Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2024, 11(1):231-237**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

SDN introduction and prospect

Alliance Kingst TONY-MAYEKO

PhD-Computer of Science
Department: Electronic and Telecommunication,
Research Unit: Laboratory Electrical Engineering
Collage: Marien NGOUABI University
School: ENSP-National Higher Polytechnic School
Country: Rep of Congo
Email: mayeko2005@yahoo.fr

Abstract The Internet has been a great success; it has become an indispensable universal tool for businesses and most individuals. However, despite their adoption, traditional networks are complex and difficult to manage. One of the reasons for this difficulty lies in the architecture of current networks, where the control plane and the data plane are vertically integrated into each network device. SDN is a new network paradigm that simplifies network management and innovation by separating network control logic from interconnects equipment, promoting centralization of control and the ability to program the network. In this article, we provide an overview of SDN. We start by introducing SDN, its architecture and its communication interfaces. We then describe the Openflow protocol, how it works and the main SDN controllers. We also examine the issues faced by SDN, focusing on key control plane challenges such as performance, scalability, security, and reliability, and discuss existing solutions to address these challenges.

Keywords Software-Defined Networking, Control Plane, Data Plane, Openflow, Programmable Networks

1. Introduction

With the advent of the Internet and new information technologies such as big data requiring distributed processing, cloud computing or the Internet of Things (IoT), traditional network architectures pose a major challenge for both operators and network administrators. In fact, for several years it has been very difficult, if not impossible, to innovate or make changes to the network. According to [1], designing or deploying a new routing protocol can take 5 to 10 years. Also, network configuration and management tasks are more complex. One of the reasons for this difficulty in designing or simply managing the networks is the strong coupling that exists between the control plane and the data plane of the connection equipment in current network architectures. In this context, the concept of Software Defined Networking (SDN) emerged to respond to cities. According to [4], SDN can also play a crucial role in the design of 5G radio networks. It is also possible to use the SDN with different approaches to improve the performance of the networks, for example using machine learning with SDN allows adding more intelligence to the networks thanks to the capabilities of the SDN.

Despite its advantages and ability to simplify networks, SDN faces challenges that can limit its functionality and performance in large networks. This article covers the main challenge.

We address the architectural rigidities of current networks, particularly by making them more programmable. The main idea of this new paradigm is to take out the intelligent part of the connection equipment and place it in a single control point called the controller, the latter offering a central view of the network, which on the one hand simplifies the network management and configuration.

SDN therefore has multiple advantages; it can serve multiple domains and be integrated with emerging technologies such as big data, machine learning, 5G, IoT and smart cities, thus offering programmability and a global, centralized view of the network [2].

For example, the programmability of SDN is particularly useful for big data applications that require numerous reconfigurations [3]. SDN also improves network resiliency and scalability, which are essential for large-scale IoT deployment, such as B. Intelligent performance, scalability, security and reliability issues with SDN controllers. Much work has focused on the topic of SDN, given its importance in the field of networks.

2. Materials and Methods

2.1 What is the software –defined network?

SDN is a new concept that describes a network architecture whose control plane is completely decoupled from the data plane. According to the ONF (Open Network Foundation) SDN is an architecture that separates the control plane from the data plane, and centralizes all network intelligence in a programmable entity called "Controller", in order to manage several elements of the data plane (e.g. switches or routers, etc.) via APIs (Application Programming Interface). =>More concretely, we can say that network architecture follows the SDN paradigm if, and only if, it verifies the following points:

The control plane is completely decoupled from the data plane, this separation is materialized through the definition a programming interface (Southbound API) -All network intelligence is outsourced to a logically centralized point called an SDN controller, which provides a global view of the entire physical infrastructure.

The SDN controller is a programmable component that exposes an API (NorthboundAPI) to specify control applications.

2.2 Architecture of software defined network

A traditional network generally consists of connecting devices such as switches and routers. This device contains both the transmission part and the network control part. With this architectural model, it is difficult to develop new services due to the strong coupling between the control plane and the transmission plane. To open network devices to innovation, the SDN architecture was born. It enables the control part to be decoupled from the transmission part of the interconnection device.

The SDN is mainly composed of three layers and communication interfaces (Figure 1), we describe in what follows these layers, as well as the communication interfaces:

The transmission layer: also called "data plane", it is composed of routing equipment such as switches or routers, its main role is to transmit data and collect statistics.

