Effectiveness of Implementing Load Balancing via SDN*

Leonardo C. F. P. Aguilar[1], **Daniel Macêdo Batista**[1]

[1]Department of Computer Science – IME
University of Sao Paulo (USP) – Sao Paulo, SP, Brazil

leonardo.aguilar@usp.br, batista@ime.usp.br

Abstract. *Software-Defined Networking (SDN) is an architecture that allows the creation, management and customization of the network through programmable switches and centralized controllers via a well-defined protocol. Despite the wide dissemination of general advantages in using SDN, it is always important to evaluate the real advantages considering specific network applications. In line with this, the purpose of this work is to analyze the effectiveness of using SDN for load balancing by developing a balancer, made available as free software, that can execute three different algorithms, giving to the administrator the possibility to choose, at run time, which will be used as well as their configurations, and the possibility to implement new algorithms.*

1. Introduction

According to [Moharana et al. 2013], load balancing in computer networks is an important technique aiming the optimization of existing resources for better distribution of data packets, avoiding equipment overload and decreasing of applications response time. The technique is based on choosing the most appropriate equipment to respond a particular request and it can be done in several ways. The most common is the deployment of a specific "black-box" hardware that aims to forward the data packets to the equipment in best current condition [Gandhi et al. 2014]. The definition of the best equipment is made by a distribution algorithm that can consider several static and dynamic attributes. However, an implementation based on this approach can cause several types of limitations, such as lack of flexibility in the choice of the algorithm and the impossibility to increase the number of equipment controlled in the pool, since these two aspects are locked by manufacturer and model. Besides, the cost of this solution is quite high, reaching US$ 2,914.99 for instance[1].

On the other hand, conforming to [Esteve Rothenberg et al. 2010], with the dissemination of the Software-Defined Networking (SDN) architecture in several environments, the data traffic within the network does not need to depend only on the hardware. This ensures greater customization and adaptation that can vary according to several network conditions (for example available servers, packets meta-data, etc.) in contrast to the previous rigid model. The use of SDN 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 assure greater control over the network. Based

*This paper summarizes the final results of the capstone project developed by Leonardo C. F. P. Aguilar available at https://linux.ime.usp.br/~lcfpadilha/mac0499/. Accessed March 21, 2019.

[1]The Barracuda Load Balancer ADC 240 was chosen as the price parameter. Available at http://www.zones.com/site/product/index.html?id=100713342. Accessed March 21, 2019.

on this, it is possible to assume that the application of the SDN architecture to implement load balancing may be a more flexible alternative, and with a lower implementation cost, when compared to the use of specific "black-box" hardware.

The objective of this paper is to evaluate the effectiveness of implementing load balancing via SDN. Implementation specifics of a prototype and the results of experiments in an emulated network are presented to attest the advantages of the approach. It was possible to observe that the response time of a server farm using an SDN based balancer is within the user acceptance range (less than 0.1 second). To guarantee the reproducibility of the research and to encourage the spread of the approach, all the code developed on the scope of this project is available as free software at `https://github.com/lcfpadilha/pox` under Apache 2.0 license.

The rest of this paper is organized as follows: Section 2 presents some basic concepts about load balancers and SDN. Section 3 describes the main contribution: an implementation of a load balancer based on SDN. Section 4 presents the experiment planning. The results obtained from the experiments are presented on Section 5 and conclusions are presented on Section 6.

2. Basic Concepts

2.1. Load Balancers

According to [Moharana et al. 2013], load balancing is a widely used mechanism to better leverage all available resources, optimizing the use of them and the execution time of the tasks. Its application is either to distribute the workload between specific resources of a computer, such as hard disks, or between network servers in order to ensure that the best machine, i.e. the one with the greatest availability, responds to a particular request.

Several methods can be used to ensure the correct balance, and it is possible to use different elements of computer networks, such as DNS servers, to achieve this objective. One very common method is the use of a physical balancing device (the load balancer) connected to the available servers. This load balancer is responsible for listening to customers and, when a request arrives, it has the responsibility to redirect the request to one of the servers in the "farm". Because the load balancer is a single point that connects customers with the application, clients do not have direct access to the servers, which improves the security level of the service being accessed.

In addition, to ensure a better quality of load balancing between servers, it is common for load balancers to use algorithms that go beyond a simple random choice to select the next server that will respond the next request, being able to apply methods that can take into account various network parameters. There is no algorithm that can be considered preferable, since each one 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. However, this has a great impact on cost of the devices, and can cause them to reach more than US$ 10,000.00 [KEMP Technologies nd].

