People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering

Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

MASTER

In **Electronics**

Option: **Computer Engineering**

Title:

Comparison between APIC-EM and Network programmability in hybrid SDN.

Presented By:

- **Bachir SABRI**
- **Ismail Belqassim RAHMANIA**

Supervisor:

Mr. A. MOHAMMED SAHNOUN

Registration Number:……../2020

Abstract

The traditional methods used for network equipment configuration are time consuming, and hard to manage. It is both difficult to configure and maintain the network according to predefined policies and difficult to meet the growth in networks.

For today's networks, software-defined networking (SDN) looks to be the best suited solution to overcome difficulties. SDN provides traffic programmability, agility and the ability to create policy driven networks. SDN simplifies the control of the network by the decoupling of control and data plane, southbound protocols and northbound APIs. But due to SDN's big and expensive requirements, organizations and industries tend to favour Network programmability. Network programmability became a trend, enhanced and inspired by Software Defined Networks, that are based on scripting methods and standard programming languages used for the control and monitoring of network elements.

The project begins with defining SDN and explaining its main concepts and how it differs from traditional Networking.

A discussion will follow on the layered SDN architecture, an analysis of the control plane then a section discussing southbound and northbound interfaces. After that, Cisco's solution APIC-EM is presented along with a working example. Finally, Network programmability is tested in a simulated network of traditional devices, then compared to its APIC-EM counterpart.

Keywords: SDN, Controller, APIC-EM, NAPALM, NETMIKO.

Dedication

I, Rahmania Ismail Belqassim, dedicate this humble work

to my father and mother who are my stars and the beacons of my hope.

to my sister and brothers who never fail to show their utmost support and always watch over me.

to my friends T.Salah, B.Bachir, B.Oussama, M.Smail, B.Riad, G.Abdelghani, B.Bilel, M.Mahdi, M.Oussama and M.Houssam who have stood by my side when I most needed help and supported me. I am blessed to have you involved in my life and hope for your continuous patronage.

Thank you very much.

“Ismail”

Every challenging work needs self-efforts as well as guidance and orientation of those who are very close to our heart. I dedicate my humble work

To my parents, the reason of what I become today and who have always supported me and oriented me towards the good.

To my brothers and sisters and to all my family and friends for their help to follow my studies, and their encouragements and sacrifices.

To my supervisor Mr. A. MOHAMMED-SAHNOUN and all the teachers and workers of IGEE

To all those whom I love and who love me

“Bachir”

Acknowledgement

We would like to first and foremost thank Allah Almighty who gave us the strength and the courage allowing us to finish this modest work.

Then we would like to extend the thanks to our supervisor

Mr. A. Mohammed-Sahnoun who was always there for us when guidance was needed.

We also thank the members of the jury for agreeing to judge and to evaluate our work and for sharing their valuable remarks with us and

We are very thankful for all our teachers for their efforts that helped us to accumulate our modest knowledge and improve our skill set while studying at IGEE

Thank you.

TABLE OF CONTENTS

Abstract.....**II**

General Introduction.....**1**

Chapter I INTRODUCTION TO SDN.....

Introduction.....3

I.1 Modern Networking Concepts and Elements.....3

I.1.1 The OSI model.....3

I.1.2 Routers.....4

I.1.3 Switches.....4

I.1.4 Servers.....5

I.1.5 Clients.....5

I.1.6 Cabling(wiring).....5

Coaxial Cables.....5

Twisted Pair Cables.....6

Fiber Optics.....6

USB Cables.....7

Serial and Parallel Cables.....7

Crossover Cables.....7

Other Types of Network Cables.....8

I.2 Traditional Networking limitations.....8

I.3 Modern Network requirements.....8

I.4 Data Center and Virtualization.....9

I.4.1 Data center.....9

I.4.2 Server Virtualization.....9

Hypervisors 9

Virtual Machines.....10

I.4.3 Advantages of Virtualization.....10

I.5 SDN architecture and operation.....10

I.5.1 SDN architecture principles.....10

I.5.2 Classic and Hybrid SDN.....11

I.5.3 SDN architecture components.....12

I.5.3.1 Data (Infrastructure) plane.....12

I.5.3.2 Control plane.....13

I.5.3.3 Application plane.....13

I.5.4 SDN and NFV.....13

SDN (Software Defined Network) 13

NFV (Network Functions Virtualization) 13

I.5.4.1 NFV architecture.....14

Network access control and segmentation of classes of users…14

Path isolation.....14

Network Services virtualization.....14

Conclusion.....16

Chapter II SDN CONTROL PLANE.....

Introduction.....17

II.1 Southbound interfaces.....17

II.2 Northbound interfaces.....18

II.3 SDN Controller.....19

II.3.1 SDN Control Plane functions.....19

II.3.2 SDN controller models.....21

II.3.3 SDN controller vendors.....21
II.3.4 SDN controller platforms.....22
Floodlight.....22
OpenDaylight.....22
OpenContrail.....22
Open Network Operating System (ONOS).....22
II.4 Application Policy Infrastructure Controller Enterprise Module (APIC-EM) 23
II.4.1 APIC-EM Main functions.....23
Host and devices discovery.....23
Devices inventory.....24
The topology.....24
II.5 Limitations of centralized SDN controller.....24
Scalability.....24
Reliability.....24
Security.....25
Conclusion.....25
Chapter III APIC-EM SOLUTION AND ITS IMPLEMENTATION.....
Introduction.....26
III.1 APIC-EM controller.....26
III.1.1 Installing APIC-EM.....26
III.1.2 REST APIs.....27
III.1.3 Features of APIC-EM.....27
III.1.3.1 Automated Discovery.....27
Discovery Jobs and Scanning Credentials.....28
III.1.3.2 Device and Host Inventories.....30

Devices and Hosts.....30
Device.....30
Host.....30
Device Inventory.....30
Host Inventory.....31
III.1.3.3 Topology.....32
III.1.4 APIC-EM applications.....33
Path Trace.....33
EasyQoS.....33
Cisco Intelligent WAN (IWAN).....33
Cisco Network Plug and Play (PnP).....34
Back Up and Restore.....34
Audit Logs.....34
III.1.5 APIC-EM example.....34
III.1.5.1 Devices configuration.....35
III.1.5.2 Basic configuration.....35
Router ALGIERS.....36
Router BOUMERDES.....37
Router BLIDA 38
SWITCH (Positioned in Algiers) 38
III.1.5.3 Using APIC-EM.....40
Discovery.....41
Inventory.....42
Topology.....43

Path Trace.....43

Conclusion.....44

Chapter IV NETWORK PROGRAMMABILITY USING PYTHON.....

Introduction.....45

IV.1 Network Automation using Python.....45

IV.1.1 Why Python for network programmability.....45

IV.1.2 Benefits of Python for Network automation.....45

IV.1.3 Definitions.....46

NAPALM.....46

PARAMIKO.....46

NETMIKO.....46

LANSCAN.....46

IV.1.4 Installing and Configuring VMware Workstation.....47

IV.1.5 Installing and Configuring the controller.....47

IV.1.6 Installation of the required Tools.....49

IV.1.7 Network Architecture.....50

IV.1.7.1 Device Configurations.....50

IV.1.7.2 Basic Configurations.....51

IV.1.7.3 Network Discovery (using lanscan) 53

IV.1.8 Device Configurations and management by the controller.....54

The Routers.....59

Forming trunk links.....59

Configure routing.....60

IV.1.9 Reading the State of Devices.....61

Reading the state of the routers.....61

Reading the state of the switches.....64

IV.2 Network Automation with ansible tool and YAML playbook……...66

IV.2.1 Installing Ansible on the controller.....66

IV.2.2 Backup of the running configuration with ansible.....67

IV.2.3 Code explanation and execution.....67

The results.....68

IV.3 APIC-EM controller vs Automation using Python in hybrid SDN.....69

Conclusion.....70

General Conclusion.....71

Bibliography.....

Figure References……

List of Tables

TABLE II.1. SDN Vendors and Platforms.....22

TABLE III.1. Addressing Table (APIC-EM example) 35

TABLE IV.1. Addressing Table (Python automation) 51

TABLE IV.2. VLAN Table.....54

TABLE IV.3. APIC-EM vs Python Automation.....70

List of Figures

FIGURE I.1. OSI model.....3

FIGURE I.2. Router.....4

FIGURE I.3. Switch.....4

FIGURE I.4. Coaxial Cable.....5

FIGURE I.5. UTP and STP Cables.....6

FIGURE I.6. Fiber Optic Cable.....6

FIGURE I.7. USB Cables.....7

FIGURE I.8. Parallel and Serial Cables.....7

FIGURE I.9. Crossover Cables.....8

FIGURE I.10. Data Center.....9

FIGURE I.11. VirtualBox and VMware (Hypervisors).....10

FIGURE I.12. Classic SDN model.....11

FIGURE I.13. Hybrid SDN model.....12

FIGURE I.14. SDN Architecture Layers.....12

FIGURE I.15. Comparison between SDN and NFV.....14

FIGURE I.16. Diagram displaying an example NFV Architecture scheme.....15

FIGURE II.1. Northbound and Southbound Interfaces.....17

FIGURE II.2. Applications communication with the Controller via Northbound APIs18

FIGURE II.3. The SDN Control layer interaction with the other layers through APIs..20

FIGURE II.4. SDN using APIC-EM 23

FIGURE III.1. New discovery window.....28

FIGURE III.2. SNMP v2c Credentials.....29

FIGURE III.3. CLI Credentials.....29

FIGURE III.4. Device Inventory.....31

FIGURE III.5. Host Inventory.....31

FIGURE III.6. Topology view.....33

FIGURE III.7. Example Network Topology.....34

FIGURE III.8. Sub-interfaces configuration.....36

FIGURE III.9. Status of IPv4 interfaces.....36

FIGURE III.10. Routing table of Algiers.....37

FIGURE III.11. SSH configuration commands.....37

FIGURE III.12. SNMP configuration commands.....37

FIGURE III.13. Routing table of Boumerdes router.....38

FIGURE III.14. Routing table of Blida router.....38

FIGURE III.15. Switch Interfaces status.....38

FIGURE III.16. Switch interfaces mode configuration.....39

FIGURE III.17. Switch trunk interface details.....39

FIGURE III.18. Switch VLAN interface details.....39

FIGURE III.19. APIC-EM home page.....40

FIGURE III.20. APIC-EM CLI Credentials window.....41

FIGURE III.21. APIC-EM SNMP configuration window.....41

FIGURE III.22. APIC-EM Discovery page.....41

FIGURE III.23. APIC-EM Discovery process results.....42

FIGURE III.24. APIC-EM inventory page.....42

FIGURE III.25. APIC-EM Topology page.....43

FIGURE III.26. APIC-EM Path Trace navigation panel.....43

FIGURE III.27. Path Trace results page.....44

FIGURE IV.1. VMware Workstation Pro Install menu.....47

FIGURE IV.2. New Virtual Machine creation.....47

FIGURE IV.3. New Virtual Machine Operating System Installation.....48

FIGURE IV.4. New Virtual Machine Hardware settings.....48

FIGURE IV.5. Python 3.8 Installation.....49

FIGURE IV.6. NAPALM installation.....49

FIGURE IV.7. NETMIKO installation.....49

FIGURE IV.8. Network Topology.....50

FIGURE IV.9. Interface configuration and static routing.....52

FIGURE IV.10. SSH configuration.....52

FIGURE IV.11. Management interface configuration.....53

FIGURE IV.12. Switch management interfaces connectivity test (pings)….....53

FIGURE IV.13. VLAN configuration from the controller using a python script.....54

FIGURE IV.14. Controller access to the Switch.....55

FIGURE IV.15. TRUNK links configuration on Switch SW1.....56

FIGURE IV.16. TRUNK links configuration on Switch SW2.....56

FIGURE IV.17. TRUNK links configuration on Switch SW3.....56

FIGURE IV.18. TRUNK links configuration on Switch SW4.....56

FIGURE IV.19. Automated configuration Python script.....57

FIGURE IV.20. Configuration script execution from the Controller.....57

FIGURE IV.21. TRUNK configuration between distribution and access layer Switches and Server assignment to VLANs.....58

FIGURE IV.22. Spanning-Tree Protocol configuration script.....59

FIGURE IV.23. Spanning-Tree Protocol primary/secondary configuration script execution……59

FIGURE IV.24. Python script to create a sub-interface with dot1q encapsulation on the Router.....60

FIGURE IV.25. Execution of the script creating a sub-interface on the Router……60

FIGURE IV.26. Python script for Routing configuration (with OSPF) 61

FIGURE IV.27. Execution of the Routing configuration script from the Controller…61

FIGURE IV.28. Python script to display the Routing configuration…….62

FIGURE IV.29. Routing information saved as Routing_info.txt……..62

FIGURE IV.30. Python script that shows the Routing table and stores it in a file……63

FIGURE IV.31. Routing table for R1.....63

FIGURE IV.32. Routing table for R2.....63

FIGURE IV.33. Ping results from the Router to homeClient and the SERVERS…….64

FIGURE IV.34. Python script showing VLANS configuration information……64

FIGURE IV.35. VLAN Table of SW6.....65

FIGURE IV.36. VLAN Table of SW7.....65

FIGURE IV.37. Python script that shows STP configuration information……65

FIGURE IV.38. SW1 STP root bridge information.....65

FIGURE IV.39. SW3 STP port states…….66

FIGURE IV.40. Verifying ansible version.....67

FIGURE IV.41. YAML playbook to backup configuration.....67

FIGURE IV.42. Hosts file.....67

FIGURE IV.43. ansible configuration file.....67

FIGURE IV.44. Ansible Playbook execution.....68

FIGURE IV.45. Ansible execution result.....69

List of abbreviations and acronyms

Abbreviation	Meaning
API	Application Programmable Interface
APIC-EM	Application Policy Infrastructure Controller – Enterprise Module
CLI	Command Line Interface
CPU	Central Processing Unit
SDN	Software Defined Networking
OSI	Open System Interconnection
PDU	Protocol Data Unit
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
UTP	Unshielded Twisted Pair
STP	Shielded Twisted Pair
WAN	Wide Area Network
USB	Universal Serial Bus
PC	Personal Computer
NIC	Network Interface Card
VM	Virtual Machine
NFV	Network Functions Virtualization
NAT	Network Address Translation
DNS	Domain Name System
IDS	Intrusion Detection System
VLAN	Virtual Local Area Network
QoS	Quality of Service
DHCP	Dynamic Host Configuration Protocol
SNMP	Simple Network Management Protocol
BGP	Border Gateway Protocol

Abbreviation	Meaning
LISP	Local Identifier Separation Protocol
I2RS	Interface to Routing Systems
REST	Representational State Transfer
OS	Operating System
ONF	Open Networking Foundation
ONOS	Open Network Operating System
ISR	Integrated Service Router
GUI	Graphical User Interface
SSH	Secure SHell
NAPALM	Network Automation and Programmability Abstraction Layer with Multivendor support
RAM	Read Only Memory
NTP	Network Time Protocol
HTTP	Hyper Text Transfer Protocol
CDP	Cisco Discovery Protocol
OSPF	Open Shortest Path First
EIGRP	Enhanced Interior Gateway Routing Protocol
AD	Administrative Distance
STP	Spanning-Tree Protocol
ACL	Access Control List
YAML	YAML Ain't Markup Language (originally Yet Another Markup Language)

GENERAL INTRODUCTION

General Introduction

Networking technologies are evolving constantly and rapidly to answer the ever-rising challenges of the industry. This evolution is bringing about great changes (solutions) to networks in terms of efficiency, reliability and security.

Network virtualization started a revolution, allowing for network programmability and efficient device management through some concepts that changed the way networks operate. SDN is one of the concepts that emerged nearly a decade ago and is still in constant development.

The theme of this project was proposed by our supervisor Mr. A. MOHAMMED-SAHNOUN, based on the fact that SDN is a new technology underdevelopment that is getting place in Algeria today.
The problematic that pushed us to find an answer through this work is:
« How does APIC-EM perform as an industrial SDN solution compared to Network Automation using Python? »

This project totals four chapters and a general conclusion. The first chapter will introduce Software Defined Networking (SDN) architecture and show what SDN brings to the table compared to traditional networking schemes.
The second chapter will focus on the SDN control plane. The SDN controller will be considered along with its functions and performance. This chapter will also touch upon the Southbound and Northbound interfaces that are used to communicate with the underlying infrastructure and the applications respectively. After that, a brief introduction to the Cisco APIC-EM is in order, followed by some limitations of SDN controllers.
The third chapter will delve into the Cisco APIC-EM previously introduced, in detail. An operational example will also be discussed by the end of the chapter.
The fourth and last chapter will be about Network Automation using Python libraries (NETMIKO, NAPALM). In this chapter too, an example will be laid then explored in detail, then at the end a comparison between Cisco's APIC-EM and Python Automation will be held to determine the better solution.

The conclusion will recapitulate what was said throughout the chapters then answer the problematic that was the incentive of this work.

CHAPTER 1
INTRODUCTION TO SDN

Introduction:

Software-defined networking (SDN) is a rising network management technology that emerged after the need to revolutionize some traditional networking aspects and constructs to achieve better efficiency. It enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring [1].

Traditional networks tend to be complex and decentralized, that is what made the new networking trends aspire for easy troubleshooting and flexibility. SDN tries to solve that by making a focal point to centralize the control in the network, then separates the forwarding scheme from the routing one. The former is referred to as the Data plane and the latter as the control plane.

I.1 Modern Networking concepts and elements:

I.1.1 The OSI model:

The OSI architecture is the highest level of abstraction. The OSI model represents communication systems in terms of layers that partition them functionally, without considering structure or technology. Each of those layers provides services to the layer above and receives services from the one below. The purpose of OSI is to allow application processes to initiate, maintain and terminate communication [2].

The table below shows and describes the different OSI layers:

OSI model				
Layer			Protocol data unit (PDU)	Function[14]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

FIGURE I.1. OSI model

I.1.2 Routers:

Routers are networking devices that forward traffic between networks. Data is sent as 'packets' from a router to another through the networks that they connect, to the final destination.

Each router uses a routing table/policy to forward packets to their destination by reading their destination addresses from the packet header [3].

FIGURE I.2. Router

I.1.3 Switches:

A network switch is a hardware device that channels incoming data from multiple input ports to a specific output port that will take it toward its intended destination [4]. It is a small device that transfers data packets between multiple network devices such as computers, routers, servers or other switches.