The control layer: also called "control plane", it consists mainly of one or more SDN controllers, its role is to control and manage infrastructure equipment through an interface called 'south-bound API '.

The application layer: represents the applications that make it possible to deploy new network functionalities, such as traffic engineering, QoS, security, etc. These applications are built through a programming interface called 'north-bound API

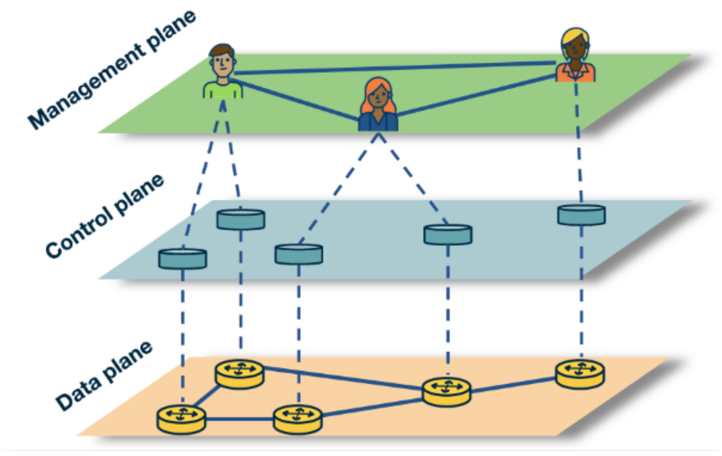


Figure 1: Architecture SDN

2.3 Openflow

Open flow is the protocol used for communication between the transmission layer and the control layer, it was originally proposed and implemented by Stanford University and then standardized by the ONF, its latest version is 1.5. This architecture is mainly based on three components:

The data plane, which consists of Openflow switches;

The control plane, consisting of OpenFlow controllers;

A secure chain that allows switches to connect to the control plane. The specification of an openflow switch is standardized by the ONF.

According to the ONF specification, an openflow switch must contain one or more flow tables, these flow tables contain several entries corresponding to rules, each mainly consisting of the following three fields (Table 1):

The packet header: it defines the data flow; it contains the information necessary to determine the packet to which this rule applies. The packet header can identify different protocols like Ethernet, IPv4, IPv6 or MPLS depending on the Openflow specification used.

The action: specifies how the packets of a stream are processed. An action can be one of the following forward the packet to one or more ports, drop the packet, forward the packet to the controller, or change the packet header field.

Counters: are reserved for collecting flow statistics. They record the number of packets and bytes received from each stream, and the time elapsed since the last stream was transmitted.

Table 1: Structure of a flow table entry of an openflow 1.0 switch

Header Fields	Counter	Actions

2.3.1 Openflow operation

When a packet arrives at a switch, the switch checks whether there is an entry in the flow table that matches the packet header. If so, the switch takes the appropriate action in the flow table. Otherwise, i.e. no corresponding entry, the switch generates an asynchronous message to the controller in the form of a Packet_in, then the controller decides an action for this packet according to its setup and sends a new forwarding rule as Packet_out and Flow-mod to the switch, and finally the switch flow table is updated to reflect the new rule installed by the controller. Figure 2 [14] describes the process of transmitting a packet with Openflow.

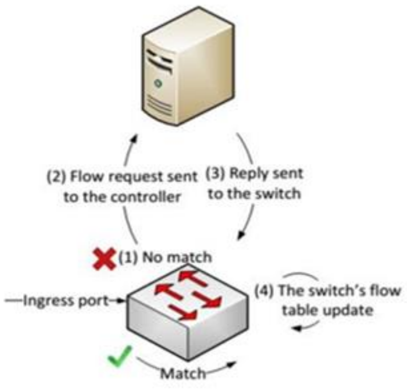

(2) Flow request sent to the controller
(3) Reply sent to the switch
(1) No match
Ingress port
(4) The switch's flow table update
Match

Figure 2: Packet transmission process with openflow

The exchange of information between the switch and the controller takes place by sending messages through a secure control channel using TLS (Transport Layer Security).

