University of Sao Paulo
Institute of Mathematics and Statistics
Bachelor Degree of Computer Science

Leonardo de Carvalho Freitas Padilha Aguilar

Analysis of the use of SDN for load balancing

São Paulo
December of 2018

Analysis of the use of SDN for load balancing

Final monograph for
MAC0499 – Supervised Graduate Project.

Supervisor: Prof. Dr. Daniel Macêdo Batista

São Paulo
December of 2018

Abstract

Software defined networking (SDN) is a recent feature that allows the creation, control and customization of the network through the use of software, unlike the traditional model where the network was fixed and defined by hardware. This paradigm change is accomplished by the existence of a central entity in the network, called controller, which is responsible for sending the rules of forwarding, modifying and/or dropping of packet flows to the switches. The purpose of this work is to analyze the use of SDN for load balancing by developing a balancer that can execute three different algorithms, giving to the user the possibility to choose and change (at run time) which will be used, as well as their parameters.
Keywords: SDN, load balancer, load balancing algorithms.

Contents

List of Abbreviations

ADC	Application Delivery Controller
ADN	Application Delivery Network
API	Application Programming Interface
ARP	Address Resolution Protocol
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DNS	Domain Name System
HLD	Hardware Load Balancer Device
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
OF	OpenFlow
OSI	Open System Interconnection
PHP	PHP: Hypertext Preprocessor
QoE	Quality of Experience
RAM	Random Access Memory
SDN	Software defined networking
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

List of Codes

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Contextualization

Load balancing in computer networks, according to Moharana (2013), is an important technique that seeks to ensure the optimization of existing resources for better distribution of data packets, avoiding equipment overload and decreasing applications response time. When applied at application layer, the technique is based on choosing the most appropriate server to respond a particular request and it can be done in several ways, Currently the most common, according to Gandhi et al. (2014), is the implementation of a specific hardware that aims to forward the data packets to the server in best condition (this selection is defined by a distribution algorithm that can vary from device to device). However, this implementation can cause several types of limitations, such as lack of flexibility in the choice of the algorithm or the number of servers supported by the network, besides the cost of its application being quite high, being able to reach U$ 2000[1].

On the other hand, with the growing of the software defined networking (SDN), the establishment of behavior and data traffic within the network do not need to depend only on the hardware, conforming to Esteve Rothenberg et al. (2018). This ensures greater customization and adaptation that varies according to network conditions (for example available servers, packets meta-data, etc.) in contrast to the previous rigid model. The use of this recent technique takes place in several aspects, ranging from the academic scope, to the development and testing of new network protocols, as well as in the commercial scope, to reduce costs and greater control over the network.

Based on this, it is possible to assume that the application of the SDN technique may be a more flexible alternative and with a lower implementation cost when compared to the use of hardware for load balancing.

1.2 Objectives and contribution

The objective of this work will be to analyze the performance (focusing on the response time) and the cost of using SDN and hardware for load balancing.

To perform this analysis, the project propose to develop an SDN application that contains three different algorithms for balancing and allows the network administrator to choose which should be executed, as well as their due parameters. This will allow greater control of the

[1]The Barracuda Load Balancer ADC 240 was chosen as the price parameter. Available in <http://www.zones.com/site/product/index.html?id=100713342>. Accessed on 04/22/2018.

entire process of balancing to the user, something that is not allowed with the physical devices commonly used for this task.

As a contribution, the application for SDN architecture that implements load balancing will be available as open source software so that other people can use it in studies about the subject and eventually improve it by adding new features.

1.3 Structuring of the monograph

At Chapter 2, it will be presented the basic concepts that involve this work, that is, the load balancing, the software defined networks and the network emulation, that will be used to execute the performance analysis.

The Chapter 3 focuses on load balancing using hardware, exposing how the task is performed, and listing the different types of devices. Finally, a general performance and the cost of applying them will be presented.

In Chapter 4 it is explained how the load balancer was developed using SDN technique, also presenting the application of the balancer and the balancing algorithms that were inserted.

The Chapter 5 is directed to the execution and results obtained from the performance analysis of the balancer using SDN.

Finally, Chapter 6 will present the final considerations about the work developed and will point out some ways of deepening the research theme worked on in this monograph.

Chapter 2

Theoretical foundation

2.1 Load balancing

According to Moharana (2013), load balancing is a widely used mechanism to better leverage all available resources, optimizing the use of it and the time of execution of the tasks. Its application is as much to distribute the work load between specific resources of a computer, as hard disks, as for servers of network in general, in order to guarantee that the best machine (that is, the one with more availability) respond to a particular request.

With the popularization of high-speed internet, the demand for a better quality response to the services available increased dramatically, and for that to be possible, applications began to no longer invest in improving their single server (by increasing the available memory, for example), but to add new machines, thus giving rise to the concept of server farm, that is, several computers with a single task: to respond client requests for a given application, providing a more economical alternative when compared to improvement of a single server.

In this sense, load balancing is an important technique because it is responsible for taking this request to one of the corresponding servers, so that it can use all available processing capacity.

Several methods can be used to ensure this balancing, as will be discussed in the following topics.

2.1.1 On the client side

For this type of balancing, a list is given with the IP addresses[1] available to the client that is in charge of selecting the server with which to communicate. This has several security implications because the client will have direct access to the available servers as well as optimization problems because the application can not guarantee that the client will always choose the best server at that moment. For example, if a large number of clients decide to communicate with server number 1, this will create an overhead on the server and consequently decrease the response time.

2.1.2 Using DNS

In this method, Domain Name System (DNS) is used by associating a domain with the list of IP addresses that are available in the server farm. The DNS then selects the next server using the round-robin algorithm (the addresses are associated to a queue and when there is a request, the one that is at the beginning of the queue is used, removing it and

[1]Devices connected to the Internet are identified at the network layer by their IP address.

re-positioning it the end). An example of how topology works for this kind of balancing can be seen in Figure 2.1. The client requests the DNS server to resolve the domain of the application (Flow 1 in Figure 2.1) and it is responsible for choosing one of the available servers, with which the client will communicate and send the data (Flow 2 in Figure 2.1).

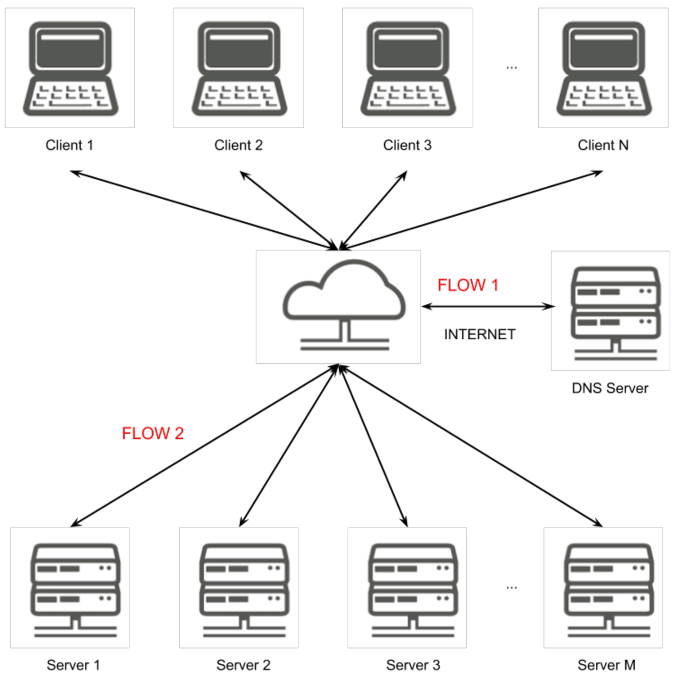

Figure 2.1: *Load balancing using DNS.*

2.1.3 Using load balancers devices on servers

This is the most commonly used method nowadays. The application has a hardware (the balancer) that will be responsible for listening to clients and, when a request arrives, it is his responsibility to redirect to one of the server from the server farm.

This balancer is the single point of contact of customers with the application (see Figure 2.2), ensuring that clients do not know about the functionality division of the servers and can not contact them directly, which is a good solution in terms of security.

