A Testbed for research and development of SDN applications using OpenFlow

Nádia Pires Gonçalves
nadia.goncalves@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa

ABSTRACT

Network technologies have been dominated by traditional paradigms resulting from the TCP/IP model and local networks, centered on traditional switching and routing concepts. The current network complexity at the data center, local and operator levels present new management challenges and flexibility requirements. The SDN paradigm emerged to tackle these challenges. The main goal of SDN is to separate the control plane from the data plane, which are usually tied together in conventional network devices, in such way that these can be managed, controlled and monitored by custom applications, enabling increased net- work flexibility independently of proprietary solutions.

OpenFlow is a communication protocol based on the SDN paradigm, which defines a communication between the data plane and the control plane. This project aims to create a testbed in order to facilitate the comprehension about Programmable Network.

Keywords

Software-Defined Networking, OpenFlow, POX

1. INTRODUCTION

Since 1970 there have not been many changes in traditional networking technologies, thus increasing network ossification [13]. The IP protocol based Internet has had huge success, being primarily centered on the traditional concepts of routing and switching. Though, the current network complexity is facing significant networking issues, such as QoS, security, mobility and management.

The current state of networking technology introduces unnecessary cost and complexity. This is a universal issue, since network architectures are increasingly complex and have low scalability [4]. Many solutions have been proposed to substitute current networks, but have never been implemented for being extremely difficult to test.

The fact that the current network technology and devices are installed at a large scale, with numerous devices and protocols, and that they are mostly based on enclosed proprietary network devices, meaning that only equipment vendors can configure and create protocols, does not help the implementation of new ideas that may arise by the network research community or by new requirements of network operators. In fact, most current network devices have an integrated control and data plane, forcing service providers to use a repetitive process to configure each device or group of devices of the same brand in an independent way [7].

In the last few years, the concept of SDN emerged as a proposal to overcome these limitations [8]. SDN triggered a great interest on researchers and network operators. SDN, have the goal of separating the data plane from the control plane, allowing the restructuring of the network management so that it is possible for programmers to control the network data plane directly [10].

The OpenFlow [11] interface is an open protocol proposal that defines a communication API between the data plane and the control plane. Openflow was the first SDN protocol widely accepted by both the research community and vendors, providing high performance and granular network traffic control through network devices.

The basic idea behind OpenFlow is to create a system that guarantees researchers and network operators the largest possible control over packet flows on network devices. For this control to happen, the decisions for packet treatment are based on a subset of information that different devices extract from the packet during processing. OpenFlow allows the storage of tuples with this data on the network device's flow-tables, associating an action with these entries flows [14].

Even though OpenFlow is recent and the number of real world applications is still limited, there are already several large scale companies interested in using OpenFlow, such as, Google[12], Verizon [6] and Yahoo [9], among others. These players have shown particular interest in standardizing the OpenFlow protocol, thus forming the Open ONF [5].

The projects already available are GENI[1] in the United States, JGN2plus [2] in Japan and OFELIA[3] in Europe.

SDN applications are still something being developed; the concepts are still theoretical, even though there are implementations on big corporations or on the Universities that founded OpenFlow. There are still many questions, about OpenFlow.

As time goes by, an increasing number of companies or Universities are implementing programmable networks with the OpenFlow interface.

1.1 Objectives and Contributions

This thesis aims to develop a testbed for SDN applications, with the purpose of academic analysis and research, as well as potential application in the internal data network at Técnico Lisboa. With this study it is known that it will be easier to understand how to implement the programmable networks in a real network.

To achieve the necessary acknowledgment about SDN with Openflow for the future application in Técnico Lisboa, it is imperative that the final goal must fulfill the following requirements and specifications:

- Implementation of a network with OpenFlow support;
- Configure and monitor the network;
- Identify use cases;
- Implement use cases in a test scenario;
- Analyze the network, extracting results regarding network performance;

1.2 Dissertation Structure

This dissertation is composed of 6 chapters which are arranged as follow; chapter 2 presents the state of the art which describes who is the SDN and how it works, chapter 3 describes the architecture of the project, chapter 4 describes the implementation strategies, the equipment and software chosen and how implements the several use cases, chapter 5 presents the evaluation of the several use cases in order to understand if the SDN is the biggest thing and the chapter 6 summarizes the work developed, the problems obtained and the future work.

