OpenDaylight vs. Floodlight: Comparative Analysis of a Load Balancing Algorithm for Software Defined Networking

Jonier Porras Duque, Daniel Ducuara Beltrán and Gustavo Puerto Leguizamón

Faculty of Engineering, Universidad Distrital Francisco José de Caldas, Colombia

***Abstract*:** This paper presents the proposal of a load balancing algorithm implemented in two of the most popular controllers for Software Defined Networks (SDN): OpenDaylight and Floodlight. A comparative study in terms of the available bandwidth and delay time of the packet forwarding was performed by means of simulation modeling in a base network in which a shortest path algorithm was implemented as well. The results show that the proposed load balancing algorithm improves significantly the performance of a SDN in terms of the offered QoS of an OpenDaylight based controller. The effect of the proposed load balancing algorithm in the Floodlight controller shows a smaller impact mainly on the bandwidth allocation due to its in-build modules that by default perform specific routing and forwarding operations efficiently according to the traffic demand.

***Keywords*:** Software Defined Networking, Load Balancing, Floodlight, OpenDaylight, MININET.

1. Introduction

Software defined Networking (SDN) is described as a dynamic, manageable, adaptable and cost-effective network architecture suitable to cope with the transport of high bandwidth demands. SDN aims at the creation of networks in which the control is detached from the hardware and is given to a software application called controller. This fact allows achieving simpler, programmable, flexible, more scalable and automatable networks as well as greater security and reliability are acquired due to the centralized control [1, 2, 3]. In the context of the Quality of Service (QoS), load balancing is emerging as an important feature for future communication networks due to its versatility and constant improvements in communication and information systems. From the network's viewpoint, it provides scalability and easy management to TCP/IP, web, proxy, Virtual Private Networks (VPN) and multimedia services. Load balancing allows the use of the existing parallel paths between input and output nodes to distribute the information flows that are transmitted in a network where its underlying contribution is to reduce the congestion through routing and traffic control according to the existing resources in the backbone. The fact of combining all these characteristics can generate new models and structures that support the appropriate and balanced distribution of traffic with assurances of QoS, thus obtaining the most optimal paths to destinations [4].

Load balancing distributes IP traffic to multiple copies or instances of TCP/IP services, each one running on a host within the farm (or cluster, if it is a server farm running a web application) of servers. Transparent partitions of client requests are made through the hosts and clients are allowed to access the farm using one or more virtual IP addresses. From the client's point of view, the farm seems to be a single server which responds to their requests. If traffic increases, the network administrator simply connects another server to the farm [5].

Most of the problems presented in current networks that prevent achieving proper load balancing are related to the routing algorithm itself. Nowadays, the routing is based on the shortest path algorithm in which each packet that enters the network looks for the path with the least number of hops that allows it to reach the destination and it is usually the same for all packets, even if there are other paths with a higher number of hops but much faster. This fact degrades operational aspects of the network such as: the congestion that occurs over the shortest route link, the overuse of some links while others are not often used and the reduction of the effective throughput of the network [4].

This paper evaluates the performance of the SDN controllers Floodlight [6] and Opendaylight [7] through the implementation of a load balancing algorithm that allows obtaining the shortest and/or lowest load paths for the transmission and forwarding of packets among the end devices of the network. In section II, the methodology and elements necessary to carry out the performance evaluation of SDN with the load balancing algorithm are presented. In section III, the obtained results are exposed as well as the analysis and observations found in this study. Finally, section IV summarizes the conclusions of this work.

2. Related work

Most recent research works regarding software defined networking are focused on important aspects of network performance such as load balancing, security, QoS, energy efficiency and traffic optimization. Real-time programmability through the centralized controller has become a great chance of enhancing and optimizing services offered in data center networking and cloud environments, campus and high speed networks, wireless communications and residential environments aiming at making better end user experience [8].

2.1 Data centers and environments

Some requirements necessary when operating at large scales in data centers environments are optimal traffic engineering, network control, and policy implementation. High levels of latency, faults, and prolonged troubleshooting may cause not only a negative end user experience but also significant cost

penalties for operators. Through SDN is possible to monitor and manage a great deal of network devices and services ensuring an effective usage of resources provided for operators. Google, for instance, has implemented SDN technology to connect its geographically dispersed data centers around the globe allowing increased resilience and manageability [8], [9]. Likewise, Cloud computing has also integrated SDN based traffic-engineering solutions to increase service scalability and provide an automated network. Microsoft implemented in their public cloud a load balancing solution based in SDN Ananta, a layer 4 load balancer employing commodity hardware to provide multitenant cloud management. This deployment has been quite important for Microsoft Azure public cloud, since it has allowed obtaining high throughput for several tenants allocated a single public IP address [8], [10]. Reducing energy consumption in data centers has also had a significant enhancement and has allowed cutting down operational costs for operators. SDN technologies such as ElasticTree allow network-wide power management by putting out redundant switches from the controller side during low traffic demand [8], [11].

2.2 Campus and high speed networks

Enterprise networks may show a great deal of variability in traffic patterns requiring proactive management to adjust network policies and fine tuning performance using a programmable SDN framework. A centralized control plane may also aid in effective monitoring and utilization of network resources for readjustment. An additional benefit may be to eliminate middle boxes providing services such as NAT, firewalls, access control, service differentiation solutions, and load balancers [8], [12]. The integration of heterogeneous networking technologies using OpenFlow enabled network elements and a centralized controller has seen a great deal of applicability in optical networking. Using centralized real-time programmability, SDN enabled hardware from multiple vendors and optical packet based as well as circuit-switched networks can be placed under the SDN controller. For instance, in [13] a Wavelength Selective Switching (WSS) facilitated by OpenFlow protocol was demonstrated, and virtual Ethernet interfaces to demonstrate OpenFlow based wavelength path controlling in optical networking is described in [14]. A commodity SDN controller, such as NOX and POX, can operate the optical light paths using OpenFlow by mapping virtual Ethernet interfaces to physical ports of an optical cross-connect node. The evaluation of network performance metrics included latency of path setup and verification of routing and wavelength assignment allocation using dynamic node control provided promising results for future software defined optical networking (SODN) [8], [15]. In contrast with typical distributed GMPLS protocol, SDON uses a unified control protocol for QoS metrics offers greater capacity and performance optimization in optical burst switching. The application of SDN and in particular OpenFlow based controls in high-speed and campus networking, therefore, continues to grow resulting in new as well as hybrid solutions to achieve greater network programmability [8], [16].