2.2. Software-Defined Networking

Software-Defined Networking (SDN) is an architecture based on the decoupling of the data plane (the layer that, in fact, forwards the data) from the control plane (the layer

responsible for take decisions about the routing of the data) in such a way that it is possible to define the routing decisions using standardized and customized software not-specific to a manufacturer.

For years, the architecture of the networks was defined only by physical interconnection elements, like routers and switches, where the data plane and the control plane were indivisible. The routing and the forward of packets in these elements were managed by a layer of control created by the manufacturer, without the possibility of change by the network administrators. This control plane was built into the element itself. Thus, a network with n interconnection elements could have n different control plane rules, one for each device, not necessarily compatible with each other. With SDN, the control plane rules can stay on a dedicated server with high throughput and this server is responsible for adding rules for packets on each network device. Several rules are possible, such as allowing the flow of a particular IP address, blocking a particular port, etc.

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 1.

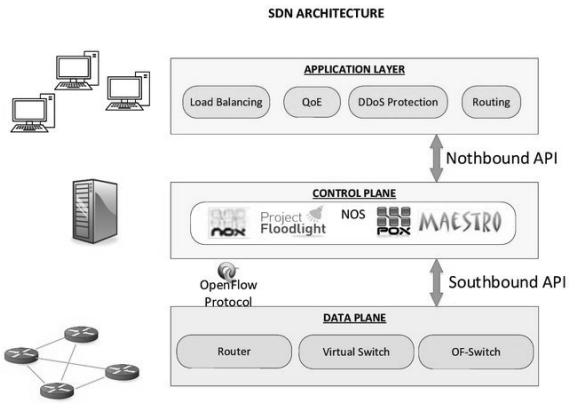

Figure 1. SDN Architecture.[Fernández et al. 2018, p.5]

The first layer, which is at the topmost level, is the **application** layer. The applications are software able to communicate with the layer below in order to define the behavior of network flows. The load balancer to be defined throughout this paper will act on this layer. The middle layer is called the control plane, 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. Several SDN controllers are available as free software. Figure 1 presents the names of some of these controllers. The layer below is the **data** plane, or layer of the network devices, whose function is to receive the data packets, perform some action with them based on the instructions passed by the controller, and update their internal counters which serve as general statistics. Communication between the application layer and the controller (or Northbound API) usually provides abstract views of the network and allows the direct expression of behavior and network requirements [Open Network Foundation 2013]. 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 devices' flow table.

3. SDN for Load Balancing

As explained on previous sections, the use of a physical "black-box" device to perform load balancing has a very high implementation cost and does not allow flexibility, since the administrator can not have full control of the balancing parameters. To solve this problem, this work proposes a SDN application to act as a load balancer, reducing the cost and allowing the administrator greater flexibility when compared to the fixed model determined by the manufacturers of balancers.

The proposed load balancer performs 3 different algorithms for load balancing (random, round-robin and least-bandwidth). It allows the administrator to select not only the algorithm but also the load ratio that each server will receive. The interface with the load balancer is made via CLI at run time.

In the proposed approach, a switch OpenFlow receives packets to be directed to one of the farm servers. The decision about what server will be selected is based on rules delivered by the controller, configured by the administrator. To implement the load balancer, the POX controller version 0.2.0, a controller written in Python, which allows a fast and easy development, widely used for prototyping projects in SDN, was used.

The source code of the POX controller comes with a sample module called `ip_loadbalancer` that implements a simple load balancer. This module has two classes: the `MemoryEntry`, an abstraction of the flows that are in the switches, and also the `ipbl` class, which contains the balancer logic itself. This class stores the servers' IP addresses, the controller's IP address, the flows that are active and the methods for selecting servers and for handling packets arriving at the controller.

The proposed module was implemented on the top of the sample POX module. It currently receives the IP address on which the balancer will be used and the server address list. Code 1 illustrates the execution of the load balancer. In the illustration, the load balancer address was set to `10.0.1.1` and the available servers were defined by the addresses `10.0.0.1`, `10.0.0.2` and `10.0.0.3`.

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

Code 1. Example of running the ip_loadbalancer component.

The controller sends ARP messages to the servers periodically to find out if there has been a crash on a server. This act like a heartbeat checking because if a machine doesn't respond to the message for a long time, it probably crashed and is removed from the list.

