Journal of Internet Services and Applications, 2023, 14:1, doi: 10.5753/jisa.2023.3068

Load Balancing between Paths using Software Defined Networks

Alisson Cavalcante e Silva [**Universidade do Estado do Rio de Janeiro** | ***alisson2000rj@gmail.com***]
Marcelo Gonçalves Rubinstein [**Universidade do Estado do Rio de Janeiro** | ***rubi@uerj.br***]

PEL/DETEL/FEN, Universidade do Estado do Rio de Janeiro, Rua São Francisco Xavier, nº 524, Sala 5019E, Maracanã, Rio de Janeiro, RJ, 20550-013, Brazil.

Received: 19 December 2022 • **Accepted:** 11 September 2023 • **Published:** 08 November 2023

Abstract Small networks usually use Ethernet devices that apply solutions such as the Spanning Tree Protocol (STP) to forward packets through a single path with no loops. However, this prevents the use of idle links that may reduce congestion and augment the aggregate bandwidth of the network. This work proposes a load balancing mechanism between paths using Software Defined Networks (SDNs). The proposed mechanism, named MLB (Multipath Load Balance), computes multiple paths with disjoint links that have the smallest number of hops between source and destination. Moreover, MLB has a "switching control" function that verifies whether the current occupation of the path exceeds a percentage of its capacity and if the potential new path computed by MLB has an occupation at least a percentage value smaller than that of the current path. MLB is implemented in Python and evaluated in Mininet. The results show that it is possible to increase the aggregate bandwidth by 95% and decrease the packet loss by about 95.5% compared with the standard operating mode of the OpenDaylight SDN controller.

Keywords: Load Balancing, Software Defined Network, Multipath Routing, Performance Evaluation

1 Introduction

Virtual services offered over the Internet by commercial clouds or within enterprise networks by corporate clouds are increasingly being consumed by users. Nowadays, there is an increasing number of new applications offering essential services for the professional scope and also for entertainment. Such growth tends to promote the increase of the underlying network infrastructure with the introduction of new hosts and, consequently, of new switches, necessary for the expansion of the network. In general, with the network expansion, new links that generate path redundancy are created. In this way, by having redundant links, it is possible to use them not only during unavailability situations, but also in the day-to-day performing load balancing between them. This type of scenario makes it possible to increase the throughput of data traffic by combining the bandwidth of two or more network paths [Liu *et al*., 2016].

This work proposes a load balancing mechanism between paths, named MLB (Multipath Load Balance), using SDNs (Software Defined Networks)[1]. MLB can be applied on small networks preferably with a large number of disjoint paths. The fact that SDN networks have a logically centralized architecture allows the controller to have a global view of the network topology and with this, it is possible to compute all the paths between sources and destinations. Another important characteristic is that traditional load balancers are not programmable, as they have instruction sets designated by the proprietary architecture of their manufacturers [Semong *et al*., 2020]. Nevertheless, SDN networks are programmable. This feature allows the network administrator to create intelligent applications, using data collected from the network, that can make decisions such as which way a network flow should be routed. In [Hamdan *et al*., 2021], the authors state that the use of a modern approach, such as SDN, allows overcoming data plane congestion by optimizing data traffic when less busy paths are chosen. In addition, Ethernet networks rely on the STP (Spanning Tree Protocol) to generate a spanning tree with only one path, which causes data traffic to be sent from one point to another in the network without loops generated by redundant paths [Singh *et al*., 2015]. Thanks to its global view of the network, the SDN controller makes the use of STP unnecessary and assumes the role of managing the network paths. This approach is employed in Amiri and Javidan [2019], but no load balancing mechanism is used. Other researchers address load balancing, such as in [Bredel *et al*., 2014; Ramdhani *et al*., 2016; Mallik and Hegde, 2014; Bhandarkar and Khan, 2015; Hassan, 2017; Zhao *et al*., 2021]. Their main differences are in the use of disjoint or non-disjoint links, the type of path selector, and the introduction of other functionalities, such as a flow admission control. Moreover, these solutions use the traditional SDN model where the controller provides static rules to the SDN switches but other solutions apply the data plane programmability paradigm using languages as P4 and NPL, such as in Kawaguchi *et al*. [2019].

The MLB mechanism is based on the load balancing mechanism proposed by Seth in Seth [2022]; Hassan [2017]. However, MLB differs from Seth's mechanism in that it has a function named "switch control" that uses some assumptions to perform the path change, avoiding modifications that lead to a very low gain. The switching control verifies if the current occupancy of the path, i.e.; highest data volume transmitted and received between the links of the path, exceeds a percentage of its capacity and if the computed potential new path has an occupancy at least a percentage value smaller

[1]This work is based on our paper published in Portuguese in the Proceedings of the XXV Workshop de Gerência e Operação de Redes e Serviços (WGRS) available at `https://sol.sbc.org.br/index.php/wgrs/article/view/12455/12320`

than the occupancy of the current path. Furthermore, MLB performs path computation with disjoint links, based on the Edmonds and Karp algorithm [Cormen *et al*., 2009]. This algorithm performs path computation using the Breadth-First Search (BFS) algorithm to find all possible paths between a source and a destination in a graph. Then it uses the maximum flow theory by applying the Ford-Fulkerson method to select paths that do not share edges [Cormen *et al*., 2009]. Seth's mechanism uses non-disjoint links, based on an adaptation of Dijkstra's algorithm.

We use Mininet [Lantz *et al*., 2010] to compare the performance of our mechanism, Seth's mechanism, and the OpenDaylight (ODL) SDN controller, which has no load balancing mechanism. The results show that MLB performs better in bandwidth aggregation and packet loss. It is possible to increase the aggregate bandwidth by 95% and decrease the packet loss by about 95.5% compared with ODL.

This work is organized as follows. Some characteristics of multipath routing are presented in Section 2. Section 3 presents related work. Section 4 describes the implementations of the load balancing mechanisms, while Section 5 presents the performance evaluation. At last, conclusions and future work are presented in Section 6.

2 Exploiting Multipath

The multipath routing technique exploits the physical resources of the traditional network using multiple paths between source and destination to send data traffic [Tsai and Moors, 2006]. The three basic components of multipath routing are: path computation, traffic division, and path selection [Singh *et al*., 2015]. These components can also be applied in the SDN context, when an SDN switch can use information from different layers and multiple paths to forward packets.

Path computation aims to find all existing paths between a source and a destination. For the search process to be effective, the path computing algorithm needs to have global knowledge of the network topology. After discovering the topology, the algorithm needs to identify the paths from source to destination based on three scenarios [Singh *et al*., 2015]:

a) Disjoint nodes: path is composed of nodes (other than source and destination) not shared by other paths. By not sharing nodes, there will also be no link sharing, except for broadcast links and Non-Broadcast Multi-Access (NBMA) networks. This configuration provides greater fault tolerance as the paths are completely independent. However, it implies greater expenditure on infrastructure, as in order to obtain more paths it is necessary to create more nodes and links. This scenario is exemplified in **Figure 1(a)**, which presents a graph composed of 10 nodes and 14 links, with node "A" as the source and node "J" as the destination of two flows that follow paths that are independent and free from the sharing of nodes and links.
b) Disjoint links: the path is composed of nodes that can be shared by other paths, but the links are not shared. This scenario has lower fault tolerance compared with the previous scenario. This is because, when a node fails, all paths that make use of it will be affected. The same will not happen if the failure occurs in a link. The use of disjoint links can help increasing the total amount of bandwidth and decreasing network congestion [Sun *et al*., 2012]. An example of this scenario is presented in **Figure 1(b)**, where there are three flows traversing paths that do not share links, but node "E" is shared by Flows 2 and 3.
c) Non-disjoint links: exemplified in **Figure 1(c)**, both the nodes and the links of a path can be shared by the other paths in the network. The lack of restrictions on the use of common nodes and links makes path computation easier [Singh *et al*., 2015]. However, this scenario is considered the worst case among the three, because if a node or link fails, this failure will affect all paths that share such physical resource.