2. SOFTWARE-DEFINED NETWORK WITH OPENFLOW PROTOCOL ARCHITECTURE

This thesis aims to create a SDN testbed, that takes advantage of the needs of network administrator to control the network and the development a testbed for OpenFlow at Técnico de Lisboa, in order to provide an experimental and research setup of SDN at IST and to envisage possible applications of SDN in the operational Técnico de Lisboa's network. This chapter describes which is the architecture that was implemented to support several use cases.

In Section 2.1 present the global architecture, that is explain the networks components, how they inter-connect with each other, the OpenFlow Switch architecture and why this architecture. In Section 2.1.1 will be define the communication between controller and switch.

2.1 Overview Architecture

The testbed consists of three switches, computers and a laptop running POX as the controller.

In order for the testbed to work, network configuration is necessary, using network cables for the connections. Some switch ports are designated as OpenFlow ports, thus the controller may used them to send its flow. The controller's IP address is also specified along with relevant information such as mode and datapath ID.

As shown the figure 1, the network topology, it is composed with 3 OpenFlow switches and 1 Controller. The Controller is linked to all OpenFlow switches by a IEEE 802.3 wired connection, and the switches are connected between each others. The controller controls all the Ethernet interface ports of switches, as well as WLAN interface.

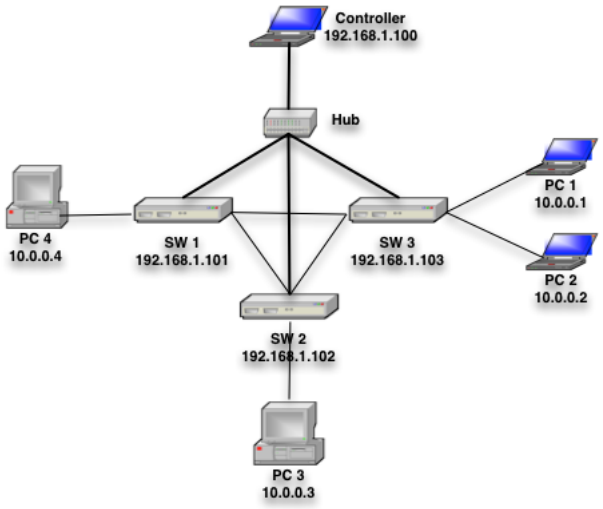

Figure 1: OpenFlow Testbed

The names that identify the OpenFlow switches are composed by a two letters "SW" followed by a number. The goal of this identifications is to have a perception and to knows which is the switch that one person it is work.

The OpenFlow switches support the instructions made by Controller, and this instructions are put in the data plane of switch. This flow tables contains several flow entries that make a matching with packet. It's means that the controller is the central piece of the network architecture, this controller manage the network, maps out the network's status, takes a given configurations and renders into OpenFlow entries and it sends this entries to OpenFlow Switches.

The OpenFlow switches architecture is a quite different than a normal switch, is installed a packet on the switches, that makes essentially a "dumb" device that forward packets between ports.

The figure **??**, represents the OpenFlow Switches and we can verified that switch is composed for 4 components, Management, Data plane, Device Firmware and OpenFlow client. The data plane contains the flow entries given by the controller, this flow entry it's add in the switch flow table, the device firmware it is a firmware.

2.1.1 Exchanged Message between Controller to Switch

Openflow is a communication protocol that provides a secure communication between controller and Switches.

It is important denote that this protocol does not responsible for the flow table definition, but is responsible for the flow table forwarding to the switches. It is mean that when a Openflow switches communicate with controller are necessary exchanged some message types. This protocol supports three message types:

Controller-to-switch messages: are initiated by the Controller and are used to manage or directly inspect the switch state. These messages are, commonly, the first used when the OpenFlow Channel is established.

Asynchronous messages: are initiated by the switch. They are used to update the Controller on events occurring on the network and are also used to change the switch state. These messages are sent independently of Controller request. Switches send this type of message to the Controller to indicate the arrival of a packet, switch state change or in case of an error.