The balancing of the proposed module is done considering TCP segments, that is, only packets of this protocol will be balanced between the servers. Other protocols can be considered as long as they are supported by OpenFlow. When a packet carrying a TCP segment is received at the balancer, the controller will be responsible for sending the IP address of one of the active servers to the balancer, and when doing so, a copy of the flow is created in the controller by a period of time. Figure 2 exemplifies how balancing is done using the proposed module considering the execution illustrated on Code 1.

The original `ip_loadbalancer` module already implements the random algorithm and we extended it by including the round-robin and the least-bandwidth algo-

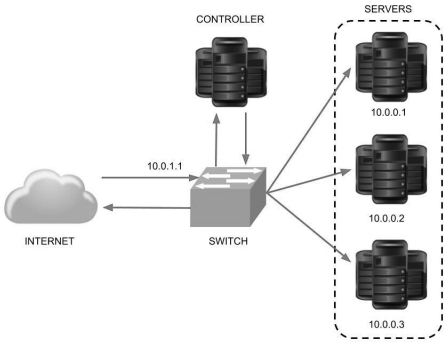

Figure 2. Exemplification of the ip_loadbalancer component.

rithms, both supporting the weights per server, and allowing the administrator to change the algorithms at run time. The round-robin algorithm works by associating the servers with a queue and when there is a request, the one that is at the beginning of the queue is used, removing it and re-positioning it to the end. The least-bandwidth algorithm selects the server with the lowest network traffic consumption to respond the next request.

4. Experiments

To evaluate the effectiveness of the proposed load balancer, it will be subjected to a load test and its response time will be measured for each one of the three algorithms and for a varying number of servers connected to the balancer.

The software used for the tests was the Apache Benchmark (`ab`) version 2.3, a 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 allows the execution of requests concurrently, similar to a real scenario where more than one user can request a service from the server. In this way, this tool was used to simulate a scenario where 10 users try to access some page of a web application. The simulation also made 1000 requisitions at the same URL to thus analyze whether the balancer distributes the load efficiently and maintains an acceptable average response rate.

Thus, the test was based on executing the command displayed in Code 2 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 2. Running the Apache Benchmark for testing, assuming the IP address of the balancer is 10.0.1.1.

The Mininet emulator was used to emulate a network with the default Mininet topology: a switch, which is the balancer, connected to other hosts, which can be both servers and clients. The type of the emulated network is Local Area Network (LAN) and it had $n+1$ 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 started.

The host number 1, whose IP address defined by Mininet is always 10.0.0.1, is the client, while the others, whose IP addresses range from 10.0.0.2 to 10.0.0.n+1, were the

servers of the application. Figure 3 exemplifies how the network topology was for a test with 4 servers and how each of the functions was separated.

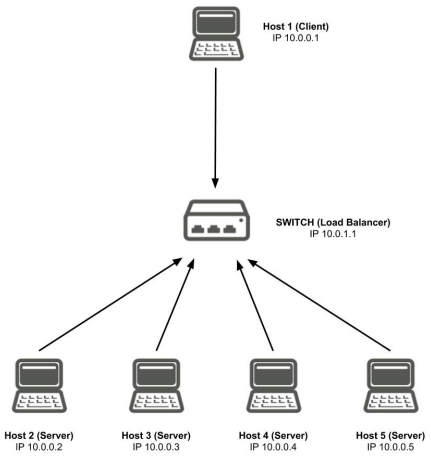

Figure 3. Exemplification of the emulated network for testing with 4 servers.

The tests were divided into two parts, using, in both, 9 different pages with sizes of 100KB, 250KB, 500KB, 1MB, 2.5MB, 5MB, 10MB, 25MB and 50MB. In the first one, the performance of the balancer, through the response time, was evaluated for 2, 3, 4, 5, 6, 7 and 8 servers, where each one delivered, in each request, a random page with the same probability to be chosen, similar to what happens on a real server, where the page size is not fixed for each 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 was used and the tests were executed with each one of the pages and the response time for each of these scenarios were measured.

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 0.2.0 (carp) were also used.

5. Results

The tests with different number of servers, whose results can be seen in Figure 4, 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 measured was 28 milliseconds which, according to [Nielsen 1993], is within the limit where the user feels that the system is reacting instantaneously. 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 with the other number of servers.