The performance of the algorithm in path computation depends on the number of nodes and links in the topology. According to Singh *et al*. [2015], establishing an ideal number of paths can reduce the complexity of the processing. Also, non-disjoint paths should be avoided, as sharing nodes and links means having to deal with low fault tolerance. Another issue regards bandwidth, as paths with shared links will also have their bandwidths shared.

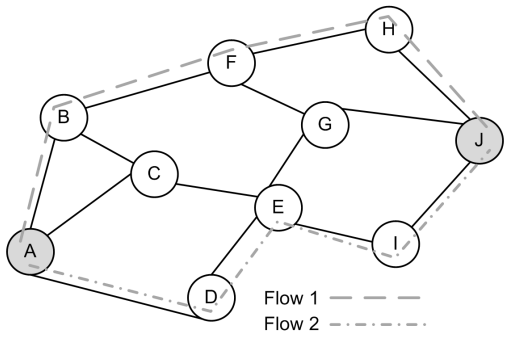
The load balancing mechanisms evaluated in this work make use of non-disjoint paths in the case of Seth's mechanism and paths with disjoint links in the case of the proposed mechanism named MLB.

According to Prabhavat *et al*. [2012], the traffic division can be done at different levels: packet, flow, subflow, superflow, and sub-superflow. In this work we use the traffic division by flow based on the following packet identifiers: source IP, destination IP, protocol, source port, and destination port. Once a flow identifier is defined, the SDN switches use it when forwarding packets.

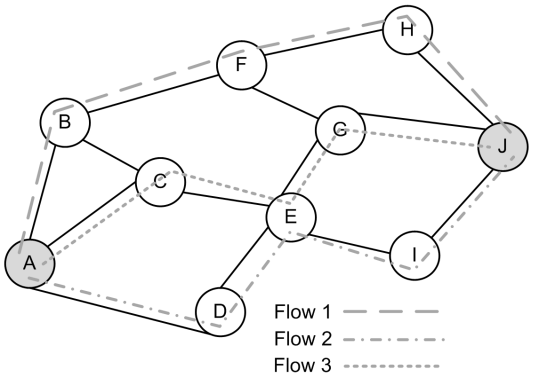
The selection of paths can be classified according to the type of selector used [Prabhavat *et al*., 2012]:

a) Round robin: traffic is forwarded by all computed paths following a cyclic sequence.
b) Packet info: traffic is forwarded based on information contained in the packet header.
c) Traffic condition: may consider the traffic load, the traffic throughput, the traffic volume, or the number of active flows in the path.
d) Network condition: may consider queue size, delay, jitter, or packet loss along the path.

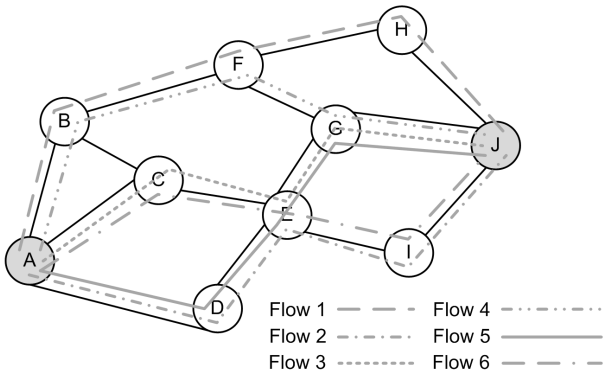
The load balancing implementations used in this work are based on the selection of paths by traffic condition. The MLB mechanism uses the throughput of the path, i.e.; the volume information of data transmitted and received in a 1 s-time frame by the SDN switches of a path. Seth's mechanism uses the volume of traffic in the path, i.e., data volume transmitted and received by the interfaces of the switches that make up the path during a 2 s-interval.



(a) Disjoint nodes



(b) Disjoint links



(c) Non-disjoint links

Figure 1. Examples of path computation scenarios considering node "A" as the source and "J" as the destination.

3 Related Work

Load balancing between links using SDN networks has been studied by some researchers, such as in recent survey papers [Hamdan *et al*., 2021; Semong *et al*., 2020].

Bredel *et al*. [2014] present a traffic optimization using multipath that extends the Floodlight controller and is based on disjoint paths. They use a path selection algorithm that chooses the path with the least number of mapped flows. The work compares different path selection algorithms that are based on: hash of packet header fields, random selection, round-robin, or the number of flows in the path. The results show that the use of the algorithm based on the number of flows performs better when compared with the others. However, the authors did not include in the work the selection by path based on the traffic rate and the traffic volume, usually used to assess congestion in the links.

Ramdhani *et al*. [2016] address the limitation of the single-path routing due to the use of STP in Ethernet networks. The authors use SDN as a way to help with multipath routing. Load balancing is proposed with a flow admission control activated whenever the data load in the path reaches 80% of its capacity, which helps to reduce network congestion. The model uses switch statistics collected by the Ryu controller that help in decision making. An evaluation shows the good performance of the proposed model in achieving a more stable transfer rate and lower latency compared with STP single-path routing. However, admission control seeks to reduce network congestion by not allowing new flows.

Mallik and Hegde [2014] present a dynamic load balancing mechanism using multipath and a congestion control mechanism that calculates the data load level on the paths to determine whether they can be used in the path selection. The application together with the Floodlight controller tries to detect if the path load is above a certain threshold and if so, instructs the switches to forward the flows through an alternative path. The authors highlight the capacity of their model to identify and immediately react to load imbalance and traffic congestion. All possible paths between source and destination are computed, characterizing the use of paths with non-disjoint links. However, it is known that this type of path computation has low fault tolerance, since a node or link failure will affect all flows that pass through such physical resource.

Bhandarkar and Khan [2015] implement a dynamic load balancing that chooses the path with the highest free bandwidth to route traffic. The work presents a performance evaluation, with data traffic generated by the Cbench tool, between the implemented solution and the load balancing of the Floodlight controller based on round-robin. According to the authors, the results show that with the use of their solution, in comparison with the round-robin load balancing, it is possible to handle more packets, as well as reduce network latency. However, the work does not present the level of network congestion, which could be done by showing the packet loss rate of the traffic.

Hassan [2017] presents the implementation of the dynamic load balancing algorithm proposed in Seth [2022]. The work aims to evaluate and validate the proposed algorithm. In his work, Seth uses the computation of shortest and non-disjoint paths between source and destination, based on an adaptation of Dijkstra's algorithm. OpenDaylight is used as a network controller in a Fat-tree-based topology. The algorithm only takes into account two network flows, a fixed flow and a load-balanced one. Thus, the first flow operates as background traffic and the second is switched between paths. In addition, switching takes into account the data volume transmitted and received on the current path comparing it with those of the computed paths. However, there is no threshold to avoid switching between paths of similar performances. Our work, on the other hand, proposes an adaptation of Seth's algorithm. Our solution supports the load balancing of more flows and performs the switching between paths in a controlled manner, avoiding switching to new paths that present flow rates similar to that of the path in use.