Symmetric messages: are sent without any solicitation in either direction and are used upon connection start up or for request/reply messaging or even other messaging purposes. This type of message can be initiated by both sides, either by the switch or by the Controller [8].

The communication between Controller and Openflow switch it is make in two phases, the initial communication and the such as: the Initial phase, this phase are divided in some sub-phases, the main goal of this phase is the communication establishment while the phase connectivity check has the intention to verified the status of all switches, sending keep alive messages, two phases; the handshake phase are send messages when the communication is established, in order words when is fired a ConnectionUp event. The other phase are send messages when is fired a event that the Openflow switch do not know handling with a specified packet sending a PacketIn message, for example.

Connection Establishment.

When a Openflow Switch joins the programmable network is initiated a TLS session establishment, after this session are established,is exchanged several messages to establishment the connection.

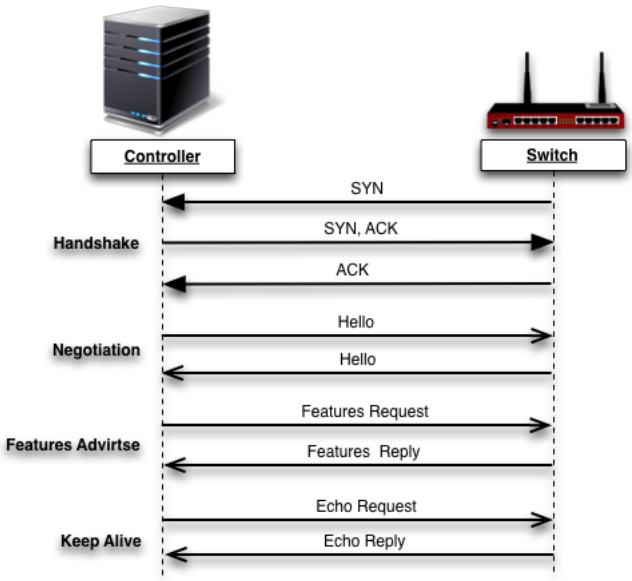

Figure 2: Connection Establishment between OpenFlow Controller and OpenFlow switch

As show the figure 2 it is send out an hello message, this message type are exchanged between the switch and controller upon connection startup.

After the hello messages the controller and the Openflow switch starts the features advertise, they sending messages features request and features Reply between the which other. The feature request message is sending by the controller to switch, while the switch reply to Controller with features reply message.

A features Request consists of an message without body that controller send to switch and this switch answer with a features reply.

A features Reply consists of an message with the body contains the switch characteristics, such as datapath identifier, the buffer length, the number of tables that the datapath supports, the switch capacities, the actions that the switch supported and list of ports and the respective speeds.

The set config message it is send by the controller to switch, when the controller want that switch send a flow expirations.

An echo Request consists of an OpenFlow header together with an random length data field. This data field might be a message timestamp to check latency, or zero-size to verify liveness between the switch and controller.

An echo Reply message consists of an Openflow header together with unmodified data field.

Event Handling.

After of all ConnectionUp are established is initiated the event handling. The controller is waiting for the events that come the switch and the switch waiting for response for Controller part.

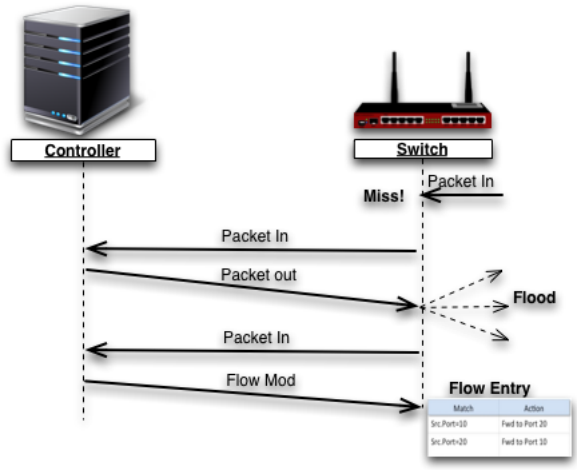

Figure 3: Event Hadling between OpenFlow switch and OpenFlow Controller

When the switch don't know what to do with a specified packet and if this switch have not a kind of drop instruction, so the switch send a packet for the controller. As shown in figure 3, the PacketIn message is a way for the switch to send a message to the controller.This is very important, because your function is pushes packet forwarding rules by controller to an Openflow switch.

