Abstract—In this paper a novel algorithm is proposed for the 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 our algorithm, we 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%, reduce the loss by 88.2%, increase the received data size by 20.8%. In addition, the proposed algorithm acts as a congestion control algorithm and a new load balancing algorithm.

Keywords—Software Defined Network; Datacenter; POX controller; Fat-Tree; Mininet; Load Balancing.

I. INTRODUCTION

Computer networks and Datacenters (DC) have been developed in a rapid pattern. New techniques and methods are required for monitoring and at the same time, managing these kinds of networks is crucial. SDN rose as a new approach that can accomplish this task [1]. For DCs it is so important to increase system's throughput and reduce the delay time since they hosts several cloud applications that are delay sensitive and require high throughput [2]. There are two approaches to deploy this; either by hardware or software. The first approach refers to the DC network topology. Recently there are several network topologies that are used in DC and one of them is the Fat-Tree topology [3].

These topologies are built with a certain manner that contains several Equal Cost Multi Path (ECMP) between any sources - destination pair. ECMP provides better throughput than single route method, in addition the network would results in less latency [4]. Nevertheless, the change of the network topology may be an expensive solution; in addition, it is not an efficient mechanism to deploy. The latter approach utilizes software by mean of controlling the routing protocols. The majority of the existing routing protocols are implemented to find one path for any source-destination pair (if there is no link failure, otherwise it will select another path). This leads to a considerable underutilization for the ECMP supported by the Fat-Tree topology [5]. Therefore, we propose a Software Defined Network (SDN) based load balancing algorithm to solve this problem. All communications in the SDN based network were captured by a centralized controller. The information then are analysed in term of deploying a load balancing facility on the existing infrastructure by means of improving the scheduling of dataflow in the network. This could only be happened by taking the advantage of having a global view by the SDN controller [6]. We will apply the proposed load balancer algorithm on a Fat-Tree topology [3] which has an ECMP property. We implemented an algorithm for dynamically balancing the load by means of re-routing based on information from SDN controller. Whereas, when the network performance decreases, other available paths are checked via the SDN controller. We implemented the topology and tested the proposed algorithm using Mininet network emulator a GUI called MiniEdite [7, 8 and 9]. We compared the results of the throughput and loss with and without the load balancing algorithm. The results clearly show how the proposed load balancing algorithm improves network performance[10]. The rest of this paper is organized as follows; Section II is a comparison between traditional and SDN based load balancing technique. Section III explains the proposed algorithm. Section IV presents the obtained results and discusses them. Section V concludes the paper.

II. Traditional vs. SDN based load balancing

A. Traditional load balancing

The traditional load balancing technique has a dedicated hardware, load balancer to manage clients’ traffic with different servers. Figure 1 shows a simple model of typical load balancing system. As shown in the figure, the clients are connected to the load balancing server that is already connected to the gateway switch. When the load balancer receives a request from any client, it forwards it to a server that has information about the available resources so that it can serve the request. Hence, the load balancer should keep
track of every single session. In addition, it should perform a Network Address Translation (NAT) and Domain Name System (DNS) functions [11]. The disadvantage of having a dedicated hardware for the purpose of load balancing is that it considered as an expensive solution and not a flexible approach. In addition, is undergoes from the problem of having single point of failure and leads to bottleneck for whole system [11, 12].

III. PROPOSED ALGORITHM

In this section, we will describe the network architecture (topology) and the proposed algorithm.

A. Architecture

Mininet network emulator is used to implement our virtual datacenter. As shown in Figure 3, a fat-tree topology is utilized to emulate a datacenter. Fat-tree topology built with k-port switches, consists of many k-pods, each pod has two layers, the upper one that is the aggregation and the lower one that is the edge. This k-pods is connected to (k/2)2 core switches, in addition there are (k/2)2 equal cost paths between any source-destination pairs. In our topology, we utilized k=4. The main advantage of a fat-tree topology is a large number of available paths between any pair [5].

This means that in fat-tree topology, there are several interconnections between different layers which are greater than the traditional tree topology. This alternative path provides more paths for transmission of the traffic [6].

B. Algorithm Description

The proposed algorithm is added to the POX controller of the SDN based datacenter. When the bandwidth utilization goes under the threshold value and/or the loss reaches a specific value, then the algorithm will start and balance the load by re-route some traffic to another path with lower bandwidth utilization [13]. Figure 4 is a flow chart that explains how the algorithm works and the steps that are taken to improve the network performance.
C. **SDN based Load Balancer Implementation**

Taking advantage from the global view of SDN based programmed POX controller the proposed algorithm would be capable of conducting monitoring and scheduling tasks. The algorithm is depicted in figure 4 below.

