running the Floodlight Controller is the first step of network simulation. Running the topology using Mininet is the next step. Then the load balancer script is executed. Here, the source host is entered by the user. Then the least loaded host is selected as the destination. The selected hosts are activated and

Fig. 3 System flow diagram



traffic is generated between them. Wireshark [8] is used to monitor the traffic between these hosts. Port statistics are obtained using the in-built APIs of Mininet.

Then using Dijkstra's algorithm, shortest path(s) is/are determined between the source and destination. If multiple shortest paths are found, then the machine learning module comes into picture otherwise the shortest path obtained becomes the best path itself. To overcome link failure, the first step is to detect link failure and the other step is to choose an alternate path. Link failure is detected by the SDN controller when a switch is unable to send a packet-in message to the controller within a predefined timeout period [9]. The SDN controller then informs the load balancer about the link failure. The load balancer temporarily removes all the paths containing the failed link and finds the least loaded path as usual. Node failure is overcome by detecting the node failure and finding the backup path while the SDN controller performs failure recovery as specified in [10]. Node failure is detected by the SDN controller when the node stops sending packet-in message to the controller within a predefined timeout

period [9]. The load balancer utilizes the remaining backup paths for finding the best path between the source host and the destination host.

4.2 Data Collection and Preprocessing

The training dataset is obtained by running the Mininet simulator repeatedly after an interval of three minutes and recording all the path information using the APIs. The port statistics obtained are then used to calculate the path features using the formulae shown in the equation below. This method ensures that there is variety in the dataset which is usually the case in real-world networks. Such variety also ensures that the model is accurate. Such collection of data is only possible because of the global view architecture of SDN. This training data collected from the simulation subsystem is stored in the form of an excel sheet. This training dataset is first loaded into the program and then examined for errors. After cleaning, the data comes scaling of data after which the training data is ready. The testing data is obtained in real time in JSON format using Django. This data is converted into proper format after which it is ready for testing.

1. BWRatio: BWRatio is the subtraction of cumulative transmitted bytes $B(t)$ and previous cumulative transmitted bytes $B(T-1)$ divided by the maximum bandwidth BWRatio (MAX).

$$\text{BWRatio} = B(T) - B(T-1)/\text{BWRatio(MAX)} \tag{2}$$

2. Packet Loss Rate (P Loss): It is the ratio of the number of packets not received and the number of packets transmitted. Packet(T) is the number of packets transmitted while Packet(R) is the number of packets received.

$$P\,\text{Loss} = \text{Packet}(T) - \text{Packet}(R)/\text{Packet}(T) \tag{3}$$

3. Transmission Latency: It is the ratio of bytes transmitted Byte and the transmission rate (trRate).

$$\text{Latency} = \text{Byte}/\text{trRate} \tag{4}$$

4. Total Node Utilization: It is the sum of node utilization values of all the switches along the path.
5. Transmission hops: This value is directly obtained by using the predefined API for hops.

4.3 Training and Testing Neural Network Model

The back-propagation artificial neural network is trained on the training data for 100 epochs to get the desired accuracy. Training is hardly time-consuming as the dataset has about 600 records collected at different times and the training time of a real-time system is less valuable than the testing time, i.e., the predicting time. The testing time needs to be negligible which is the case here. The model is evaluated with 'Mean Squared Error' as the loss function while 'Mean Absolute Error' is used as a metric to test the accuracy of the model. Now the shortest paths with their features are used to predict the integrated load on each one of them.

5 Results

5.1 Training Results of BPANN

The result after training the model is a graph of mean absolute error versus the number of epochs. The graph in Fig. 4 is for topology with 8 hosts. In this graph, the mean absolute error at the end of training the model with $n = 7$ is found to be closest to zero which means that the model is most accurate when $n = 7$. So, the value of n is set to 7. The graph in Fig. 5 is for topology with 16 hosts. In this graph, the mean absolute error at the end of training the model with $n = 9$ is found to be closest to