When this type of message happens is fired an event handler allowing that controller processes the packet-in, and generates one or more flow-mod type message and sends it to one or more switches or send a packet-out to one or more

switches depending on functionality.

After the controller processing the packet it is send out to all of devices with actions that the devices have to handler. The controller can sends a packet-out or flow-mod message replying to the switch.

Controller sends packet-out message to one or more switches depending on functionality. A packet-in has the same packet payload as a packet-out. If the controller did not send a packet-out to the datapath, then the client sending the original dataflow packet would have to resend the packet found in the packet-in.

While, the flow-mod message permits to instruct a switch to add a particular flow entry in the flow table. This type of message can happen when the connection starts or when is fired a event to the controller, while the packet-out it is only sent when the switch send a packet-in message.

3. EVALUATION

This chapter reproduces the results that allow a short evaluation of the SDN behavior comparing with traditional networks and of how SDN networks perform in reactive versus proactive mode.

3.1 Tests Objectives

The main goal of this thesis is to prove the concept of the SDN with OpenFlow protocol, to accomplish this goal it's necessary to realize several tests, with packet captures from Wireshark.

These scenarios will allow the measurement of the packet transmission time, the switch flow table installation cost and the time it takes for a switch to establish a match with a packet.

3.1.1 Metrics

This section presents a brief explanation of the evaluation metrics used to test the project. The evaluation metrics used are flexibility and performance evaluation.

Flexibility: this metric verifies device functionality after successive configuration changes with OpenFlow, due to the creation of several test scenarios;

Performance: this metric evaluates the performance of an OpenFlow network and is composed of latency and throughput measurements.

3.2 SDN Network Vs Traditional Network

This section evaluate the behavior of the SDN and the traditional network.

The two test that permits to make this comparison is latency and throughput.

3.2.1 Latency

The main goal of this metric is to prove that SDN networks basically have better or equal latency compared to traditional networks.

To achieve this goal a ping that measures the latency was necessary. These pings are made in SDN hybrid mode.

As shown in figure 4 the latency of the packets is better than the latency of the packets in traditional networks, though when the first packet is received by the destination the latency in SDN is higher (118 ms) than in traditional networks, but was not shown in the figure because it would make the data impossible to read.

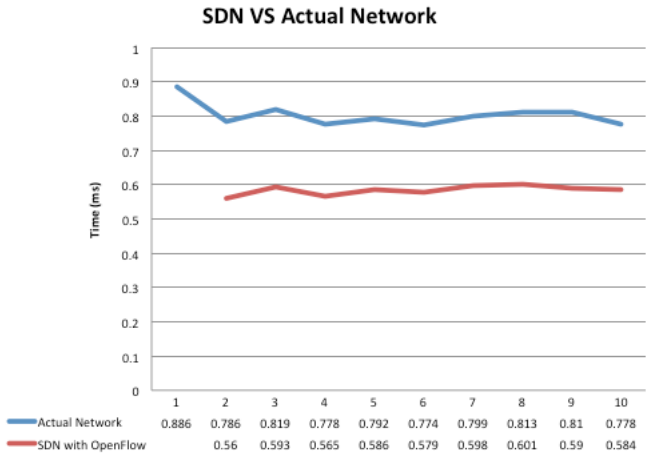

Figure 4: Performance Comparison between SDN and Traditional Network

This behavior happens because in a first phase the packet is sent to the controller and the flow tables entries are installed and only after this procedure are the other packets sent to the destination.

Network Type	Minimum	Average	Maximum
Traditional Network	0.774 ms	0.803 ms	0.886 ms
Programmable Network	0.560 ms	12.336 ms	118.336 ms

Table 1: RTT comparison between SDN and Traditional Network

As shown in the table 1 the average latency is worse, because of the first packet that went to the Controller.

3.2.2 Throughput

Throughput is the rate of successful message delivery over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The throughput is usually measured in bits per second.