2.3 Wireless communications

Due to the real-time programmability and potential to seamlessly introduce new services and applications to consumers, the SDN paradigm has been ported to mobile communication networks. A programmable wireless data plane offering flexible physical and MAC address based routing, in contrast layer 3 logical address based traffic forwarding, has allowed developers to fine tune mobile communications performance [17], [18]. Using the control plane, user traffic can be segregated and routed over different protocols such as WiMAX, 3GPP, and LTE advanced [8].

Current cellular technologies are relatively inflexible by limitations in link capacities, making real-time service provisioning difficult and prone to errors. The redesigning of cellular networks using SDN principles adds modularity to the existing infrastructure, with each layer encapsulating horizontally chained protocol stacks orchestrated by a network operating system residing at the top [19]. For example, with cellular SDN (CSDN) is possible to take advantage of network function virtualization (NFV) to optimize the control through contextual analyses of user data to create intelligent traffic forwarding policies [8], [20].

2.4 Residential environments

Software defined networking has also been considered as an efficient way to manage residential and small office networks. To relieve the burden of network management on residential gateways, the creation of virtual residential gateway (data plane) using software defined networking controller at the service provider side to remotely allow management flexibility innovative service delivery in homes was presented in [21].

The residential router or gateway may be controlled and managed remotely via an SDN controller at the service provider premises, with the latter mainly responsible for fine tuning and troubleshooting the residential network [21, 22, 23]. Incorporation of the SDN in residential networks offers improved scalability and privacy for network management. From a security perspective it has been argued that an anomaly detection system in a programmable residential SDN provides more accuracy and higher scalability than intrusion detection systems deployed at Internet service provider side [8], [24].

3. Materials and Methods

The principle of load balancing to the packet transmission management in software defined networking is similar to that implemented in traditional networks. In the case of SDN networks, the controller is responsible for carrying out the process of selecting the most appropriate route through the routing protocols considering the load of the network links.

For a network topology as shown in Figure I, in order to obtain the shortest path through Dijkstra's algorithm, each node v of the graph G(V,E) has an associated label L(v), this label indicates the smallest known distance to go from a fixed node u to this node. Initially, if the edge exists, then the value of L(v) corresponds to the weight w(u,v) of the edge

joining the nodes u and v where L(v)=∞. Otherwise, if the distance is unknown, the algorithm works by creating a set of nodes T⊂V, for which the shortest path from the node u to each one of them has been obtained up to that point. At the end of the algorithm, L(v) contains the cost of the shortest path to go from u to v [25], [26].

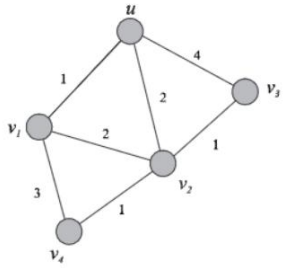

Figure 1. Network scenario [25].

In this work the shortest path based on the Dijkstra's algorithm for the load balancing implementation was programmed in Python where a new node in the list T is added for each iteration of the algorithm. This is achieved by choosing a node v' that does not yet belong to T and that has a minimum tag L(v'). In other words, the node v' closest to u and external to the T list is chosen. Once this is done, the labels of the nodes over which v' influences are updated, so that a new calculation of the distances from u to these nodes is made and this node v' is added to T. The process is repeated until all the nodes of the graph are in the list. The pseudo code is shown as follows [25]:

Algorithm 1 Shortest path using Dijkstra algorithm

```
for all v ≠ u such that L(v) = w(u,v) do
    L(u) = 0
    T = u
    while T ≠ V do
        Find v' ∉ T such that ∀ v∉ T, L(v') ≤ L(v)
        T = T ∪ v'
        for all v∉ T such that v' is adjacent to v do
            if L(v) > L(v') + w(v',v) then
                L(v) = L(v') + w(v',v)
            end if
        end for
    end while
end for
```

In this context, u and v are the nodes that require communication. u is the initial node and v is the destination node. The labels L(u) and L(v) symbolize the weights of each link that represent the distance between a pair of nodes. The set T stores the nodes that will form the shortest route from an origin node u to a destination node v. w(u,v) represents the weight or cost between the nodes u and v.

Initially, when the algorithm evaluates the first node of the network, the distance must be equal to 0 (L(u)=0) and therefore the first node evaluated must be assigned to T (T=u). Then, the algorithm evaluates that while T does not have all the nodes of the network (while not all the nodes have been evaluated to find the shortest path from u to each node v of the network), it must analyses a neighbour node v' which does not belong to the set T and its distance or weight must be less than the neighbour node v previously evaluated. v and v' are adjacent nodes that connect directly to the previous node. The algorithm analyses which one of these two paths have a lower cost to form the link, so if L(v') ≤ L(v) the set T adds the new node v'. The next step is to analyse that for every node v that does not belong to T and that is adjacent to v', it must be added to the cost of the total link L(v), the sum of the distance L(v') and the weight w(v',v) which corresponds to the connection between v and v'. This should be done whenever L(v) > L(v') + w(v',v), that is, if the new analysed link has a cost lower than L(v) (the lower distance obtained previously), in this way L(v) is updated with the new node that forms one route with a lower cost. This algorithm must be repeated until all possible routes between the nodes of the network have been evaluated [25].

In order to evaluate the performance of the Dijkstra's algorithm, a customized network comprised of many possible routes that allow the packets to take different paths was created in the environment of the Mininet software, where its multiple tools for bandwidth, delay times, and traffic measurements, among others were used. Thus, it is intended to observe how the algorithm finds the shortest path taking into account that the network was previously loaded by the controller.