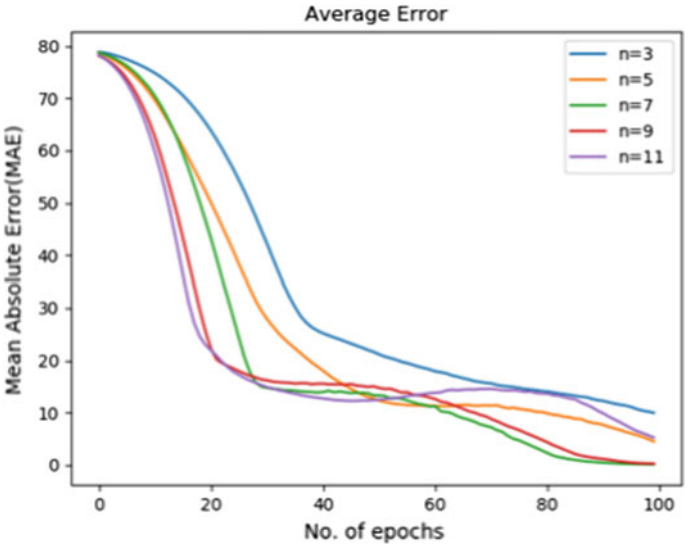


Fig. 4 Mean absolute error for topology with 8 hosts

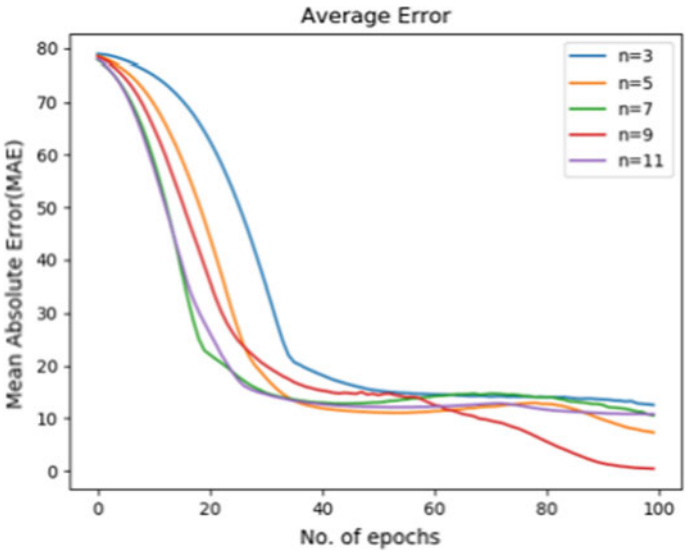


Fig. 5 Mean absolute error for topology with 16 hosts

zero which means that the model is most accurate when $n = 9$. So, the value of n is set to 9.

5.2 Comparing Latency Before and After Load Balancing

The iperf tool [11] is used to get the network statistics for analysis. The iperf command gives the throughput while the ping command is used to measure the latency. Figure 6 depicts the final result in the form of latency and throughput recorded before and after load balancing. In Table 1, the average latency between h1 and h8 decreases and the throughput increases. In Table 2, the average latency between h1 and h4 decreases and the throughput increases. Similar results are noted for a different pair of hosts in both topologies. This means that the load balancing is working.

6 Conclusion

Dynamic load balancing is achieved by predicting the effective load on all the shortest paths and selecting the path with minimum load in real time. The output of the system is nothing but this least loaded path. The model is trained successfully on topologies with 8 and 16 hosts. This means that the proposed system is scalable. The system can also handle link failure and node failure. This means that the system is reliable. The

Table 1: Results for Topology with 16 hosts recorded between h1 and h8			
Before Load Balancing		After Load Balancing	
Average Latency (ms)	Throughput (Gbits/sec)	Average Latency (ms)	Throughput (Gbits/sec)
0.2	3	0.15	3.67

Table 2: Results for Topology with 8 hosts recorded between h1 and h4			
Before Load Balancing		After Load Balancing	
Average Latency (ms)	Throughput (Gbits/sec)	Latency (ms)	Throughput (Gbits/sec)
0.159	3.96	0.152	3.98

Fig. 6 Result tables

proposed system can be used as a load balancing module in an SDN-based system to improve its performance.

References

1. http://mininet.org/
2. Chen-xiao C, Ya-bin X (2016) Research on load balance method in SDN. Int J Grid Distrib Comput 9(1):25–36. http://dx.doi.org/10.14257/ijgdc.2016.9.1.03
3. Lan Y-L, Wang K, Hsu Y-H (2016) Dynamic load-balanced path optimization in SDN-based data center networks. In: 2016 10th international symposium on communication systems, networks and digital signal processing (CSNDSP). https://doi.org/10.1109/csndsp.2016.7573945
4. Li J, Chang X, Ren Y, Zhang Z, Wang G (2014) An effective path load balancing mechanism based on SDN. In: 2014 IEEE 13th international conference on trust, security and privacy in computing and communications. https://doi.org/10.1109/trustcom.2014.67
5. Zakia U, Ben Yedder H (2017) Dynamic load balancing in SDN-based data center networks. In: 2017 8th IEEE annual information technology, electronics and mobile communication conference (IEMCON). https://doi.org/10.1109/iemcon.2017.8117206
6. Wang T, Guo X, Song M, Peng Y (2017) A fuzzy synthetic evaluation algorithm with dynamic weight for SDN. In: 2017 IEEE 2nd information technology, networking, electronic and automation control conference (ITNEC). https://doi.org/10.1109/itnec.2017.8284896
7. https://networkx.github.io/
8. https://www.wireshark.org/
9. Xu H, Yan L, Xing H, Cui Y, Li S (2017) Link failure detection in software defined networks: an active feedback mechanism. Electron Lett 53(11):722724. https://doi.org/10.1049/el.2017.082

10. Zhang S, Wang Y, He Q, Yu J, Guo S (2016) Backup-resource based failure recovery approach in SDN data plane. In: 2016 18th Asia-Pacific network operations and management symposium (APNOMS). https://doi.org/10.1109/apnoms.2016.7737211
11. https://iperf.fr/
12. https://keras.io/getting-started/sequential-model-guide/