To realize this test the iperf tool was used, it was possible to measure this parameter over a TCP connection between the nodes. Following the statistics obtained from the previous sections, there is a relational logic on the performance of the protocol. The result of this test is the same of that in subsection 3.2.1, which is a minimal difference between programmable networks versus traditional networks.

As show the figure 5, the only difference between this two networks are when the controller react in reactive mode and hybrid mode.

3.3 SDN behavior Evaluation

It is difficult to prove with values the flexibility of SDN, the only way to prove this flexibility is to apply several use cases and to add one OpenFlow switch at a time.

With tested use cases it was possible to conclude, that the controller has the same behavior in all topics, although, after several runs the controller could no longer communicate with the equipment, making a Reboot of the controller as well as the equipment necessary. This was the only defect that was more noticeable, it is noteworthy to refer that the controller was run on a personal computer.

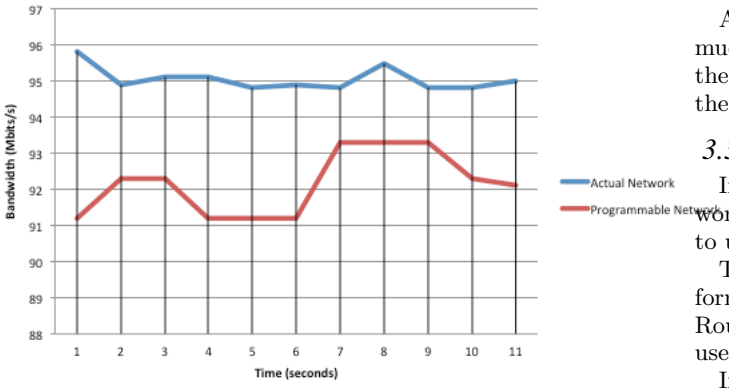

Figure 5: Throughput Comparison between SDN and Traditional Network

3.3.1 *Proactive Versus Reactive Mode*

Before presenting the results of several uses cases, it was also necessary to show the difference between reactive and proactive mode, to not speak of the hybrid mode, because it's the best of both worlds.

The tests made were exactly the same as in the previous section. It can be seen in the graph that when one speaks of a proactive network its as if there is talk of a traditional network.

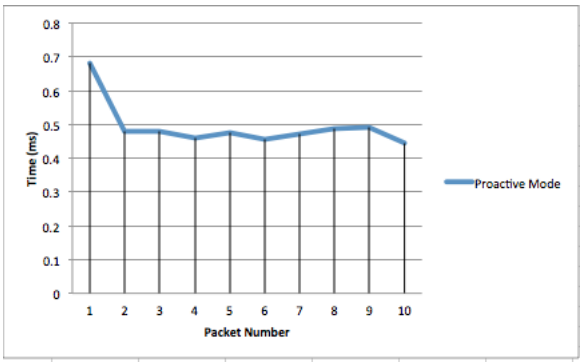

Figure 6: Latency in Proactive Mode Network

As can be seen, by analyzing figure 6, the proactive mode behaves similarly to traditional networks due to the installation of the flow table entries in the beginning of the communication.

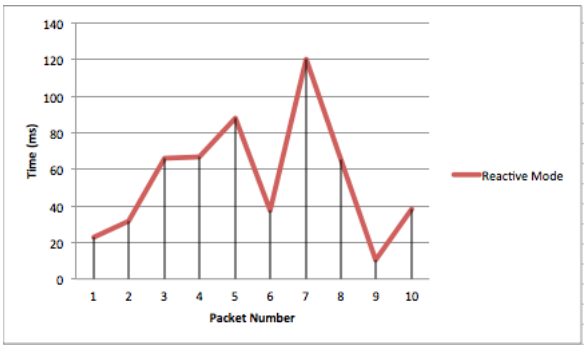

Figure 7: Latency in Reactive Mode Network

As show the figure 7, in reactive mode the latency is much higher because every packet has to be processed by the controller as there is no flow table entry installation in the switches, bringing about the bottleneck problem.

3.3.2 *Packets Capture*

In order to better understand how SDNs with OpenFlow works, the capture of some packets was necessary, in order to understand what are the messages exchanged.