For the sake of the comparative study, the SDN controllers Floodlight and OpenDaylight operate over the same network topology and under the same conditions were implemented. The proposed load balancing algorithm improves the performance of a former proposal in terms of compatibility and suitability to work with the new versions of the SDN controllers. In this context, measurements of the available bandwidth and response times of ICMP requests were performed for a scenario in which the load balancing algorithm was used and also when it was not used, in such a way that the improvements can be appreciated. In addition, the selection of the shortest and/or least loaded paths will be shown. The flowgraph of the load balancing algorithm is shown in Figure 2.

First of all, a pair of hosts must be defined for the load balancing algorithm, these hosts will communicate to each other. Then, a neighbor device near Host 2 will serve as a reference in the network for knowing where is located Host 2. Subsequently, the algorithm will carry out a mapping of the network to find out and extract information regarding IP address, MAC address and ports of all hosts and switches into the network. Then, it will evaluate what routes are shorter and with the lower load between Host 1 and Host 2 and it will select one of them for the communication. This last process is performed through Dijkstra's algorithm mentioned and explained before. Finally, the controller will save the new rule with information about the shortest path and with the lower load in a routing table, meanwhile the algorithm keeps on seeking the best routes for the communication taking into account the current network performance. Because of its transversal operation, the proposed load balancing algorithm could also be used in environments as those presented in [27] and [28] in order to improve the QoS features.

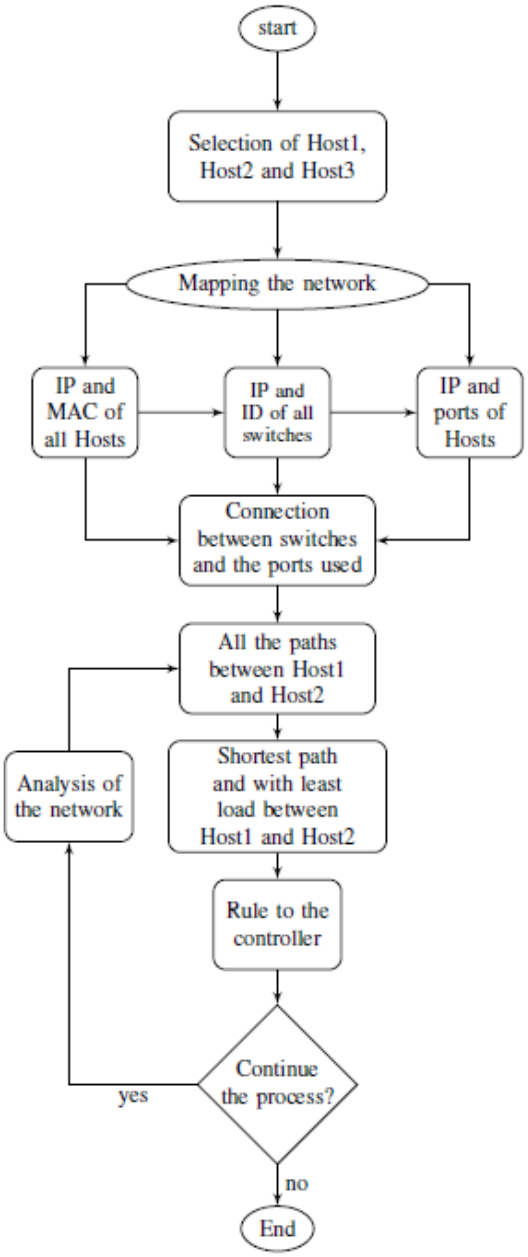

Figure 2. Flow diagram of the proposed load balancing algorithm.

4. Results and Analysis

This section describes the outcomes obtained by the OpenDaylight and Floodlight controllers in the context of the shortest path and load balancing algorithms described in the previous section. It is worth to point out that both controllers are operating over the same network topology.

4.1 Results for OpenDaylight controller

Figure 3 shows the network topology in which the tests with the OpenDaylight controller were performed. Bandwidth measurements between end devices h1 and h12 (gray circles) were initially chosen. Results show that the average values are 0.377 Gbit/s for the network without the load balancing module and 32.6 Gbit/s when applying the load balancing algorithm, this results in roughly an improvement of 86 times the initial bandwidth. Table 1 summarizes the results of three different tests performed.

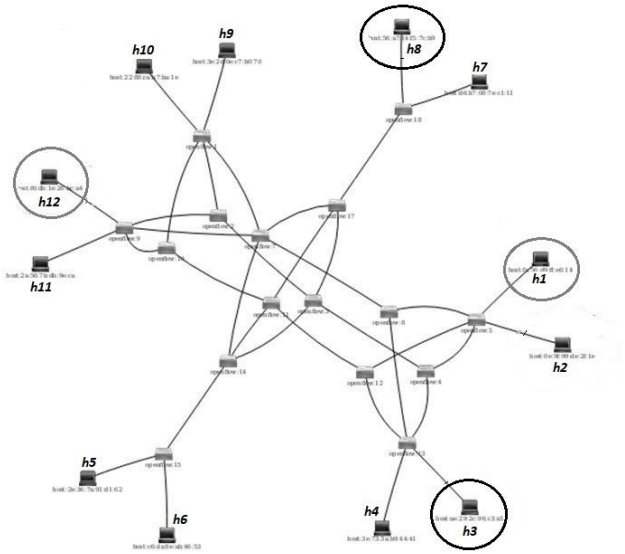

Figure 3. Network from Opendaylight Web API

Table 1. Bandwidth between h1 and h12

Without load balancing	With load balancing
0,376 Gbit/s	33,6 Gbit/s
0,369 Gbit/s	31,9 Gbit/s
0,386 Gbit/s	32,3 Gbit/s

Likewise, the bandwidth between the hosts h3 and h8 (black circles in Figure 3) was evaluated. The obtained average values obtained from three different measurements are shown in Table 2. The results indicate that without using the module of load balancing the bandwidth is 0.227 Gbit/s whereas 32.766 Gbit/s for the network using the load balancing algorithm is feasible. That is, the bandwidth had an improvement of 144.34 times.