Load balancing may increase energy consumption by spreading traffic to as many active SDN devices as possible.

So, Zhao *et al*. [2021] propose PLOFR, a Power-efficient and Load-balanced Online Flow Route framework for SDN networks. A power optimization problem of satisfying load balancing constraint is formulated and the corresponding algorithms are proposed. However, energy consumption is out of the scope of this paper.

We highlight the main characteristics of the related work in **Table 1**. This work, similarly to [Ramdhani *et al*., 2016; Mallik and Hegde, 2014], also proposes an SDN-based load balancing mechanism that has a switching control between links activated whenever the load level in the path reaches a certain threshold. In addition, this mechanism also verifies whether the potential new path has more free bandwidth compared with the current path. Thus, in general, it prevents unnecessary switches from being performed that could lead to congestion of the new path chosen by the load balancer. However, unlike [Ramdhani *et al*., 2016; Mallik and Hegde, 2014], we do not use a selective policy of packet discard, as the retransmission of discarded packets directly affects the flow transmission time. In another way, the use of paths with disjoint links avoids the sharing of links between the computed paths. This choice contributes to the load balancing to have greater fault tolerance when a link is down since the link unavailability will affect only a single path.

4 Evaluated Load Balancing Mechanisms

In this work, two load balancing mechanisms between paths are used, the one proposed in this work named MLB and the one by Seth. The two mechanisms act in the application layer and interact with the network controller using its northbound interface. Furthermore, they use the "NetworkX" library written in Python for the creation, manipulation, and study of graphs and networks [Hagberg *et al*., 2008]. In our case, nodes are simple elements (SDN switches) that use flow identifiers to forward packets. Traditional layer-2 switches forward packets using only the destination MAC address and do not perform load balancing. In the following, we present implementations of both mechanisms. Our mechanism implementation is openly available[2].

4.1 Implementation of Seth's Mechanism

This implementation is based on Dijkstra's algorithm for computing non-disjoint multiple paths that have the smallest number of hops between the source and the destination. The algorithm only works for one load-balanced flow. Moreover, switching takes into account the data volume on the current path comparing it with those of the computed paths. We use Seth's mechanism since its source code is available under a General Public License [Seth, 2022]. The pseudocode of this implementation, derived from its source code, is presented in Algorithm 1.

The source and destination hosts are defined at the beginning of the execution, as shown in Lines 2 and 3. Seth's implementation performs load balancing of just one flow.

[2]The code is available at https://github.com/alisson2000rj/MLB

Algorithm 1: Seth's Implementation

```
1  begin
2      src = source host
3      dst = destination host
4      while true do
5          connected_nodes = proc_topology()
6          G = networkx.graph(connected_nodes)
7          paths = networkx.all_shortest(G, src, dst, dijkstra)
8          for i from 1 to number(paths) do
9              t1 = returns_data_tx_rx(OpenDaylight_restconf,
                 paths[i])
10             wait 2 s
11             t2 = returns_data_tx_rx(OpenDaylight_restconf,
                 paths[i])
12             path_cost[i] = t2 - t1
13         end
14         best_path = returns_index(min(path_cost))
15         configure_flow(paths[best_path], source, destination)
16     end
17 end
```

In Line 4, the repetition structure begins, which keeps the application in constant execution. In Line 5 the "proc_topology()" function asks the network controller for information about the connections established between the switches of the network. Such information is generally known to the controller, which uses the OFDP (OpenFlow Discovery Protocol) and the LLDP (Link Layer Discovery Protocol) protocols to discover existing network devices in the topology. In the next line, with the information received from the controller, the network application assembles the global graph of the network topology. In Line 7, the "G" graph structure is processed by the "all_shortest" function, based on Dijkstra's algorithm. As a result, all computed paths between source and destination are returned and stored in the variable "paths". On Line 8, a loop that will be repeated given the number of computed paths starts. Inside the loop, in Line 9, "t1" receives from the controller, at the current time, information of the data volume transmitted and received by the interfaces of the switches that make up the path and after a 2 s-waiting, "t2" also receives from the controller the same data volume information, as shown in Line 11. Then, in Line 12 the value of "t2" is subtracted by "t1" and the result is stored in the vector "path_cost". The variable "best_path" receives the path with the lowest recorded data volume, in Line 14. The idea is to use the path with less data transmitted and received during that 2 s-interval. And finally, in Line 15, the application submits to the controller information about the new path that the flow must use.

4.2 Implementation of the MLB Mechanism

This implementation is based on the Edmonds and Karp [Cormen *et al*., 2009] algorithm, which computes multiple disjoint paths between a source and a destination. Their algorithm applies the Breadth First Search (BFS) algorithm, the maximum flow theory, and the minimum cut of links with capacity equal to 1 to compute the shortest disjoint paths between a source and a destination. Our implementation differs from Seth's not only in the type of path calculation but also in the number of flows it can handle. In addition, the MLB has a "switching control" function that verifies whether the

Table 1. Main characteristics of related work.

Work	Scenario	Path selector	Other functionalities
Bredel *et al*. [2014]	disjoint links	traffic condition	-
Ramdhani *et al*. [2016]	non-disjoint links	traffic condition	flow admission control
Mallik and Hegde [2014]	non-disjoint links	traffic condition	congestion control
Bhandarkar and Khan [2015]	non-disjoint links	traffic condition	-
Hassan [2017]	non-disjoint links	traffic condition	-
Zhao *et al*. [2021]	non-disjoint links	traffic condition	power saving
This work	disjoint links	traffic condition	switching control

current occupation of the path exceeds a percentage value (named "condition1") of its capacity and if the potential new path computed by MLB has an occupation at least a percentage value ("condition2") smaller than that of the current path. The pseudocode of this implementation is presented in Algorithm 2.

Algorithm 2: MLB Implementation

```
1  begin
2      src = source switch
3      dst = destination switch
4      no_flows = number of flows
5      path_capacity = 10,000,000
6      current[flow] = ∅
7      while true do
8          connected_nodes = proc_topology()
9          G = networkx.graph(connected_nodes)
10         for flow from 1 to no_flows do
11             paths =
                 networkx.edge_disjoint(G, src, dst, edmonds_karp)
12             for i from 1 to number(paths) do
13                 paths_cost[i] =
                     calculate_occupation(OpenDaylight_restconf,
                     paths[i])
14             end
15             best_path = returns_index(min(paths_cost))
16             if current[flow] = ∅ then
17                 configure_flow(paths[best_path],
                     src, dst, flow)
18                 current[flow] = best_path
19             end
20             else if paths_cost[current[flow]] ≥ condition1 ×
                 path_capacity & path_cost[best_path]
                 ≤ condition2 × path_cost[current[flow]] then
21                 configure_flow(paths[best_path],
                     src, dst, flow)
22                 current[flow] = best_path
23             end
24         end
25     end
26 end
```

In Lines 2 and 3, the source and destination switches are defined. Line 4 defines the number of flows that will be balanced. In Lines 5 and 6 reference values for the switching control function are defined. In Line 7 the repetition structure begins, which keeps the application in constant execution. Line 10 presents a repetition structure conditioned to the number of flows in use.