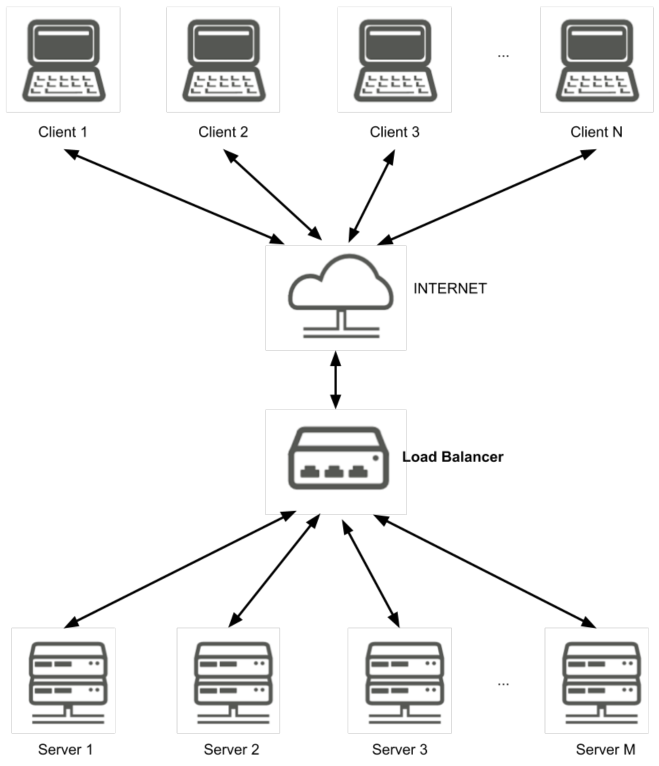

Figure 2.2: *Server-side load balancer.*

2.2 Load balancing algorithms

To ensure a better quality of load balancing between servers, it is very common for balancers to use algorithms that go beyond a simple random choice to select the next server that will respond the next request.

There is no technique that can be considered preferable, since each algorithm has its peculiarity, being able to act better in different scenarios. For this reason, some manufacturers use a combination of more than one algorithm in their devices, increasing the scenarios in which the distribution is most efficient.

The following three algorithms will be using during the project.

2.2.1 Random

One of the simplest techniques of load balancing is to choose one of the servers randomly for each client request.

The random algorithm can allow a good load distribution for servers with the same characteristics because it guarantees that servers receive requests in an equiprobable way (provided that the random number generator used follows a uniform distribution).

2.2.2 Round-robin

As in DNS balancing, which was seen in the previous section, the round-robin algorithm makes use of an IP address queue and, whenever a client makes a request, the server whose address is in the beginning of the queue is selected, then it is removed from that position and inserting it at the end. Figure 2.3 exemplifies a scenario using the round-robin algorithm, in this example it is possible to see a balancer with three different servers. The first request arrives and is sent to the first server (which IP address is 10.0.0.1), the second to the second server (which IP address is 10.0.0.2) and the third server to the last server (IP 10.0.0.3). The next request will then be sent to the IP server 10.0.0.1, because it is now the first in the queue.

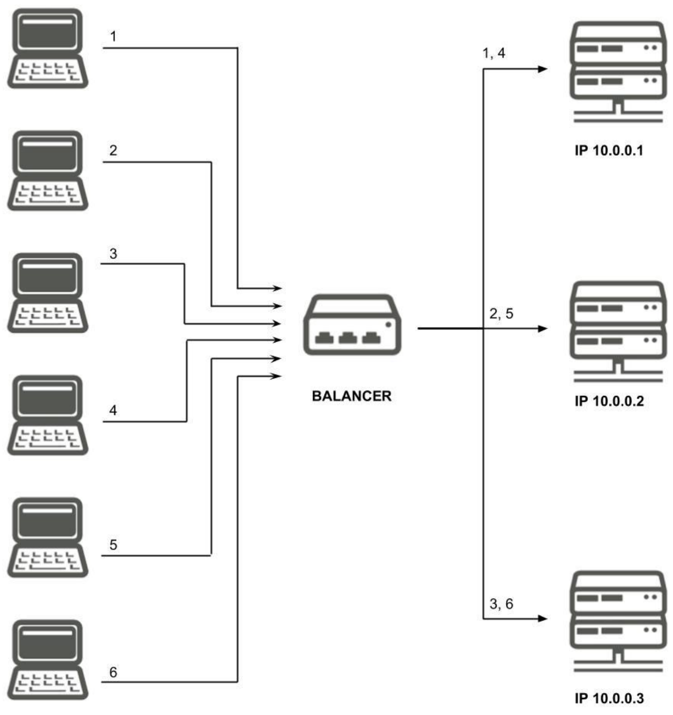

Figure 2.3: *Example of a round-robin algorithm for load balancing.*

This technique is interesting for server farm with computers that basically have the same specifications (for example the same amount of RAM, same CPU, etc.) because it distributes the load in an equivalent way, ignoring these characteristics.

For cases in which the servers have different characteristics, weights can be added to the servers and, therefore, machines with larger weights will receive more load than machines with smaller weights, thus allowing machines with more capacity to receive more requests than the others. Figure 2.4 exemplifies a scenario with the use of the round-robin algorithm with weights. In it, it can be seen that the first server has weight 3 (soon, receives the first three requests), the second has weight 1 (receiving the fourth requests) and the third weight 2 (receiving the last 2 requests).

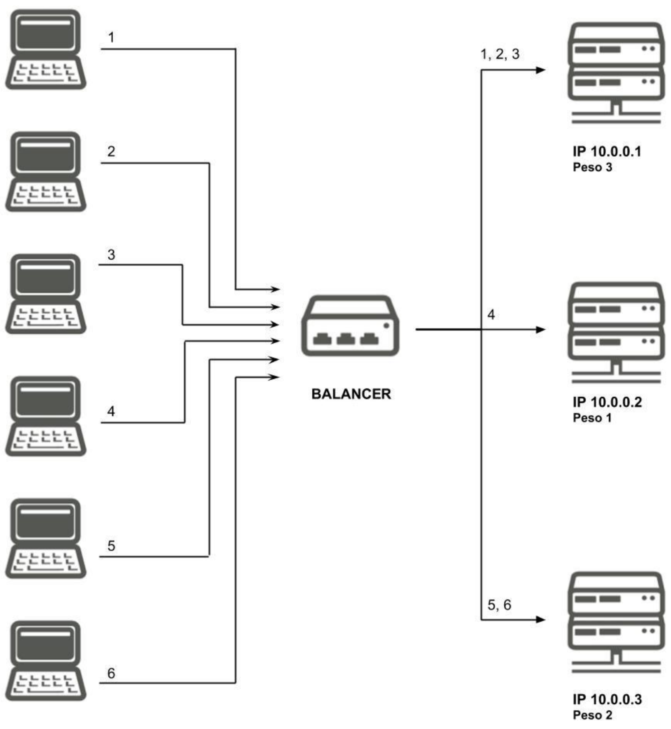

Figure 2.4: *Example of the weighted round-robin algorithm.*

2.2.3 Least-bandwidth

The least-bandwidth algorithm selects the server with the lowest network traffic consumption to respond the next request. The balancer parses and stores the amount of bytes that are transmitted to each server and can then determine the next machine (the one that transmitted the least data so far). Figure 2.5 exemplifies a scenario using the least-bandwidth algorithm, the servers have initial traffic of 4 MB for the server with IP address 10.0.0.1, 2 MB for the server with IP address 10.0.0.2 and 1 MB for the server 10.0.0.3. When the request is taken to some server, the number of bytes related to that packet is added in the total traffic of that machine.