Table 2. Bandwidth between h3 and h8

Without load balancing	With load balancing
0,226 Gbit/s	33,5 Gbit/s
0,237 Gbit/s	32,6 Gbit/s
0,219 Gbit/s	32,2 Gbit/s

Subsequently, the packet delivery delay time was evaluated by sending 100 ping requests between each pair of the previously defined selected hosts. As seen in Figure 4, the minimum delay time obtained between the hosts h1 and h12 without load balancing was 0.253 ms (black trace) and whereas with load balancing (grey trace) the obtained value was 0.031 ms. As far as the maximum obtained values is concerned, the absence of load balancing imposes a delay time of 0.649 ms (black trace) whereas 0.369 ms was found when using the load balancing algorithm (grey trace). Overall, the average delay time of the ping requests was 0.361 ms without load balancing and roughly 0.065 ms with load balancing. The measured average delay times indicate that the load balancing algorithm reduces the time needed to select the most optimal path in approximately 5.55 times.

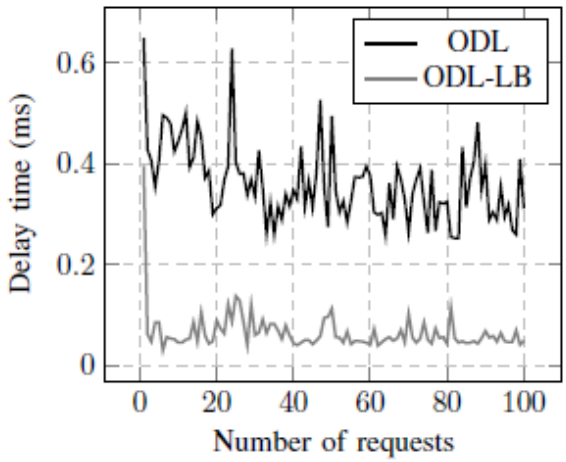

Figure 4. Delay time for 100 ping requests between h1 and h12.

As far as the delay time between the hosts h3 and h8 is concerned, the Figure 5 shows that the minimum delay time obtained without load balancing was 0.176 ms (black trace), with load balancing (grey trace) the obtained value was 0.041 ms. Figure 5 also shows that the maximum values in the absence of load balancing imposes a delay time of 1.199 ms (black trace) whereas 0.538 ms was obtained when using the load balancing algorithm (grey trace). As a result, the average delay time of the ping requests was 0.330 ms without load balancing and 0.067 ms with load balancing. Thus, the average time of the ping requests was reduced 4.93 times when the load balance module was used for the most optimal path selection.

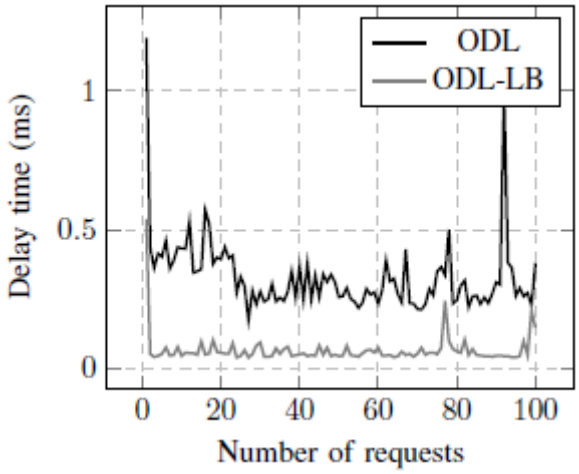

Figure 5. Delay time for 100 ping requests between h3 and h8.

The considerable differences in the time statistics between the test with and without the load balancing algorithm, are mainly due to the fact that the OpenDaylight controller does not have default modules that allow selecting the most suitable path between two points, instead, the controller floods the network with packets and then selects the right path. In the same way, the bandwidth is assigned to each one of the possible paths used for communication such that each path have only a fraction of the maximum bandwidth and not the maximum.

In general, the results obtained with the implementation of the algorithm for load balancing in a network controlled with OpenDaylight, show a great improvement in the management of bandwidth when selecting the shortest path and/or with the lowest load for the pair of hosts. The increase in bandwidth was between 80 and 150 times greater.

In the same way, the results obtained with the implementation of the load balancing algorithm in a network controlled with OpenDaylight, show a significant improvement by reducing the time of ping requests between two ends of the network when selecting the shortest path and/or with less load. The reduction of the times was 5.55 and 4.93 times in each of the hots pairs respectively.

4.1.1 Shortest path analysis and load balancing

Figure 6 shows the shortest and least loaded path that was obtained between devices h1 and 12 (grey color) and devices h3 and h8 (black color). As can be seen, the algorithm selected the shortest and/or with the least load between each pair of hosts. For the pair of hosts h1 and h12 there is no shorter path alternative than the one chosen by the algorithm. In the case of hosts h3 and h8, one of the shorter paths was chosen, since there were three paths that had the same number of hops. In this case the algorithm selected the path with the lowest load out of the three.

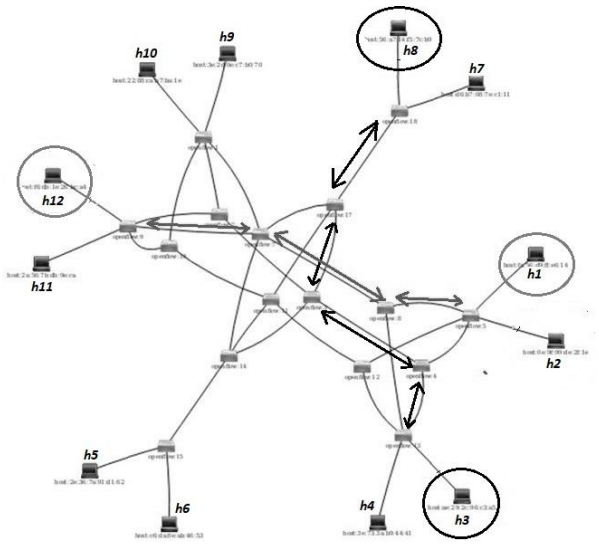

Figure 6. Shortest path.