In Line 11, the G graph structure is processed by the "edge_disjoint()" function using the path computation method of Edmonds and Karp. On Line 12, a loop that will be repeated given the number of computed paths starts. In this one, the "calculate_occupation" function estimates the occupation on the computed paths using the volume information of data transmitted and received in a 1s-time frame by the switches of each path. At this point, the network application registers the highest data load verified between the links of the path, an action that is repeated in all the computed paths. In Line 15, the variable "best_path" receives the path with the lowest occupation. And finally, Line 16 tests if it is the first execution of the application for the current flow. If so, the path of the flow is configured in Line 17 and "current" vector receives the index of the best path for the current flow. If not, Line 20 presents the switching control function, which allows switching the flow to the new path if the current occupation of the path is greater than or equal to the value defined in the variable "condition1", and the new path has an occupation that is at least "condition2" smaller than the occupation of the current path. We present tests regarding the setting of "condition1" and "condition2" values in Sections 5.1 and 5.2. For cases where both conditions are satisfied the new alternative path will be set to the current flow in Line 21 and the "current" vector will receive the index of the best path for the current flow, as shown in Line 22.

5 Performance Evaluation

This section presents the performance evaluation of the ODL controller, which does not use load balancing mechanisms, Seth's and the MLB mechanisms. The metrics used to obtain the results are presented hereafter, followed by the mechanisms and the traffic models used in the evaluation.

Evaluated metrics are:

a) aggregate throughput: the use of load balancing tends to increase the aggregate bandwidth. For its calculation, flows use the TCP (Transmission Control Protocol) protocol.

b) fairness: the use of a path by more than one flow is considered fair when all flows make use of equal parts of the available bandwidth. The fairness index [Jain, 1991] returns a value between 0 and 1. If all flows have a nearly equal flow throughput between them, the result is close to 1. If there are significant differences in flow throughputs, the result tends to 0. The fairness index [Jain, 1991] is given by:

$$f(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2},$$

where "n" is the total number of flows competing for bandwidth and "x_i" is the throughput value of the "i" flow.

c) packet loss: packet loss is mainly due to congestion in the path. In this case, flows use the UDP (User Datagram Protocol) protocol.

The three mechanisms used in the experiments are presented hereafter in more detail:

a) ODL: the only mechanism with no load balancing. With the use of the ODL controller, all flows are switched by the same default path defined in the spanning tree.
b) Seth's: performs load balancing using Seth's implementation based on non-disjoint paths.
c) MLB: performs load balancing using the path-based MLB implementation with disjoint links and the switching control function.

Seth's mechanism performs load balancing of only one of the flows, in this case the main flow. The MLB mechanism can balance more flows. However, to make this evaluation fair, the experiments performed are divided into two groups named two-flow- and four-flow-experiments. In the experiments with two flows, MLB is configured to balance only the main flow as in Seth's mechanism. Furthermore, another flow that corresponds to background traffic is used and this traffic is not balanced. For the two-flow experiments, two types of traffic models are used:

a) TCP-2f model: composed of two flows, the main flow has host "h1" as its source and host "s5" on port TCP/5001 as its destination, and the background flow has as its source the host "h2" and "s6" as the destination on port TCP/5002 (see **Figure 2**). The duration of the flows is fixed at 600 s.
b) UDP-2f model: composed of two flows, the main flow has host "h1" as its source and host "s5" on the UDP/5001 port as its destination, and the background flow has as its source "h2" and as destination "s6" on the UDP/5002 port (see **Figure 2**). The duration of the flows is also fixed at 600 s.

In the experiments with four flows, MLB is configured to balance three flows, which are main flows. The fourth flow is background traffic, being switched by the default path generated in the spanning tree. For the experiments with four flows, two types of traffic models are used:

a) TCP-4f model: composed of four flows, three of them main flows with sources, destinations, and ports defined as follows: for flow 1, the source is the host "h1" and destination is the host "s5" on the port TCP/5001; for flow 2, the source is the host "h2" and destination is the host "s6" on port TCP/5002; and for flow 3, the source is the host "h3" and destination is the host "s7" on port TCP/5003. The background flow has host "h4" as source and host "s8" on port TCP/5004 as destination (see **Figure 2**). The duration of the flows is fixed at 600 s.
b) UDP-4f model: composed of four flows, three of them main flows with sources, destinations, and ports defined as follows: flow 1, the source is the host "h1" and destination is the host "s5" on port UDP/5001; flow 2, the source is the host "h2" and destination is the host "s6" on port UDP/5002; and flow 3, the source is the host "h3" and destination is the host "s7" on port UDP/5003. The background flow is defined as having host "h4" as source and host "s8" on port UDP/5004 as destination (see **Figure 2**). The duration of the flows is fixed at 600 s.

TCP flows are used to measure the aggregate throughput and the fairness index, whereas UDP flows are used to measure the packet loss. UDP flows are configured to transmit data at 9.5 Mbps; 95% of the capacity of the network links. This transmission rate has been chosen after running several tests in which packet loss was measured. The transmission rate was successively increased until the limit of transmissions without packet loss was reached; 9.5 Mbps in the experiments. We aim to evaluate the packet loss when the network is near congestion, even if only one flow is sent. All flows are generated using the iPerf tool. Each test is run 30 times. Means and 95% confidence intervals are used.

Figure 2 shows the network topology used in the experiments, which is composed of eight nodes, three disjoint paths, and five non-disjoint paths considering the sources and destinations already mentioned. The topology is emulated in Mininet [Lantz *et al*., 2010]. Network nodes are emulated using the Open vSwitch virtual switch. The capacity of all links in the network is configured at 10 Mbps due to available computational resources. The OpenDaylight controller was chosen because it is an open project that is already used in the academic and corporate world [Kreutz *et al*., 2015]. Developed in the Java programming language, it has a native application named "l2switch" composed by the "Loop Remover" module responsible for generating the network spanning tree, similar to the one generated by the STP protocol. The spanning tree is composed of standard paths that cover all nodes in the network and aims to eliminate redundant paths that may cause loops. ODL maintains a spanning tree inventory with the STP "status" of the "forwarding" for the active links and "discarding" for the inactive links. In the example of the figure, for host "h1" to communicate with host "s5", path 1-2-6-8 is used. The status of the links can be verified through the ODL management web console [OpenDaylight, 2022].

5.1 Two-Flow-Experiments

First, "condition1" and "condition2" parameters of the MLB mechanism are varied to evaluate their impact on the aggregate throughput using the TCP-2f traffic model. We make $condition1 \in \{30\%, 40\%, 50\%, 60\%, 70\%\}$ and $condition2 \in \{10\%, 30\%, 50\%, 70\%\}$.

Figure 3 shows that there is no significative impact of both parameters on the aggregate throughput. In this way, we chose to set the first condition to 50% so that the switching may occur only when the current path has an average or large bandwidth occupation. By doing this, we avoid dividing large flows and consequently the delays caused by excessive reordering of packets at the destination due to this division. Likewise, the second condition is set to 10% to avoid switching between paths with very close bandwidths, which could also contribute to the occurrence of out-of-order

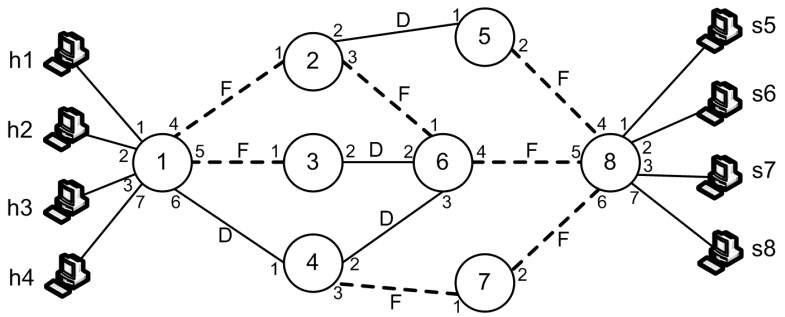


Figure 2. Network topology used in the experiments. The dashed links indicate the default paths designated by the spanning tree generated by the ODL. F stands for forwarding and D stands for discarding, both related to the status of links when using ODL.

packets [Singh *et al*., 2015].