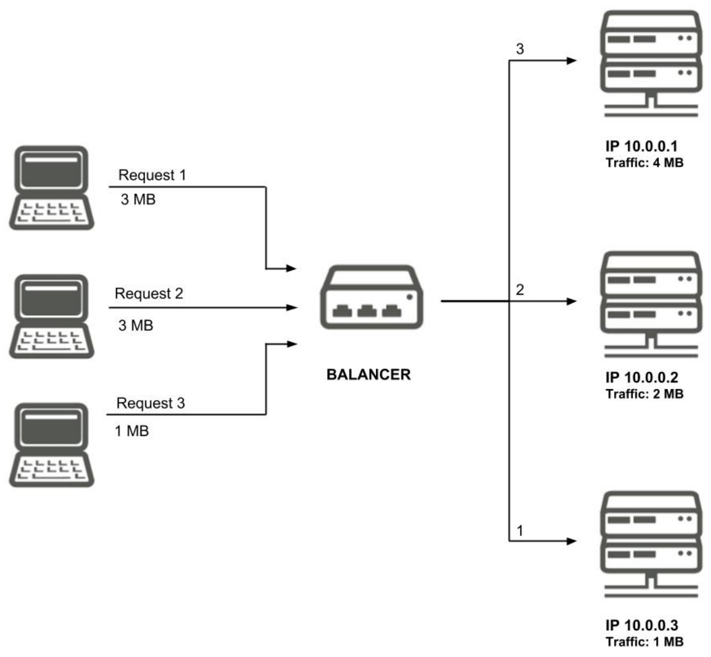

Figure 2.5: *Example of the least-bandwidth algorithm.*

Like the round-robin method, the least-bandwidth can also accept weights on the servers. If w_i is the weight of the server i and b_i the network traffic consumption of it, then the machine with the lowest ratio $\frac{b_i}{w_i}$ is selected to respond the next request, thus, machines with larger weights receive more load than machines with smaller weights.

2.3 Software defined networking

Software Defined Networking (SDN) is a concept of decoupling the data plane (the layer that carries the user data) and the control plane (the layer responsible for routing the data) of the traditional networks in such a way that it is possible to define the routing of the data using software.

For years the architecture of the networks was defined only by physical devices where these planes were indivisible, that is, the routing of packets of the routers and switches[2] was managed by a layer of control created by the manufacturer, without the possibility of change by the network administrators. This control plan was built into the equipment itself. Thus, a network with n interconnection equipment could have n different control plan rules, one for each device, not necessarily compatible with each other.

With computing becoming more and more dynamic, the need arose to separate these two layers, giving more power to the administrator and allowing the network to be more adaptable and centralized in the control plan.

With SDN, the routing definition is allowed to stay on a dedicated server with high throughput and this is responsible for adding routing rules for packets on each network device (such as allowing the flow of a particular IP address, blocking a particular port, etc.).

[2]The switch is a network device capable of connecting computers, allowing communication between them.

This is of great value not only in the research area, where it is possible to use SDN as a tool to test new protocols without the manufacturer exposing the internal operation of equipment (Duque et al., 2012), but also in the commercial scope, the complexity of the internal network of applications.

The SDN architecture divides the networks into three distinct layers, superimposed conceptually one above the other, which have specific tasks and communicate only with their adjacent layer, as can be seen in Figure 2.6.

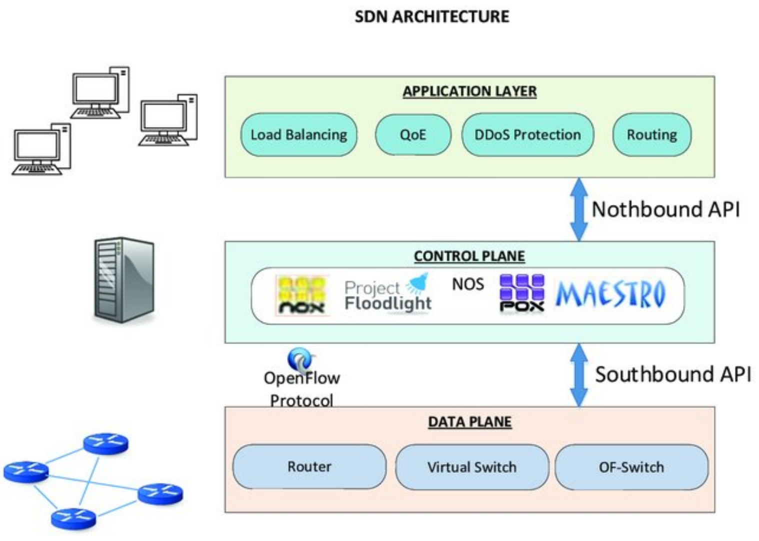

Figure 2.6: *SDN Architecture.(Fernández et al., 2018, p.5)*

The first layer, which is at the topmost level, is the **application** layer. The applications are software that can communicate with the layer below in order to define the behavior of the network flows. The balancer to be defined throughout this work will act on this layer.

The middle layer is called the control layer (commonly referred to as the **SDN controller**). It is characterized by communicating with both the level below and the level above, giving the upper layer an abstraction of the routing devices and allowing their flows to be modified by it.

The layer below is the **data** layer (or layer of the network devices), whose function is to only receive the data packets, perform some action with them and update their internal counters (general statistics such as width of band flow).

Communication between the application layer and the controller (or Northbound API), according to Open Network Foundation (nd), usually provides abstract views of the network and allows the direct expression of behavior and network requirements. Already the communication between the layers of lower level with the controller (also known as Southbound API) is made through a protocol like the **OpenFlow**, that allows the control of the flows and the flow table of the devices.

2.3.1 OpenFlow architecture

The OpenFlow architecture is one of the most widespread software defined networking architectures. It defines a series of patterns that allow the control of **flow tables** from each routing device.

Flow tables are responsible for storing network flow information, that is, they are tables whose entries define information such as routing rule (some specific value of the packet header), action (what to do if the rule is met), and general statistics on the flow, as in Figure 2.7.

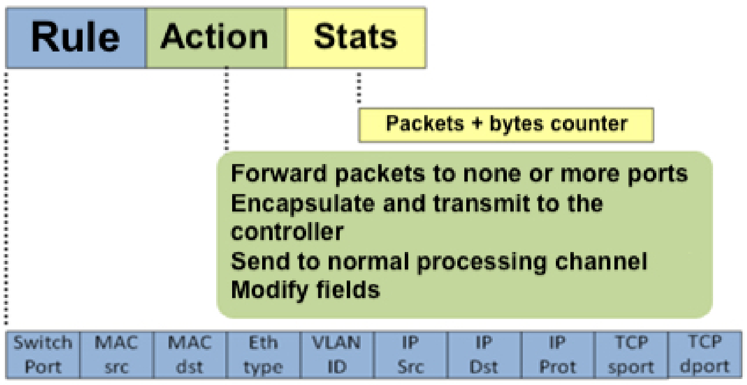

Figure 2.7: *Representation of a flow in a flow table.* (Duque et al., 2012)

In addition, the OpenFlow architecture also has a device called **controller** (which is, as previously seen, responsible for allowing an abstraction of flows to network applications), as well as a secure channel where the controller can communicate with the devices and a well-defined protocol for this communication, the OpenFlow protocol.

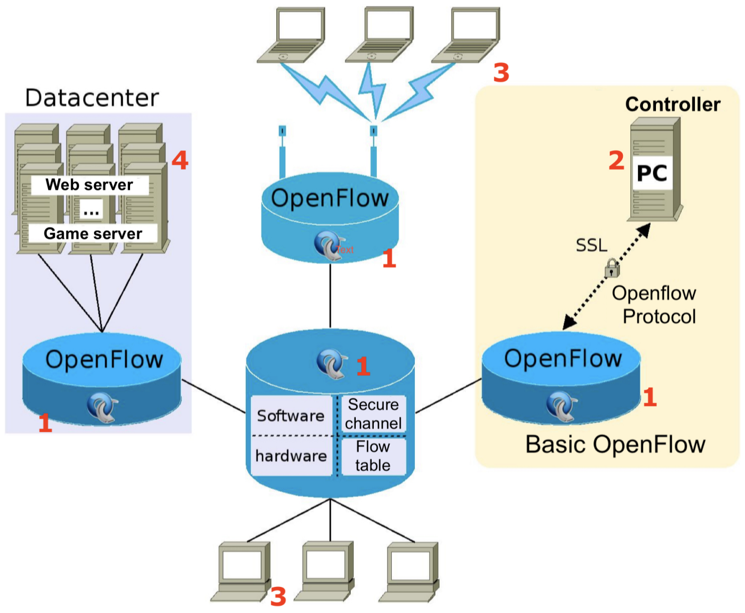

Figure 2.8: *Openflow Basic Architecture.* (Esteve Rothenberg et al., 2018)

In the example of Figure 2.8, the items identified by 1 are OpenFlow switches, devices responsible for network communication. These include the flow tables and the secure channel, along with the data plan and the control plan. The controller, which is identified by item 2, is the one who remotely manages packet forwarding. Communication between controller and switches is done through the secure channel using the OpenFlow protocol. Items 3 and 4 identify the end users of the network (Silva, 2016).