Subsequently, in order to demonstrate the load balancing in the network, hosts h4 and h8 are selected and applied the load balancing algorithm. The path defined for this pair of hosts is in black color in Figure 7 Then, between these two devices, ping requests are made permanently while load balancing is running between a different pair of hosts, h3 and h7. The algorithm determines the most appropriate path for these two hosts, which may be the shortest and/or least loaded. The resulting path must share the least number of switches belonging to the previously path selected for h4 and h8 (the path obtained is marked with grey color in the Figure 7). Finally, devices h1 and h8 are selected while pairs h4-h8 and h3-h7 communicate permanently. The algorithm is applied and the most appropriate path is obtained (shortest and/or least loaded path), which shares the least number of switches belonging to the paths selected for the previous pairs of hosts.

By making ping requests between each pair of hosts, improvements in the delay time can be reached because the

network load is balanced. The average time was reduced approximately 5 times in all pairs of hosts, considerably improving the behavior of the network. Thanks to the load balancing algorithm, the packet transmission rate between different points of the network was improved, especially in large networks where there were many paths through which packets can be routed.

It should be noted that normally the load balancing modules in a network do not always work in the most appropriate way. This is due to the fact that the most optimal links are not always chosen, since these algorithms are only based on obtaining the shortest path. Thus, they can choose the path with the least number of hops but with an excessive load.

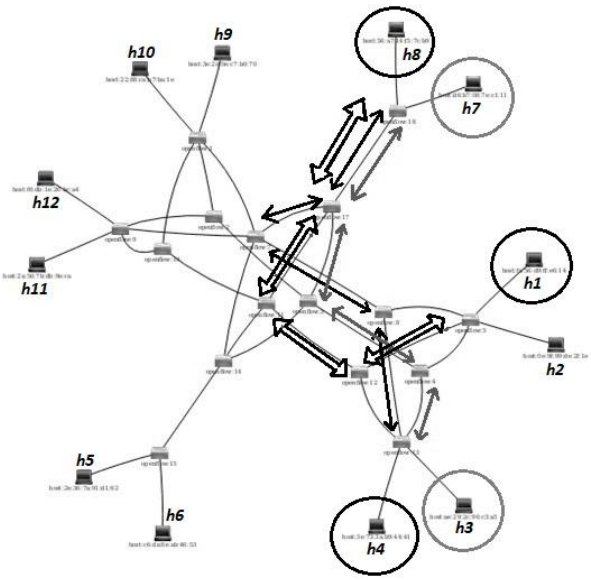

Figure 7. Network load balancing.

4.2 Results for Floodlight controller

The evaluation of the shortest path with the Floodlight controller was performed in the network topology shown in figure 8 with hosts h1 and h12 (black circle) and hosts h5 and h10 (grey color). It should be pointed out that the network topology is exactly the same as that used for the evaluation of the OpenDaylight controller in the previous section.

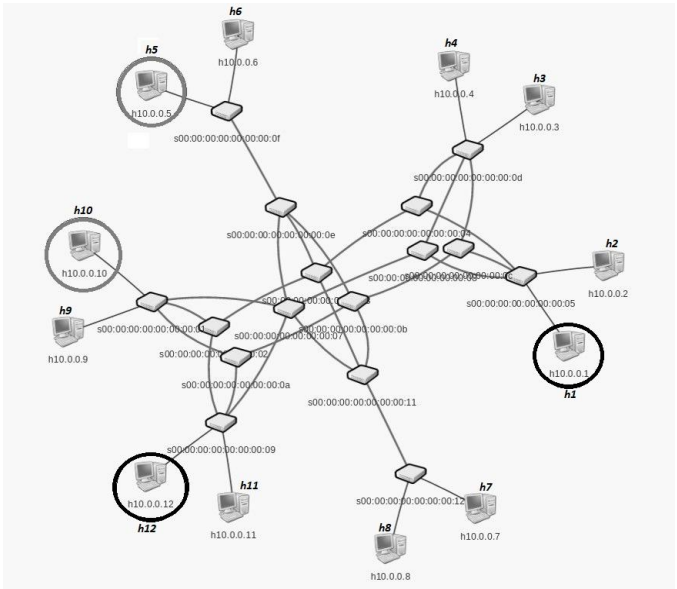

Figure 8. Network from Floodlight API Web.

Bandwidth measurements were performed between the two hosts in which the effect of having load balancing is contrasted with the operation without the load balancing algorithm. The results are summarized in Table 3:

Table 3. Bandwidth between h1 and h12

Without load balancing	With load balancing
35,6 Gbit/s	34,9 Gbit/s
36,2 Gbit/s	37,1 Gbit/s
32,3 Gbit/s	35,0 Gbit/s
35,8 Gbit/s	33,8 Gbit/s
36,3 Gbit/s	37,4 Gbit/s

The average values of the results presented in Table 3 are: 35.24 Gbit/s for the network without the load balancing module and 35.64 Gbit/s when applying the load balancing module.

Similarly, Table 4 shows the bandwidth measurements between hosts h5 and h10. The average values found for this pair of hosts are: 33.72 Gbit/s for the network without the load balancing module and 36.54 Gbit/s for the network with the load balancing module.

Table 4. Bandwidth between h5 and h10

Without load balancing	With load balancing
38,5 Gbit/s	35,2 Gbit/s
33,8 Gbit/s	37,0 Gbit/s
32,3 Gbit/s	34,9 Gbit/s
32,2 Gbit/s	39,1 Gbit/s
31,8 Gbit/s	36,5 Gbit/s

Subsequently, 100 ping requests were sent out and the time statistics of the transmitted packets were obtained. In Table 5 is observed that the average time was 2.19 times lower when the load balancing algorithm was implemented for the pair of hosts h1-h12.

Table 5. Time statistics between h1 and h12

	Without load balancing	With load balancing
Minimum time	0,039 ms	0,042 ms
Maximum time	14,532 ms	0,515 ms
Average time	0,221 ms	0,101 ms

In the Figure 9 the black color trace corresponds to the ping requests without using the load balancing module, and the grey color trace when it was implemented in hosts h1-h11. The figure shows that the general behavior in the two tests is very similar and the only difference is in the first ping request. The delay time of the first ping request with the load balancing module was 0.515 ms, while for the other case the value was 14.532 ms (see maximum time at Table 5).