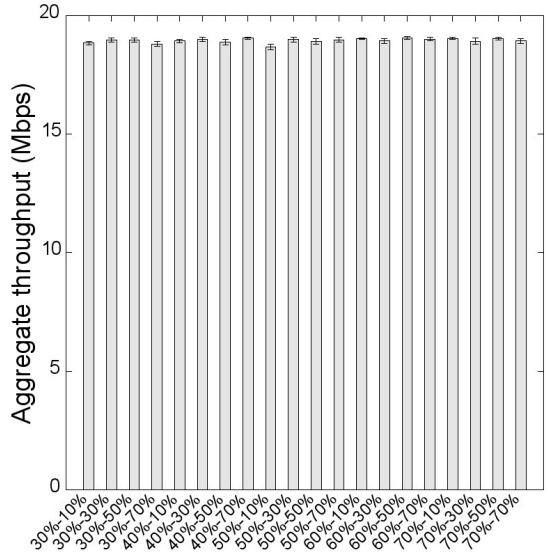


Figure 3. Aggregate throughput for MLB when different values of the pair condition1-condition2 and two flows are used.

Then we evaluate the results for the ODL controller, Seth's and MLB mechanisms. **Figure 4** presents the aggregate throughput results. It is possible to observe that using only the ODL, the aggregate throughput is approximately 10 Mbps. In this case, the main flow and the background one share the bandwidth of the standard path generated by the ODL (spanning tree) formed by nodes 1-2-6-8. Thus, as there is no load balancing, the aggregate throughput will be less than or equal to the bandwidth of the standard path. In Seth's load balancing, the aggregate throughput is increased by 41% when compared with the ODL and reaches 13.5 Mbps. This is because Seth's balancing performs the switching of the main flow to a new alternative path that presents a lower volume of traffic. The main flow is now switched to a path without shared links with the background flow, formed by nodes 1-4-7-8, to the default path used by the background flow, or to paths with links shared with the default path, formed by nodes 1-2-5-8, 1-3-6-8, and 1-4-6-8. Finally, for the MLB balancing, aggregate throughput reaches 18.6 Mbps, a 37%-gain when compared with Seth's one. This better performance is due to the use of paths with disjoint links and to the switching control function that only allows switching for the main flow when 50% or more of its path bandwidth is occupied and the new path has an occupied bandwidth that is at least 10% smaller than that of the current path. This prevents switching to a new path that has an occupancy level similar to the current path in use.

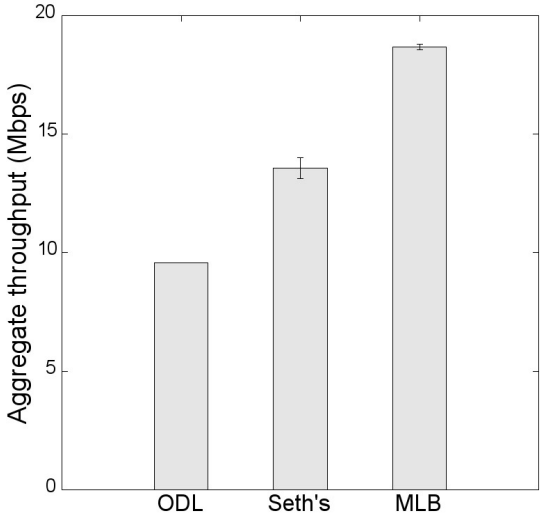


Figure 4. Aggregate throughput for ODL, Seth's and MLB mechanisms when the number of flows is 2.

Figure 5 presents the results of the fairness index using the TCP-2f traffic model. For ODL, the contention for the use of bandwidth between the flows resulted in a fairness index of 0.99, which means that both flows have similar throughputs: background flow throughput reaches 4.8 Mbps versus 4.7 Mbps of the main flow. In Seth's load balancing, the fairness index is also 0.99; which demonstrates that the flows have similar throughputs, even with the path switching generated by the load balancing. This means that at times when the background flow and the main flow use the standard path, the throughputs remain approximately equal and at times when the main flow is switched to an alternative path, both flows also have similar throughputs. The average throughput for the background flow is 6.8 Mbps against 6.7 Mbps for the balanced main flow. For the MLB balancing, the fairness index is also 0.99. Both flows reach 9.3 Mbps. It should be noted that even the background flow is indirectly benefited by the load balancing of MLB since it uses the 1-2-6-8 path

exclusively for most of the time.

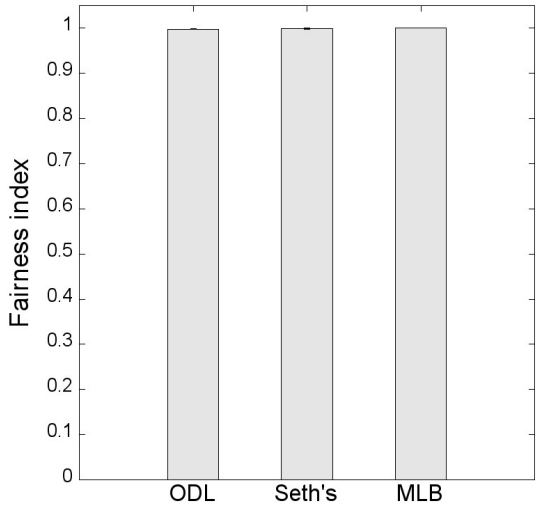


Figure 5. Fairness index for ODL, Seth's and MLB mechanisms when the number of flows is 2.

Finally, **Figure 6** presents the packet loss results using the UDP-2f traffic model. With the use of ODL, packet loss is close to 50%. This high value can be explained by the fact that the flows use the standard path formed by nodes 1-2-6-8. The bandwidth of the default path is shared by the two UDP flows that are not subject to congestion control. Thus, the two source hosts transmit the flow at a constant rate of 9.5 Mbps, which leads to a large packet loss. In Seth's balancing, packet loss is close to 27%, as the loss mainly occurs when the balanced main flow shares the standard path with the background flow or when it uses some alternative path with links shared by the standard path; for example for paths formed by nodes 1-2-5-8, 1-3-6-8, and 1-4-6-8. With the switching of the main flow to the path formed by nodes 1-4-7-8, the flows no longer compete with each other for bandwidth and the packet loss is not significant. In MLB balancing packet loss is below 5%. This is because the switch control function only allows the main flow switching to occur when there is a difference of 10% or more in the occupancy of the current path compared with that of the new path. When paths have their bandwidth occupied with similar load levels, switching does not take place. Thus, in the case where the main flow follows a path that does not share any link with the standard path, such as the path formed by nodes 1-4-7-8, the occupation of the paths will occur similarly. This happens because, with the switching control, the flows will remain for a longer time following different paths without any switching taking place. And in this way, as there is no bandwidth dispute between the flows, more packets are successfully delivered at their destinations and, consequently, packet loss decreases. MLB presented a packet loss 13 times less compared with Seth's one and 24 times less compared with ODL one.