2.4 The Orchestrator

The Orchestrator responds to the need to be able to program the network from end to end. When the application is deployed in a data center, users can be connected via WiFi, connected to the LAN, behind the WAN and firewalls or other network devices, which are often independent domains that form a chain of execution, making the task virtually impossible for the controller.

For example, in operators' service provider architectures; it is not the controller but the orchestrator that offers this end-to-end programming capability. The latter receives an order from an application and then performs the following actions necessary to complete the requested task. In order to fulfill its mission, the orchestrator can rely on the deployed controllers for each element of the chain.

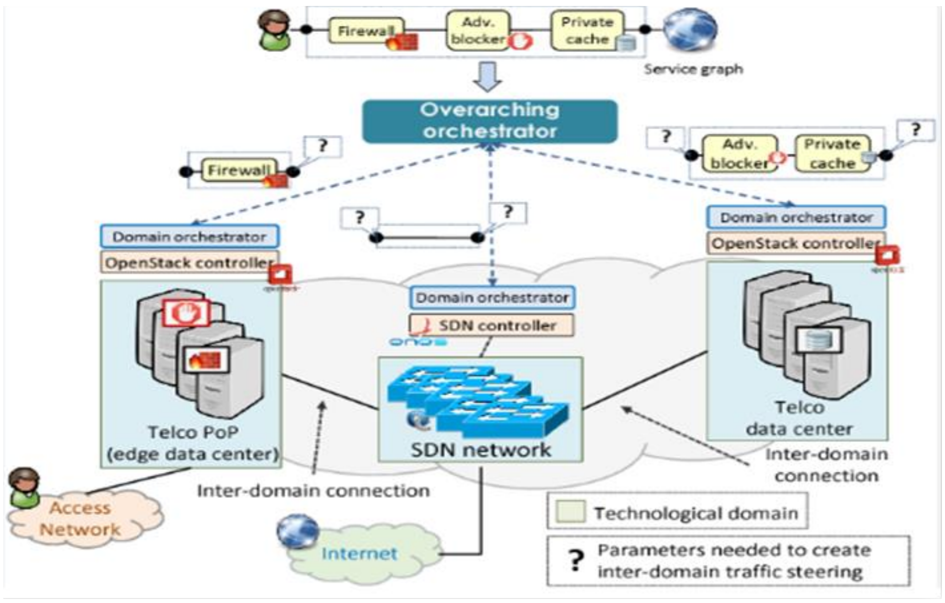


Figure 3: SDN architecture with an orchestrator

2.5 Cases of SDN

The SDN market has been constantly evolving since its appearance on the world stage in 2011, many advantages provided by technology in the various areas. This means that the use cases are very varied and span all the parts of the network.

2.5.1 QoS on the Internet

Internet has always been an architecture that does not guarantee stability, or new generation video streaming, VoD or other video conferencing services are not very tolerant of delays and transmission errors and therefore require a stable network for packet forwarding. Based on the centralized view of the network that SDN offers, we can choose different paths for the different traffic flows depending on the rate. Hence the concept of VSDN (Video over SDN), an architecture that determines the optimal path based on the global view offered by the centralization of control.

2.5.2 Data Center

Oracle SDN is an example of a system that provides a virtualized data center network. This system dynamically connects virtual machines to network servers. With the Oracle Fabric Manager interface, configuration and monitoring of virtual networks is possible from anywhere, and provisioning of new services such as firewall, load balancing and routing is done on demand. According to Oracle, their proposal increases application performance by 30x and can save up to 80 Gbps server-to-server.

2.5.3 Cellular Networks:

With the advent of 5G, operators must optimize their infrastructure to handle ever-increasing amounts of data while ensuring the quality of service that comes with this technology. SDN is one of the building blocks that enable operators to unlock infrastructure potential and maximize flow rates while maintaining significant visibility into the behavior of their network devices. An architecture called CSDN (Cellular SDN) has been proposed, an approach that uses SDN concepts for dynamic and centralized utilization of resources, collects user data and network conditions, and escalates them to the controller to optimize power consumption, resource usage and customization of customer service user.

2.5.4 Security

The dynamic nature of SDN was exploited by a group of researchers (Jafarian, Al Shaer & Qi Duan 2012) who successfully reduced the recovery risk by 99% of the information from an external packet analyzer. Their approach is based on random mutation, and host IP address suggestions have been made quite frequently, including mechanisms to mitigate Denial of Service (DDoS) attacks. Two key elements were exploited in the proposed architecture: IDS for intrusion detection in the LAN and Openflow switches for dynamic isolation of infected devices (Juba, Huang & Kawagoe, 2013).