FIGURE I.3. Switch

The destination of a frame in a LAN is determined by the switch by looking at the physical device address (also known as the Media Access Control address or MAC address). Switches maintain tables that match each MAC address to the port from which the MAC address is received.

A network switch operates on the network layer, called layer 2 of the OSI model.

I.1.4 Servers:

Servers are computers that provide services to one or more end devices using specific software for each service. Additionally, a single computer can run multiple types of server software [5]. Many types of servers exist including web servers, mail servers and file servers, each type runs software specific to the purpose of the server.

I.1.5 Clients:

Clients are end devices (computers) in the network. They are equipped with software that allows them to receive the services provided by Servers [5]. A single computer can run multiple types of client software.

I.1.6 Cabling(wiring):

Cables are still widely used in networking systems for data transfer nowadays despite the advent of wireless options.

- **Coaxial Cables**

Coaxial cables are most known for being used to connect television sets to home antennas. They are also a standard for 10MBps Ethernet cables.

FIGURE I.4. Coaxial Cable

- **Twisted Pair Cables**

Twisted pair cables started as a standard for Ethernet and evolved as improvements in transfer speed were made with each new version (10MBps then 100MBps then later 10GBps). The wires inside the cable are twisted to minimize electromagnetic interference and as a protection against cross talk [6].

FIGURE I.5. UTP and STP Cables

- **Fiber Optics**

Fiber optic cables are made of strands of glass that transmit light pulses. They are flexible and see a lot of usage in long distance and/or high traffic WAN.

There are 2 types of fiber optic cables: Single mode and Multimode. The former is mainly used in WAN for its high bandwidth capacity, while the latter sees use in LAN networks to reduce costs [7].

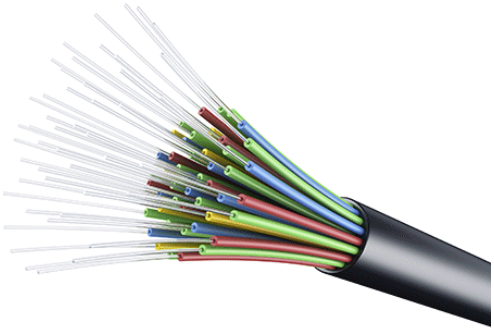

FIGURE I.6. Fiber Optic Cable

- **USB Cables**

USB cables are mainly used in interconnections between a computer and a peripheral device. But they can be used to link two computers via adapters.

FIGURE I.7. USB Cables

- **Serial and Parallel Cables**

Serial and Parallel cables are outdated today in favor of Ethernet. They still can be used nevertheless to establish a (relatively slow) connection between the Serial ports of two computers.

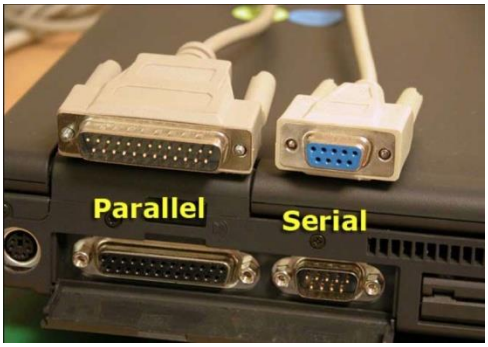

FIGURE I.8. Parallel and Serial Cables

- **Crossover Cables**

Crossover cables are typically used to link the same type of devices. They have color coded wires visible on their connectors (the order differs to distinguish manufacturers).

Crossover cables do not see use anymore because routers nowadays have built-in crossover capabilities [7].

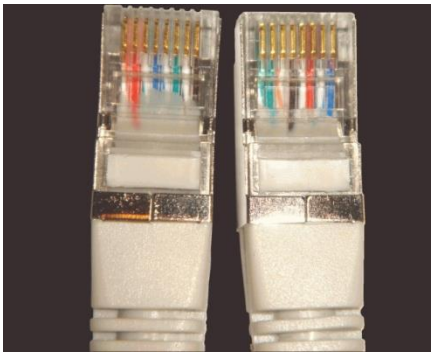

FIGURE I.9. Crossover Cables

- **Other Types of Network Cables**

Patch cables as their name suggests are cables used temporarily in the network usually for testing or troubleshooting purposes. They are available in many types (coaxial, twisted pair, fiber optic etc...) and tend to be short [7].

I.2 Traditional Networking limitations:

Great changes in the networking field were expected, but they were too fast for traditional networking. In fact, even though traditional networking schemes are still operational and network devices are being constantly upgraded, they are showing their limitations in terms of dealing with complexity, variability and high load volume [8]. Also, the embedded and delocalized nature of the control plane makes maintenance and troubleshooting a daunting task the larger the network is, because it must be done single handedly on every device, which opens possibility for human errors.

As networks become more complex, it becomes hard to manage them while dealing with varying security requirements, traffic volumes and quality of service. For example, scaling a network demands from the administrator to reconfigure many devices manually, a task that gets impractical and inefficient the bigger the network is.

I.3 Modern Network requirements:

The ever-growing demands of the field ask for revolutionary changes in networking to meet the needs of the users in terms of:

- Flexibility
- Operational maintainability
- Scalability

- Embedded security
- Mobility
- Automation
- Layered management

I.4 Data Center and Virtualization:

I.4.1 Data center:

Data centers are facilities equipped with computers, power supplies, storage systems and connections to establish the data center network. They need to be highly secure and reliable to answer the large traffic, provide quality services and stay operational in case a problem arises.

Large data centers are industrial-scale operational structures that use as much electricity as a small town [9].

FIGURE I.10. Data Center

Definition:

Managing, developing and troubleshooting software is usually easier than its hardware counterpart. This advantage can be manifested in the form of Data center virtualization. Virtualization allows to virtualize the physical devices and networks [10].

I.4.2 Server Virtualization:

- **Hypervisors:**
  Hypervisors are usually software (or hardware) used to enable the creation of virtual machines on abstract platforms set up on the real physical hardware.

Each of these virtual machines runs a complete and separate operating system. Examples of hypervisor software include VMware Fusion, Oracle Virtual Box, Solaris Zones and VMware Workstation.

FIGURE I.11. VirtualBox and VMware (Hypervisors)

- **Virtual Machines:**

  A virtual machine (VM) is a software program or operating system that works as a separate computer, it is also capable of running applications and programs like a real computer. The physical computer is the host, and the VMs (one or many) created on it are called guests.

I.4.3 Advantages of Virtualization:

1. Less equipment is required
2. Less energy is consumed
3. Less space is needed
4. Faster server provisioning
5. Improved disaster recovery

I.5 SDN architecture and operation:

SDN is an architecture that uses a 'Controller' (control plane) to mediate between the applications (application plane) and the underlying infrastructure (data plane). This approach presents an optimized and simplified network model compared to traditional schemes. The controller allows applications to request services from the infrastructure, and the infrastructure to provide them in return [11].

I.5.1 SDN architecture principles:

The SDN architecture operates on some principles, mainly:

- Separation of data plane from control plane.
- A main (central) controller or a set of coordinated controllers implement the control plane.

- Open interfaces are defined between the devices in the control plane (controllers) and those in the data plane.
- Applications running on the SDN controllers enable the programming of the network.

I.5.2 Classic and Hybrid SDN:

There are two approaches in SDN: Classic and Hybrid SDN. The classic SDN approach may or may not fit the user depending on what is required and is not without its limitations.

In Classic SDN, the control plane is completely parted from the data plane and implemented in a central control module 'The Controller'. This approach goes against most networking designs nowadays. The controller must be highly performant, available, scalable and secure. Satisfying those conditions is a big challenge, especially under heavy traffic and fast requests.

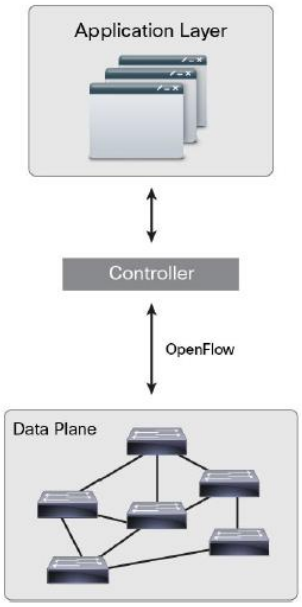

FIGURE I.12. Classic SDN model

On the other hand, Hybrid SDN makes use of network programmability to allow the application layer (APIs) to interact with network devices and take advantage of their inherent hardware intelligence. There are many advantages to traditional networks, and the Hybrid SDN approach tries to capitalize on that by adding a centralized controller without discarding those features, hence a partial separation of the control plane from

the infrastructure. This approach is more efficient and performant when compared to traditional networking schemes, while being more practical and less challenging to implement compared to Classic SDN [12].

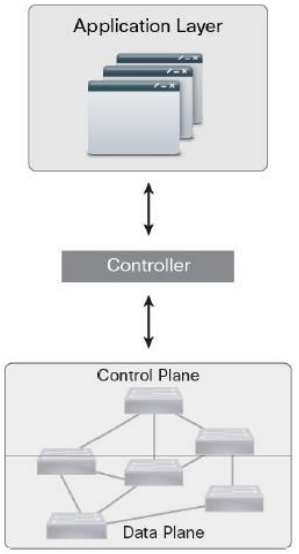

FIGURE I.13. Hybrid SDN model

I.5.3 SDN architecture components:

SDN architecture is structured as three layers that communicate through bounds (APIs) as shown in the FIGURE I.14.

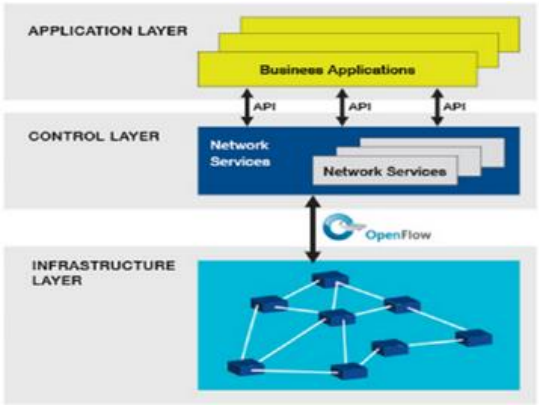

FIGURE I.14. SDN Architecture Layers

I.5.3.1 Data (Infrastructure) plane:

The data plane is the part of the network through which user packets are transmitted. It is a theoretical term used to conceptualize the flow of data packets through a network

infrastructure. It is often included in diagrams and illustrations to give a visual representation of user traffic [13].

The data plane is also known as the user plane, the forwarding plane or the carrier plane.

Routing protocols manage network traffic using routing/forwarding tables. The data plane forwards traffic to the next hop along the path based on information learned from data plane packets.

In software-defined networking (SDN), the data plane is found in the software rather than the firmware. The decoupling of the user plane and the control plane allows for greater flexibility and dynamic control in state-of-the-art network architectures.

I.5.3.2 Control plane:

The control plane handles the communications between the application plane and the data plane. It translates the requirements of the applications and provides them with an abstract view on the data layer via northbound interfaces.

I.5.3.3 Application plane:

In the application layer, applications interact with the controller to communicate their needs and the ideal network behaviour required for their operations.

I.5.4 SDN and NFV

We will distinguish the two architecture concepts SDN and NFV.

- **SDN (Software Defined Network)**: architecture that separates physically the data transfer (forwarding) plane from the control plane which controls several devices.
- **NFV (Network Functions Virtualization)**: is an architecture that uses virtualization technologies to manage network functions essentially through software rather than hardware. The NFV concept is based on blocks of visualized network functions which can be combined to provide tailor-made and upgradable (scalable) network services [14].

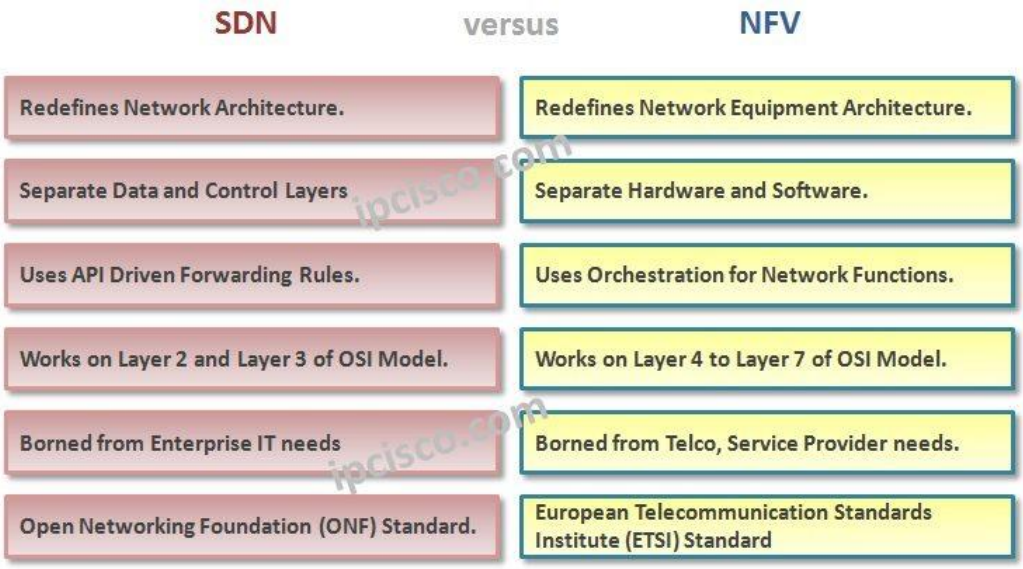

FIGURE I.15. Comparison between SDN and NFV

Concretely, these functions are executed by virtual machines running on the hardware; these are routers, switches, servers or cloud computing systems. The proposed functions are security and firewall, NAT, DNS, caches, intrusion detection (IDS), etc.

NFV and SDN both use an abstraction layer. While SDN seeks to separate control and transfer, NFV puts an abstraction layer in network transfer functions (and others) provided by hardware. The two concepts are implemented in a complementary way but also exist independently of each other.

I.5.4.1 NFV architecture:

Network Functions Virtualization architecture is made up of three elements:

- Network access control and user classes segmentation: Access control prompts users to provide credentials to access their appropriate logical partition. Those partitions are accessible to some and denied to some other users depending on the class attribution attributed to the specific user.
- Path isolation: Traffic is partitioned through isolated networks. These networks are connected via routers. The isolated paths are mapped to VLANs and virtual services.
- Network services virtualization: Network services such as security, quality of service (QoS), address management (DHCP/DNS), policy partition and application isolation are provided [15].

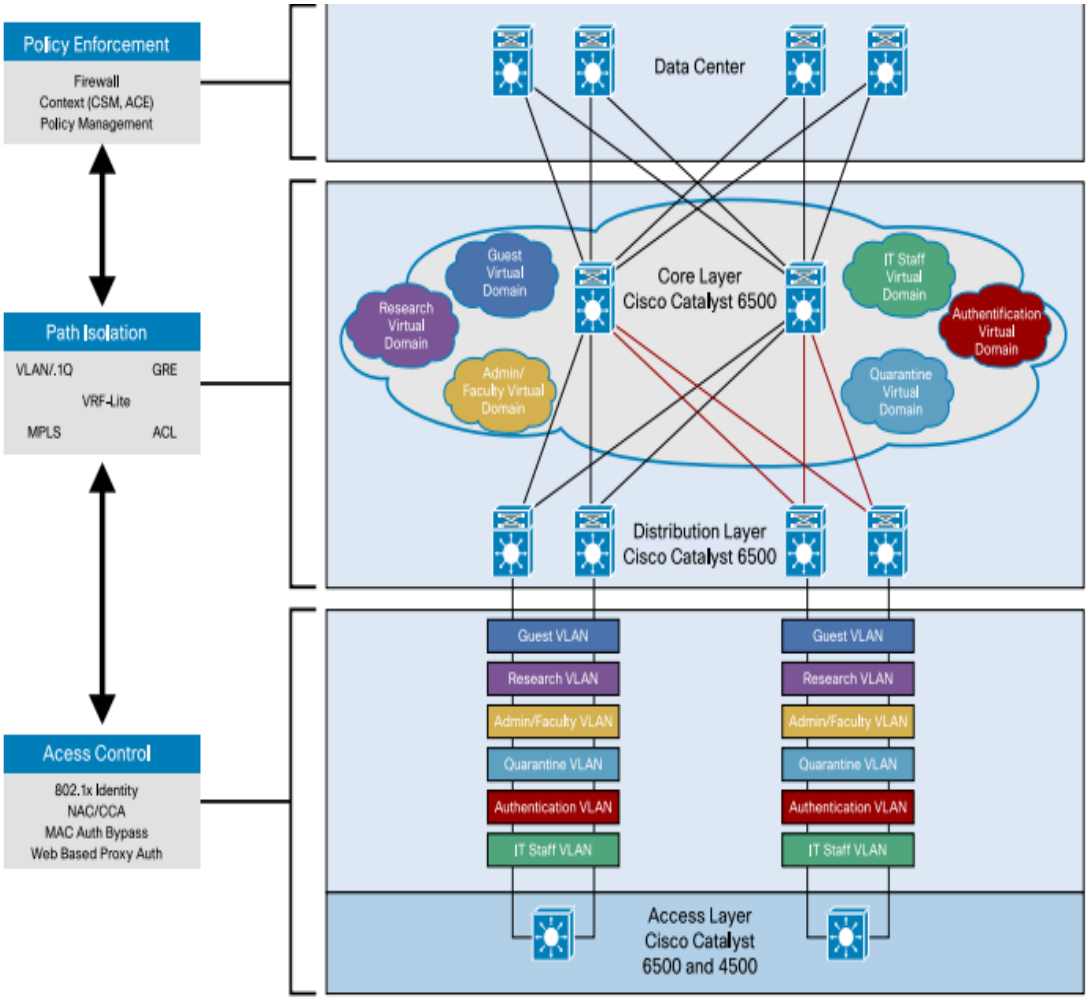

FIGURE I.16. Diagram displaying an example NFV Architecture scheme

The division and sharing of hardware resources allows for an efficient use of physical hardware, especially when there is a need to buy new devices to handle network services. That is the strength of network services virtualization. It has numerous advantages, namely:

1. Reduce costs in purchasing network equipment.
2. Faster time to deployment.
3. Flexibility in scale up or down of capacity.
4. Efficiencies in space, power and cooling.