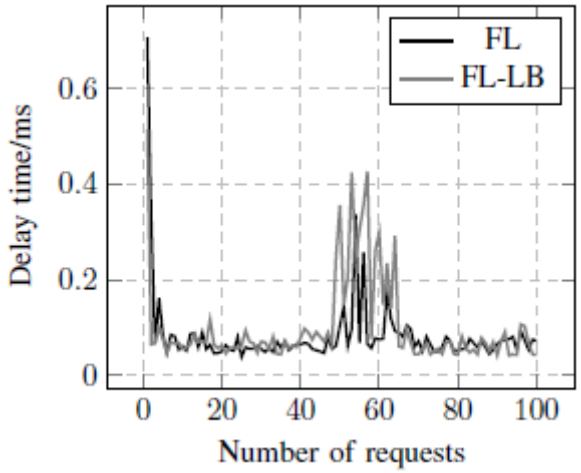

Figure 9. Delay time for 100 ping requests between h1 and h12.

As far as the hosts h5 and h10 is concerned, Table 6 shows the minimum and maximum values of the delay time in which the average delay time was 2.88 times lower when the load balancing module was used.

Table 6. Time statistics between h5 and h10

	Without load balancing	**With load balancing**
Minimum time	0,045 ms	0,040 ms
Maximum time	9,536 ms	0,520 ms
Average time	0,150 ms	0,052 ms

Figure 10 shows in more detail the general behavior of all ping requests. The black trace is related to ping requests without using the load balancing module, and the grey trace when implementing load balancing. The general behavior of the two configurations is very similar, however, the first ping request without the load balancing module presented a very high time response of 9.536 ms, while the other has a response of 0.520 ms (see maximum value Table 6).

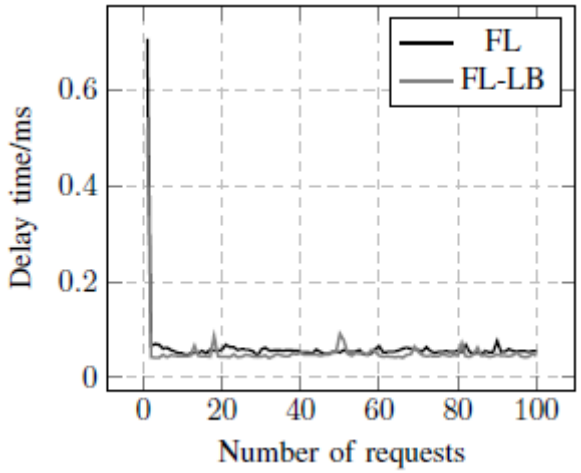

Figure 10. Delay time for 100 ping request between h5 and h10.

The load balancing algorithm implementation in a SDN network controlled by Floodlight shows a significant improvement by drastically reducing the time of the first ping request, where the pairs of hosts communicate for the first time. This causes the average time of ping requests to decrease considerably. On the other hand, the general behavior of the network in the two evaluations was quite similar, showing that the Floodlight controller has a pretty good performance on its own.

Although the average value of the bandwidth increases from one case to another, the improvements are very small unlike the results obtained with the OpenDaylight controller. The reason why this occurs is because the Floodlight controller has a large number of default modules that are automatically loaded each time the controller is executed and derives in an improvement of the network performance, among these modules is the load balancing. This default module for load balancing is developed in Java, so the one proposed and implemented in this paper is an important alternative since it was developed in Python.

4.2.1 Shortest path analysis and load balancing

Unlike the OpenDaylight controller, Floodlight does not send ping requests throughout the network when attempting to communicate two hosts. On the contrary, this controller determines the most appropriate way to establish the connection and generate a flow within its tables. In most cases, the selected path is usually the shortest and usually only one. The following test is similar to the one performed in OpenDaylight where it is observed which paths the controller defines for communication between two hosts when the load balancing algorithm is used and when it is not.

For demonstrating purposes of the load balancing in the network, initially the communication is carried out between hosts h3 and h8. The paths selected when establishing the connection between these two hosts with and without our load balancing algorithm are the following: paths between h3 and h8 without algorithm 0d::0c::0b::11::12 - 0d::08::07::11::12 (in Figure 11 the path in grey color) and path between h3 and h8 with algorithm 0d::0c::0b::11::12 (the path in black color). When the connection is established without the implementation of the load balancing algorithm, the default modules with which the controller works assign two paths for communication, one forward path and one return path. On the contrary, the load balancing algorithm only allocates one way for communication. The assignment of two paths is advantageous if we consider it as a way to avoid the saturation of the links while avoiding the load increment in one path. On the other hand, it is inefficient if we consider that several hosts try to communicate among them. In this way, more than the necessary resources are used.

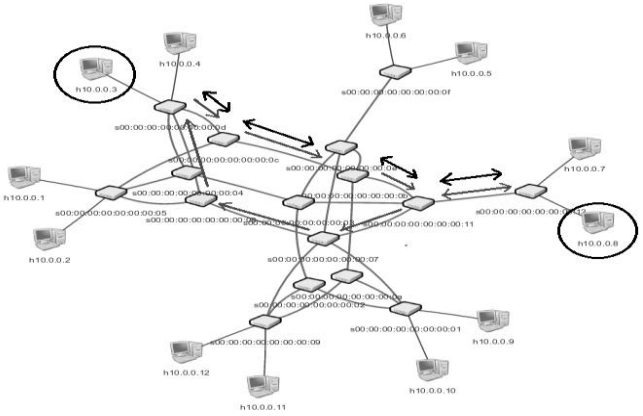

Figure 11. Load balancing between h3 and h8.

Then, host h3 remains sending ping requests to host h8 in order to occupy this link, while trying to establish the connection between hosts h4 and h7. In both cases, a single path is assigned and different from those previously assigned: path between h4 and h7 without the load balancing algorithm 0d::04::03::11::12 (in Figure 12 the path in grey color) and Path between h4 and h7 with the load balancing algorithm 0d::08::07::11::12 (the path in black color).