- **Monitoring phase**: In fat-tree topology, there are multiple paths from a source to the destination. The controller will send a request to all the switches in the network periodically, up on analysing the reply from those request, the controller will be able to determine the best alternative path to the destination. This conclusion could be utilized by the second phase (scheduling phase). The monitoring phase adds an overhead, but it’s minimal, because the size of each request is 8 bytes only, while the size of replay is 104 bytes [14].

- **Scheduling phase**: By exploiting the global view provided by the SDN based POX controller. The proposed algorithm suggests transmission of periodic request by the controller to obtain a detailed information about the network status and parameters.

When the specific parameter (Qos metrics) reaches a threshold value for instance the throughput drop to a specific value, the latency or/and loss reaches a defined value, then load balancing function will be initiated. Using information gathered in the first phase (monitoring phase), the controller will change the flow-table of switches and send the traffic over multi paths instead of a single path based on the best available paths. This is conducted by sending a flow-mod message to all the switches in the new (added) path, and update the flow-table information to rout the incoming packets to the destination.

D. **Evaluation**

This section covers the explanation of the points related to the simulation environment, traffic distribution, and then the outcomes will be explained.

1- **Environment**

All experiments were conduct using HP Pavilion (HEWLETT-PACKARD) PC, with the processor Intel Core (TM)i7-4500U CPU @ 2.4GHz, and with an 8 GB RAM , which runs a 64-bit Windows 8 operating system and a Virtual Box Oracle VM version 5.0.10 r104061. The guest OS installed in the VM was Linux OS Ubuntu 14.04 32bit with 1GB RAM. Mininet 2.1 emulator was installed on this VM, with POX 2.0 controller. The aforementioned system is utilized to emulate a datacenter with a fat-tree topology and k=4, which has 20 switches and 16 hosts. The capacity of all the links between switches was (10Mbps and link with 0 loss).While the link capacity between hosts and corresponding switches was (20Mbps), to prevent them from being a bottleneck [15].

2- **Traffic Distribution**

To generate a traffic that can validate the proposed scheme and measure network parameters, we utilized Iperf that is an open source code. Iperf is utilized for evaluation purpose and traffic generation (which can work with TCP and UDP) [16]. Several research ([2], [12], [13] and [17]) that emulate datacenters, test their algorithms with a load-free environment. To make our simulation closer to real datacenter environment and more reliable, we generated an average network load of (15 %) of the total link capacity . This means that each host in the network communicates with other host by sending a traffic rate of 1.5Mbps. At the client side, traffic is generated by means of UDP traffic, while at the receiver side, the incoming data size, throughput, jitter, and the loss every 1 second are recorded for a time period 60 seconds. In addition to make the results more reliable, small packets and relatively small link bandwidth are utilized, because the performance of Open Virtual Switch (OVS) and OpenFlow controller created by Mininet is effected by underlying OS, available processor and the allocated memory [6, 9 and 18].
3- Test Scenario

We built a DC based on SDN architecture, as shown in Figure 3. We assume our datacenter is a YouTube datacenter, and our servers contain video data. In our test utilized 720p video format as test items. To make our results more reliable, we generated an overall network load to so that our DCN is not free of load at any time. This was conducted by sending 15% of the link capacity, in other words, each host sends traffic with 1.5Mbps throughput from one host to another. To make our results more reliable, we generated an overall network load to so that our DCN is not free of load at any time. This was conducted by sending 15% of the link capacity, in other words, each host sends traffic with 1.5Mbps throughput from one host to another. To make our results more reliable, we generated an overall network load to so that our DCN is not free of load at any time. This was conducted by sending 15% of the link capacity, in other words, each host sends traffic with 1.5Mbps throughput from one host to another.

Table 1 , shows which hosts pairs are responsible for generating the overall network traffic (load).

Then the simulated videos (720p, with standard frame rate 24, 25 or 30) are sent with a bitrate equal to 5Mbps from (H2) and (H3) to (H14). At this time, the incoming data parameters are recorded such data size, throughput, jitter, and loss at the receiver node (H14). Then, another host (H4) joins the network and start to send data to H14 (with 5Mbps). As an initial condition, the traffic from all the three hosts use path1 which is depicted in blue color in Figure 3. This traffic will overload the path and will lead to congesting the network. This will consequently leads to packet loss. When the throughput drops to 2.5Mbps that is the minimum throughput allowed to stream a 720p video [19] and/or the loss reach 20% then the proposed algorithm will move one of the host’s traffic to the least loaded path. It is important to mention that threshold values of throughput and loss could be configured with different values based on the application requirements.