Conclusion:

Software Defined Networking announces major changes to networks in the years to come. These will see their architecture profoundly evolve, facilitating new uses. All of this will be possible thanks to the programmability, openness, virtualization and orchestration of the SDN architecture. No area seems to be spared: WAN, data centers, campus, security… The challenge for network administrators is to support this new step in order to be able to take advantage of these new capabilities.

The next chapter will focus on the Control plane.

CHAPTER 2
SDN CONTROL PLANE

Introduction:

SDN architecture separates the Control plane (layer) from the Data plane, enabling the use of software through open interfaces to control the network in terms of traffic and connectivity [16]. The SDN Control plane is the centralized logic entity controlling how data traffic is managed. It is in that sense the 'medium' that gives a view on the infrastructure layer to the Application plane.

In this second chapter, the SDN Controller and its functions, performances and some controller platforms from different vendors will be discussed, as well as Southbound and Northbound interfaces. Then, “Application Policy Infrastructure Controller Enterprise Module” known as “APIC-EM” which is Cisco's SDN controller implementation will be introduced. Finally, some disadvantages and limitations of the SDN controller will be presented.

II.1 Southbound interfaces:

In order to communicate between the Controller and the infrastructure, SDN southbound APIs (Application Program Interfaces) are used. They can be either subject to intellectual propriety or open source. Southbound APIs enable the Controller to manage the network efficiently and flexibly depending on the needs [17].

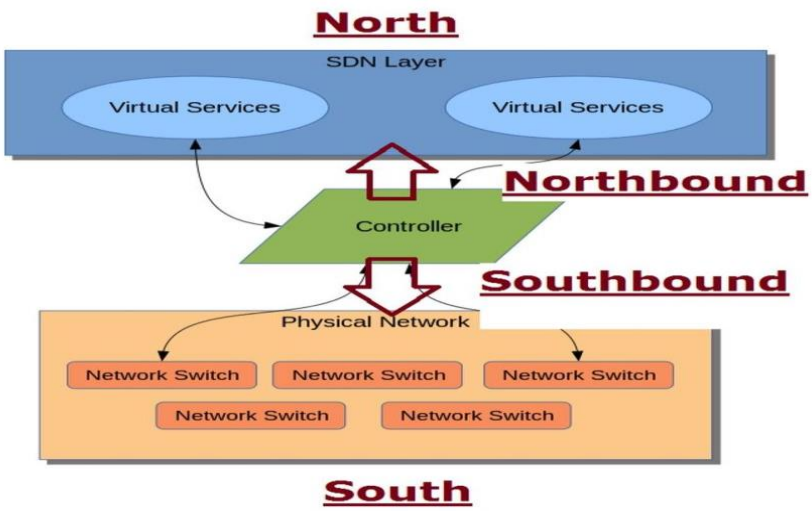

FIGURE II.1. Northbound and Southbound Interfaces

Among southbound interfaces, OpenFlow is the most widely recognized one. It was developed by the Open Networking Foundation (ONF). OpenFlow sets the way in

which the controller interacts with the data layer. It manages the flow tables of the switches/routers in order to efficiently deal with the incoming traffic [17].

For traditional network devices, they work with SNMP as management southbound API to collect information about the traffic and status of the elements. Beside OpenFlow and SNMP, there exist other southbound API protocols like I2RS (Interface to Routing Systems), the Open vSwitch Data Base (OvSDB), NetConf, and LISP and BGP.

II.2 Northbound interfaces:

SDN Northbound APIs handle the communication between the Controller and the services/applications running over the network. The role of these APIs (taking advantage of SDN network programmability) is to enable network automation by sending application requests from the application layer to the controller. The network then provides services/information back to those applications.

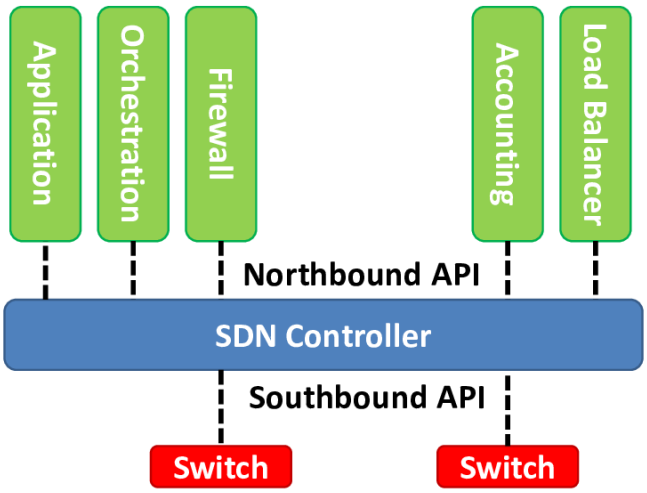

FIGURE II.2 Applications communication with the Controller via Northbound APIs

The good thing about Northbound APIs (usually REST APIs) is that there is a myriad of interfaces to choose from, depending on what applications are to be used.

These applications range from load balancers, firewalls to other SDN security services.

The SDN Controller is integrated with automation stacks, namely: Puppet, Chef, SaltStack, Ansible and CFEngine via Northbound APIs. The same can be said about

orchestration platforms like Openstack, vCloudDirector of VMware and Apaches's open source CloudStack [18]. Northbound APIs make work easier for the application developers, because they enable the applications to interact with the network as a black box, thus removing the need to know the inner workings of the network.

II.3 SDN Controller:

When speaking of the control plane, the discussion will inevitably revolve around the SDN Controller. It is an application that allows for improved network management and better application performance. The SDN controller is usually installed on a server, from which it can control the switches.

An SDN controller manages flow control to the switches/routers 'below' (via southbound APIs) and the applications and business logic 'above' (via northbound APIs) to deploy intelligent networks.

The SDN controller uses forwarding policies instilled by the network administrator to manage traffic, thus reducing the need to configure each device manually. The separation between the control and data layers makes network management and automation easy, because the centralized control acts as an operating system (software) for the network (hardware) [19].

II.3.1 SDN Control Plane functions:

It is not an understatement to say that the SDN Controller is the heart of the network. It interacts and mediates between the network devices on one side, and the applications on the other.

Northbound interfaces enable the SDN controller to communicate with applications such as firewalls or load balancers. Southbound interfaces like Openflow for example, allow the Controller to interact with the network devices to configure them or choose traffic path.

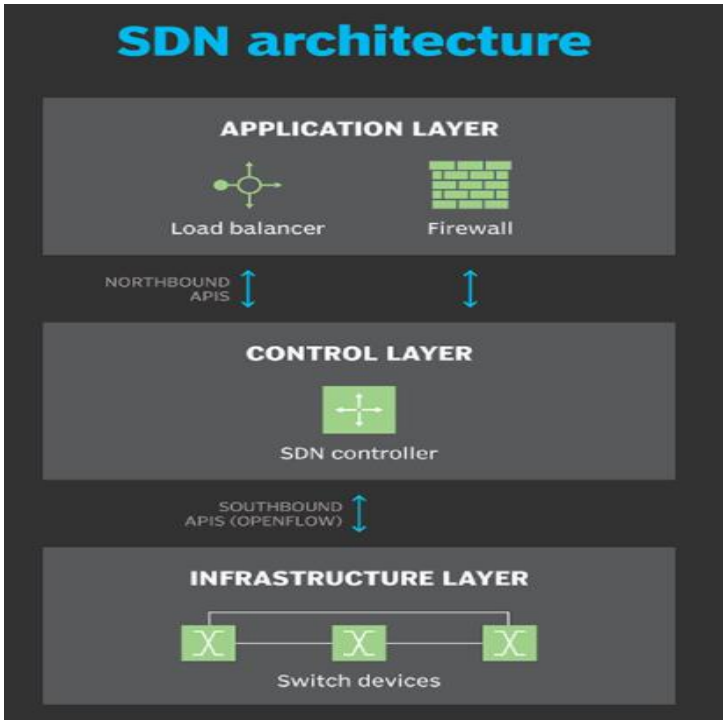

FIGURE II.3. The SDN architecture Layers

The ONF created OpenFlow in 2011, and a work group focused on Northbound APIs development in 2013. But because application requirements vary a lot, there was no standardized set in the industry.

SDN controllers have many functions that enable them to mediate between the Application plane and the infrastructure layer, these functions are:

- Shortest path forwarding: finding and choosing the optimal route by gathering information from the network.
- Notification managing: Receives, processes, and forwards to/from application events, such as alarm notifications, security alarms, and state changes.
- Security mechanisms: Provides isolation and security enforcement between applications and services.
- Topology managing: Builds and maintains switch interconnection topology information.
- Statistics managing: Collects data on traffic through the switches.
- Device managing: Configures switch parameters and attributes and manages flow tables [20].

An advantage of SDN centralized control is that the controller has a bird eye on the network. That allows it to redirect traffic flow through the available paths depending on what is required. The controller also notifies the network operator in case of congested links.

II.3.2 SDN controller models:

There are mainly 2 approaches in terms of SDN controller models: The Centralized model and the Distributed model.

The centralized SDN model makes a centralized controller manage and oversee the entire network. This controller acts like a single logically centralized decision point, the brain of the network [21].

The distributed SDN model operates by making several controllers share traffic loads and tasks. These controllers communicate through East/Westbound interfaces to provide more reactivity, redundancy, consistency and scaling, especially in data centers [22].

Both models have their inherent advantages and should be carefully considered. But companies can -- and should -- use more than one controller (Distributed Model), adding a backup for redundancy. In the industry, three controllers tend to be a common number among both commercial and open source SDN options. This adds redundancy allowing the network to continue running in the event of connectivity loss or controller susceptibility.

The main inconvenience in the centralized model is the fact that if the SDN Controller encounters a problem, it will cause the whole network to malfunction or stop activity. In such an event, the lack of redundancy is very noticeable, and the network must wait for the Controller to resume normal operation. The Distributed model is less susceptible to this problem due to the redundancy provided by the presence of multiple controllers.

II.3.3 SDN controller vendors:

Vendors that provide SDN controllers are numerous; some of them are listed in the table below:

TABLE II.1. SDN Vendors and Platforms

Company	SDN Platform
Big Switch Networks	Big Switch Controller
Cisco	APIC-EM
Juniper Networks	Open Contrail
VMware	VMware NSV
Plexxi	Plexxi Controller
Nuage Networks	Virtualized Services Platform (VSP)
Midokura	Midonet
Andara Networks	Andara sky controller

II.3.4 SDN controller platforms:

SDN controllers are available in a variety of open source options, including:

- **Floodlight**: Floodlight is a community-developed, open source, Java OpenFlow controller, supporting OpenFlow protocols. The project is managed by SDN start-up Big Switch Networks. It's a widely popular platform and is commonly thought to be one of the easier SDN platforms to implement
- **OpenDaylight**: OpenDaylight is by far the most well-known. OpenDaylight is a collaborative project led by the Linux Foundation. The software is written in Java and claims extensive compatibility support from vendors such as Big Switch and Cisco [23].
- **OpenContrail**: OpenContrail is an Apache 2.0 licensed open source project that provides a complete SDN and network virtualization solution for cloud infrastructure. The OpenContrail project architecture addresses networking requirements [24].
- **Open Network Operating System (ONOS)**: An open source alternative to OpenDaylight and similar spinoff platforms is ONOS, ONOS uses a unique method for exposing northbound and southbound interfaces in such a way that the application policy engines, interface protocols and orchestration systems are completely independent [20].

II.4 Application Policy Infrastructure Controller Enterprise Module (APIC-EM):

In Enterprises today the hardware that Cisco offers are the catalyst switches, ISR routers, ASA firewalls and wireless controllers. All of these do not support the new protocols of SDN like OpenFlow. And in order not to replace all the hardware and infrastructure, Cisco introduced a solution to keep the hardware and benefits/advantages of the SDN model, which is the Cisco application policy infrastructure controller-Enterprise Module (APIC-EM) [25].

FIGURE II.4. SDN using APIC-EM

It is provided with a built-in application like:

1. Network Visibility Application
2. EasyQoS Application
3. Path Trace Application
4. IWAN (Intelligent WAN) Application
5. Network Plug and Play Application

II.4.1 APIC-EM Main functions:

Host and devices discovery:

To start using the APIC-EM controller, the first important task to do is to discover hosts and devices and add them to the inventory database. To start discovering devices you must configure SNMPv2c credentials or SNMPv3 credentials or both SNMPv2c and

SNMPv3 credentials (depending on your network) as well as the CLI credentials which are mandatory [26].

Devices inventory:

After the discovery of devices is completed, all the devices discovered throughout the scan that are not Hosts are displayed in the Device Inventory page. There, detailed information about the devices can be consulted.

Each item in the Device Name column of the Device Inventory view is a link to additional information about a specific device. You can view the running-configuration which is in the router or switch's memory just by one click.

In the host inventory you can see the different information related to each device such as its MAC address, IP address, host name and the connected interfaces [26].

The topology:

The Topology window displays a graphical view of your network. Using the discovery settings that you have configured, the Cisco APIC-EM discovers and maps devices to a physical topology with detailed device-level data.

II.5 Limitations of centralized SDN controller:

The logically centralized SDN model faces some problems and challenges in terms of scalability, reliability and security as the surrounding environment of the controller is enhanced. These points hurt the viability and the network performance of the centralized model, especially when compared to its distributed model counterpart.

Scalability:

The number of devices an SDN controller can manage is limited, which makes it a daunting task to deal with tremendous traffic intensity and large numbers of devices, like in data centers.

This makes scalability quite difficult to achieve in wide area SDN.

Reliability:

A centralized controller is a single point of failure. This means that if the controller fails for whatever reason, the network will stop working. This hurts the reliability of the network.

Security:

A centralized controller is susceptible to attacks because it is hard to secure. In order to protect domain information, we choose to implement different privacy policies in different SDN domains, some networking information in one domain should not be disclosed to an external entity, which is impossible to make in case of a centralized controller. These attacks can compromise the whole network [27].

Conclusion:

The Control plane is the logical center of the SDN. It eases network management through the controller which has the critical task of providing abstractions, essential services, and common APIs to developers. Network administrators can opt for a centralized or a distributed SDN model. While the former is more likely to cost less and is easier to setup, the latter provides a better performance in terms of reliability, security and scalability. APIC-EM is an SDN controller that provides common functionalities as network state and network topology information, device discovery, and distribution of network configuration.

In the next chapter, Cisco's APIC-EM will be discussed in further detail.

CHAPTER 3
APIC-EM SOLUTION AND ITS IMPLEMENTATION

Introduction:

The previous chapters have introduced some SDN concepts and features. Some of those features will be useful in discussing the SDN deployment on a network infrastructure.
In this chapter, the APIC-EM controller will be presented and discussed in detail. Then an example will be provided to have a look at its operation.

III.1 APIC-EM controller:

The Cisco Application Policy Infrastructure Controller - Enterprise Module (APIC-EM) is Cisco's Software Defined Networking (SDN) Controller for Enterprise Networks (Access, Campus, WAN and Wireless).

The platform hosts multiple applications (SDN apps) that use open northbound REST APIs that drive core network automation solutions. The platform also supports a number of south-bound protocols that enable it to communicate with the breadth of network devices that customers already have in place and extend SDN benefits to both green field and brown field environments.

The Cisco APIC-EM controller upholds both wired and remote industry networks over the Campus, Branch and WAN foundations. It offers the accompanying advantages:

- Creating intelligent and programmable network using APIs.
- Saves time, assets and costs through progressed automation.
- Turn business policies into dynamic network configuration.
- Providing a centralized network control and automation [28].

The APIC-EM has many features to map and manage the network.

III.1.1 Installing APIC-EM:

The installation is done on a VMware hypervisor. There are many hardware requirements: A minimum storage capacity of 500 GB with a disc in/out speed of at least 200 MBps. A minimum of 32 GB RAM must be configured for the virtual machine that contains the Cisco APIC-EM when a single host is being deployed. The single host server that contains the virtual machine must have this much RAM physically available. For a multi-host deployment (2 or 3 hosts), 32 GB of RAM is required for each of the virtual machines that contains the Cisco APIC-EM. Three servers are required for high availability and redundancy.

For the virtual machine options, 6 vCPUs are the minimum number required for the virtual machine configuration.

Web access is required along with an updated web browser (Google chrome or Mozilla Firefox for example) or VMware vSphere Client. Configuration of the timing of the guest VM to an NTP server is also required to avoid conflicting time settings [29].

To install Cisco APIC-EM, the ISO image containing the controller must be downloaded.

For detailed installation steps, refer to last year's work.[38]

III.1.2 REST APIs:

REST, short for Representational State Transfer, is a programming model used for developing and to invoke Web services. Its flexibility, scalability and portability made it the 'go to' choice in the industry [30].

- **Lightweight:** No special manipulation, programming, knowledge or skills are required to use REST applications.
- **Flexible:** Users can use REST for simple requests or for complex operations.
- **Scalable:** Depending on the server specs, REST requests can scale as much the servers can handle because they run on the Web server.
- **Platform-agnostic:** REST operates on all platforms, any HTTP-enabled host can send REST requests and receive REST responses, regardless of OS (whether on Windows, Mac OS, Linux) and device(on a mobile device, such as a smartphone or a tablet.)

The GUI of APIC-EM uses REST API so that applications communicate with the Controller.

III.1.3 Features of APIC-EM:

III.1.3.1 Automated Discovery:

The Cisco APIC-EM controller finds network devices and hosts by applying a scan over the network, the scan tries to authenticate each device it finds. The process of finding network devices is known as discovery.