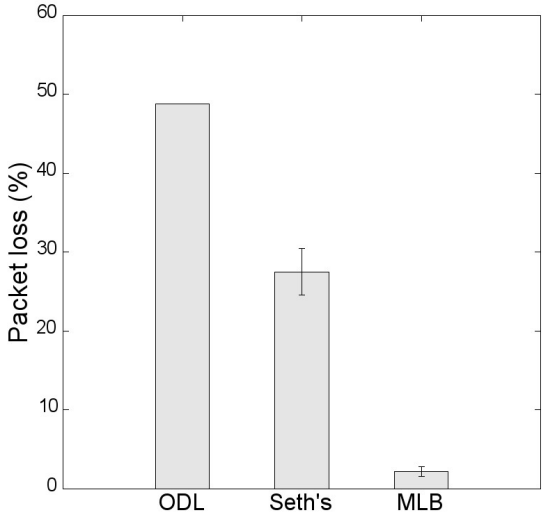


Figure 6. Packet loss for ODL, Seth's and MLB mechanisms when the number of flows is 2.

5.2 Four-Flow-Experiments

Again, condition1 and condition2 parameters of the MLB mechanism are varied to evaluate their impact on the aggregate throughput but using the TCP-4f traffic model. We use the same set of values as in Section 5.1.

Figure 7 shows similar results for the aggregate throughput, except when condition2 is equal to 70%. This result can be attributed to the difficulty of finding a new path with less than 30% of the occupation of the current path in use, a very low value for a scenario with a considerable number of flows and, consequently, a considerable traffic volume. We maintain the first condition equal to 50% and the second one equal to 10% for the following experiments due to the same reasons presented in Section 5.1.

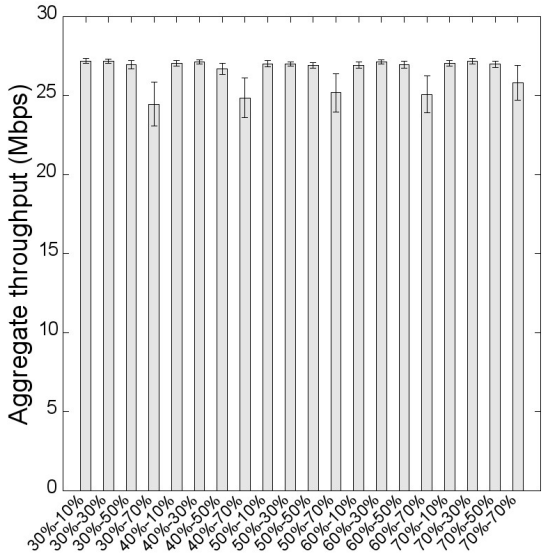


Figure 7. Aggregate throughput for MLB when different values of the pair condition1-condition2 and four flows are used.

]

Also taking into account the ODL controller and Seth's mechanism, **Figure 8** presents the aggregate throughput re-

sults. It is possible to observe that the aggregate throughput using ODL is below 10 Mbps. Without load balancing, the three main flows and the background flow are routed through the default path.

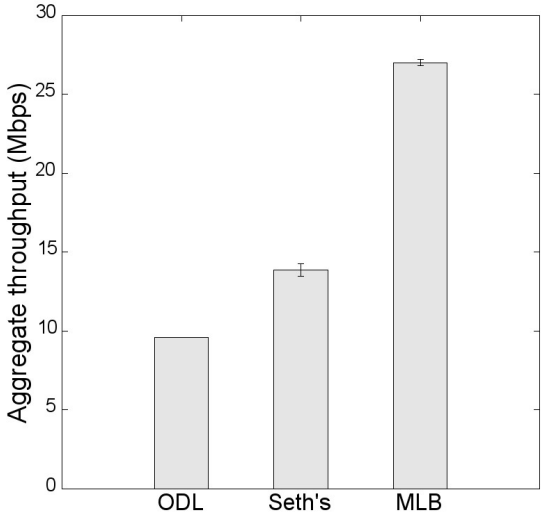


Figure 8. Aggregate throughput for ODL, Seth's and MLB mechanisms when the number of flows is 4.

In Seth's mechanism, load balancing allows the aggregate throughput to approach 14 Mbps. In this mechanism, only one of the main flows is balanced, the other unbalanced main flows are routed by the default path formed by nodes 1-2-6-8 together with the background flow, as can be seen in **Figure 9**. In addition, during balancing, the balanced main flow is switched to the default path, starting the bandwidth dispute with the other flows. This occurs because the mechanism always verifies which of the paths presented the lowest volume of transmitted data. In this way, any slight variation in the values of the data volume makes Seth's mechanism perform the switching between them, a situation that tends to occur with great frequency.

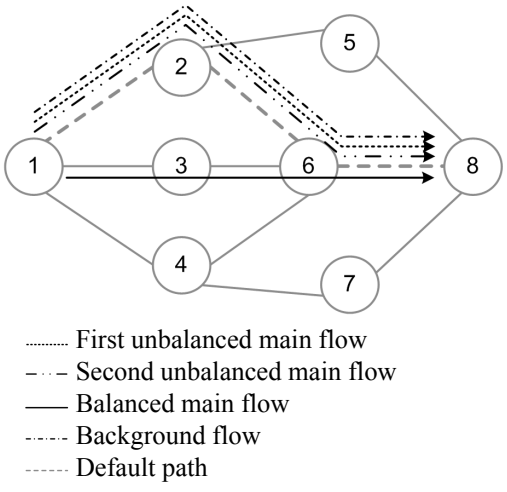


Figure 9. Switching for one of the main flows to the path 1-3-6-8 using Seth's mechanism.

In the MLB mechanism, the aggregate throughput reaches 27 Mbps. This result can be attributed to the following facts: load balancing is applied to the three main flows, the computed paths do not share links, and the switching control function prevents switching from occurring when its operating conditions are not met (see Section 4.2). One of the main flows has an average throughput of 8.8 Mbps, the other two main flows have an average throughput of 8.5 Mbps and the background flow reaches 1.2 Mbps. According to the path switching example shown in **Figure 10**, assume the following. The first main flow is switched by the path formed by nodes 1-4-7-8. The second main flow is switched by the path formed by nodes 1-3-6-8, performing bandwidth dispute with the background flow in the link composed by nodes 6-8. And the third main flow is switched by the path formed by nodes 1-2-5-8, performing dispute with the background flow in the link formed by nodes 1-2. This configuration of flow switching and similar ones (each flow following a different disjoint path) provide a high value of aggregated throughput. **Table 2** presents one execution of the MLB algorithm during 10 rounds, with each round corresponding to one execution of the while loop (Lines 7 to 25 of the Algorithm 2). As previously stated, our algorithm does not change the path of the background traffic, so this traffic is not shown in the table. At Round 0, all three main flows use path 1-4-7-8 since its occupation was zero (the background traffic uses path 1-2-6-8). Then at Round 1 first and third flows change to other paths since their occupations are lower. At this moment, all main flows use disjount paths. This behavior is maintained until Round 8. At Round 9, as the first main flow and the background traffic were disputing the link 1-2, one or both the flows reduce the sending rate and so the third flow starts to use the path 1-2-5-8. Nevertheless, at Round 10, the first main flow moves to the path 1-4-7-8 and again all three flows use disjoint paths. Considering **Table 2**, disjoint paths are used for most of the time. It is worth mentioning that its aggregated throughput exceeds by 181% the aggregate throughput presented by the ODL and by 94% Seth's one.