The packet arrives at the routing device and it checks if there is any flow whose rule fits with the packet. If so, the action is performed with this packet, otherwise the packet is sent to the controller and this allows the network applications to decide what to do with this packet, adding a small overhead to the data stream. When this new decision is made, it is stored in the routing devices to avoid heavy traffic with the controller and stays there for a certain time.

2.3.2 Controllers

As previously mentioned, controllers are the center of the control plane of software defined networks, acting as true network operating systems, where they manage the flow tables of the routing devices (Esteve Rothenberg et al., 2018) and abstract this concept for applications, making it easier to develop new features for networks.

There are several types of controllers in different types of languages, such as NOX, developed in C++ and Python, or Beacon, developed in Java and it allows use in the Android platform. However this work will focus on a third controller called POX, written in Python, because of its simplicity and better performance when compared to NOX.

2.4 Network emulation

A major problem when it comes to research in the area of networks are the methodologies used to perform tests and experiments, because a very large infrastructure is needed, with physical equipment interconnected in order to create an environment that can reproduce reality. In this sense, the concept of network emulation arises, which basically allows the creation of a virtual environment that emulates the characteristics of a real network, with routers, switches and links between them.

The emulation tool to be used in this work is Mininet, which acts as a collection of physical devices connected in a single Linux kernel (Silva, 2016). It also allows the creation of network topologies with ease and with full support of OpenFlow technology in such a way that the code developed for Mininet can be implemented in the real world with minimal changes (Mininet, nd).

Chapter 3

Hardware load balancing

The hardware load balancing (also known as HLD) occurs through a physical device, the balancer, that connects physically with the available servers, positioning itself usually between the external network and the internal network of the organization that provides the service to be balanced. This device has its own processor, memories and network interfaces to receive and distribute the packages in the best possible way.

This was the only method used in the 1990s, conforming Sekar (2017), since the use of a way that involving software was very difficult and slow because the hardware was not evolved (according to Figure 3.1, in 1995, the RAM had average capacity of approximately 100Mb, which has a negative impact on the performance of the programs), requiring a device with focus on load distribution to optimize this task.

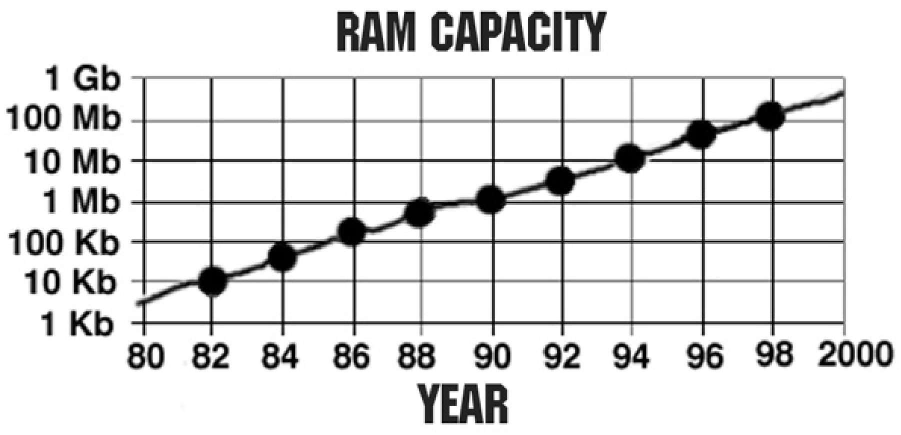

Figure 3.1: *Graph of RAM memory capacity per year. (Pagano, 2012, p.55)*

The use of operating systems focused on load distribution, specialized network firmwares and also optimized distribution algorithms for particular hardware are characteristics that allow greater efficiency for the execution of the balancing. But not all balancers have all these features. Some, such as the TP-Link TL-R480+, are devices that have only one network interface that supports multiple protocols and a simple routing policy, without the need for software support. [1] This shows how much this market is varied, having several types of balancers that serve different groups of users.

Because they can act on Layer 4 (Transport) and Layer 7 (Application) of the OSI model

[1]The information presented here about TP-Link TL-R480+ load balancer is available at <https://www.tp-link.com/br/products/details/cat-4910_TL-R480T+.html>. Accessed on 08/16/2018.

(the standard that defines a characterization of computer networks in layers), it is common to classify the routers in L4 or L4/L7, depending on their layer of action.

Layer 7 routers can access the content of the message (URL and cookies, for example) because the HTTP protocol is part of the application layer, and therefore, these routers form an Application Delivery Network (ADN) where delivery is based on the content of that layer, for example, PHP scripts for a set of specific servers and HTML pages for another set. Because of this set of available resources, L4/L7 routers are generally much more expensive than L4 routers, which act only on the transport layer, performing routing of packets.

The use of hardware for balancing also gives the organization greater security because the device can not be accessed physically by people who do not have access to the dependencies of the place where it is maintained.

In addition to security, the devices also have the ability to enable Secure Sockets Layer protocol[2] acceleration, so decoding of data would happen on the balancer before passing the request forward. This allows servers to not spend extra computing time on decryption, thereby improving their performance.

However, there are several problems of physical device implementation for this task. The first and most significant is the very high cost, both for acquiring a next-generation balancer and for maintaining the physical network.

In addition, the devices have a limit of servers that can be connected to them, which makes it difficult to scale the system because if it is necessary to add more machines than the limit to server farm, we must add one more balancer or change it for one that has a larger limit, making the use of hardware for this task less flexible for changes.

3.1 Cost of devices

The cost of the devices is quite varied, as stated above. It all depends on its features, the amount of traffic that is estimated, and the number of servers available. Table 3.1 lists some load balancers, their features and their respective prices.

Device	**SSL Acceleration**	**Max Throughput (Gbps)**	**Max Number of Servers**	**Price (USD)**
KEMP LM-2200	Yes	0.95	4	100
KEMP LM-X3	Yes	3.4	8	4,000
Citrix Netscaler MPX 7500	Yes	5	8	17,300
F5 LTM-2000S	Yes	5	8	17,300
Citrix MPX-8015	Yes	Not Published	12	48,000
KEMP LM-8020M	Yes	30	8	60,000
F5-BIG-BT-i7800	Yes	40	12	144,900

Table 3.1: *Table of characteristics of some physical devices.(KEMP Technologies, nd)*

[2]SSL protocol, the standard for establishing secure connections between the server and client

Chapter 4

SDN load balancing

4.1 SDN as a load balancer

As seen in Chapter 3, the use of a physical device to perform load balancing has a very high implementation cost (the price of good devices varies by thousands of dollars). To solve this problem, a SDN application will be proposed as a load balancer, reducing the cost and allowing the administrator greater flexibility when compared to the fixed model determined by the manufacturers of the balancers.

The application of SDN to solve this problem is quite clear, since the main characteristic of these devices is the understanding of the current state of the internal network and how the servers are behaving, exactly what the software-defined network technique allows.

It will then benefit from both the flexibility and ease that SDN allows to create a load balancer that performs 3 different algorithms for load balancing (random, round-robin and least-bandwidth) and that allows the administrator to select not only the algorithm but also the load ratio that each server will receive, all at run time using the terminal.

In this way, who will be responsible for receiving and redirecting the packages to the servers will be the switch OpenFlow. The controller will be responsible for receiving the administrator information and applying it to the switch, changing the configuration of the balancer, and selecting which server should receive the next request, as shown in Figure 4.1.