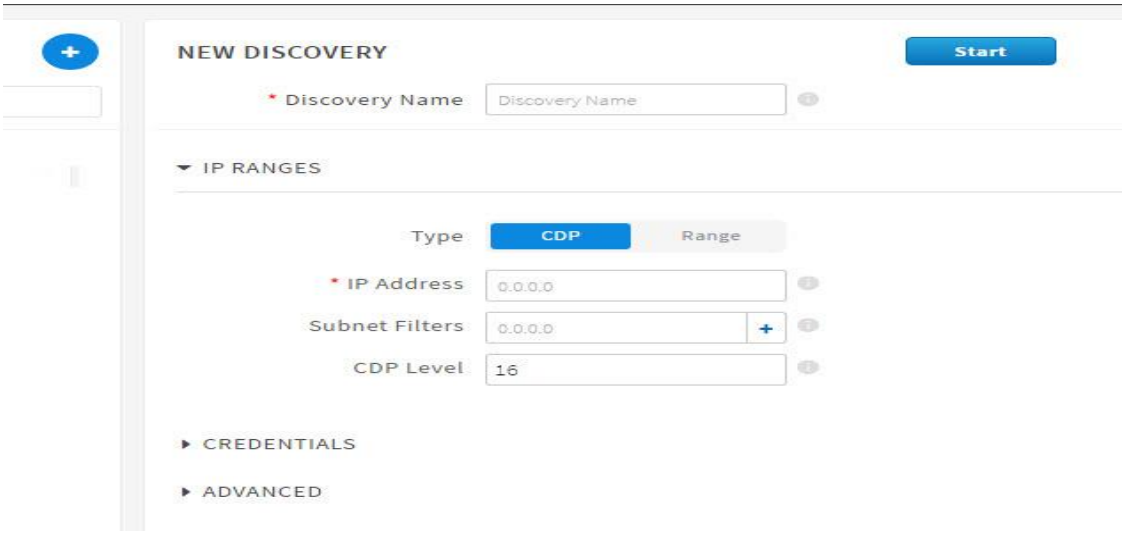

FIGURE III.1. New discovery window

The Scan of devices is done by trying to connect to devices using SSH or SNMPv2/3 (Simple Network Management Protocol v2/3) by default. TELNET is an option as well, but its usage is discouraged since it sends logging information in plain text, hence insecure. The order in which the controller attempts the protocols can be defined and the network administrator can eliminate the use of some protocol for a particular device [31].

Cisco Discovery Protocol (CDP) can be used to scan a specific IP address ranges or scan upwards starting from a CDP-enables device.

Discovery Jobs and Scanning Credentials:

Before starting to scan for network devices, an administrator must first supply scanning credentials. To enable automated discovery of a variety of devices, APIC-EM support different protocols including SSH, TELNET and SNMPv2/3.

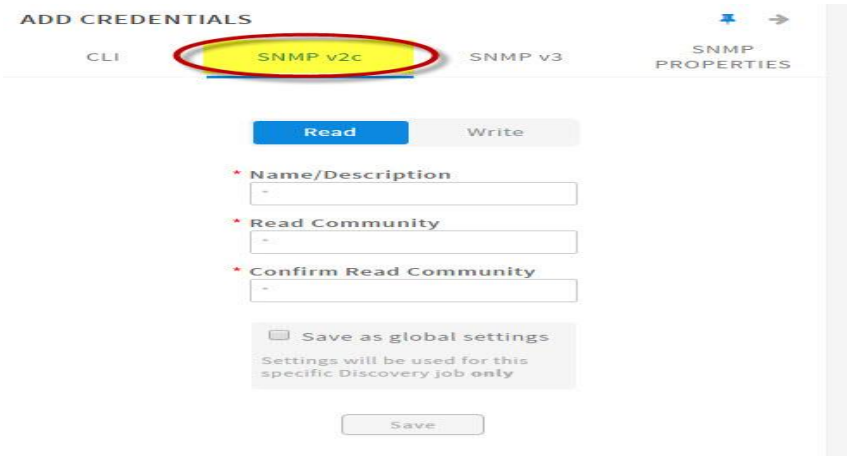

FIGURE III.2. SNMP v2c Credentials

For a successful scan, the devices must support SNMPv2c with the public read-only community string or SNMPv3. If additional information such as device configurations is available, the scanner can be configured with additional credentials known as CLI credentials. Once done the scanner uses the command-line interface (CLI) on Cisco device to retrieve its configuration [32].

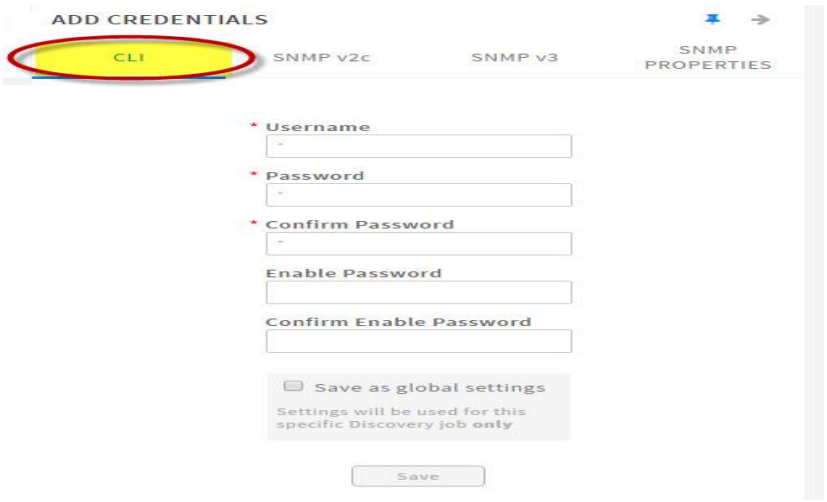

FIGURE III.3. CLI Credentials

The supplied credentials must be the same with the ones configured on network devices (valid credentials) which enables the controller to log into the device. To have access to privileged and sensitive information, some devices require an enable password that must be provided by the scanning credentials.

These scanning credentials can be saved as a named item for use in a particular scan that can be executed from the controller GUI or from the REST API. A named set of

discovery settings is a discovery request or discovery job. Each discovery job specifies the method of the scan (CDP or IP range) and the set of credentials to establish the connection with devices.

In order to support a wide variety of devices, APIC-EM can save multiple sets, one set for each type of credentials (SNMPv2c, SNMPv3, CLI, SSH or TELNET). Some credentials can be made available for all discovery requests or jobs. They are known as global credentials.

Some credentials are not available for all sets. They are only available for a specific request and can only be used by it. They are known as job-specific credentials or request-specific credentials. A discovery job can either use its specific credentials alone or in addition to the global credentials [32].

III.1.3.2 Device and Host Inventories:

Devices and Hosts:

The Device Inventory and Host Inventory windows display information about network elements that the network scanner discovers. The scanner categorizes each network element as a device or as a host.

Device: Each network element that participates in traffic-management over the network and mediates data transmission is considered a device. Switches, Routers and Access Points are examples of devices.

Host: Each network element that has not been identified as part of the traffic-management and not authenticated via 802.1x standard is a host in APIC-EM GUI. A network device may be in the host section until the controller identifies it as a device and sends it back to the device section.

The Device Inventory and Host Inventory tables are generated after the network scan. They contain information about the discovered devices and hosts, respectively.

Device Inventory:

The Device Inventory table can display information about the Device's name, Reachability status, IP address, MAC address, IOS/firmware on device, Platform, Serial Number, Up Time, Configuration, Device Role, Device Family, Device Series, Device

Tag, Policy Tag, Location, Last Updated Time and Last Inventory Collection Status [33].

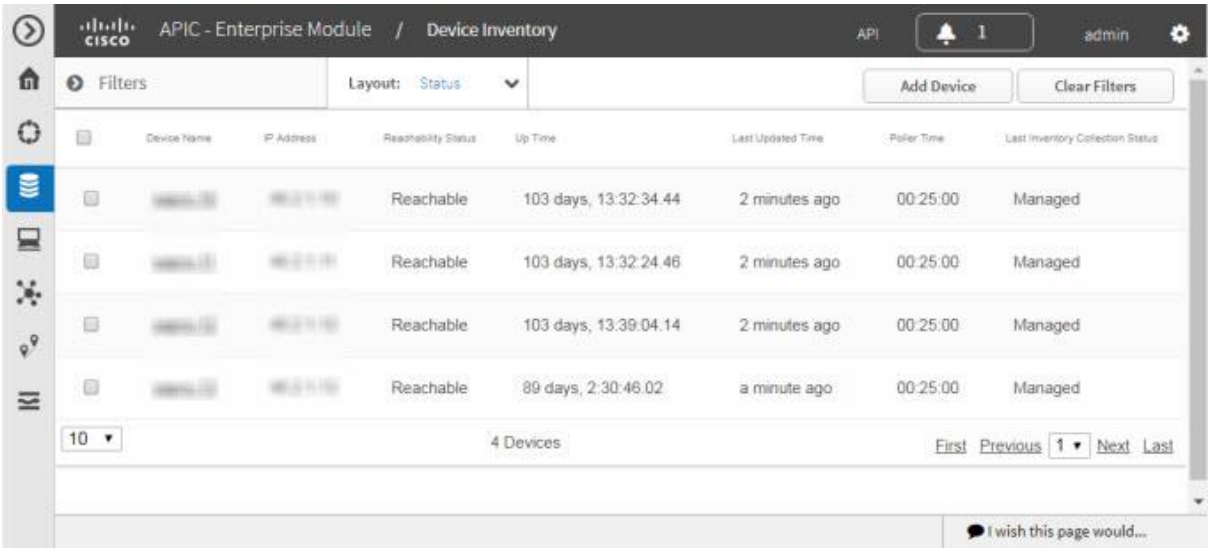

FIGURE III.4. Device Inventory

Host Inventory:

If the discovery scan identifies a network element as a host (or cannot identify it as a device), that element appears in the Host Inventory instead of the Device Inventory.

The Host Inventory table can display information about the Host's Name, MAC Address, IP Address and Type. It also shows the Connected Network Device IP Address and the Connected Interface Name [33].

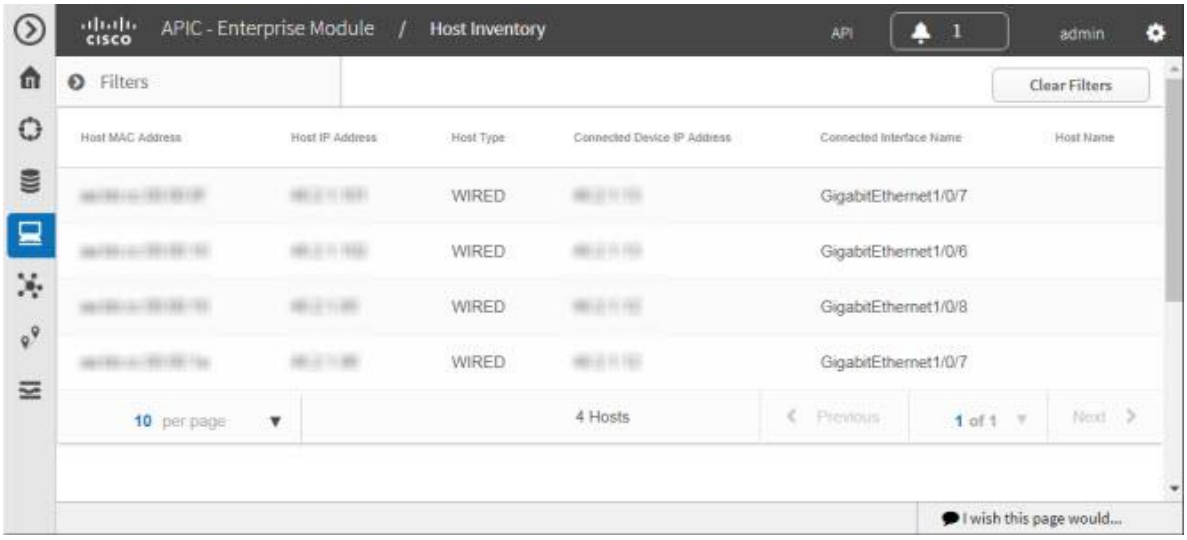

FIGURE III.5. Host Inventory

III.1.3.3 Topology:

Topology displays a graphical view of the network. Filters are available to the user to set the view according to device location, device role, virtual topologies (VLANs, VRFs), and/or OSPF areas, IS-IS, EIGRP, static routes, tags.

Below are the different views of the network topology available for the user to choose from:

- **Enterprise Collapsed:** (default view) in this view, the physical connections between devices are displayed, separated by connection branches.
- **Enterprise Expanded:** this view sorts devices from left to right according to the number of current connections to each. Devices at the leftmost side of the view have 0 connections. Increasing numbers of connections sort towards the right side of the view.
- **Device Type and Role:** Sorts devices according to their type and role in the network, such as access router, distribution switch, core switch and hub, or border router.

There are also other device filtering criteria, namely:

- **Layer 2**: Available VLANs can be viewed from the menu Layer 2. The devices of any VLAN are highlighted when an item is clicked on the menu.
- **Layer 3**: other criteria can be chosen by clicking Layer 3. The menu shows ISIS, OSPF, EIRGP or static-route items; selecting any of the items displays where the protocol is used (routes).
- **VRF**: VRF stands for Virtual Routing and Forwarding. Clicking it shows the routes that use it [34].

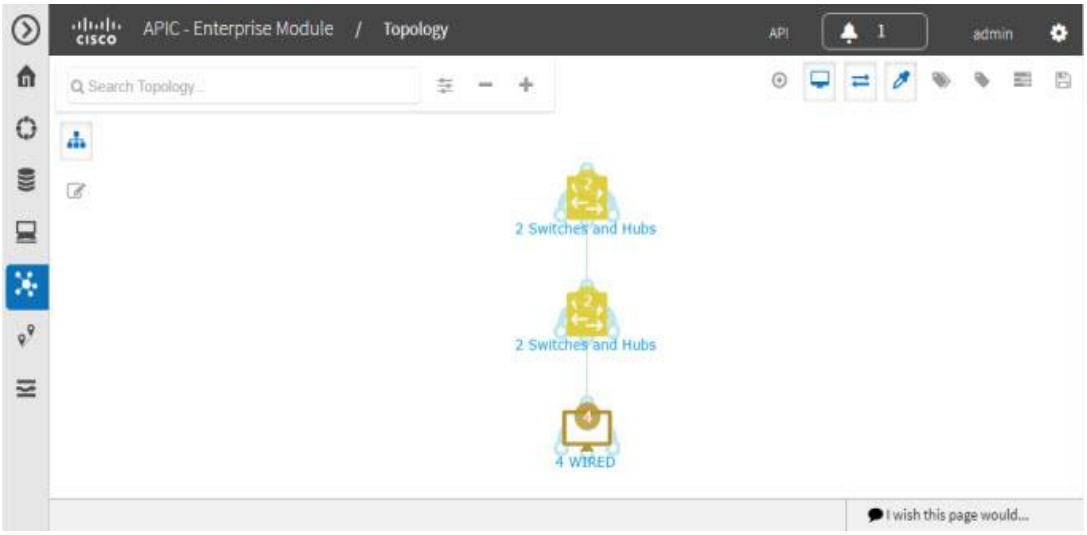

FIGURE III.6. Topology view

III.1.4 APIC-EM applications:

Path trace:

One powerful tool the Cisco APIC-EM brings is Path Trace**.** It shows which path the traffic takes, which is very useful for overseeing and troubleshooting traffic paths that are distributed throughout the network. Path Trace is especially useful the bigger the network gets because paths tend to get entangled, which makes manual debugging hard [35].

EasyQoS:

EasyQoS is an APIC-EM application responsible for deploying Quality of Service policies across the network. It uses a declarative approach in which the QoS is expressed as business intent (WHAT must be accomplished instead of HOW). The APIC-EM then translates that intent into device configuration [36].

Cisco Intelligent WAN (IWAN):

IWAN network profiles are provided by the IWAN application with simple business policies. The applications business preferences are determined from the preferred path for hybrid WAN links by the IWAN application. This in turn saves telecom costs because it uses cheaper WAN links, all while providing a better experience [37].

Cisco Network Plug and Play (PnP):

Cisco Network PnP application is part of the Cisco Network PnP solution which provides security, scalability and a hands-free deployment experience for users on various Cisco devices (routers, switches etc...).

Back Up and Restore:

All the database can be backed up then restored in case such a need arises using the Back Up and Restore application from the graphical user interface of the controller.

Audit Logs:

All the user's and network activities and event records are kept and recorded in the Audit Logs application and can be checked anytime.

III.1.5 APIC-EM example:

In this section, a working example (from a precedent project) [38] of an APIC-EM controlled network will be shown and discussed. The figure below shows the network topology:

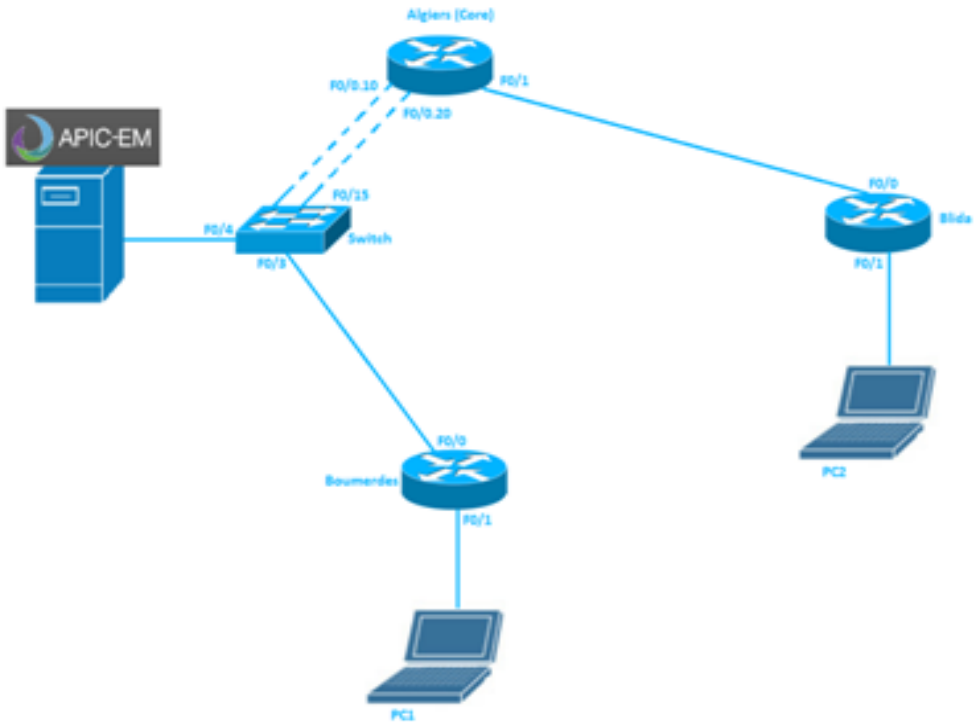

FIGURE III.7. Example Network Topology

III.1.5.1 Devices configuration:

Before beginning with the network management with APIC-EM, initial configuration is given to the network devices (That construct the physical topology shown in the figure above). The address configuration of the devices is based on the table below:

TABLE III.1. Addressing Table (APIC-EM example).