The exchange of messages and their capture was performed in several use-cases, some of them were the Static Routing use case, Learning Switch use case and the VLANs use case.

In all the captures taken, they all have common features, they are either the entry of a packet that the network device does not recognize or they're a flow table entry insertion.

When the network is in reactive mode flow_mod type messages are ignored, using instead packet_out type messages.

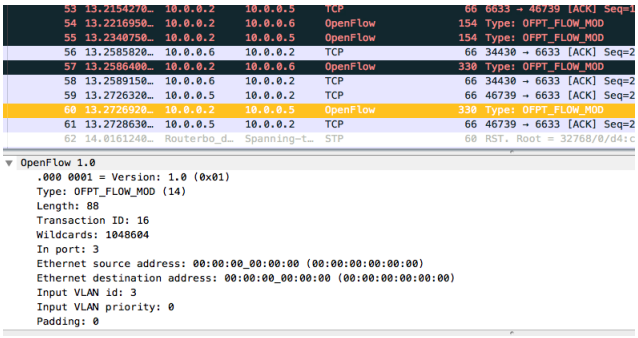

Figure 8: Latency in Reactive Mode Network

Another focus of analysis regarding captures and flow table entries are VLANs, as can be seen by analyzing the figure 8, all packets that are sent to VLAN 3 will match the correct flow entry in the switch's flow table and will be forwarded to the correct port.

It is noteworthy that in the vlans case the results were limited, making vlans work much better in a network with only one device rather than 3.

As can be seen in figure 9, the devices react as soon as an unexpected change occurs alerting the switch immediatly.

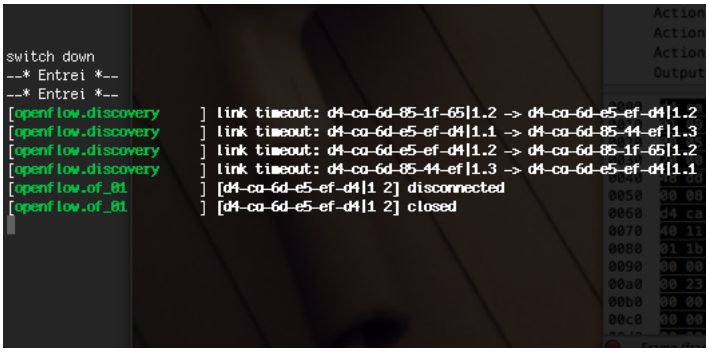

Figure 9: LLDP Switch Down

3.4 Summary

In summary, the hosts do not know what is going on behind the network or how the network is being managed, the

results obtained show that there are no high latency values in programmable networks.

4. CONCLUSIONS

4.1 Summary

Programmable networks (SDN) with the OpenFlow interface have distinguished themselves in the past few years as a potential substitute for current networks. In order to understand the behavior of this new paradigm the creation of a realistic environment that implements OpenFlow is necessary.

In order to make SDN more perceptible, a testbed was created implementing several use cases in a way that this new paradigm could be understood. To make this testbed the purchase of OpenFlow supporting devices was necessary.

This project introduces SDN as a test platform with the main goal of serving the academic community of Técnico Lisboa - Taguspark. This Thesis aims to create the testbed that will allow some insight into OpenFlow, such as scalability, mobility and versatility possibilities.

This document is orgnaized in 4 chapters. The first chapter pertains state-of-the-art, then in chapter 2 the proposed architecture is explained, leading to chapter 3 where implementation of the architecture is executed, and finally to prove that that programmable networks are a valuable asset an evaluation was performed in chapter 4.

In state-of-the-art we explain what OpenFlow and SDNs are, the applications made for this new paradigm, what tools exist and the choices made regarding them.

In chapter 2 we analyze the architecture to create the testbed. This architecture will always have a controller that allows transmission of instructions, management of the network, among others. This type of architecture is centralized,bringing with it bottleneck problems.

The implementation of this architecture involved the choice of equipment and software, and justifying the choices made. We also explain why we chose to implement these use-cases, how we did it and what was learned from implementing them. It is in the implementation chapter that problems start to arise and things don't turn out as expected.