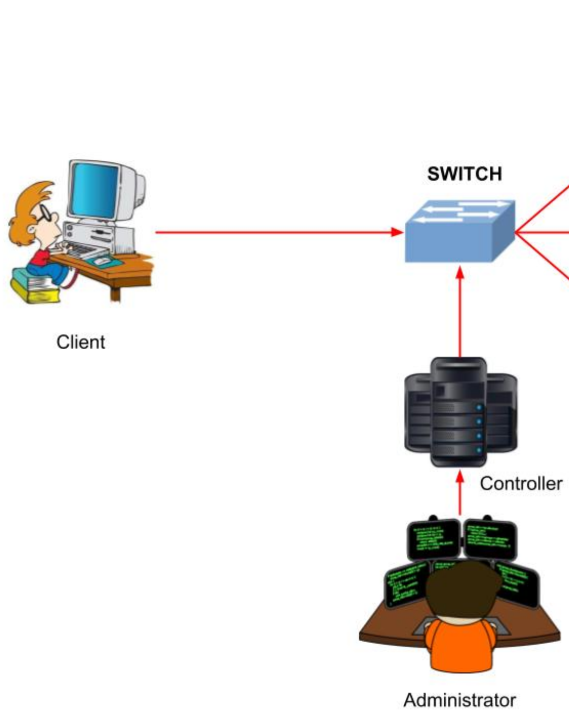

Figure 4.1: *Exemplification of the use of SDN for load balancing.*

To do this, the POX controller will be used, a controller written in Python, which allows a fast and easy development, widely used for prototyping projects in SDN.

The execution of the POX is also quite simple, as we can see in the Code 4.1. Simply run the command pox.py, passing as optional argument controller options and as a required argument at least one component name (a Python module) that will run, more specifically the SDN application that will communicate with the controller.

```
./pox.py [POX options] [C1 [C1 options]] [C2 [C2 options]] ...
```

Code 4.1: *Using the POX controller, where C1, C2, etc. are names of the controllers (Python modules).*

It will therefore need to create a POX component that will be the load balancer. Conveniently, the controller already has an example component that performs this task, but does not contain all the features that were previously defined. Therefore, this module will be used as the basis for the development of the balancer.

4.2 Balancing module

The POX balancing module is called *ip_loadbalancer* and can be found in the *misc* directory of the controller. In it, there are two classes: the MemoryEntry, an abstraction of the flows that are in the switches and also the ipbl class, containing the balancer logic itself. This class stores the IP addresses of the servers, the IP address of the controller, the flows that are active and the methods for selecting servers and for handling packets arriving at the controller.

The component currently receives the IP address on which the balancer will be used (as shown in the Code 4.2). In the example it can be seen that this address was set to 10.0.1.1) and the server address list (in the Code 4.2, this list is defined by 10.0.0.1, 10.0.0.2 and 10.0.0.3). The controller then uses this information from the servers to send an ARP message to these addresses from time to time to find out if there has been a crash on a server (the machine that has not responded to the message for a long time has probably crashed and must be removed from the list), thus performing the health checking of the servers.

```
./pox.py misc.ip_loadbalancer \\
ip=10.0.1.1 \\
servers=10.0.0.1,10.0.0.2,10.0.0.3
```

Code 4.2: *Example of running the ip_ loadbalancer component.*

The balancing of this module is done by TCP/IP protocols, that is, only packets of this protocol will be balanced between the servers (which are the most used protocols in web applications). When a packet of this protocol is received at the balancer IP address, the controller will be responsible for sending to one of the IP addresses of the list of active servers using the random algorithm, and when doing so, a copy of the flow is created in the controller by a period of time.

Figure 4.2 exemplifies how balancing is done using this component. In the example, the servers connected to the switch have IP addresses 10.0.0.1, 10.0.0.2 and 10.0.0.3 and the balancer itself has an external IP address of 10.0.1.1 and is directly connected to the internet. The packets that arrive on the balancer are sent to the controller and it will decide which server the switch should redirect to.

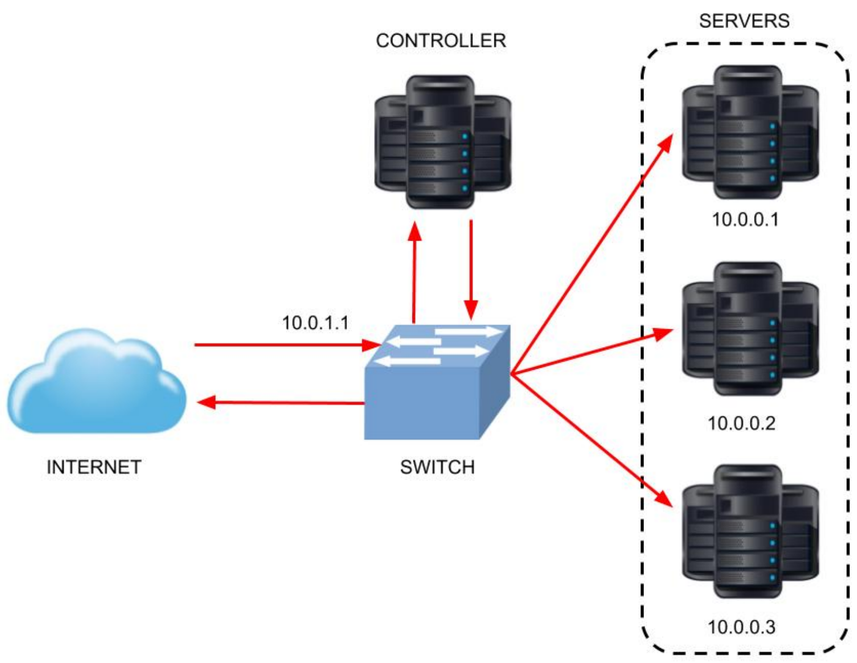

Figure 4.2: *Exemplification of the ip_ loadbalancer component.*

It will be added to this module two more load balancing algorithms (round-robin and least-bandwidth), both with support the weights per server, and will allow the administrator to change the algorithms at run time using a terminal in the controller.

4.3 Balancer development

The features of the balancer were listed below, in order of dependence:

1. Add the algorithms (round-robin e least-bandwidth);
2. Weight support in algorithms;
3. Add user interaction.

4.3.1 Round-robin algorithm

All new algorithms must be added in the _pick_server method, which aims to select the next server to meet the next request.

For the addition of round-robin specifically, it will first be needed the list of servers. The balancer class (*iplb*) already has this structure through the variable live_servers that has all the IP addresses of the active servers. That way, it only be needed to add a counter to know which index of that list is the server that will meet the next request.

In the constructor of the balancer, the index (which is a class variable of class iplb) starts with 0. Each new call of _pick_server, the IP address of that position will be retrieved to respond the request and finally, it will be added 1 to the index.

If the value of the variable exceeds the size of the list, it will be assigned 0 to it. Thus maintaining the circularity of the algorithm without adding new data structures to the balancer.

Finally, the user will be able to choose between the two possible balancing algorithms. For this, the support for a command line flag will be added, which may be random (if the user want the random algorithm, which is the default value of the flag), round-robin, or least-bandwidth. This value will be passed to the balancer and it will select the algorithm based on it.

4.3.2 Least-bandwidth algorithm

For this algorithm, in addition to the IP address list, it will also be necessary to have access to the amount of traffic transferred to the servers and thus select the one with the lowest value.

With the SDN technique, based on everything that has been exposed to this point, this is quite simple to do because traffic information can be obtained by analyzing the statistics of each stream that are on the switches. OpenFlow allows the controller to request these statistics through a specific request (the *StatsRequest*), so the controller needs to request this data continuously to update the traffic values for each server.

The balancer class will then have a dictionary where the keys are the IP addresses of the servers and the value is the amount of traffic to each server in the last 14 seconds (an arbitrary value it was chosen such that the controller does not spend as much time in charge to handle the statistics requests of the switches). In this way, a timer will be implemented so that every 14 seconds the controller sends the *StatsRequest* to each of the switches and so, when the response is received, the dictionary values will be updated for each of the servers involved in the flows.