Device	Interface		IP Address	Subnet mask
Router Algiers	F0/0	F0/0.10	192.168.1.1	**255.255.255.0**
		F0/0.20	192.168.2.1	**255.255.255.0**
	F0/1		192.168.3.1	**255.255.255.0**
Switch (in Algiers)	F0/4 (vlan10)		192.168.1.3	**255.255.255.0**
	F0/3 (vlan20)		192.168.2.3	**255.255.255.0**
Router Boumerdes	F0/0		192.168.2.2	**255.255.255.0**
	F0/1		192.168.4.1	**255.255.255.0**
Router Blida	F0/0		192.168.3.2	**255.255.255.0**
	F0/1		192.168.5.1	**255.255.255.0**
PC1 (in Boumerdes)	/		192.168.4.2	**255.255.255.0**
PC2 (in Blida)	/		192.168.5.2	**255.255.255.0**

III.1.5.2 Basic configuration:

As a first step, it is obligatory to establish a basic configuration for each router and switch. This configuration is more or less similar to all network devices. Results are shown in figures below associated with each device of the network.

The basic configuration steps are as follow:

1. Configuration of the names of the routers.

2. Remote Access Configuration (SSH): to access remote devices.

3. Configuration of the console line.

4. Setting the password for the run mode.

5. Encryption service activation.

6. Addressing configuration.

7. Static route configuration.

8. SNMP protocol configuration.

Router ALGIERS:

In order to form a trunk link with the switch from ALGIERS router, it is necessary to create one sub-interface of interface FastEthernet 0/0 as shown in FIGURE III.8, for every VLAN configured on the switch. The results of IP address, status and protocol of interfaces by **show ip int brief** command are shown in the FIGURE III.9:

```
Algiers(config)#int fa0/0.10
Algiers(config-subif)#encaplsulation dot1Q 10
                             ^
% Invalid input detected at '^' marker.

Algiers(config-subif)#encapsulation dot1Q 10
Algiers(config-subif)#ip address 192.168.1.1 255.255.255.0
Algiers(config-subif)#no shut
Algiers(config-subif)#int fa0/0.20
Algiers(config-subif)#encapsulation dot1Q 20
Algiers(config-subif)#ip address 192.168.2.1 255.255.255.0
Algiers(config-subif)#no shut
Algiers(config-subif)#end
```

FIGURE III.8. Sub-interfaces configuration

```
Algiers#show ip int brief
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        unassigned      YES unset  up                    down
FastEthernet0/0.10     192.168.1.1     YES manual up                    down
FastEthernet0/0.20     192.168.2.1     YES manual up                    down
FastEthernet0/1        192.168.3.1     YES manual up                    down
Serial0/0/0            unassigned      YES unset  administratively down down
```

FIGURE III.9. Status of IPv4 interfaces

Results of routing table by **show ip route** command are shown in FIGURE III.10.

```
Algiers#sh ip rout
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

S    192.168.4.0/24 [1/0] via 192.168.2.2
S    192.168.5.0/24 [1/0] via 192.168.3.2
C    192.168.1.0/24 is directly connected, FastEthernet0/0.10
C    192.168.2.0/24 is directly connected, FastEthernet0/0.20
C    192.168.3.0/24 is directly connected, FastEthernet0/1
```

FIGURE III.10. Routing table of Algiers

SSH configuration (Same for Boumerdes and Blida routers) is shown in FIGURE III.11

```
Algiers(config)#ip domain-name icosnet.com
Algiers(config)#crypto key generate rsa
The name for the keys will be: Algiers.icosnet.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]

Algiers(config)#
*Mar 19 08:36:21.179: %SSH-5-ENABLED: SSH 1.99 has been enabled
Algiers(config)#username SDN password 07032019
```

FIGURE III.11. SSH configuration commands

SNMP configuration (Same for Boumerdes and Blida routers) is shown below:

```
Algiers>enable
Password:
Algiers#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Algiers(config)#snmp-server community public RO
Algiers(config)#end
Algiers#wr
Building configuration...
[OK]
Algiers#
```

FIGURE III.12. SNMP configuration commands

Router BOUMERDES :

Results of routing table by **show ip route** command are shown in FIGURE III.13.

```
Boumerdes#sh ip rout
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.4.0/24 is directly connected, FastEthernet0/1
S    192.168.5.0/24 [1/0] via 192.168.2.1
S    192.168.1.0/24 [1/0] via 192.168.2.1
C    192.168.2.0/24 is directly connected, FastEthernet0/0
S    192.168.3.0/24 [1/0] via 192.168.2.1
```

FIGURE III.13. Routing table of Boumerdes router

Router BLIDA:

Results of routing table by **show ip route** command are shown in FIGURE III.14.

```
Blida#sh ip rout
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

S    192.168.4.0/24 [1/0] via 192.168.3.1
C    192.168.5.0/24 is directly connected, FastEthernet0/1
S    192.168.1.0/24 [1/0] via 192.168.3.1
S    192.168.2.0/24 [1/0] via 192.168.3.1
C    192.168.3.0/24 is directly connected, FastEthernet0/0
```

FIGURE III.14. Routing table of Blida router

SWITCH (Positioned in Algiers):

Results of **show ip int brief command** to show address, status and protocol of VLAN interfaces are described in FIGURE III.15.

```
Switch#sh ip int brief
Interface              IP-Address      OK? Method Status                Protocol
Vlan1                  unassigned      YES manual up                    up
Vlan10                 192.168.1.3     YES manual up                    up
Vlan20                 192.168.2.3     YES manual up                    up
```

FIGURE III.15. Switch Interfaces status

Mode configuration to each switch port, Fa0/15 mode trunk, Fa0/3 mode access, also for Fa0/4 mode access is shown in FIGURE III.16.

```
Switch(config-if)# switchport mode trunk
Switch(config-if)# switchport trunk native vlan 1
Switch(config-if)# switchport trunk allowed vlan 10,20
Switch(config-if)#exit
Switch(config)#int fa0/3
Switch(config-if)#switchport mode access
Switch(config-if)#switchport access vlan 20
Switch(config-if)#end
Switch#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#int fa0/4
Switch(config-if)#switchport mode access
Switch(config-if)#switchport access vlan 10
Switch(config-if)#end
```

FIGURE III.16. Switch interfaces mode configuration

Results of **show int trunk** command are shown in FIGURE III.17.

```
Switch#sh int trunk

Port        Mode             Encapsulation  Status        Native vlan
Fa0/15      on               802.1q         trunking      1

Port        Vlans allowed on trunk
Fa0/15      1-4094

Port        Vlans allowed and active in management domain
Fa0/15      1,10,20

Port        Vlans in spanning tree forwarding state and not pruned
Fa0/15      none
```

FIGURE III.17. Switch trunk interface details

Results of **show vlan** command showing each interface VLAN name, status and ports are described in FIGURE III.18.

```
Switch#sh vlan

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/5, Fa0/6
                                                Fa0/8, Fa0/9, Fa0/10, Fa0/11
                                                Fa0/12, Fa0/13, Fa0/14, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Fa0/23, Fa0/24
                                                Gi0/1, Gi0/2
10   server                           active    Fa0/4, Fa0/7
20   bmds                             active    Fa0/3
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------
```

FIGURE III.18. Switch VLAN interface details

III.1.5.3 Using APIC-EM:

After devices initial configurations, logging in into APIC-EM is as shown in FIGURE III.19.

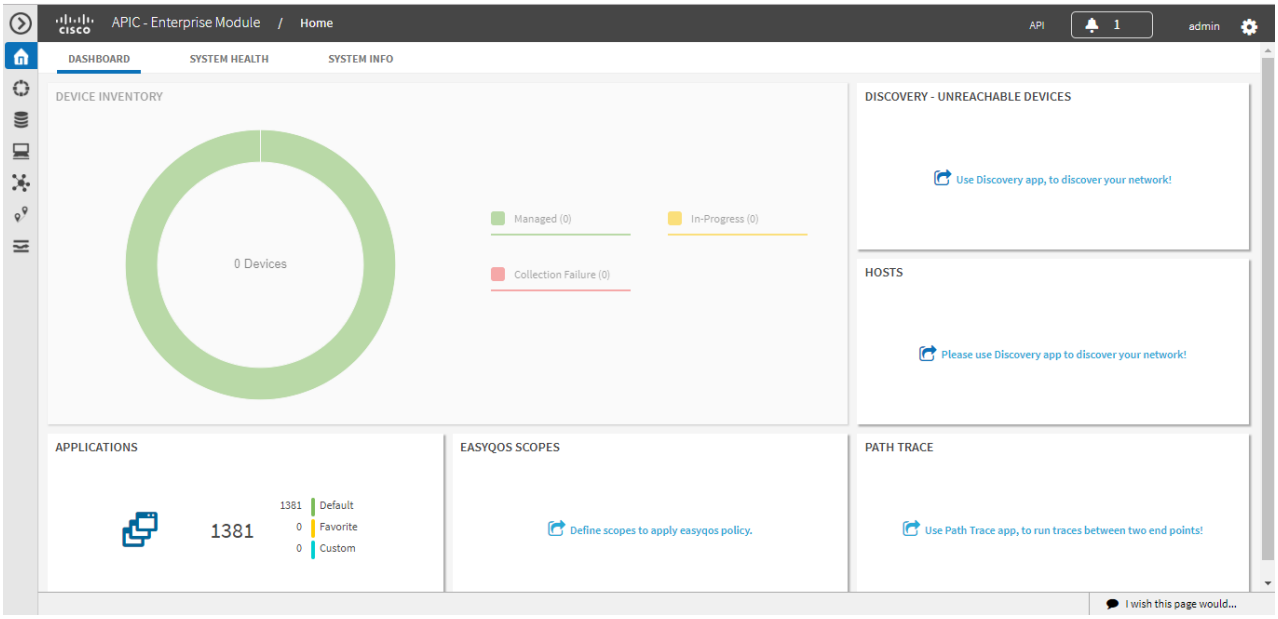

FIGURE III.19. APIC-EM home page.

In order to discover these devices on the network, CDP, SSH and SNMP are used. CLI, SSH and SNMP are required so that APIC-EM can access devices. So, they are configured by clicking on the "gear" icon in the top right corner and selecting "settings".

In the discovery credentials menu, "CLI Credentials "is selected.

Enter the username and password used for SSH and the enable password as shown in FIGURE III.20, then click on the Add button. After that, choose the read only, the write and the SNMP community as in FIGURE III.21. Finally, click on the save button.

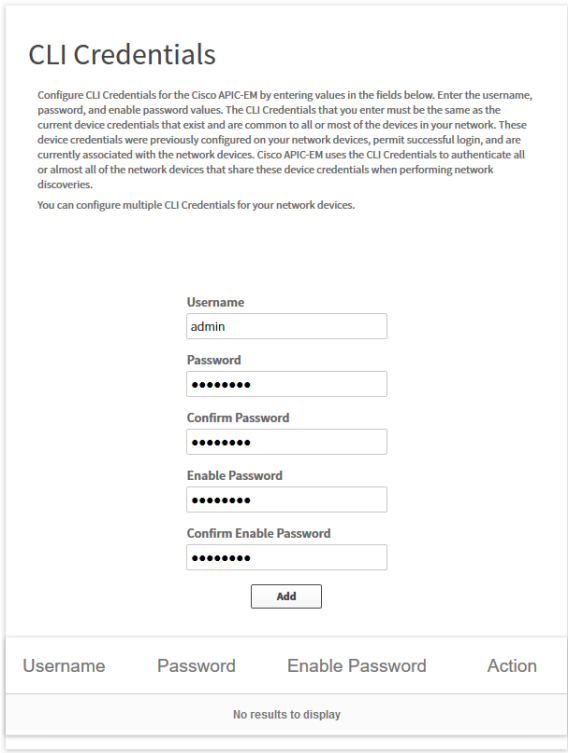

FIGURE III.20. APIC-EM CLI Credentials window.

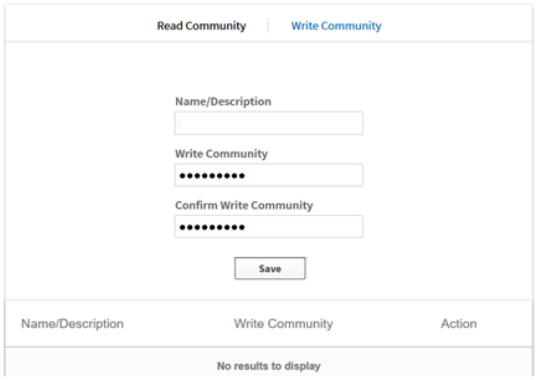

FIGURE III.21. APIC-EM SNMP configuration window.

Discovery:

In order to discover the network, the discovery icon on the left side is selected. First, entering the name of the discovery and the seed IP address from the network as shown in FIGURE III.22, leaving the subnet filter empty so that the controller does not exclude any subnet, with the CDP level with its default value.

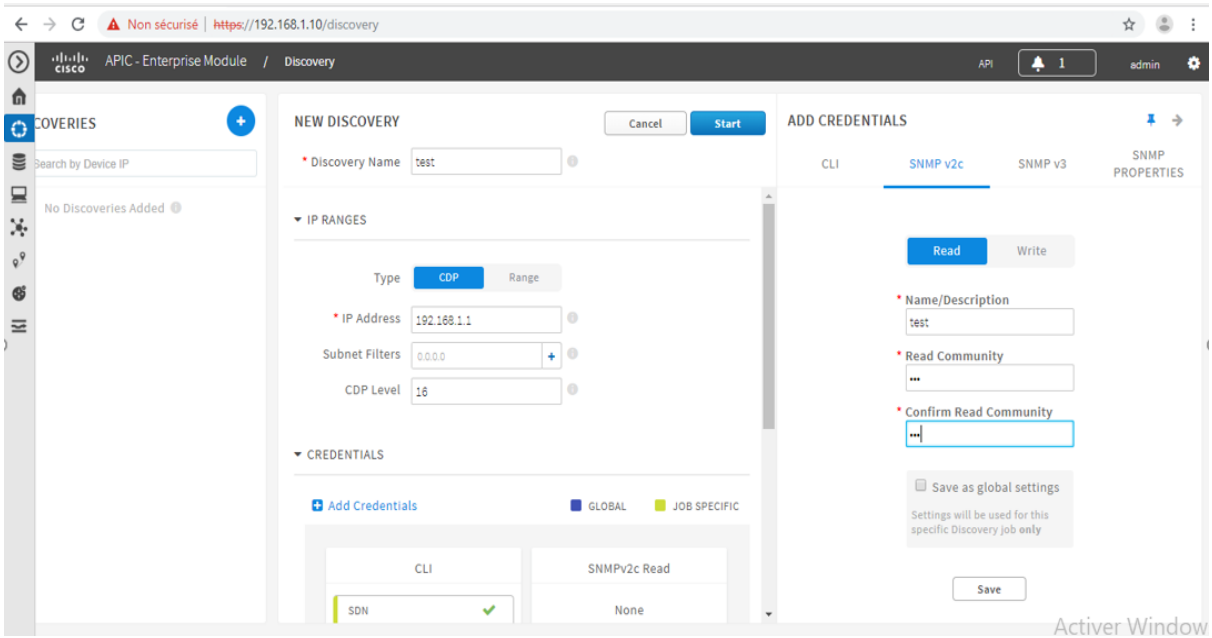

FIGURE III.22. APIC-EM Discovery page

Click on SNMPv2c in the Add Credentials panel.

Click “Read” in the SNMPv2c panel. Then, type “public” in the Read Community and in “confirm Read Community” field and click “save”.

Then, click Start. The user interface displays an In-progress message to indicate that scanning is taking place. Results of the discovery are shown in FIGURE III.23.

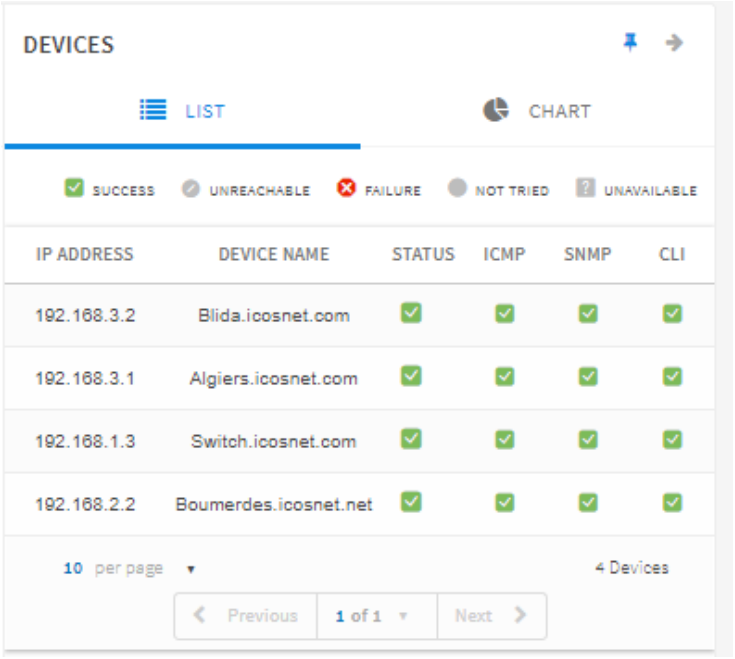

FIGURE III.23. APIC-EM Discovery process results

Inventory:

After performing the discovery scan, Device Inventory icon is clicked to view the discovered devices information as FIGURE III.24 describes.

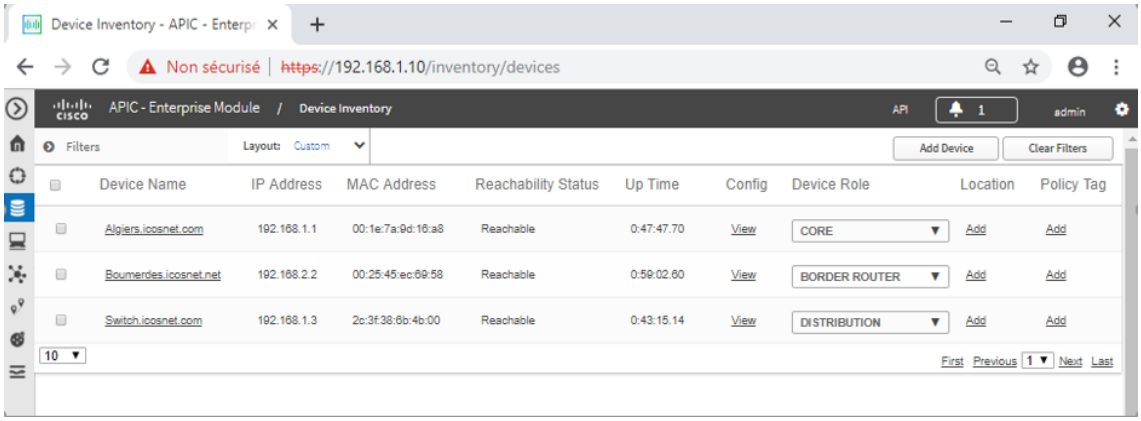

FIGURE III.24. APIC-EM inventory page

Topology:

By clicking on the topology icon on the left-hand side of the menu, the network topology is shown as in FIGURE III.25.