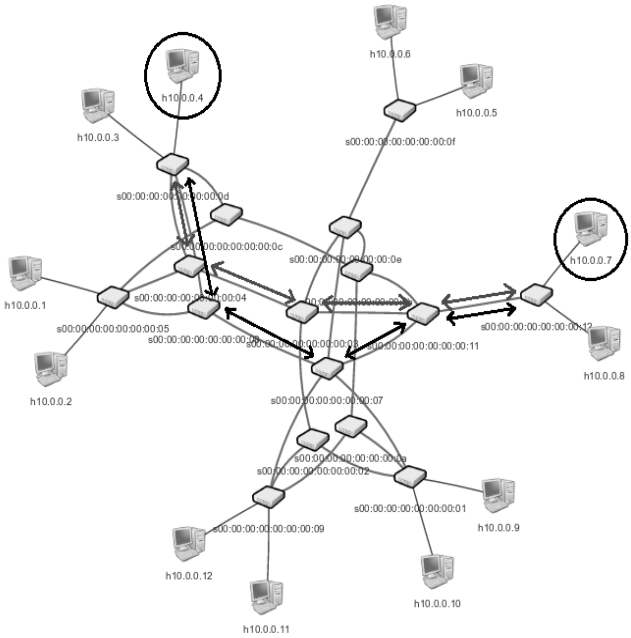

Figure 12. Load balancing between h4 and h7.

In the same way, host h3 remains sending ping requests to host h8 and host h4 sending ping requests to host h7 in order to occupy these links. With the new pair of hosts h12 and h6, the following paths are obtained: without the algorithm 0f::0e::07::09 (in Figure 13 the path in grey color) and with the algorithm 0f::0e::07::09 (the path in black color). In this case, the assigned paths are the same.

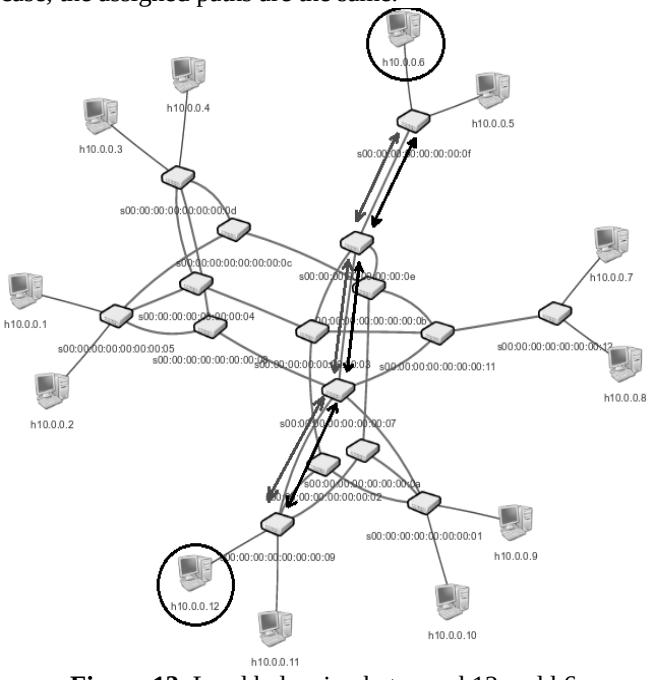

Figure 13. Load balancing between h12 and h6.

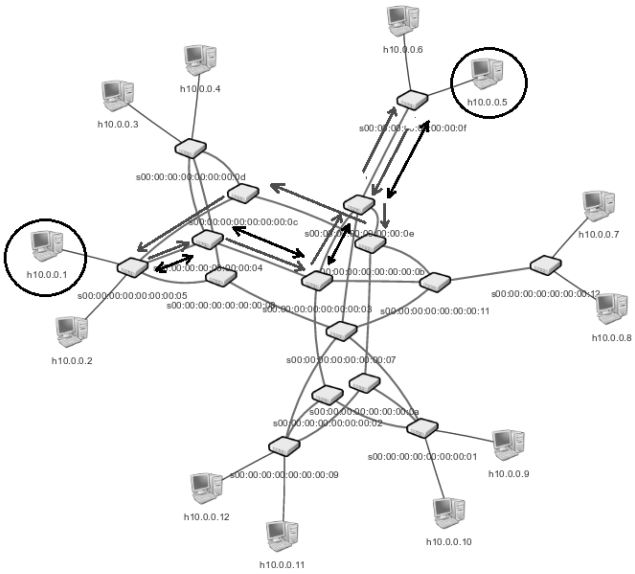

Figure 14. Load balancing between h1 and h5.

Finally, for the communication between h1 and h5, the controller assigns two paths by default, while the load balancing algorithm assigns only one: paths 0f::0e::0b::0c::05 - 0f::0e::03::04::05 (in Figure 14 the path in grey color) and the path 0f::0e::03::04::05 (path in black color). The general behavior in both tests shows a very good allocation and distribution of the resources of the network, which depending on the situation, will provide better results our proposal or the default module implemented by the controller. In this test, we demonstrate a better performance with our approach since the communication was made among 8 hosts.

5. Conclusions

Software defined networking or programmable networks have a wide range of controllers for traffic management in the network, where each one of them differs from each other due to its complexity and support given by their developers. The OpenDaylight controller has low performance when handling response times and bandwidth allocation, and it presents an excessive consumption of RAM. However, it has good information support from its developers and it has a Web API. On the other hand, the Floodlight controller has a very good throughput in managing response times and bandwidth by having a large number of modules that together perform specific actions that improve its efficiency. In addition, it has good documentation and easy handling through simple instructions in the terminal or through its Web API.

The implementation of the load balancing algorithm allowed improving the behavior of a software defined networking in terms of quality of service, improving bandwidth, decreasing response times and optimally distributing the load of the links. For the case in which the OpenDaylight controller was used, there were improvements of the available bandwidth of around 100 times when finding a more optimal path for the packets. On the other hand, the packets delay time was improved by reducing it roughly 5 times on average.