Figure 5 presents the results of the second part of the tests. It can be seen that the 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)

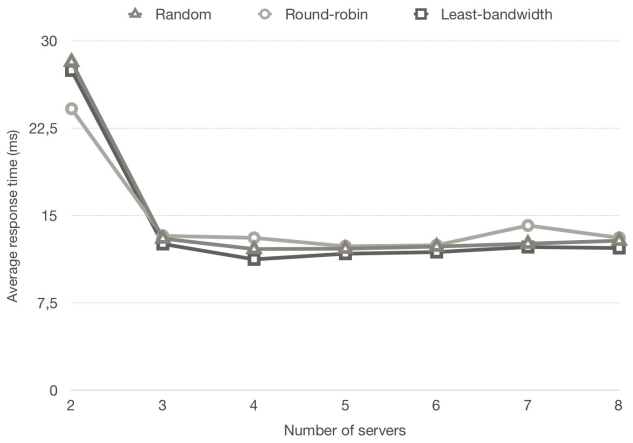

Figure 4. Average response time in relation to the number of servers. Scatter data has been suppressed since it does not exceed 2 seconds.

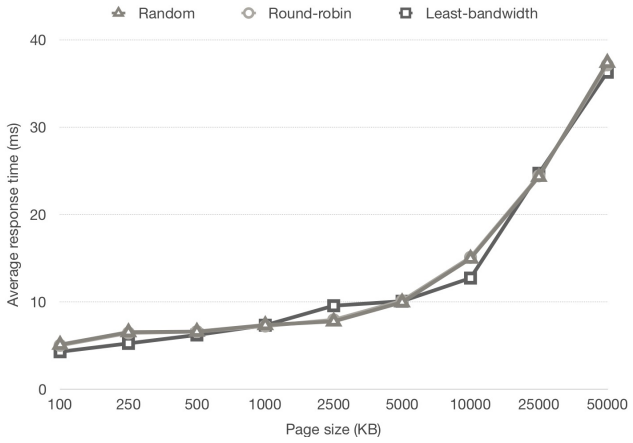

Figure 5. Average response time in relation to the pages size (Test performed with a server-farm of 8 servers). Scatter data has been suppressed since it does not exceed 1 second.

is quite small, reaching 5 milliseconds. For larger pages (between 10MB and 50MB) the difference is much greater, reaching 23 milliseconds. However, average response time remains acceptable for both small and large pages, with the maximum being 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 (mean difference less than 1 ms). Although the small difference, it is perceived that the least-bandwidth method has a generally low response time when compared to other algorithms.

6. Conclusions

Despite the advantages of using SDN, specific deployments of SDN applications are important to evaluate the real advantages of the architecture. This paper presented the results obtained in a capstone project which evaluated the effectiveness of using SDN for load balancing by developing a balancer made available as free software.

Load tests showed that the response time of a server farm using the proposed

SDN-based balancer is within the user acceptance range (less than 0.1 second), exposing how efficient a load balancer can be using SDN. In addition, the balancer elaborated in this work has the characteristic of being more flexible when compared to a physical device. Lastly, one of the main advantages when using SDN balancing lies in the price of specific and optimized hardware. While the cost of implementing a physical load balancer is around thousands of dollars 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[2], has a throughput close to 128Gbps and allows the connection of more than 20 servers.

Acknowledgments

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

References

Esteve Rothenberg, C., Nascimento, M., Salvador, M., and Magalhaes, M. (2010). OpenFlow e Redes Definidas por Software: um Novo Paradigma de Controle e Inovação em Redes de Pacotes. *Cad. CPqD Tecnologia*, 7:65–76.

Fernández, J., García Villalba, L., and Kim, T.-H. (2018). Software Defined Networks in Wireless Sensor Architectures. *Entropy*, 20:225.

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. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 27–38.

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 March 21, 2019.

Moharana, S. S., Ramesh, R. D., and Powar, D. (2013). Analysis of Load Balancers in Cloud Computing. *International Journal of Computer Science and Engineering (IJCSE)*, 2:101–108.

Nielsen, J. (1993). Response Times: The 3 Important Limits. `https://www.nngroup.com/articles/response-times-3-important-limits/`. Last access at March 21, 2019.

Open Network Foundation (2013). OpenFlow Switch Specification Version 1.4.0. `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf`. Last access at March 21, 2019.

[2]HP Aruba 2920 24G PoE+ Switch. Available at `https://www.hpe.com/br/pt/product-catalog/networking/networking-switches/pip.aruba-2920-switch-series.5354494.html`. Accessed March 21, 2019.