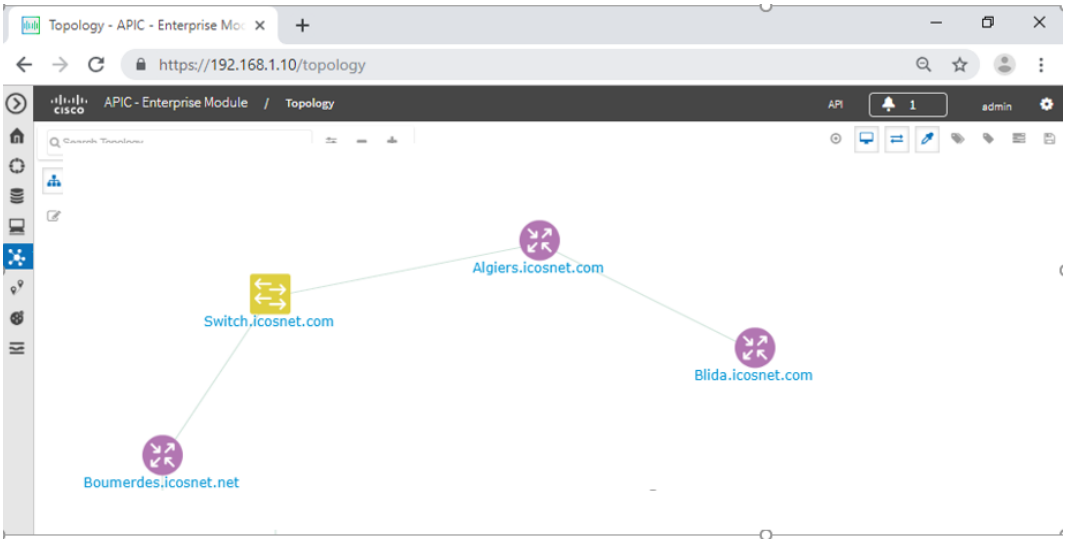

FIGURE III.25. APIC-EM Topology page

Path Trace:

The Path Trace application of APIC-EM is used to determine the path that traffic takes from one network node to another:

Open the Path Trace navigation-panel item and follow the steps described in FIGURE III.26. Select the source Algiers router with 192.168.1.1 port address to destination Boumerdes router port address 192.168.4.1.

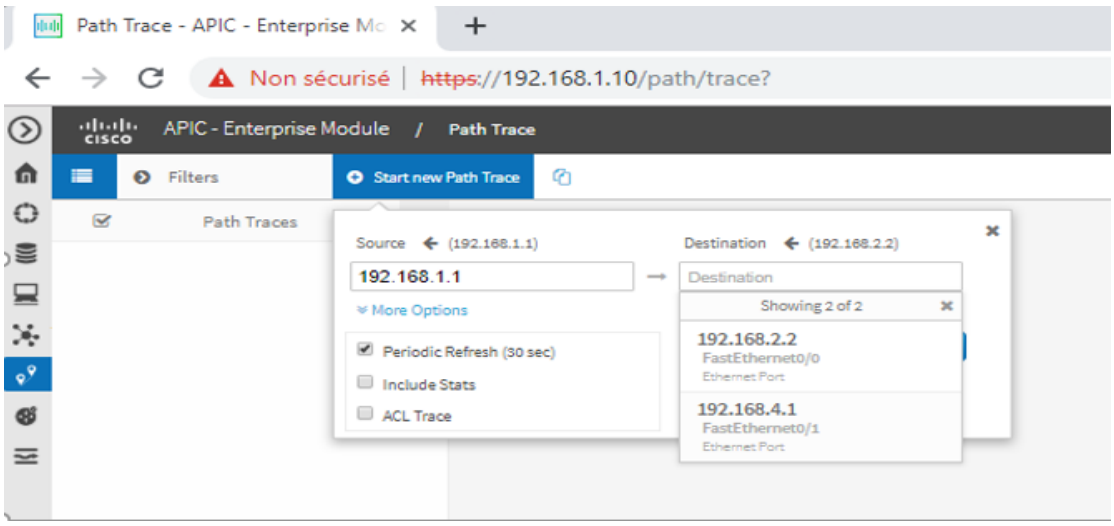

FIGURE III.26. APIC-EM Path Trace navigation panel

After some time passes, the traffic route results, from the source to the destination are shown in FIGURE III.27.

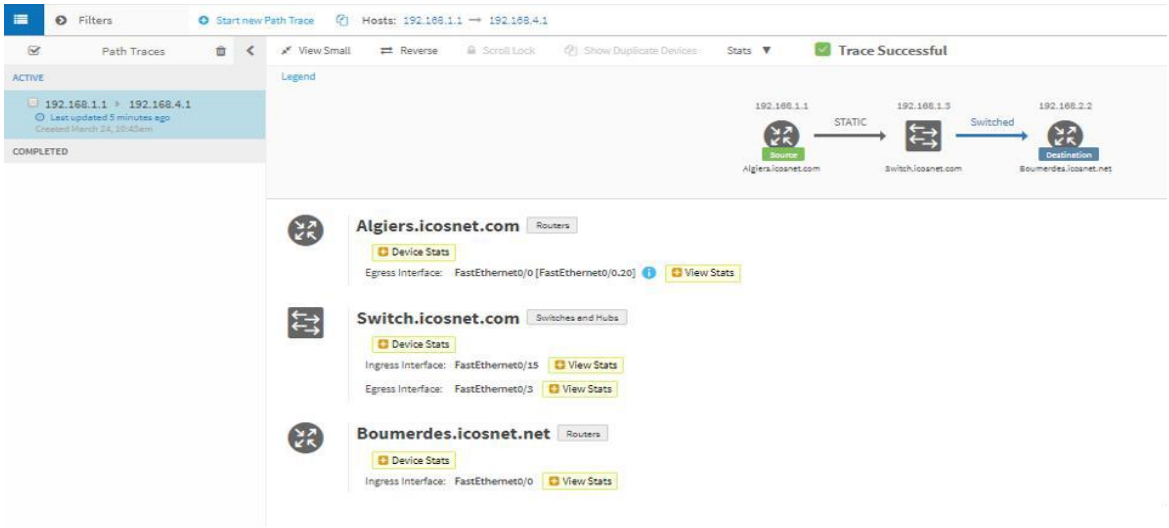

FIGURE III.27. Path Trace results page

Conclusion:

The Cisco Application Policy Infrastructure Controller is Cisco's answer in terms of SDN controllers. It leaves nothing to be desired: versatile applications in the form of Network Discovery, Devices/Hosts Inventories, Topology view and more. It also provides powerful tools like Path Trace, which trivializes troubleshooting. EasyQoS, Network PnP, Backup and Restore, etc...

Seemingly, the only downside of APIC-EM is the fact that it is only compatible with Cisco devices.

In the next chapter, Network automation using Python libraries (NETMIKO, NAPALM) will be presented and discussed, then a comparison between APIC-EM and Python automation will be held.

CHAPTER 4
NETWORK PROGRAMMABILITY USING PYTHON

Introduction:

In the previous chapter, the Cisco APIC-EM was presented and discussed. But that does not mean that there are no alternatives to it in the industry. Network Automation using Python libraries (NETMIKO, NAPALM) as well as the automation with ansible tool will be the topic in this chapter, then at the end, a comparison between the two approaches will be held to assess their overall performance.

IV.1 Network Automation using Python:

The networking industry is changing drastically, and the need for businesses to be more agile and more flexible in order to compete are drawing the days of configuring networks only with the command line interface (CLI) to a close.

Network programmability and automation has the main goal of simplifying the tasks involved in configuring, managing and operating network equipment, network topologies, network services and connectivity [39].

Python is widely used to perform network automation. With its wide set of libraries (such as **Netmiko** and **Paramiko**), there are endless possibilities for network device interactions for different vendors. Both Netmiko and Paramiko are using SSH connection to control the devices.

IV.1.1 Why Python for network programmability:

Python is a widely used programming language by programmers is the recent years. Networking companies prefer python because of its features and its easy codes. Python is used for controlling, monitoring and troubleshooting of network elements. Programmers prefer it over other languages because it is high level, simple, open source and the has extensive libraries that can be used in network automation, reducing time for equipment configuration and easing maintenance [40].

IV.1.2 Benefits of Python for Network automation:

Python is widely used in software development and networking companies because of the following features:

- **Extensive Support Libraries:** it provides different sets of libraries that covers a lot of domains to facilitate configuring and maintaining the network.

- **Improved Programmer's Productivity:** the extensive set of libraries and the clean structure simplifies the work for programmers.
- **Efficiency:** the enhanced control capabilities and testing frameworks increases speed for most applications and helps creating reliable and multi-protocol network applications [40].

IV.1.3 Definitions:

NAPALM: Napalm is a Python library that you can use to automate and interact with network devices and OSs using a unified API. As this library provides an abstraction layer, it makes it easier to configure multiple vendor devices [41].

PARAMIKO: Paramiko is a Python implementation of the SSHv2 protocol, providing both client and server functionality. It is a pure Python interface around SSH networking concepts, and it leverages a Python C extension for low level cryptography.

NETMIKO: Netmiko is an open source library designed to simplify SSH management across a wide range of network devices from various vendors including Cisco, Arista, and Juniper Network [41].

LANSCAN: lanscan is module written using Python 3.5 on Linux. It provides a network discovery tool by enabling the administrator to scan the network in order to find all the present devices. lanscan can also provide information about the scanned devices and interfaces on the network. [44]

In order to start automating the network with python, the controller machine should have python installed in it. And it should be able to reach all devices including router interfaces, switches' management interfaces and the hosts. One other requirement is that logging credentials should be present on all devices (with Telnet, SSH) and the enable secret password.

In this part, the work will be focused on installing the controller in a Linux VM, then creating a network on GNS3. The network is controlled using the controller.

The control of the network is going to be performed using python scripts, with libraries (NETMIKO and NAPALM). These scripts enable the network administrator to send configuration commands from the controller to any device on the network, as well as reading the state and retrieving information about devices and storing them.

IV.1.4 Installing and Configuring VMware Workstation:

The first step of the project is to install VMware workstation software on a Windows 10 machine with i5 CPU and 8GB of RAM, to allow the creation of virtual machines and in this case **the controller**.

To begin, the installer file (for Windows) is downloaded from the official website.

First, run the installation file as an administrator.

FIGURE IV.1. VMware Workstation Pro Install menu

After that, continue through the steps until the end.

IV.1.5 Installing and Configuring the controller:

Now that the VMware Workstation is installed, the next step is installing the Linux Virtual machine which will be used as the controller.

On VMware Workstation, Go to **File** > **New Virtual Machine**.

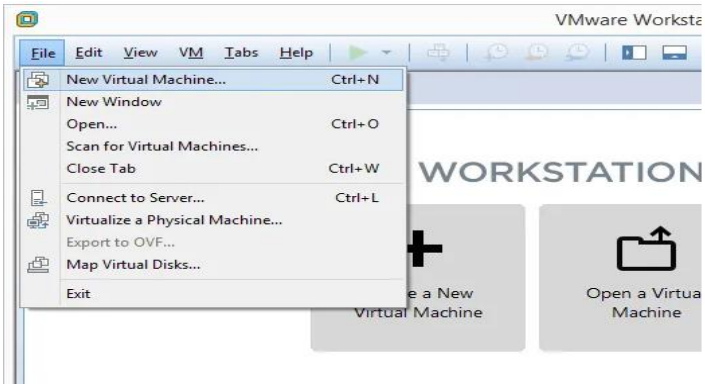

FIGURE IV.2. New Virtual Machine creation

Pick how to install the operating system:

Choose the **Installer disc image file (iso)** that you have.

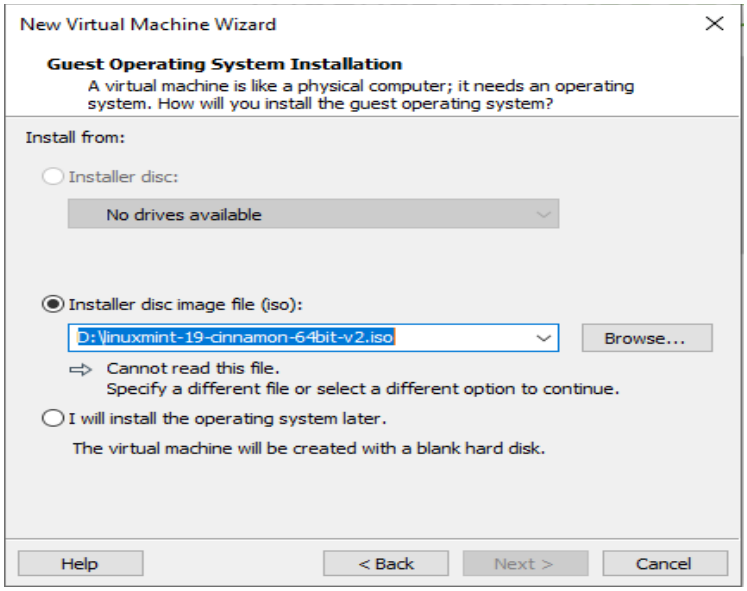

FIGURE IV.3. New Virtual Machine Operating System Installation

Define the storage capacity for the virtual machine, and then press **Next**. Select **Customize Hardware** and make any necessary changes. You can change details about the memory, processors, disc drive, network adapter, USB controller, sound card, printer, and display.

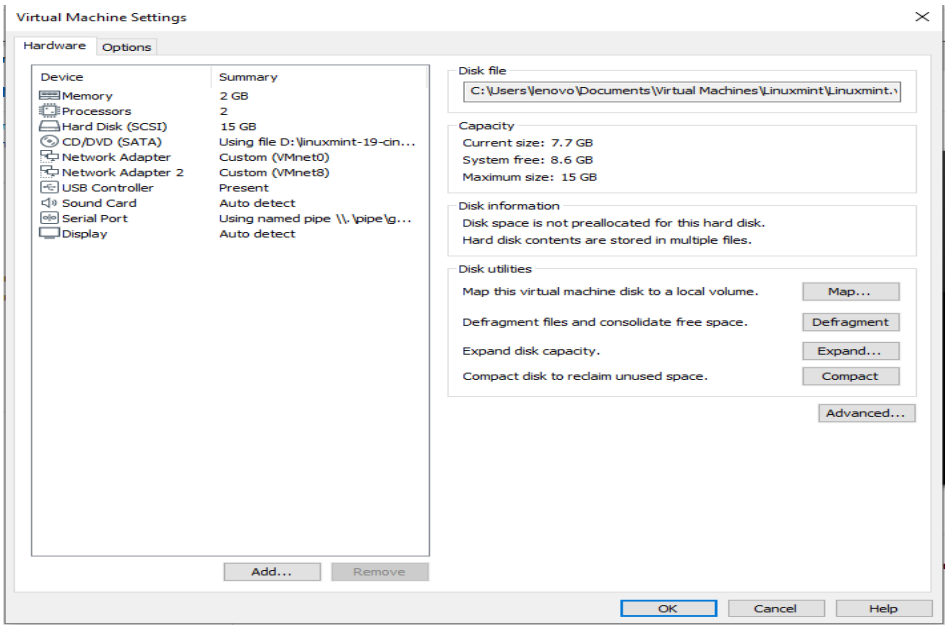

FIGURE IV.4. New Virtual Machine Hardware settings

After the installation of the Linux Machine which will act as a controller. It is now mandatory to install the required tools that will be used to control the Network functionalities.

IV.1.6 Installation of the required Tools:

For the controller to be able to control the network, some tools must be installed in the machine.

To install python, first the references must be updated using “sudo apt-get update”. After that, install python using “sudo apt-get install python3.8”

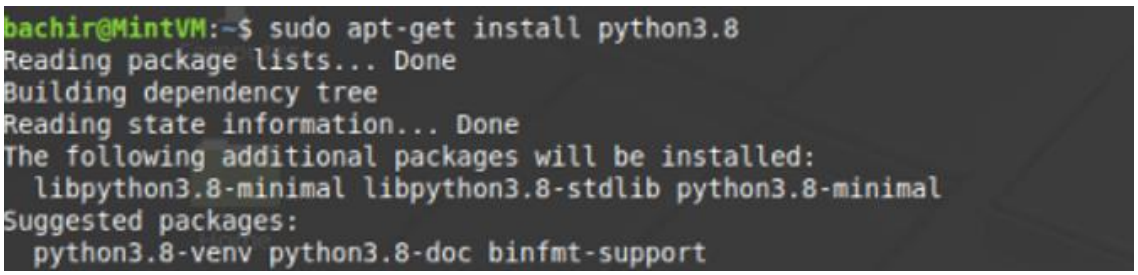

FIGURE IV.5. Python 3.8 Installation

Two specific python tools (libraries) are needed for controlling the network, NAPALM and NETMIKO.

First, install NAPALM using “pip install napalm “

FIGURE IV.6. NAPALM installation

Following that, install NETMIKO using “sudo apt-get install -y python3-netmiko “

FIGURE IV.7. NETMIKO installation

Finally, install lanscan using “pip3 install lanscan “

Now that the needed libraries for the controller are installed, it is ready for use.

IV.1.7 Network Architecture:

The complete network system is simulated in the GNS3 simulation software including all routers, switches and servers. GNS3 allows the simulation of real network scenarios since it uses the real iOS images in the devices.

The whole GNS3 network is in the host pc, whereas the ControllerPC is a guest in VMware. The controller is connected to the network using VMware network adapter VMnet8.

The network is composed of the following devices:

- Two Routers.
- Four distribution layer switches.
- Two access layer switches.
- Four Servers.
- Home Client pc which is a simulated pc in GNS3.