<table>
<thead>
<tr>
<th>Source host</th>
<th>Destination host</th>
</tr>
</thead>
<tbody>
<tr>
<td>H10</td>
<td>H1</td>
</tr>
<tr>
<td>H11</td>
<td>H6</td>
</tr>
<tr>
<td>H7</td>
<td>H12</td>
</tr>
<tr>
<td>H13</td>
<td>H8</td>
</tr>
<tr>
<td>H9</td>
<td>H16</td>
</tr>
<tr>
<td>H15</td>
<td>H5</td>
</tr>
</tbody>
</table>

IV. RESULTS

In this experiment, the algorithm changes the path of (H4) to path2, and from our results (Figures 5, 6 and 7), it is noticed how the received data size, throughput, and loss are improved when the algorithm takes action and initiate a load balancer (and a congestion control).

From the trend above, we can observe that at the beginning of the experiment; both (H2) and (H3) hosts had an average throughput of 4.1 Mbps. Then after 20 seconds, (H4) joined the network and start transmitting. Obviously, we could see the throughput of the hosts decreased gradually. According to the global view information collected by the POX controller, the algorithm automatically tries to find another path to route one of the host’s traffic. In this test the algorithm re-route the traffic from (H4) to path2 based on the best alternative path that is selected by the proposed algorithm. Therefore, the throughput is improved as it is shown in Figure 5. In addition, packet loss and received data are improved as shown in Figures 6 and 7 respectively.

The results evidently showed that the proposed Software Defined Network based load balancing algorithm can improve the performance of datacenters, and it works as a congestion control, as well as the load balancing technique.
The aforementioned results were obtained when the algorithm was applied on a datacenter. Figure 8, 9 and 10 show the results of same scenario without applying the proposed algorithm. When the network performance is compared with and without using the proposed algorithm, the proposed algorithm as it is depicted in Table II shows a superiority over the traditional algorithm. The proposed algorithm increases the throughput by a ratio of 21.923% to 36.94% and reduces the loss by 88.2% to 90.4%. Finally, it improves the transferred data size by 20.82% to 35.64%.

![Throughput VS Time](image1)

**Figure 8.** Throughput VS time (without algorithm)

![Losses Ratio](image2)

**Figure 9.** loss during the test (with algorithm)

![Data Received from hosts](image3)

**Figure 10.** data size received from each host (without algorithm)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Host2</th>
<th>Host3</th>
<th>Host4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>21.923%</td>
<td>23.94%</td>
<td>36.94%</td>
</tr>
<tr>
<td>Loss percentage</td>
<td>88.2%</td>
<td>90.4%</td>
<td>85.25%</td>
</tr>
<tr>
<td>Data size received</td>
<td>20.82%</td>
<td>22.24%</td>
<td>35.64%</td>
</tr>
</tbody>
</table>

V. Conclusion

Software Defined Network raised as a new paradigm that lead the developments of networking management. Its main aim is to convert the traditional and static way of network configuration into a dynamical and programmable way. Those characteristics help datacenters to utilize their resources in an intelligent and efficient way. In addition, they would be more flexible and dynamic for virtualization and for hosting, managing and running cloud applications. This study put a spot light on the SDN based datacenters implementation using Mininet emulator and how the network could be managed utilizing the capabilities of SDN POX controller. A datacenter with Fat-Tree topology was built and a novel algorithm for conducting a load balancing using SDN was proposed. Then the proposed algorithm was evaluated using Mininet emulator (that provide ECMP), then we proposed a novel algorithm. Results show that the proposed algorithm improves the throughput by a ratio of 21.923% to 36.94% compared to when the proposed algorithm is not used. The results also show that the loss rate is reduced 88.2% to 90.4 %, moreover, the received data size (in each tested host) was increased by 20.82% to 35.64%. The main idea of proposed algorithm was load balancing through re-routting the traffic from original path to alternative path when the throughput decreased to specific value and/or the loss reaches the threshold value. Figure 5 and 6 showed that, at the beginning ( in first 20 seconds) of our test, the throughput of (H2 and H3) had an average value equal to (4.1Mbps) and in the same time period the loss was (0%) for both hosts. Then a new host (H4) joined the network and start transmitting. We note that the throughput of (H2 and H3) dropped to ( 2.8 Mbps) and (2.4 Mbps) respectively ; also the loss after that increased to (16%) for (H2) and (20%) H3. This drop in network performance initiated our algorithm, which firstly start the Monitoring phase that search for alternative available path, if it find one the Scheduling phase re-direct some of the traffic (traffic of one of hosts) to the alternative path. From same figures we could note that after the algorithm started, the throughput of all host increased and reached (4.1Mbps) for (H2), (4.12Mbps) for (H3) and (4.23Mbps) for (H4); also the loss decreased to (0%) for all hosts. Finally we conclude that the novel proposed SDN based algorithm work as a load balancing technique as well as congestion control (because it eliminate and control the congestion in network links). Also our results showed that this algorithm improve the overall network performance.
REFERENCES