4.3.3 Load ratio per server

Assuming that the class of the balancer receives in its constructor a dictionary where the key is an IP address and the value an integer that represents the proportion of load to which the server with that IP address must receive, it is possible to change the algorithms seen previously to support weights (or load rates) per server.

In round-robin, if the server weight is x then it will be needed to send x requests before moving on to the next element in the list. Thus, it is enough to create another counter that will represent how many requests that server answered with that index. Once this value is equal to that server's weight, the index may advance to the next position.

For the least-bandwidth algorithm, it will be used the formula based on the weighted least-connection algorithm (Linux Virtual Server Knowledge Base, nd), assuming a set of servers $S_1, S_2, ..., S_n$, lets also assume that $W(S_i)$ and $B(S_i)$ are the server weight S_i and the traffic consumed by it, respectively. Let B_t the total amount of traffic consumed by all servers, thus, $B_t = \sum_{i=1}^{n} B(S_i)$. Therefore, the server to respond to the next request is the server m whose value $\frac{\frac{B(S_m)}{B_t}}{W(S_m)}$ is the smallest of all the others.

Since B_t is a constant, it can be omitted, and thus it is sufficient to find the server m where $\frac{B(S_i)}{W(S_i)} > \frac{BS_m}{W(S_m)}$ for all i such that $1 <= i < n$. As division consumes a lot more processing cycles when compared to multiplication, it is possible to optimize this formula for $B(S_i) * W(S_m) > B(S_i) * W(S_m)$. It will then traverse the entire list of server IP addresses only once, as seen in the Code 4.3, to find the server that satisfies this condition. Thus, the time consumed by the algorithm is $O(n)$.

```
    for i in range(n):
        if weights[servers[i]] > 0:
            best_server = servers[i]

            for ii in range(i + 1, n):
                if data_transferred[best_server]*weights[servers[ii]] > \
                 data_transferred[servers[ii]]*weights[best_server]:
                   best_server = servers[ii]
            return best_server
```

Code 4.3: *Code of the least-bandwidth algorithm with weight support.*

Finally, the user will be allowed to choose server weights through a command line argument called weights_val, which is a vector containing the weights of all servers in the same order as the IP address list, i.e. , if the IP address server 10.0.0.3 was the third to be added to the list of servers on the command line, then its weight is represented by the third value in the argument weights_val, as can be seen in Code 4.4, where the weight of the address server 10.0.0.3 is 2.

```
./pox.py misc.ip_loadbalancer \\
ip=10.0.1.1 \\
servers=10.0.0.1,10.0.0.2,10.0.0.3 \\
algorithm=round-robin \\
weights_val=1,2,2
```

Code 4.4: *Example of running load balancer with weights on servers.*

If no weight value is specified, weight 1 will be used for all servers. If the amount of values specified in this argument is also different from the number of servers, an error is returned to the user and execution of the controller is canceled.

4.3.4 User interaction

The interaction with the user will be done through a terminal located in the controller. In it, the user can change the server weights and the balancing algorithm.

The POX controller has a way of adding interactivity to the controller via the Interactive object, which allows other applications to interact with the Python interpreter running the controller. Thus, it is enough to associate the name of the variable to which the user will type in the interpreter with the function that is wanted to be executed. The user will also need to express on the command line, when executing the controller, that they would like to access the interpreter through flag py, as can be seen in Code 4.5.

```
./pox.py misc.ip_loadbalancer \\
ip=10.0.1.1 \\
servers=10.0.0.1,10.0.0.2,10.0.0.3 \\
algorithm=round-robin \\
weights_val=1,2,3 \\
py
```

Code 4.5: *Example of running the load balancer with the interpreter flag.*

Each of these functions is defined as a method of the balancer class. In order to change the algorithm, the user must enter the change_algorithm command in the interpreter and pass the name of the new algorithm (which may be random, round-robin or least-bandwidth), as you can see in Code 4.6.

```
> change_algorithm("random")
```

Code 4.6: *Example of changing the algorithm from the balancer to the random one.*

In order to change the server weights, the user must enter the change_weights function and pass as a dictionary argument, where the key is the IP address of each server and the value of its respective weight (as exemplified in Code 4.7).

```
> change_weights({"10.0.0.1": 1, "10.0.0.2": 4, "10.0.0.3": 3 })
```

Code 4.7: *Example of changing weights from 3 servers with IP addresses 10.0.0.1, 10.0.0.2, 10.0.0.3 to 1, 4 and 3, respectively.*

Chapter 5

Performance Analysis

5.1 Methodology

To test the efficiency of the load balancer, it will be subjected to a load test and its response time evaluated for each type of algorithm implemented and for a varying number of servers connected to the balancer.

The software used for the tests will be the Apache Benchmark, a command line program provided by the Apache Software Foundation that allows a comparative evaluation of HTTP servers, evaluating how many requests per second the servers can serve optimally. It was originally created to test only Apache HTTP servers, however it is generic enough to test any kind of server (Apache HTTP Server Version 2.4, nd).

The Apache Benchmark tool is free and distributed under the terms of the Apache License, and can be obtained in a Linux environment based on the Debian distribution through the `apt-get install ab` command. After installation, the user will be able to run the tool with the `ab` command. The version used in the experiments was 2.3.

The Apache Benchmark allows the execution of requests concurrently, that is, more than one request to the server can be done simultaneously. This is similar to a real scenario where more than one user can request a service from the server. In this way, this tool will be used to simulate a scenario where 10 users try to access some page of our application. The simulation will also make 1000 requisitions at the same URL to thus analyze whether the balancer distributes efficiently and maintains the average response rate acceptable.

Thus, the test is based on executing the command displayed in Code 5.1 50 times and calculating the arithmetic mean of the response time of each of the tests.

```
ab -n 1000 -c 10 http://10.0.1.1
```

Code 5.1: *Running the Apache Benchmark for testing, assuming the IP address of the balancer is 10.0.1.1.*

As we do not have hardware available for running tests with several servers, the Mininet emulator will be used, as explained in Section 2.4, to emulate a network with the default Mininet topology, that is, a switch (which will be the balancer) connected to other hosts, which can be both servers and clients. Then, a network will be created that will have $n + 1$ hosts, where n represents the number of servers that one wants to test, leaving 1 host to serve only as client of the application, where the tests will be run.

The host number 1 (whose IP defined by Mininet is always 10.0.0.1) will be the client, while the others (whose IP addresses range from 10.0.0.2 to 10.0.0.n) will be the servers of the application.

Figure 5.1 exemplifies how the network topology will be for a test with 5 servers and how each of the functions will be separated. The command to be executed to create such a network can be seen in Code 5.2. The controller will be remote because it will be executed simultaneously, using the command in the Code 5.3, the balancing module that was presented in Section 4, which will wait for connections in port number 6633.

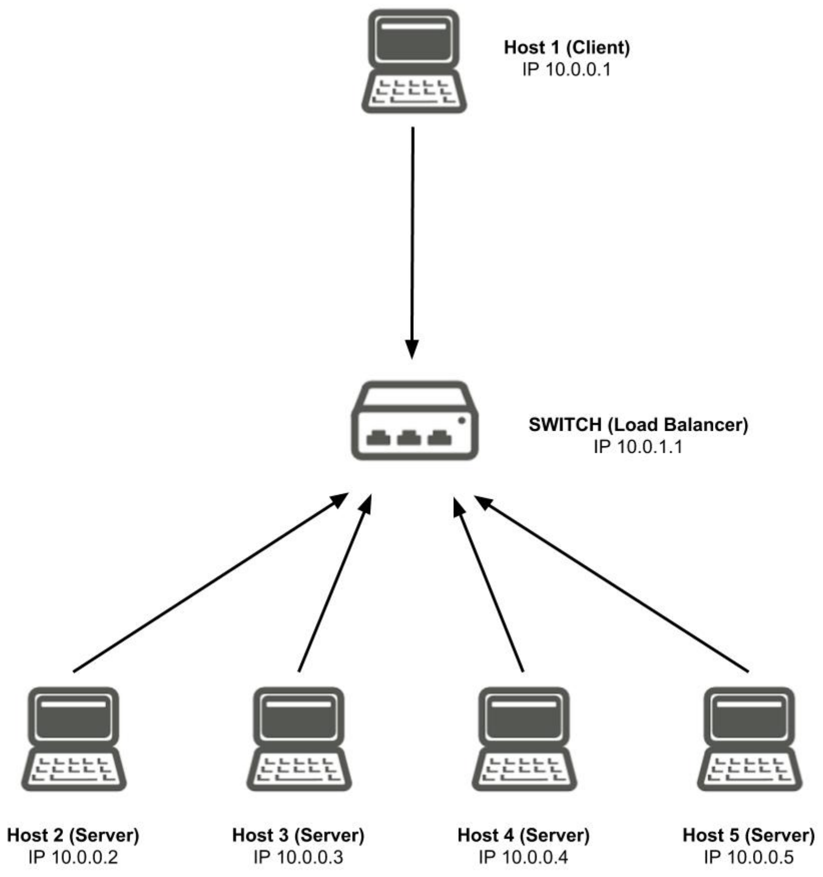

Figure 5.1: *Exemplification of the emulated network for testing with 5 servers.*

```
sudo mn --topo single,5 --controller=remote,port=6633
```