The logical topology is presented in the diagram below:

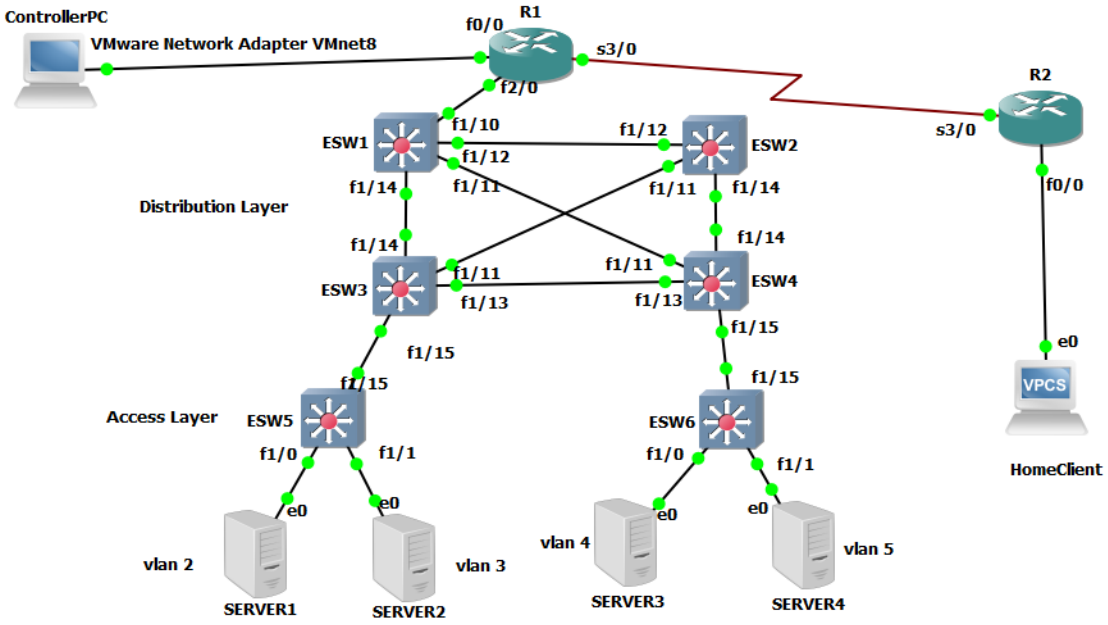

FIGURE IV.8. Network Topology

IV.1.7.1 Device Configurations:

Before beginning with the network management with The Controller, the initial configuration is given to the network devices.

The device addresses are configured according to the following table:

TABLE IV.1. Addressing Table (Python automation)

The device	The interface	The IP-address	The subnet mask
R1	F0/0	192.168.244.130	255.255.255.0
R1	F2/0.1	192.168.1.1	255.255.255.0
R1	F2/0.2	192.168.2.1	255.255.255.0
R1	F2/0.3	192.168.3.1	255.255.255.0
R1	F2/0.4	192.168.4.1	255.255.255.0
R1	F2/0.5	192.168.5.1	255.255.255.0
R1	S3/0	192.168.10.1	255.255.255.252
R2	F0/0	192.168.30.1	255.255.255.0
R2	S3/0	192.168.10.2	255.255.255.252
ESW1	Vlan1	192.168.1.2	255.255.255.0
ESW2	Vlan1	192.168.1.3	255.255.255.0
ESW3	Vlan1	192.168.1.4	255.255.255.0
ESW4	Vlan1	192.168.1.5	255.255.255.0
ESW5	Vlan1	192.168.1.6	255.255.255.0
ESW6	Vlan1	192.168.1.7	255.255.255.0
The controller	VMnet8	192.168.244.129	255.255.255.0
SERVER1	NIC	192.168.2.2	255.255.255.0
SERVER2	NIC	192.168.3.2	255.255.255.0
SERVER3	NIC	192.168.4.2	255.255.255.0
SERVER4	NIC	192.168.5.2	255.255.255.0
Home client	NIC	192.168.30.2	255.255.255.0

IV.1.7.2 Basic Configurations:

As a first step, it is obligatory to establish a basic configuration for each router and switch. This configuration is more or less similar for all network devices.

The basic configuration steps are listed below:

- Configuring hostnames for routers and switches.
- Configuring Remote Access Telnet and secure shell (SSH).
- Configuring Console Line.
- Setting passwords and usernames.

- Activate password Encryption.
- Configuring the interface addresses.
- Configuring the management interfaces.
- Configure Static Routing.

- **Configuration examples are shown below:**

The interface configuration and the static routing:

```
IGEE#sh ip int bri
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            192.168.244.130 YES NVRAM  up                    up
ATM1/0                     unassigned      YES NVRAM  administratively down down
FastEthernet2/0            unassigned      YES NVRAM  up                    up
FastEthernet2/0.1          192.168.1.1     YES NVRAM  up                    up
FastEthernet2/0.2          192.168.2.1     YES NVRAM  up                    up
FastEthernet2/0.3          192.168.3.1     YES NVRAM  up                    up
FastEthernet2/0.4          192.168.4.1     YES NVRAM  up                    up
FastEthernet2/0.5          192.168.5.1     YES NVRAM  up                    up
FastEthernet2/1            unassigned      YES NVRAM  administratively down down
Serial3/0                  192.168.10.1    YES NVRAM  up                    up
Serial3/1                  unassigned      YES NVRAM  administratively down down
Serial3/2                  unassigned      YES NVRAM  administratively down down
Serial3/3                  unassigned      YES NVRAM  administratively down down
FastEthernet4/0            unassigned      YES NVRAM  administratively down down
IGEE#
IGEE#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#ip route 192.168.30.0 255.255.255.0 s3/0
```

FIGURE IV.9. Interface configuration and static routing

Remote access configuration is essential for the controller. A username and a password must also be configured for the sessions. In this case the username is set to “bachir” and the password to “12345pass”:

```
ESW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW1(config)#hostname SW1
SW1(config)#ip domain-name igee.com
SW1(config)#cry
SW1(config)#crypto key generate rsa
The name for the keys will be: SW1.igee.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]

SW1(config)#
*Mar  1 00:31:08.747: %SSH-5-ENABLED: SSH 1.99 has been enabled
SW1(config)#line vty 0 4
SW1(config-line)#transport input all
SW1(config-line)#login local
SW1(config-line)#exit
SW1(config)#line console 0
SW1(config-line)#login local
SW1(config-line)#exit
SW1(config)#username bachir password 12345pass
SW1(config)#enable secret 12345pass
SW1(config)#service pass
SW1(config)#service password-encryption
SW1(config)#
```

FIGURE IV.10. SSH configuration

To be able to control the switches, the management interfaces are set up as follows:

```
ESW7#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW7(config)#int vlan 1
ESW7(config-if)#ip address 192.168.1.2 255.255.255.0
ESW7(config-if)#no shut
ESW7(config-if)#
*Mar  1 00:01:00.023: %LINK-3-UPDOWN: Interface Vlan1, changed state to up
ESW7(config-if)#exit
ESW7(config)#ip default-ga
ESW7(config)#ip default-gateway 192.168.1.1
```

FIGURE IV.11. Management interface configuration

Verify that the controller is able to ping all the switches' management interfaces:

```
bachir@MintVM:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=9.70 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=6.61 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=3.47 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 3.472/6.595/9.700/2.543 ms
bachir@MintVM:~$ ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=2 ttl=254 time=20.8 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=254 time=36.9 ms
^C
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2006ms
rtt min/avg/max/mdev = 20.833/28.894/36.955/8.061 ms
bachir@MintVM:~$ ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=2 ttl=254 time=32.8 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=254 time=30.1 ms
64 bytes from 192.168.1.3: icmp_seq=4 ttl=254 time=32.2 ms
c64 bytes from 192.168.1.3: icmp_seq=5 ttl=254 time=33.9 ms
64 bytes from 192.168.1.3: icmp_seq=6 ttl=254 time=22.8 ms
^C
--- 192.168.1.3 ping statistics ---
6 packets transmitted, 5 received, 16% packet loss, time 5008ms
rtt min/avg/max/mdev = 22.896/30.430/33.989/3.974 ms
bachir@MintVM:~$ ping 192.168.1.4
PING 192.168.1.4 (192.168.1.4) 56(84) bytes of data.
64 bytes from 192.168.1.4: icmp_seq=2 ttl=254 time=34.6 ms
64 bytes from 192.168.1.4: icmp_seq=3 ttl=254 time=23.7 ms
64 bytes from 192.168.1.4: icmp_seq=4 ttl=254 time=30.9 ms
64 bytes from 192.168.1.4: icmp_seq=5 ttl=254 time=22.7 ms
^C
--- 192.168.1.4 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4010ms
rtt min/avg/max/mdev = 22.753/28.012/34.600/4.953 ms
```

FIGURE IV.12. Switch management interfaces connectivity test (pings)

Since the basic configuration of devices is done, it means that the network is ready to be configured and managed from the controller.

IV.1.7.3 Network Discovery (using lanscan):

Network discovery is achieved using Lanscan, a Python 3 module. It scans and lists all devices along with their open ports and other information.

Lanscan can provide interfaces information such as Interface, Driver and hardware using: "sudo lanscan interfaces".

It can also display the available network details, namely: network address in CIDR notation and the associated interface via: "sudo lanscan networks".

Additionally, a network scan can be initiated from the terminal (admin privileges). It shows IP addresses, MAC addresses, device name, vendor, etc. Using: "sudo lanscan scan".[44]

Scanning local network:

According to the network configuration (IP address and network mask), lanscan sweeps the network permitting:

Device MAC addresses discovery

Device hostname setup

Scanning external IP ranges:

In case of a scan of IP addresses that are not within the local network, lanscan uses ICMP (Layer 3) packets instead of ARP (Layer 2) packets because of the need to cross over a router to reach the destination.[45]

The external scan has a range that can comprise private or public IP addresses.

Scanning external networks has some limitations, mainly:

MAC addresses of devices cannot be seen. (no ARP)

Custom hostnames can't be used.

IV.1.8 Device Configurations and management by the controller:

As the initial configurations are done, it is time now to configure the devices according to the requirement.

In our network, each server belongs to a different VLAN; In addition to the VLAN of the management interfaces, which means a total of five VLANS.

The table below presents the VLAN information:

TABLE IV.2. VLAN Table

VLAN	Address/mask	Interfaces and host
VLAN 1 (default)	192.168.1.0/24	Management interfaces
VLAN 2	192.168.2.0/24	Server 1
VLAN 3	192.168.3.0/24	Server 2
VLAN 4	192.168.4.0/24	Server 3
VLAN 5	192.168.5.0/24	Server 4

To allow the access to the servers from outside, or even the servers to communicate with each other, all the links between the switches should be configured as trunk-links allowing all the VLANS to pass. Additionally, a trunk link must be established between the SW1 switch and the Router. It is thus necessary to create sub-interfaces with dot1q encapsulation on the router.

First, the four VLANS are configured for all the switches from the controller using the following python script.

```
#!/usr/bin/env python
import getpass
import sys
import telnetlib
import time

user = raw_input("Enter your telnet username: ")
password = getpass.getpass()


for n in range (2,8):
        HOST = "192.168.1."+str(n)
        print ("-------------Connecting to host " + HOST + "------------\n")

        tn = telnetlib.Telnet(HOST)

        tn.read_until("Username: ")
        tn.write(user + "\n")
        if password:
                tn.read_until("Password: ")
                tn.write(password + "\n")

        print ("---------------CONNECTED-----------------\n")

        tn.write("enable\n")
        tn.write(password +"\n")
        tn.write("vlan database\n")
        for i in range (2,6):
                tn.write("vlan "+str(i))
        tn.write("exit\n")
        time.sleep(3)
        print ("---------------DONE CONFIGURING HOST "+ HOST + "-------------\n\n\n")
```

FIGURE IV.13. VLANS configuration from the controller using a python script

This script asks for the username and password then connects to all the switches one by one according to the IP address using Telnet function of **telnetlib** library.

The script then sends configuration commands by the function write () to create four VLANS according to the loop.

As we can see on the switch, we received a notification that shows a connection from device “192.168.244.129” which is the IP of the controller on the vty remote line.

```
SW1#
*Mar  1 00:17:04.975: %SYS-5-CONFIG_I: Configured from console by bachir on vty0 (192.168.244.129)
SW1#
SW1#
```

FIGURE IV.14. Controller access to the Switch

The next step is to configure trunk-links on the distribution layer switches to allow the connection to different VLANS to pass. The links that must be configured as trunks are listed below:

- Ports f1/10, f1/11, f1/12 and f 1/14 of switch SW1.
- Ports f1/11, f1/12 and f 1/14 of switch SW2.
- Ports f1/11, f1/13, f1/14 and f 1/15 of switch SW3.
- Ports f1/11, f1/13, f1/14 and f 1/15 of switch SW4.

The configuration commands are written in text files, one for each switch, they are presented below:

```
GNU nano 2.9.3                  trunk sw1.txt

int range f1/10 - 11
siwtchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
int f 1/12
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
int f1/14
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
end
```

FIGURE IV.15. TRUNK links configuration on Switch SW1

```
GNU nano 2.9.3                  trunk sw2.txt

int f1/11
siwtchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
int f 1/12
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
int f1/14
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
end
```

FIGURE IV.16. TRUNK links configuration on Switch SW2

```
GNU nano 2.9.3                  trunk sw3.txt

int f1/11
siwtchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
int range f 1/13 - 15
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
end
```

FIGURE IV.17. TRUNK links configuration on Switch SW3

```
GNU nano 2.9.3                  trunk sw4.txt

int f1/11
siwtchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
int range f 1/13 - 15
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk allowed vlan 1,2,3,4,5,1002-1005
exit
end
```

FIGURE IV.18. TRUNK links configuration on Switch SW4

The code for the automated configuration is show below in FIGURE IV.19:

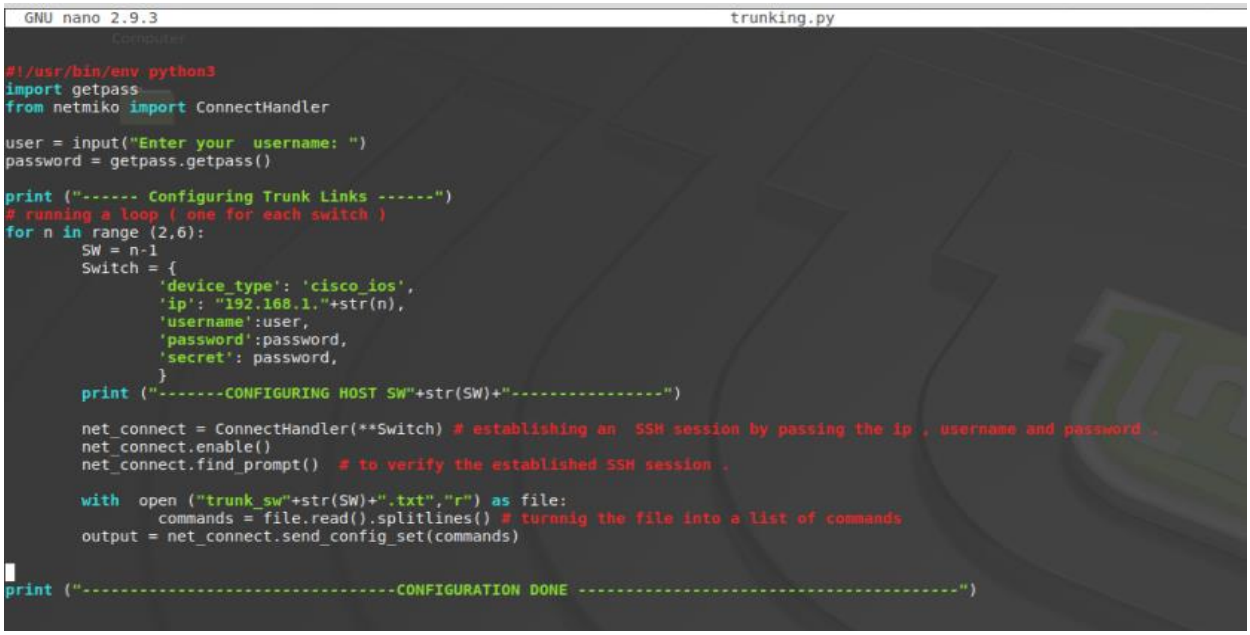

FIGURE IV.19. Automated configuration Python script

The following code will ask for a username and password once because it is the same for all the switches, then through a loop it connects to the switches one by one according to the IP address. Once connected, it opens the configuration file, turn it into a list of commands one command per line.

The last part is to send the commands by the function send_config_set (). It iterates through all the switches, once done it prints (Configuration done).

Below is the code execution from the controller shown in FIGURE IV.20:

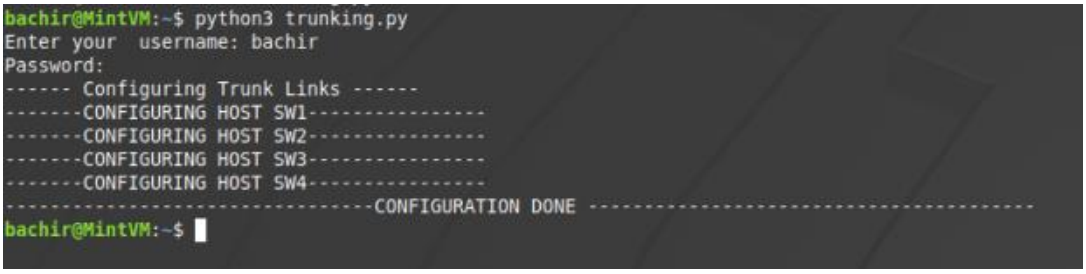

FIGURE IV.20. Configuration script execution from the Controller

Now that the Trunking is configured for the distribution layer switches, there is still the link between the Access and Distribution layer to be Trunk, as well as assigning the SERVERS to their correct VLANs.

To do so, we configure the switches as follows:

- Port f1/15 of both SW5 and SW6 is put to Trunk.
- Port f1/0 of SW5 is assigned to VLAN 2.
- port f1/1 of SW5 is assigned to VLAN 3.
- Port f1/0 of SW6 is assigned to VLAN 4.
- port f1/1 of SW6 is assigned to VLAN 5.