In general, the results in this item were quite satisfactory. In the same way, the results obtained with the Floodlight controller were not very far from the results obtained with the load balancing algorithm in terms of bandwidth and response times. Thus, in terms of these network parameters there is no need to make drastic changes to the controller.

6. Acknowledgement

The authors wish to acknowledge and thank the Universidad Distrital Francisco José de Caldas for supporting the development of this paper.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015.

[2] R. Adrian, A. Dahlan and K. Anam, "OSPF cost impact analysis on SDN network," 2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia, 2017, pp. 198-201.

[3] M. I. Hamed, B. M. ElHalawany, M. M. Fouda and A. S. T. Eldien, "A novel approach for resource utilization and management in SDN," 2017 13th International Computer Engineering Conference (ICENCO), Cairo, Egypt, 2017, pp. 337-342.

[4] M. F. Ramdhani, S. N. Hertiana and B. Dirgantara, "Multipath routing with load balancing and admission control in Software-Defined Networking (SDN)," 2016 4th International Conference on Information and Communication Technology (ICoICT), Bandung, 2016, pp. 1-6.

[5] H. Sufiev and Y. Haddad, "A dynamic load balancing architecture for SDN," 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE), Eilat, 2016, pp. 1-3.

[6] S. Asadollahi and B. Goswami, "Experimenting with scalability of floodlight controller in software defined networks," 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, 2017, pp. 288-292.

[7] Z. K. Khattak, M. Awais and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, 2014, pp. 671-676.

[8] Taimur Bakhshi, "State of the Art and Recent Research Advances in Software Defined Networking," Wireless Communications and Mobile Computing, vol. 2017, Article ID 7191647, 35 pages, 2017.

[9] A. Kumar, S. Jain, U. Naik et al., "BwE: flexible, hierarchical bandwidth allocation for WAN distributed computing," in Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15), pp. 1–14, London, UK, August 2015.

[10] P. Patel, D. Bansal, L. Yuan et al., "Ananta: cloud scale load balancing," in Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '13), pp. 207–218, August 2013.

[11] B. Heller, S. Seetharaman, P. Mahadevan et al., "Elastictree: saving energy in data center networks," in Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, p. 17, USENIX Associatio, 2010.

[12] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks," in Proceedings of the IEEE Global Communications Conference (Globecom '12), pp. 2689–2694, Ericsson Research, Anaheim, Calif, USA, 2012.

[13] V. Gudla, S. Das, A. Shastri et al., "Experimental demonstration of openflow control of packet and circuit switches," in Proceedings of the Optical Fiber Communication Conference (OFC '11), Collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC '10), vol. 45, pp. 1–3, IEEE, Los Angeles, Calif, USA, 2011.

[14] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "OpenFlow-based wavelength path control in transparent optical networks: a proof-of-concept demonstration," in Proceedings of the 37th European Conference on Optical Communication and Exhibition (ECOC '11), pp. 1–3, September 2011.

[15] D. Simeonidou, R. Nejabati, and M. P. Channegowda, "Software defined optical networks technology and infrastructure: enabling software-defined optical network operations," in Proceedings of the Optical Fiber Communication Conference (OFC '13), Optical Society of America, March 2013.

[16] A. N. Patel, P. N. Ji, and T. Wang, "Qos-aware optical burst switching in openflow based software-defined optical networks," in Proceedings of the 17th International Conference on Optical Network Design and Modeling (ONDM '13), pp. 275–280, Brest, France, April 2013.

[17] M. Bansal, J. Mehlman, S. Katti, and P. Levis, "OpenRadio: a programmable wireless dataplane," in Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN '12), pp. 109–114, Helsinki, Finland, August 2012.

[18] L. E. Li, Z. M. Mao, and J. Rexford, "Toward software-defined cellular networks," in Proceedings of the 1st European Workshop on Software Defined Networks (EWSDN '12), pp. 7–12, Darmstadt, Germany, October 2012.

[19] X. Mi, Z. Tian, X. Xu, M. Zhao, and J. Wang, "NO stack: a SDN-based framework for future cellular networks," in Proceedings of the International Symposium on Wireless Personal Multimedia Communications (WPMC '14), pp. 497–502, Sydney, Australia, September 2014.

[20] A. Bradai, K. Singh, T. Ahmed, and T. Rasheed, "Cellular software defined networking: a framework," IEEE Communications Magazine, vol. 53, no. 6, pp. 36–43, 2015.

[21] M. Dillon and T. Winters, "Virtualization of home network gateways," Computer, vol. 47, no. 11, Article ID 6965269, pp. 62–65, 2014.

[22] N. Feamster, "Outsourcing home network security," in Proceedings of the ACM SIGCOMM Workshop on Home Networks (HomeNets '10), pp. 37–42, New Delhi, India, September 2010.

[23] J. Jo, S. Lee, and J. W. Kim, "Software-defined home networking devices for multi-home visual sharing," IEEE Transactions on Consumer Electronics, vol. 60, no. 3, pp. 534–539, 2014.

[24] S. Mehdi, J. Khalid, and S. Khayam, "Revisiting traffic anomaly detection using software defined networking," in Recent Advances in Intrusion Detection, pp. 161–180, Springer, Berlin, Germany, 2011.

[25] E. W. Zegura, K. L. Calvert and S. Bhattacharjee, "How to model an internetwork," INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE, San Francisco, CA, 1996, pp. 594-602 vol.2.

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to Algorithms: Dijkstra's algorithm", Second Edition. MIT Press and McGraw-Hill. Section 24.3. 2001. Pags.595–601.

[27] Annop Monsakul, "SRAD: Smart Routing Algorithm Design for Supporting IoT Network Architecture," International

Journal of Communication Networks and Information Security (IJCNIS) Vol. 10, No. 1, pp. 91–98, 2018.

[28] I. D. Irawati, A. B. Suksmono, I. Joseph, and M. Edward "Missing Internet Traffic Reconstruction using Compressive Sampling" International Journal of Communication Networks and Information Security (IJCNIS) Vol. 9, No. 1, pp. 57–66, 2017.