Code 5.2: *Exemplification of the command for the creation of a network with 5 servers for the load tests of the balancer.*

```
./pox.py misc.ip_loadbalancer \\
--ip=10.0.1.1 \\
--servers=10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5 \\
--algorithm=random
```

Code 5.3: *Example of the command used to run the controller that will load load balancing for 5 servers using the random algorithm.*

The tests will be divided into two parts. In the first one, the performance of the balancer (through the response time) will be evaluated for 2, 3, 4, 5, 6, 7 and 8 servers, where each one will deliver a randomly sized web page varies from 100 kilobytes to 50 megabytes) in a way similar to what happens in a real server, since the pages that the clients request are not always the same size, and can vary from request to request.

The second part aims to evaluate how the balancer behaves when servers deliver a single page of defined size. For this test, a server-farm with 8 servers will be used and we will run the tests with pages of size of 100KB, 250KB, 500KB, 1MB, 2.5MB, 5MB, 10MB, 25MB and 50MB and the response time for each of these scenarios will be analyzed.

Finally, the tests were performed on an Ubuntu 14.04.4 virtual machine with 4096MB of reserved RAM running on a Macbook Pro with Intel Core i5 2.3 GHz processor, 8 GB RAM and MacOS High Sierra v10.13.6 operating system. Python 2.7.6, Mininet 2.2.2 and POX 0.5.0 (eel) were also used.

5.2 Results obtained

The tests with different number of servers, whose results can be seen in Figure 5.2, showed that the controller behaves as expected, that is, when we add more servers, the response time tends to fall. In addition, the average response time for the tests was quite acceptable (the maximum reached was 28 milliseconds which, according to Nielsen (1993), is within what the user expects as an instantaneous response). Another consideration that can be made regarding the results is that the response time is halved when we compare a server-farm with two servers and one with three servers, but the result remains almost constant at number of servers.

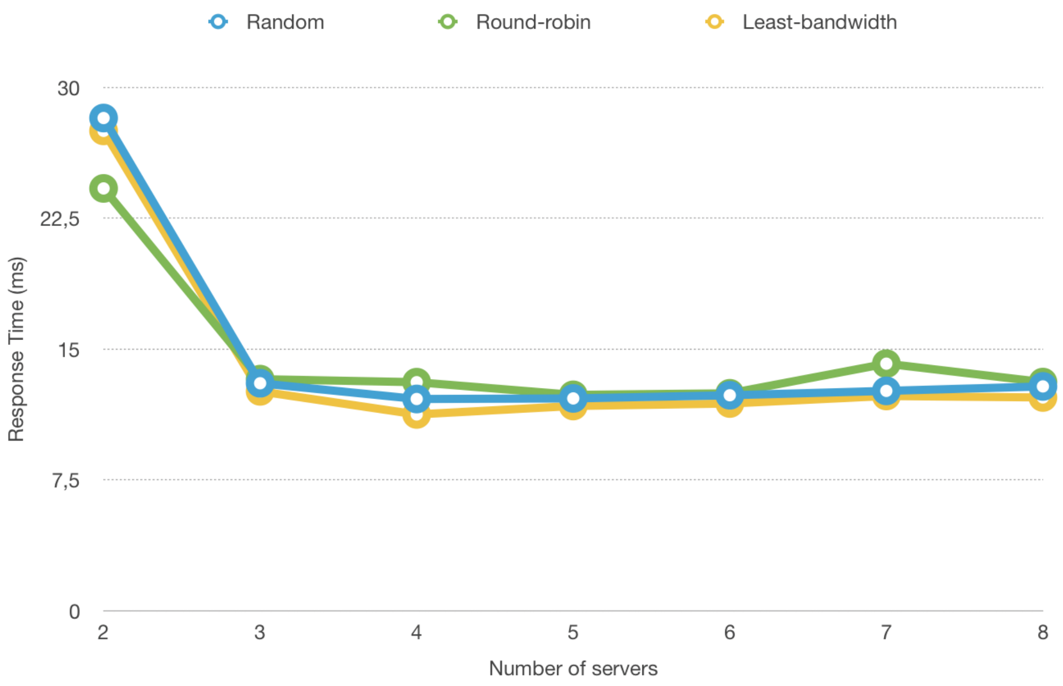

Figure 5.2: *Graph showing the response time in relation to the number of servers for each of the algorithms.*

When the results between the algorithms are compared, it is noticed that the difference between them is minimal (the maximum difference between the results is 1ms). However, it may be noted that the response time of the random algorithm is generally higher than the other algorithms. When the round-robin and least-bandwidth methods are compared, it is noticed that, for a few servers, the first works relatively better than the second, but the contrary pattern is presented as we add more servers.

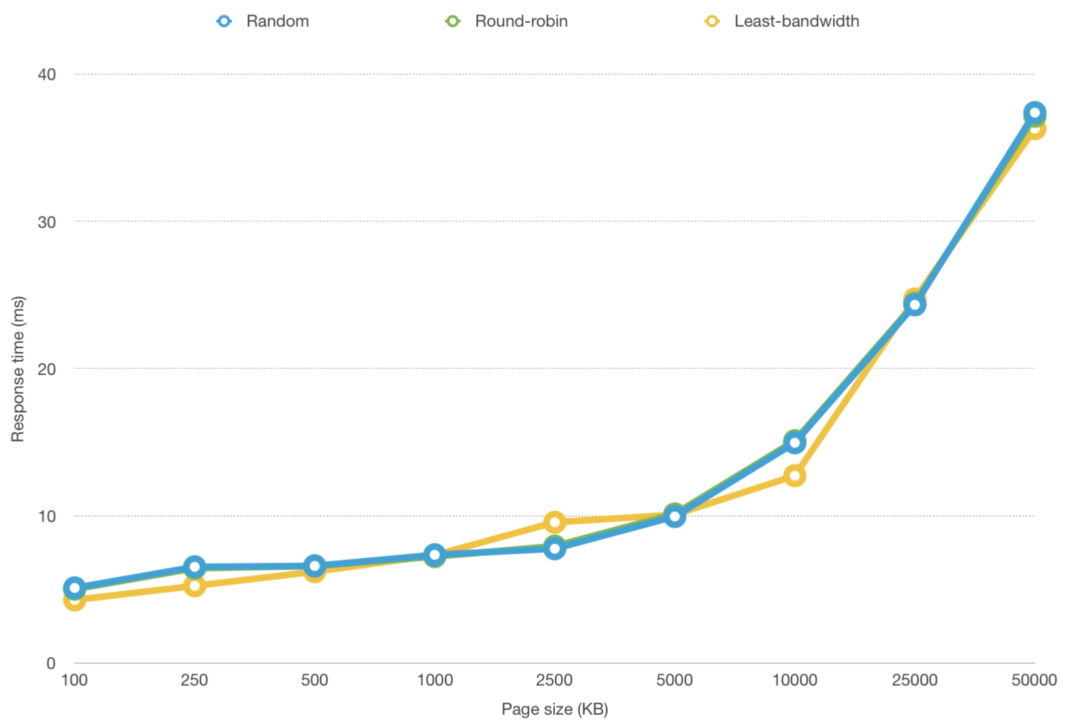

Figure 5.3: *Graph showing the response time with respect to the size of the pages offered by the servers for each one of the algorithms. Test performed with a server-farm of 8 servers.*

Figure 5.3 presents the results of the second part of the tests. In it, it can be seen that the created balancer also has an expected behavior in the sense that the response time increases while increasing the size of the resource offered by the servers. Further analysis shows that the difference between response times for relatively small pages (between 100KB and 5MB) is quite small, reaching the 5 milliseconds. Already for larger pages (between 10MB and 50MB) the difference is much greater, reaching the house of 23 milliseconds.