Finally, to prove SDN is a valuable asset, several tests were necessary. These tests consist of network analysis, how the performance compared against current networks, and if the solution is scalable or not comparing to current networks.

After doing the network analysis, we found that SDNs are a valuable asset to the new networking paradigm, because besides being esaily configured, a significant reduction in configuration time was observed as well reduced frustration comparing to tradicional networks, which have to be configured device by device. Beyond these advantages the results were surprising, they showed that, performance-wise, SDNs are better than a traditional network, even though the first packet always has high latency values, because it has to be processed by the controller, the remaining packets have lower latency comparing to a traditional network.

To close the conclusion, according to demand, the created applications, the implementation of this type of networks in big enterprises, the network's performance level and reduction of CAPEX and OPEX, among others, allow us to state that the networking paradigm will change in 10 years time.

4.2 Learnings and Challenges

With this Thesis we learned that what is expected, alot of times does not happen. This investigation required a lot of research, working to understand the theory and trying to apply the knowledge at an applicational level, which did not happen due to obstacles that will be referenced next. It is important to note that this thesis should serve as a teaching to whoever chooses to study SDN. We encountered several obstacles such as:

- The existence of little information about OpenFlow, all of it being of a more theoretical nature than usual.
- Pox is an accessible and easily usable software, though being made by the comunity and non-comercial it has many flaws, such as:
  - Up until the lastest version, lauched in April de 2014, this software did not allow instructions to be sent to specific network devices.
  - Incomplete network protocols, the biggest example being the dhcp module.

    The problem with this module is that there is no way to distinguish between the interfaces, becoming a problem to the creation of vlans. In this case the solution adopted was to divide IP ranges thorughout the vlans and then create a firewall rule, forbidding them to communicate with each other.
  - Another problem was encountered creating vlans, caused by the software itself which did not allow the creation of vlans. The solution adopted was to create the vlans on the switch using them as interfaces and configuring them from the controller.
- The utilization of the oldest version of OpenFlow was another problem, making it difficult to configure the network and analysing if it could become a valuable asset.
- When the equipment arrived there were no OpenFlow packages installed, it took considerable time to find the OpenFlow package, at the time we were seriously considering the installation of OpenWrt.

Despite all these obstacles, we were able to overcome them. We learned that even though with many research on theory the investigation of a theme less known is very complex.

We realized that in 2013 there were already developments on SDNs with OpenFlow, most of it was theoretical, though many different software controllers were developed by equipment vendors The impact was huge and there are an increasing number of implementation contests for use-cases on this theme. It is also noteworthy that equipment vendors are coming up with their own solutions for SDNs, because they do not want to lose complete control over their equipment, making researchers dependent on using their solutions.

5. ACKNOWLEDGMENTS

I want to thank my advisor Prof. Mira da Silva who gave me unconditional support in all phases of the thesis.

6. REFERENCES

[1] Exploring networks of the future, 2013.
[2] Jgn2plus.

[3] Ofelia, 2010.

[4] Liberating network architectures with the open SDN. Big switch networks, February 2013.

[5] Open networking foundation, 2013.

[6] S. Elby. How openflow can revolutionize the carrier business, September 2012.

[7] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison. Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134, 2013.

[8] O. N. Foundation. Openflow switch specification version 1.3.1. Technical report, Open Networking Foundation, September 2012.

[9] I. Gashinsky. Sdn in warehouse scale datacenters sdn in warehouse scale datacenters v2.0, April 2012.

[10] N. Kim and J. Kim. Building netopen networking services over openflow-based programmable networks. In *Information Networking (ICOIN), 2011 International Conference on*, pages 525–529, January 2011.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[12] ONF. The google sdn wan. 2012.

[13] M.-K. Shin, K.-H. Nam, and H.-J. Kim. Software-defined networking (sdn): A reference architecture and open apis. In *ICT Convergence (ICTC), 2012 International Conference on*, pages 360–361, October 2012.

[14] S. Shin, N. Kim, N. Kim, and J. Kim. Flow-based performance enhancements of sage visualcasting using openflow programmable networking. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 1270–1274, February 2011.