2.5.5 Campus and corporate network

The last few years have been marked by a radical shift in perception of work equipment in companies and on the university campus. Indeed ModeBYOD (Bring Your Own Device), in which each employee and student uses their own hardware on campus, leading us to rethink the network to give it a touch of flexibility, without the need of the network administrator very quickly be surprised by the amount of traffic managed manually and by the manual operations that could allow, for example, to connect all Internet users. SDN, whose advantages are automation and network programming, seems like a no-brainer to fill the gaps in campus networks that are becoming increasingly complex. One of the most exciting applications for the campus beyond virtualization is the programming of an application-aware network (application-aware routing).

3. Results & Discussion

3.1 Results

The OpenFlow Manager application written in **Node.js** is deployed to the north interface of the controller on a web server listening on PORT 9000. The topology is immediately retrieved and displayed on the GUI as follows

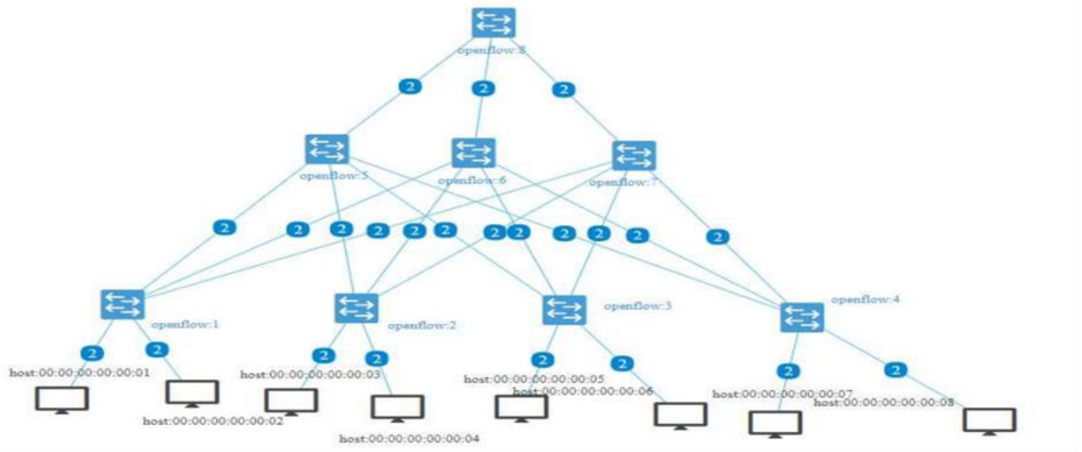


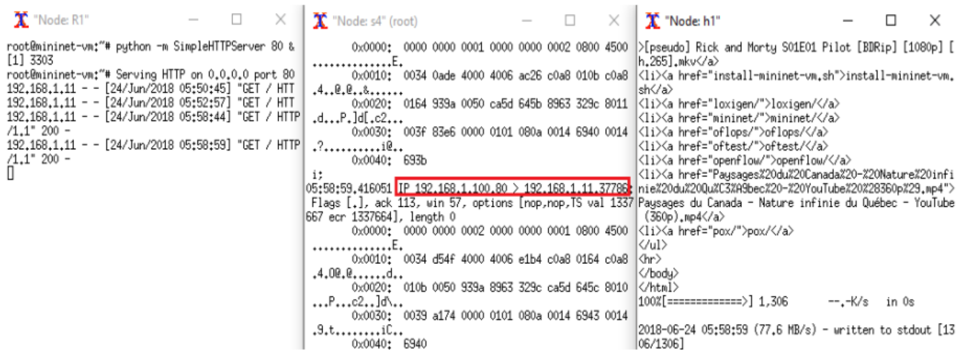
Figure 4: Scenario of topology

To confirm the proper functioning of our solution, 3 tests were performed.

3.1.1 HTTP testing

To confirm traffic is going through Switch S4, we simulated a Server HTTP on R1 using a Simple HTTP Server included in Python's standard library. A **wget** command is run on client h1 with the server address. The **tcpdump** tool was used on the switch terminal to display all data traffic over the selected interface.