However, average response time remains acceptable for both small and large pages, with the maximum reached at 38 milliseconds, which is still within what is expected of an instant response.

In a similar way to the first test, we noticed that the differences between the algorithms are very small, especially when we compare round-robin with the random method (mean difference less than 1 ms).

Despite the small difference, it is perceived that the least-bandwidth method has a generally low response time when compared to other algorithms. This is due to the way in which the method works, that is, its evaluation of the traffic consumed between the servers allows the server that has consumed less (i.e. was less requested) to respond the next request.

The tests evaluated here did not take into account the server weights, that is, the load was equally separated for each of the server-farm machines. This decision was made because it is expected that the use of the load rate will be done when it has servers with different characteristics (that is, different RAM capacity, different processors) that allow the load to be distributed in order to take advantage of those resources of better servers. In the case of the tests performed, the machines had similar characteristics, so the rates should be distributed equally in order to obtain the best balance.

Chapter 6

Conclusions and future work

With this work, it was exposed how efficient a load balancer can be using only software defined networks. Based on the POX controller, the tests showed that the response time of a server farm using an SDN based balancer is within the user acceptance range (less than 0.1 second).

In addition, the balancer elaborated in this work has the characteristic of being more flexible when compared to a physical device, that is, the controller allows the interaction with the user in such a way that they can choose the algorithms at run time without the need to disable any application or physically change the balancing device. This gives the administrator an advantage, which does not have to be tied to the specific hardware features of a particular manufacturer.

The administrator, when using a balancer like the one elaborated in this work can also benefit from the flexibility with respect to the number of available servers, that is, while physical balancing devices have a generally small limit, the balancer using SDN is limited only to the quantity available on routing devices (which are typically much larger than physical balancers).

Lastly, the biggest gain when using SDN balancing lies in the price of specific and optimized hardware to do this. While the cost of implementing HLDs is around thousands of dollars (as seen in Section 3) to achieve quality balancing and with different balancing algorithms, the estimated cost for implementing a balancer like the one elaborated in this work is limited by the price of the *switch* used to forward the packages to the servers (an example of a good-quality OpenFlow switch is the HP Aruba 2920 24G PoE+ Switch, whose price is no more than U$1200.00[1], has a throughput close to 128Gbps and allows the connection of more than 20 servers).

In addition to the results obtained with the performance analysis of the load balancer developed and the comparison of the cost of this solution in relation to the use of HLDs, this work also contributed to the availability of the developed balancer code. The code is at https://github.com/lcfpadilha/pox under Apache 2.0 license. As a continuation of the work, one can focus on adding features that make our load balancing module closer to the available ADLs (application delivery load balancers), thereby increasing its value compared to other balancers. Adding SSL unloading (decrypting the packet before sending it to the server) and DDoS protection can be characteristics that differentiate the balancer from others.

It can also bring more interaction to the user, allowing him to check the health of the servers, a log with the balancer status, and to disable the balancing for specific servers.

In addition, it can be given more intelligence to the balancer by letting it itself select the

[1]HP Aruba 2920 24G PoE+ Switch. Available in <https://www.hpe.com/br/pt/product-catalog/networking/networking-switches/pip.aruba-2920-switch-series.5354494.html>. Accessed on 11/23/2018.

algorithm that will best perform load balancing at run time through machine learning, for example.

Finally, it is also expected that the ease of the language used and the SDN technique will allow the administrator to elaborate the balancer to their needs, being no longer a complex and rigid device like the current physical balancers.

Bibliography

Apache HTTP Server Version 2.4 (n.d.). *ab - Apache HTTP server benchmarking tool*. Last access at 10/22/2018. 21

Duque, D. H., Couto, E. J., Pinto, M. H., and Magalhães, T. L. (2012). Redes Definidas por Software. Monografia (Graduação em Engenharia Elétrica), Inatel (Instituto Nacional de Telecomunicações). ix, 9, 10

Esteve Rothenberg, C., Nascimento, M., Salvador, M., and Ferreira, M. (2018). OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. ix, 1, 10, 11

Fernández, J., García Villalba, L., and Kim, T.-H. (2018). Software defined networks in wireless sensor architectures. *Entropy*, 20:225. ix, 9

Gandhi, R., Liu, H. H., Hu, Y. C., Lu, G., Padhye, J., Yuan, L., and Zhang, M. (2014). Duet: Cloud scale load balancing with hardware and software. *SIGCOMM Comput. Commun. Rev.*, 44(4):27–38. 1

Godha, R. and Prateek, S. (2014). Load balancing in a network. *International Journal of Scientific and Research Publications*, 4.

Kaur, S., Singh, J., Saluja, K., and Singh Ghumman, N. (2015). Round-robin based load balancing in software defined networking.

KEMP Technologies (n.d.). Comparison of the KEMP LoadMaster with F5 Big-IP LTM and Citrix Netscaler MPX Hardware Load Balancers and ADCs. https://kemptechnologies.com/compare-kemp-f5-big-ip-citrix-netscaler-hardware-load-balancers/. Last access at 10/22/2018. xi, 14

Lantz, B., Heller, B., and McKeown, N. (2010). A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19.

Linkletter, B. (2015). Open-Source Routing and Network Simulation. http://www.brianlinkletter.com/using-the-pox-sdn-controller/. Last access at 10/22/2018.

Linux Virtual Server Knowledge Base (n.d.). Weighted Least-Connection Scheduling. http://kb.linuxvirtualserver.org/wiki/Weighted_Least-Connection_Scheduling. Last access at 05/14/2018. 19

Mahler, D. (n.d.). Introduction to SDN (Software-defined Networking). https://www.youtube.com/watch?v=DiChnu_PAzA. Last access at 10/22/2018.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., L. Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. 38:69–74.

Mininet (n.d.). Mininet Overview. http://mininet.org. Last access at 10/22/2018. 11

Moharana, S. (2013). Analysis of load balancers in cloud computing. *International Journal of Computer Science and Engineering*, 2:101–108. 1, 3

Nielsen, J. (1993). Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits/. Last access at 09/24/2018. 23

Ominike, A., Seun, E., A. O., A., and Osisanwo, F. (2016). Introduction to software defined networks (sdn). *International Journal of Applied Information Systems*, 11:10–14.

Open Network Foundation (n.d.). OpenFlow Specification version 1.4.0. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf. Last access at 10/22/2018. 9

Pagano, F. (2012). A Distributed Approach to Privacy on the Cloud. *CoRR*, abs/1503.08115. ix, 13

Sekar, C. (2017). Software Load Balancer Advantages and 9 Reasons to Switch. https://blog.avinetworks.com/9-reasons-to-make-the-jump-to-a-software-load-balancer. Last access at 10/22/2018. 13

Silva, G. H. (2016). Redes Definidas por Software: aplicações práticas. Monografia (Tecnólogo em Sistemas de Telecomunicações), UTFPR (Universidade Tecnológica Federal do Paraná). 10, 11

Sreedhar, P. (2018). Team NetClaws - Traffic Based Load Balancing using SDN. https://www.youtube.com/watch?v=0eJHf-epyCI. Last access at 10/22/2018.

Vashistha, J. and Kumar Jayswal, A. (2013). DNS Based Approach Load Balancing In Distributed Web Server System. *IOSR Journal of Engineering*, 3:46–55.