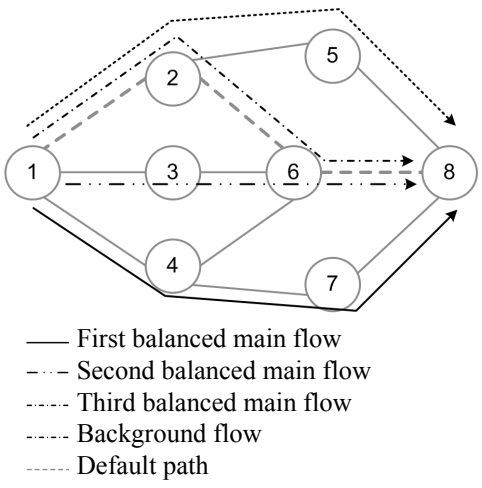


Figure 10. Example of switching for the main flows using the MLB mechanism.

Figure 11 presents the fairness index results for the TCP-4f traffic model. ODL reaches 0.99, which means that the four flows similarly occupied the default path bandwidth. The average throughput of each flow reached 2.3 Mbps.

Table 2. Example of switching for the main flows using the MLB mechanism in function of time.

Round	Time (hh:mm:ss)	Main flow	Path	Changed?
1	00:00:00	First	1-4-7-8	-
	00:00:28	Second	1-4-7-8	-
	00:00:56	Third	1-4-7-8	-
2	00:01:23	First	1-2-5-8	Yes
	00:01:50	Second	1-4-7-8	No
	00:02:18	Third	1-3-6-8	Yes
3	00:02:45	First	1-2-5-8	No
	00:03:12	Second	1-4-7-8	No
	00:03:39	Third	1-3-6-8	No
4	00:04:06	First	1-2-5-8	No
	00:04:33	Second	1-4-7-8	No
	00:05:00	Third	1-3-6-8	No
5	00:05:28	First	1-2-5-8	No
	00:05:55	Second	1-4-7-8	No
	00:06:22	Third	1-3-6-8	No
6	00:06:49	First	1-2-5-8	No
	00:07:16	Second	1-4-7-8	No
	00:07:43	Third	1-3-6-8	No
7	00:08:10	First	1-2-5-8	No
	00:08:38	Second	1-3-6-8	Yes
	00:09:05	Third	1-4-7-8	Yes
8	00:09:32	First	1-2-5-8	No
	00:09:59	Second	1-3-6-8	No
	00:10:27	Third	1-4-7-8	No
9	00:10:54	First	1-2-5-8	No
	00:11:21	Second	1-3-6-8	No
	00:11:48	Third	1-2-5-8	Yes
10	00:12:16	First	1-4-7-8	Yes
	00:12:43	Second	1-3-6-8	No
	00:13:10	Third	1-2-5-8	No

The use of Seth's load balancing results in a fairness index close to 0.89. This can be explained by the fact that it balances only one main flow, the other flows are routed through the standard path formed by nodes 1-2-6-8. When the balanced flow shares the bandwidth with the other flows, either because it is on the standard path or because it is on a path that has a shared link, the bandwidth occupation between the flows tends to be equal. However, when the balanced flow is switched to a path in which there is no link sharing with the other flows, such as the path formed by nodes 1-4-7-8, as shown in **Figure 12**, the balanced flow tends to occupy the entire bandwidth of this path. With this, its bandwidth occupancy becomes higher when compared with that reached by the other flows that continue to share the bandwidth of the standard path. In this case, the three flows have similar throughputs of 2.8 Mbps while the balanced main flow reaches an average throughput of 5.5 Mbps.

For the MLB load balancing, the fairness index has reached 0.81. Although the load balancing of the three main flows allows each flow to be routed through one of the three paths formed by disjoint links, the background flow routed through the default path consumes the bandwidth of two of the paths with disjoint links, paths 1-2-5-8 and 1-3-6-8 (see **Figure 10**). That way, the flows that are switched to these two paths compete for bandwidth with the background flow routed through the default path. In this case, the background flow has an average throughput of 1.2 Mbps, while one of the balanced flows has an average throughput of 8.6 Mbps

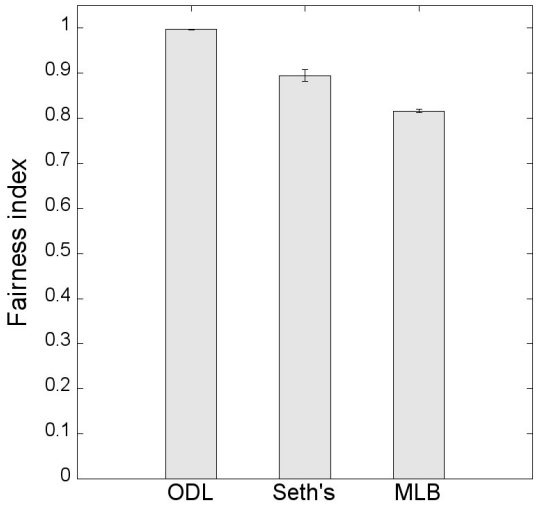


Figure 11. Fairness index for ODL, Seth's and MLB mechanisms when the number of flows is 4.

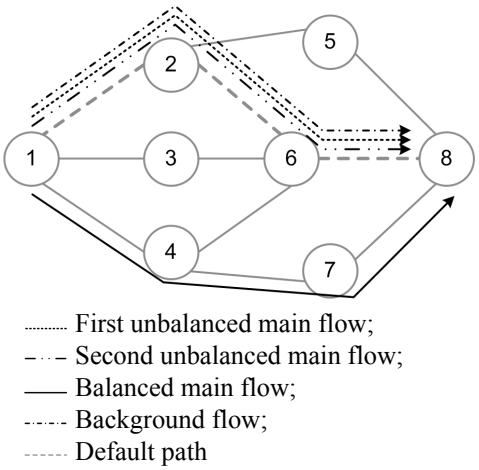


Figure 12. Switching for one of the main flows to the path 1-4-7-8 using Seth's mechanism.

and the other two have an average throughput of 8.5 Mbps.

Figure 13 presents the packet loss results for the UDP-4f traffic model. The results presented in this figure are similar to those obtained in the experiment with two flows, but with higher loss values, since now four flows are transmitted. The difference between the ODL and Seth's mechanism decreased. The ODL reached a 74%-packet loss and Seth's mechanism has a loss close to 65%. This is likely due to the congestion caused by the simultaneous transmission of the four flows. Seth's mechanism turns out to behave very similarly to ODL. This is because, as already mentioned, in Seth's mechanism only one main flow is balanced; the rest are transmitted through the standard path, similar to ODL. The MLB packet loss rate is close to 42%. This is because the three main flows are switched to less busy paths. MLB's load balancing and switching control allow more packets to be delivered to their destinations over less congested paths.

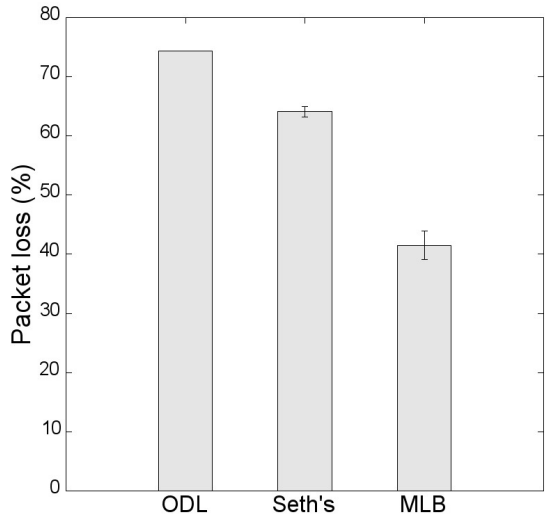


Figure 13. Packet loss for ODL, Seth's and MLB mechanisms when the number of flows is 4.