The test shows the following:



```
root@mininet-vm:~# python -m SimpleHTTPServer 80 &
[1] 3303
root@mininet-vm:~# Serving HTTP on 0.0.0.0 port 80
192.168.1.11 - - [24/Jun/2018 05:50:45] "GET / HTT
192.168.1.11 - - [24/Jun/2018 05:52:57] "GET / HTT
192.168.1.11 - - [24/Jun/2018 05:58:44] "GET / HTTP
/1.1" 200 -
192.168.1.11 - - [24/Jun/2018 05:58:59] "GET / HTTP
/1.1" 200 -
▯
```

```
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500
 ..............E.
        0x0010:  0034 0ade 4000 4006 ac26 c0a8 010b c0a8
 .4..@.@..&......
        0x0020:  0164 939a 0050 ca5d 645b 8963 329c 8011
 .d...P.]d[.c2...
        0x0030:  003f 83e6 0000 0101 080a 0014 6940 0014
 .?..........i@..
        0x0040:  693b
 i;
05:58:59.416051 IP 192.168.1.100.80 > 192.168.1.11.37786:
 Flags [.], ack 113, win 57, options [nop,nop,TS val 1337
667 ecr 1337664], length 0
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500
 ..............E.
        0x0010:  0034 d54f 4000 4006 e1b4 c0a8 0164 c0a8
 .4.O@.@......d..
        0x0020:  010b 0050 939a 8963 329c ca5d 645c 8010
 ...P...c2..]d\..
        0x0030:  0039 a174 0000 0101 080a 0014 6943 0014
 .9.t........iC..
        0x0040:  6940
```

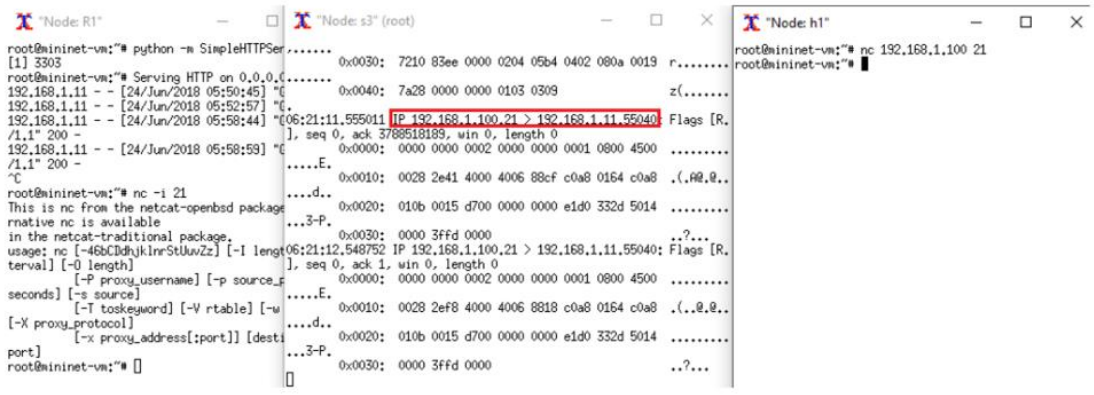
```
>[pseudo] Rick and Morty S01E01 Pilot [BDRip] [1080p] [
h.265].mkv</a>
<li><a href="install-mininet-vm.sh">install-mininet-vm.
sh</a>
<li><a href="loxigen/">loxigen/</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="Paysages%20du%20Canada%20-%20Nature%20infi
nie%20du%20Qu%C3%A9bec%20-%20YouTube%20%28360p%29.mp4">
Paysages du Canada - Nature infinie du Québec - YouTube
 (360p).mp4</a>
<li><a href="pox/">pox/</a>
</ul>
<hr>
</body>
</html>
100%[============>] 1,306       --.-K/s   in 0s

2018-06-24 05:58:59 (77.6 MB/s) - written to stdout [13
06/1306]
```

Figure 5: HTTP application test

The result is self-explanatory:
HTTP communication between client and server is well established

3.1.2 FTP testing

The tool used for the FTP transfer is Netcap. To do this, the command on the client side is n/a
On switch s3 the command used is: **tcpdump –XX –n –i s3-eth1**



```
root@mininet-vm:~# python -m SimpleHTTPSer
[1] 3303
root@mininet-vm:~# Serving HTTP on 0.0.0.(
192.168.1.11 - - [24/Jun/2018 05:50:45] "(
192.168.1.11 - - [24/Jun/2018 05:52:57] "(
192.168.1.11 - - [24/Jun/2018 05:58:44] "(
/1.1" 200 -
192.168.1.11 - - [24/Jun/2018 05:58:59] "(
/1.1" 200 -
^C
root@mininet-vm:~# nc -i 21
This is nc from the netcat-openbsd package
rnative nc is available
in the netcat-traditional package.
usage: nc [-46bCDdhjklnrStUuvZz] [-I lengt
terval] [-O length]
          [-P proxy_username] [-p source_p
seconds] [-s source]
          [-T toskeyword] [-V rtable] [-w
[-X proxy_protocol]
          [-x proxy_address[:port]] [desti
port]
root@mininet-vm:~# ▯
```

```
.......
        0x0030:  7210 83ee 0000 0204 05b4 0402 080a 0019  r.......
.......
        0x0040:  7a28 0000 0000 0103 0309                 z(.......
.
06:21:11.555011 IP 192.168.1.100.21 > 192.168.1.11.55040: Flags [R.
], seq 0, ack 3788518189, win 0, length 0
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ........
.....E.
        0x0010:  0028 2e41 4000 4006 88cf c0a8 0164 c0a8  .(.A@.@..
....d..
        0x0020:  010b 0015 d700 0000 0000 e1d0 332d 5014  ........
...3-P.
        0x0030:  0000 3ffd 0000                           ..?...
06:21:12.548752 IP 192.168.1.100.21 > 192.168.1.11.55040: Flags [R.
], seq 0, ack 1, win 0, length 0
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ........
.....E.
        0x0010:  0028 2ef8 4000 4006 8818 c0a8 0164 c0a8  .(..@.@..
....d..
        0x0020:  010b 0015 d700 0000 0000 e1d0 332d 5014  ........
...3-P.
        0x0030:  0000 3ffd 0000                           ..?...
▯
```

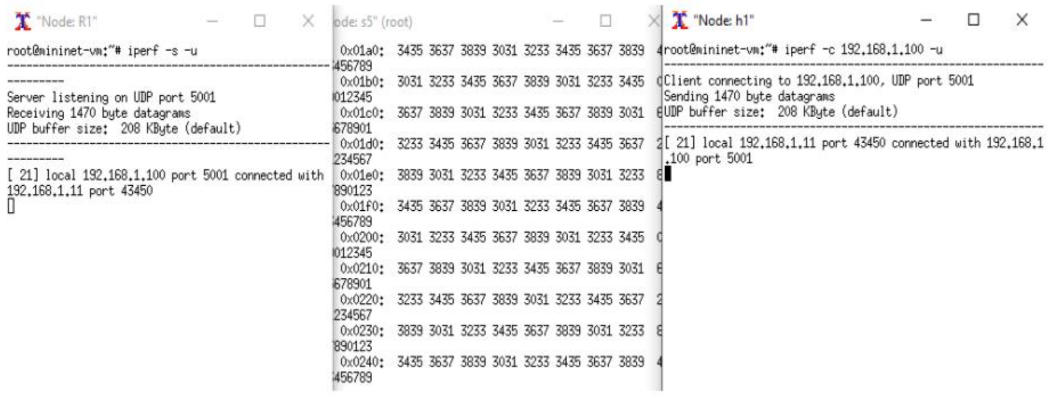
```
root@mininet-vm:~# nc 192.168.1.100 21
root@mininet-vm:~# █
```

Figure 6: FTP application test

The results confirm that the objective has been achieved.