6 Conclusion

This work has proposed a load balancing mechanism between paths that includes a switching control function. We conclude that the MLB mechanism is a good solution to avoid idle links caused by protocols such as STP and consequently contributes to increasing the aggregate bandwidth of the network. An evaluation comparing pure ODL, Seth's, and MLB mechanisms is performed. The MLB mechanism outperforms the standard ODL mode of operation and Seth's mechanism. This better performance can be attributed to the use of path computation with disjoint links and the switching control function, which is based on the occupation of the path by the data load. The use of paths with disjoint links can avoid bandwidth sharing by flows and consequently can lead to a reduction in packet loss. Switching control, on the other hand, prevents flow switching to paths with saturated occupancy levels that may cause congestion in the network instead of avoiding it.

As a direction for future work, new load balancing experiments with the MLB mechanism in a real environment can be performed, in which hardware switches compatible with the OpenFlow protocol and data traffic generated by real hosts are necessary. A second path to be followed is to use more than one network controller simultaneously. In this way, it is expected to make the network more robust and tolerant to faults caused by unpredictable situations, such as the drop in communications between the switch and the network controller, which may be caused by unavailability in the server that hosts the controller. Furthermore, with the use of more controllers, it is possible to increase the scalability of the network by adding new hosts. In this way, it will be possible to monitor the performance of the MLB mechanism in a network composed of a larger number of switches and with communications between sources and destinations through a larger number of hops. In this network, a greater number of disjoint paths and a greater number of flows can be used. Moreover, we need to investigate an approach to fine-tune condition1 and condition2 values in function of the network traffic. We also aim to evaluate the complexity of both algorithms. Moreover, in our performance evaluation, we assume that the network links have no failure or intermittency. We aim to evaluate our mechanism by taking into account link failures. At last, it is not easy to obtain near-real-time information regarding link occupation. For this, we can use solutions such as in-band network telemetry.

Declarations

Funding

This research has been funded in part by CNPq and FAPERJ.

Authors' Contributions

ACS and MGR have contributed to the conception of this study. ACS has performed the experiments. ACS is the main contributor and MGR is the main writer of this manuscript. All authors have read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

Our mechanism implementation code and result data are available at https://github.com/alisson2000rj/MLB.

References

Amiri, E. and Javidan, R. (2019). A new method for layer 2 loop prevention in software defined networks. *Telecommunication Systems*, 73(1):47–57. DOI: 10.1007/s11235-019-00594-4.

Bhandarkar, S. and Khan, K. A. (2015). Load balancing in software-defined network (SDN) based on traffic volume. *Advances in Computer Science and Information Technology (ACSIT)*, 2(7):72–76. Available at:`https://www.ttcenter.ir/ArticleFiles/ENARTICLE/3833.pdf`.

Bredel, M., Bozakov, Z., Barczyk, A., and Newman, H. (2014). Flow-based load balancing in multipathed layer-2 networks using openflow and multipath-TCP. In *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 213–214, New York, NY, USA. ACM. DOI: 10.1145/2620728.2620770.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, 3 edition. Book.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy)*, pages 11–15, Pasadena, CA USA. Available at:`https://www.osti.gov/biblio/960616`.

Hamdan, M., Hassan, E., Abdelaziz, A., Elhigazi, A., Mohammed, B., Khan, S., Vasilakos, A. V., and Marsono, M.

(2021). A comprehensive survey of load balancing techniques in software-defined network. *Journal of Network and Computer Applications*, 174. Article 102856. DOI: 10.1016/j.jnca.2020.102856.

Hassan, M. H. O. (2017). Implementing Nayan Seth's dynamic load balancing algorithm in software-defined networks: a case study. Technical report, Sudan University of Science and Technology, Master's thesis, Sudan University of Science and Technology, Sudan.

Jain, R. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1 edition. Book.

Kawaguchi, E., Kasuga, H., and Shinomiya, N. (2019). Unsplittable flow edge load factor balancing in SDN using P4 runtime. In *29th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. DOI: 10.1109/ITNAC46935.2019.9077984.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-Defined Networking: a comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76. DOI: 10.1109/JPROC.2014.2371999.

Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of Workshop on Hot Topics in Networks (SIGCOMM)*, Hotnets-IX, pages 1–6, New York, NY, USA. ACM. DOI: 10.1145/1868447.1868466.

Liu, W., Zhou, W., Wang, Y., Duan, Y., and Gao, Z. (2016). Flexible multi-path routing for global optimization in software-defined datacenters. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 190–195. DOI: 10.1109/ISCC.2016.7543738.

Mallik, A. and Hegde, S. (2014). A novel proposal to effectively combine multipath data forwarding for data center networks with congestion control and load balancing using Software-Defined Networking approach. In *International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1–7, Chennai, India. IEEE. DOI: 10.1109/ICRTIT.2014.6996178.

OpenDaylight (2022). Opendaylight - The L2 Switch project provides Layer2 switch functionality. Available at:`https://docs.opendaylight.org/en/stable-fluorine/user-guide/l2switch-user-guide.html`.

Prabhavat, S., Nishiyama, H., Ansari, N., and Kato, N. (2012). On load distribution over multipath networks. *IEEE Communications Surveys & Tutorials*, 14(3):662–680. DOI: 10.1109/SURV.2011.082511.00013.

Ramdhani, M. F., Hertiana, S. N., and Dirgantara, B. (2016). Multipath routing with load balancing and admission control in software-defined networking (SDN). In *International Conference on Information and Communication Technology (ICoICT)*, pages 1–6, Bandung, Indonesia. IEEE. DOI: 10.1109/ICoICT.2016.7571949.

Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G., and Sarefo, S. (2020). Intelligent load balancing techniques in software defined networks: a survey. *Electronics*, 9(7). DOI: 10.3390/electronics9071091.

Seth, N. (2022). SDN load balancing. Available at:`https://github.com/nayanseth/sdn-loadbalancing`.

Singh, S. K., Das, T., and Jukan, A. (2015). A survey on internet multipath routing and provisioning. *IEEE Communications Surveys & Tutorials*, 17(4):2157–2175. DOI: 10.1109/COMST.2015.2460222.

Sun, Z., Xie, Z., Chen, Z., and Dai, L. (2012). An algorithm for the shortest pairs of arc-disjoint paths problem. In *International Conference on Natural Computation (ICNC2012)*, pages 1001–1006, Chongqing, China. IEEE. DOI: 10.1109/ICNC.2012.6234600.

Tsai, J. and Moors, T. (2006). A review of multipath routing protocols: from wireless ad hoc to mesh networks. In *ACoRN Early Career Researcher Workshop on Wireless Multihop Networking*, Sydney, Australia. Available at:`https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a503452eeac9fb6b302e2d9af1a63361312da886`.

Zhao, Y., Wang, X., He, Q., Zhang, C., and Huang, M. (2021). Plofr: An online flow route framework for power saving and load balance in SDN. *IEEE Systems Journal*, 15(1):526–537. DOI: 10.1109/JSYST.2020.3010971.