3.1.3 UDP testing

For the streaming or VoIP application that is transported over the UDP protocol (e.g. SIP with asterisk), iperf is used as the test tool. As shown in the picture below



```
root@mininet-vm:~# iperf -s -u
------------------------------------------------------------
---------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
---------
[ 21] local 192.168.1.100 port 5001 connected with
192.168.1.11 port 43450
▯
```

```
  0x01a0:  3435 3637 3839 3031 3233 3435 3637 3839  4
456789
  0x01b0:  3031 3233 3435 3637 3839 3031 3233 3435  0
012345
  0x01c0:  3637 3839 3031 3233 3435 3637 3839 3031  6
678901
  0x01d0:  3233 3435 3637 3839 3031 3233 3435 3637  2
234567
  0x01e0:  3839 3031 3233 3435 3637 3839 3031 3233  8
890123
  0x01f0:  3435 3637 3839 3031 3233 3435 3637 3839  4
456789
  0x0200:  3031 3233 3435 3637 3839 3031 3233 3435  0
012345
  0x0210:  3637 3839 3031 3233 3435 3637 3839 3031  6
678901
  0x0220:  3233 3435 3637 3839 3031 3233 3435 3637  2
234567
  0x0230:  3839 3031 3233 3435 3637 3839 3031 3233  8
890123
  0x0240:  3435 3637 3839 3031 3233 3435 3637 3839  4
456789
```

```
root@mininet-vm:~# iperf -c 192.168.1.100 -u
------------------------------------------------------------
Client connecting to 192.168.1.100, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 21] local 192.168.1.11 port 43450 connected with 192.168.1
.100 port 5001
█
```

Figure 7: UDP application test

On the s5 switch, we see the influx of data between the client and the server, which demonstrates passing traffic through s5.

3.2 Discussion

We have described the most common management operations. In addition to the simplicity of implementation and the significant time savings, this affects the programming of the behavior of the networks. The results showed the reliability of the architecture in terms of adhering to predefined policies. A reachability test showed the behavior of the network in relation to restrictions, the latter confirming the compliance of the communication with what was set in advance. A test to determine the direction of traffic validated the method used to define the paths taken by the flows.
The goal was achieved on the redundancy side. On the other hand, we came to the conclusion that OpenDaylight is not the ideal controller for an environment off-fault tolerant production. The fact that the user interface has to be refreshed to see the disappearance of a faulty connection can lead to a deactivation, unlike some controllers such as ONOS, which have shown very high reliability in such situations.

4. Key Contributions

The strong development of Big Data in organizations requires an IT infrastructure to match.

The contributions of SDN (Software defined network) are very valuable: it makes it possible to streamline data flows and avoid certain setbacks by guaranteeing the best network performance.

Big Data can benefit SDN, in particular for traffic engineering, for cross-layer developments (between layers of services), for cyber-security or for intra and inter-SDN data center networks, and other.

SDN platforms based on NFV (Network Function Virtualization) functionalities aim to automate decisions that were previously made by human managers responsible for operations. The decision-making process relies heavily on the use of APIs (programmatic interfaces).

In a software-defined network, security is centralized. In this central controller, we can easily create security policies and distribute them throughout the company.

SDN enables the rapid movement of workloads across a network.

SDN-WAN brings more simplicity and automation to branch networks

5. Conclusion

Our objective in this work is to understand what the SDN represented, to then implement it and deduce its contribution to the behavior of the network while testing its reliability.

Through this work, we have used many testing tools that confirm the reliability of this architecture. The tests we carried out have confirmed the possibility of using an SDN network in the company network. Regarding the routing of sensitive applications, we proposed a method to direct data flows according to applications. We used the standard OpenFlow application and feature listening ports. Because latency is a disruptive element for real-time applications, our approach avoided the inconvenience caused by this factor by redirecting UDP and http application traffic in a way that avoids their latency dysfunction. Since the operation of an FTP application is not disturbed by the lags, we did not think it worthwhile to redirect it to another connection, which allows traffic optimization.

From the test results, we can say that the SDN challenges the model networks and offers many solutions for programming the behavior of data flows.

As perspectives for this work, we plan to conduct further in-depth studies on security and quality of service in the SDN.

The strength of this layer lies in the application layer and its permeability for the immediate delivery of services. We will therefore look at methods that make better use of this ability.

Acknowledgements

We thank to the journal team, for their valuable comments and suggestions which have led to an improvement of the manuscript.

References:

[1]. [D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, et S. Uhlig], « Software-Defined Networking: A Comprehensive Survey », Proc. IEEE, vol. 103, no 1, p. 14-76, janv. 2015.

[2]. [P. C. da R. Fonseca and E. S. Mota], « A Survey on Fault Management in Software-Defined Networks », IEEE Commun. Surv. Tutor. vol. 19, no 4, p. 2284-2321, 2017.