The script for this requirement is shown in FIGURE IV.21:

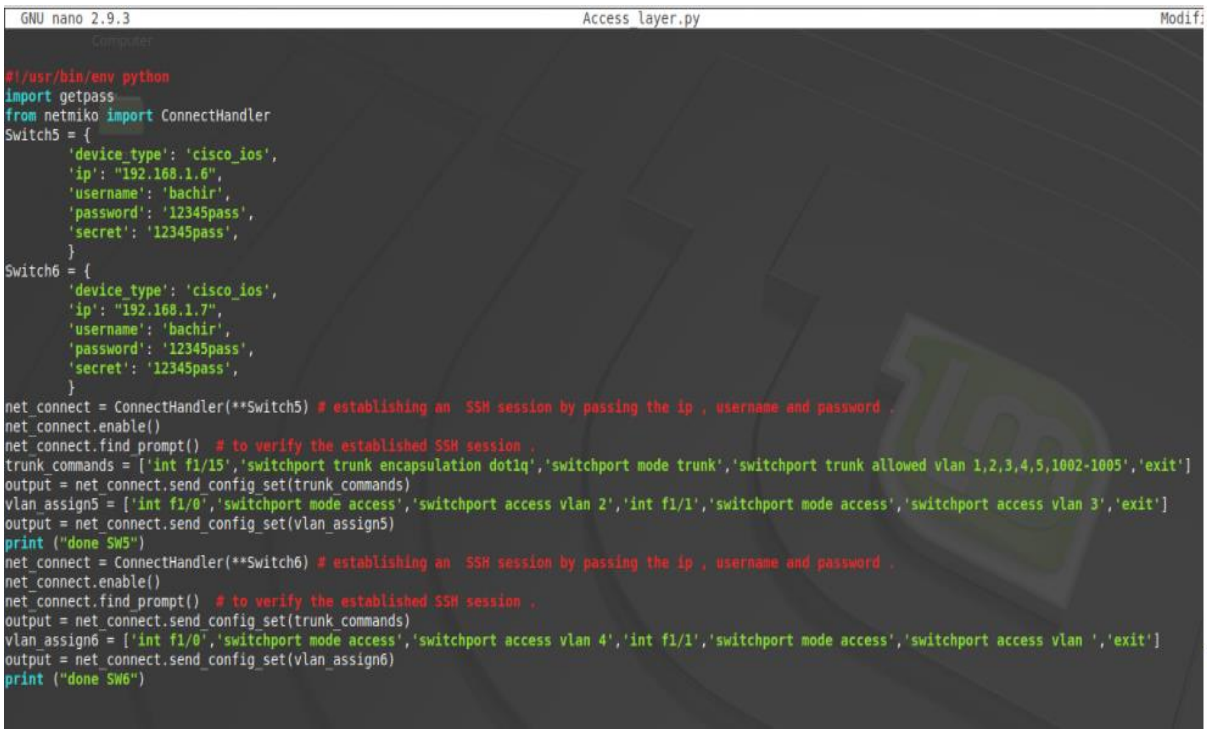

FIGURE IV.21. TRUNK configuration between distribution and access layer Switches and Server assignment to VLANs

One last requirement is the configuration of Spanning-Tree Protocol (STP) to avoid switching loops in the distribution layer since there is more than one path. STP is activated by default but we want to do load balancing between VLANS to avoid link congestion, so we choose a root bridge for each set of VLANS as follow:

- SW1 is the root bridge for VLANS 1,2 and 3 and the secondary for VLANS 4 and 5.
- SW2 is the root bridge for VLANS 4 and 5 and the secondary for VLANS 1,2 and 3.

```
#!/usr/bin/env python3
import getpass
from netmiko import ConnectHandler

Switch1 = {
        'device_type': 'cisco_ios',
        'ip': "192.168.1.2",
        'username': 'bachir',
        'password': '12345pass',
        'secret': '12345pass',
        }
Switch2 = {
        'device_type': 'cisco_ios',
        'ip': "192.168.1.3",
        'username': 'bachir',
        'password': '12345pass',
        'secret': '12345pass',
        }

net_connect = ConnectHandler(**Switch1) # establishing an  SSH session by passing the ip , username and password .
net_connect.enable()
net_connect.find_prompt()  # to verify the established SSH session .

stp_primary1 = ['spanning-tree vlan 1 root primary','spanning-tree vlan 2 root primary','spanning-tree vlan 3 root primary']
stp_primary2 = ['spanning-tree vlan 4 root primary','spanning-tree vlan 5 root primary']
output = net_connect.send_config_set(stp_primary1)
print (output)

stp_secondary1 = ['spanning-tree vlan 4 root secondary','spanning-tree vlan 5 root secondary']
stp_secondary2 = ['spanning-tree vlan 1 root secondary','spanning-tree vlan 2 root secondary','spanning-tree vlan 3 root secondary']
output = net_connect.send_config_set(stp_secondary1)
print (output)

net_connect = ConnectHandler(**Switch2) # establishing an  SSH session by passing the ip , username and password .
net_connect.enable()
net_connect.find_prompt()  # to verify the established SSH session .

output = net_connect.send_config_set(stp_primary2)
print (output)
output = net_connect.send_config_set(stp_secondary2)
print (output)
```

FIGURE IV.22. Spanning-Tree Protocol configuration script

Below is a sample of the execution:

```
bachir@MintVM:~$ python3 Stp.py
config term
Enter configuration commands, one per line.   End with CNTL/Z.
SW1(config)#spanning-tree vlan 1 root primary
 VLAN 1 bridge priority set to 8191
 VLAN 1 bridge max aging time unchanged at 20
 VLAN 1 bridge hello time unchanged at 2
 VLAN 1 bridge forward delay unchanged at 15
SW1(config)#spanning-tree vlan 2 root primary
 VLAN 2 bridge priority set to 8191
 VLAN 2 bridge max aging time unchanged at 20
 VLAN 2 bridge hello time unchanged at 2
 VLAN 2 bridge forward delay unchanged at 15
SW1(config)#spanning-tree vlan 3 root primary
 VLAN 3 bridge priority set to 8191
 VLAN 3 bridge max aging time unchanged at 20
 VLAN 3 bridge hello time unchanged at 2
 VLAN 3 bridge forward delay unchanged at 15
SW1(config)#end
SW1#
config term
Enter configuration commands, one per line.   End with CNTL/Z.
SW1(config)#spanning-tree vlan 4 root secondary
 VLAN 4 bridge priority unchanged at 16384
 VLAN 4 bridge max aging time unchanged at 20
 VLAN 4 bridge hello time unchanged at 2
 VLAN 4 bridge forward delay unchanged at 15
SW1(config)#spanning-tree vlan 5 root secondary
 VLAN 5 bridge priority unchanged at 16384
 VLAN 5 bridge max aging time unchanged at 20
 VLAN 5 bridge hello time unchanged at 2
 VLAN 5 bridge forward delay unchanged at 15
SW1(config)#end
SW1#
config term
Enter configuration commands, one per line.   End with CNTL/Z.
SW2(config)#spanning-tree vlan 4 root primary
 VLAN 4 bridge priority set to 8192
```

FIGURE IV.23. Spanning-Tree Protocol primary/secondary configuration script execution

- **The Routers:**

  After finishing all the necessary configurations for both Access and distribution layer switches, we move now to router configuration.

  In the router, two main things must be configured:

  - Forming trunk link with the switch using sub-interfaces.
  - Configuring OSPF routing protocol for both Routers.

Forming trunk links:

In order to form a trunk link with the switch from the router, it is necessary to create a sub-interface for each VLAN with dot1q encapsulation on the FastEthernet2/0.

The following script and its execution implement that as shown in FIGURE IV.24 and FIGURE IV.25 respectively:

```
#!/usr/bin/env python3
import getpass
from netmiko import ConnectHandler

Router = {
        'device_type': 'cisco_ios',
        'ip': "192.168.244.130",
        'username': 'bachir',
        'password': '12345pass',
        'secret': '12345pass',
        }

net_connect = ConnectHandler(**Router) # establishing an  SSH session by passing the ip , username and password .
net_connect.enable()
net_connect.find_prompt()  # to verify the established SSH session .

for n in range(1,6):

        sub_interfaces = ['int f2/0.'+str(n),'encapsulation dot1q '+str(n),'ip address 192.168.'+str(n)+'.1 255.255.255.0','exit']
        output = net_connect.send_config_set(sub_interfaces)
        print (output)
output = net_connect.send_config_set(['int f2/0','no shut','end'])
print (output)
```

FIGURE IV.24. Python script to create a sub-interface with dot1q encapsulation on the Router

```
bachir@MintVM:~$ nano sub_interfaces.py
bachir@MintVM:~$ python3 sub_interfaces.py
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#int f2/0.1
IGEE(config-subif)#encapsulation dot1q 1
IGEE(config-subif)#ip address 192.168.1.1 255.255.255.0
IGEE(config-subif)#exit
IGEE(config)#end
IGEE#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#int f2/0.2
IGEE(config-subif)#encapsulation dot1q 2
IGEE(config-subif)#ip address 192.168.2.1 255.255.255.0
IGEE(config-subif)#exit
IGEE(config)#end
IGEE#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#int f2/0.3
IGEE(config-subif)#encapsulation dot1q 3
IGEE(config-subif)#ip address 192.168.3.1 255.255.255.0
IGEE(config-subif)#exit
IGEE(config)#end
IGEE#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#int f2/0.4
IGEE(config-subif)#encapsulation dot1q 4
IGEE(config-subif)#ip address 192.168.4.1 255.255.255.0
IGEE(config-subif)#exit
IGEE(config)#end
IGEE#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#int f2/0.5
IGEE(config-subif)#encapsulation dot1q 5
IGEE(config-subif)#ip address 192.168.5.1 255.255.255.0
```

FIGURE IV.25. Execution of the script creating a sub-interface on the Router

Configure routing:

In the basic configuration, static routing is set so that the controller is able to connect to both routers. After that, dynamic routing with OSPF protocol is configured, and since static routing has the lowest Administrative distance (AD) it will be the primary route. So, the AD of static routing should be changed to a higher value than that of OSPF (110), say 150. See FIGURE IV.26/27:

```
#!/usr/bin/env python
import getpass
from netmiko import ConnectHandler

print ("--- Configuring OSPF ----")

for n in range (1,3):
        Router = {
                'device_type': 'cisco_ios',
                'ip': "192.168.10."+str(n),
                'username': 'bachir',
                'password': '12345pass',
                'secret': '12345pass',
                }


        net_connect = ConnectHandler(**Router) # establishing an  SSH session by passing the ip , username and password .
        net_connect.enable()
        net_connect.find_prompt()  # to verify the established SSH session .

        OSPF_commands=['router ospf 1','network 0.0.0.0 255.255.255.255 area 0','router-id '+str(n)+'.'+str(n)+'.'+str(n)+'.'+str(n),'exit']

        output = net_connect.send_config_set(OSPF_commands)
        print (output )
        if n == 1 :
                output = net_connect.send_config_set(['ip route 192.168.30.0 255.255.255.0 s3/0 150'])
        if n == 2 :
                output = net_connect.send_config_set(['ip route 192.168.244.0 255.255.255.0 s3/0 150'])
        print (output)

print ("--------------CONFIGURATION DONE -------------------------------------")
```

FIGURE IV.26. Python script for Routing configuration (with OSPF)

```
bachir@MintVM:~$ python3 ospf.py
--- Configuring OSPF ----
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#router ospf 1
IGEE(config-router)#network 0.0.0.0 255.255.255.255 area 0
IGEE(config-router)#router-id 1.1.1.1
IGEE(config-router)#exit
IGEE(config)#end
IGEE#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
IGEE(config)#ip route 192.168.30.0 255.255.255.0 s3/0 150
IGEE(config)#end
IGEE#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
LIBRARY(config)#router ospf 1
LIBRARY(config-router)#network 0.0.0.0 255.255.255.255 area 0
LIBRARY(config-router)#router-id 2.2.2.2
LIBRARY(config-router)#exit
LIBRARY(config)#end
LIBRARY#
config term
Enter configuration commands, one per line.  End with CNTL/Z.
LIBRARY(config)#ip route 192.168.244.0 255.255.255.0 s3/0 150
LIBRARY(config)#end
LIBRARY#
--------------CONFIGURATION DONE -------------------------------------
```

FIGURE IV.27. Execution of the Routing configuration script from the Controller

IV.1.9 Reading the State of Devices:

Since the configuration of the network devices is finished from the controller, the next task is to verify this configuration to see if everything works as intended.

- **Reading the state of the routers:**

First, checking if the OSPF routing protocol is properly working: is it configured? Does the router establish an adjacency with the other router? And are all interfaces participating in the routing update?

The following script gathers the necessary information and stores it:

```
#!/usr/bin/env python3
import getpass
from netmiko import ConnectHandler

print ("--- Retreiving the routing information ----")

Router = {
        'device_type': 'cisco_ios',
        'ip': "192.168.10.1",
        'username': 'bachir',
        'password': '12345pass',
        'secret': '12345pass',
        }

net_connect = ConnectHandler(**Router) # establishing an  SSH session by passing the ip , username and password .
net_connect.enable()
net_connect.find_prompt()  # to verify the established SSH session .

net_connect.send_command('terminal length 0')
output = net_connect.send_command('show ip protocols')
save = open ("/home/bachir/Router_R1/Routing_info.txt","w")
save.write(output)
save.write("\n\n_____________________________________________\n")
output = net_connect.send_command('show ip ospf neighbor')
save.write(output)
save.write("\n\n_____________________________________________\n")
output = net_connect.send_command('show ip ospf interface brief')
save.write(output)
save.write("\n\n_____________________________________________\n")
save.close()
print ("------------------------- DONE -------------------------------------")
```

FIGURE IV.28. Python script to display the Routing configuration

The gathered information is stored in the directory of Router R1 in the name of "Routing_info.txt". It is shown in the next figure:

```
bachir@MintVM:~$ cd Router_R1
bachir@MintVM:~/Router_R1$ cat Routing_info.txt
Routing Protocol is "ospf 1"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 1.1.1.1
  Number of areas in this router is 1. 1 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    0.0.0.0 255.255.255.255 area 0
 Reference bandwidth unit is 100 mbps
  Routing Information Sources:
    Gateway         Distance      Last Update
    2.2.2.2              110      00:28:41
  Distance: (default is 110)


_____________________________________________

Neighbor ID     Pri   State           Dead Time   Address         Interface
2.2.2.2           0   FULL/  -        00:00:39    192.168.10.2    Serial3/0

_____________________________________________

Interface    PID   Area            IP Address/Mask    Cost  State Nbrs F/C
Se3/0        1     0               192.168.10.1/30    64    P2P   1/1
Fa2/0.5      1     0               192.168.5.1/24     1     DR    0/0
Fa2/0.4      1     0               192.168.4.1/24     1     DR    0/0
Fa2/0.3      1     0               192.168.3.1/24     1     DR    0/0
Fa2/0.2      1     0               192.168.2.1/24     1     DR    0/0
Fa2/0.1      1     0               192.168.1.1/24     1     DR    0/0
Fa0/0        1     0               192.168.244.130/24 1     DR    0/0
```

FIGURE IV.29. Routing information saved as Routing_info.txt

From the gathered information, it can clearly be seen that OSPF is configured with a router-id of 1.1.1.1.

Another observation is that it has established a neighbouring with router R2, and it is in the full state.

The last part shows that all the interfaces are participating in the routing-update process.

Second, the routing table is displayed and stored in a text file so that it can be accessed any time. To do so, connect to the router, execute a show command and store the output to a text file.

```
#!/usr/bin/env python3
import getpass
from netmiko import ConnectHandler

print ("--- Retreiving the routing table ----")
for n in range (1,3):
        Router = {
                'device_type': 'cisco_ios',
                'ip': "192.168.10."+str(n),
                'username': 'bachir',
                'password': '12345pass',
                'secret': '12345pass',
                }


        net_connect = ConnectHandler(**Router) # establishing an  SSH session by passing the ip , username and password .
        net_connect.enable()
        net_connect.find_prompt()  # to verify the established SSH session .

        output = net_connect.send_command('terminal length 0')
        output2 = net_connect.send_command('sh ip route')

        save = open ("/home/bachir/Router_R"+str(n)+"/Routing_Table.txt","w")
        save.write(output2)
        save.close()
print ("--------------------------- DONE ---------------------------------------")
```

FIGURE IV.30. Python script that shows the Routing table and stores it in a file

This script will connect to the first router, get the routing table and store it in a text file in the directory specified for it (Router_R1). Then it will do the same for the second Router.

After executing the script, we get the following results.

```
bachir@MintVM: ~/Router_R1
File  Edit  View  Search  Terminal  Help
 GNU nano 2.9.3                  Routing Table.txt

Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

O    192.168.30.0/24 [110/65] via 192.168.10.2, 00:49:41, Serial3/0
     192.168.10.0/24 is variably subnetted, 2 subnets, 2 masks
C       192.168.10.0/30 is directly connected, Serial3/0
O       192.168.10.0/24 [110/128] via 192.168.10.2, 04:19:48, Serial3/0
C    192.168.244.0/24 is directly connected, FastEthernet0/0
C    192.168.4.0/24 is directly connected, FastEthernet2/0.4
C    192.168.5.0/24 is directly connected, FastEthernet2/0.5
C    192.168.1.0/24 is directly connected, FastEthernet2/0.1
C    192.168.2.0/24 is directly connected, FastEthernet2/0.2
```

FIGURE IV.31. Routing table for R1

```
bachir@MintVM: ~/Router_R2
File  Edit  View  Search  Terminal  Help
 GNU nano 2.9.3                  Routing Table.txt

Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.30.0/24 is directly connected, FastEthernet0/0
     192.168.10.0/24 is variably subnetted, 2 subnets, 2 masks
O       192.168.10.0/30 [110/128] via 192.168.10.1, 04:19:56, Serial3/0
C       192.168.10.0/24 is directly connected, Serial3/0
O    192.168.244.0/24 [110/65] via 192.168.10.1, 00:44:06, Serial3/0
O    192.168.4.0/24 [110/65] via 192.168.10.1, 04:19:56, Serial3/0
O    192.168.5.0/24 [110/65] via 192.168.10.1, 04:19:56, Serial3/0
O    192.168.1.0/24 [110/65] via 192.168.10.1, 04:19:56, Serial3/0
O    192.168.2.0/24 [110/65] via 192.168.10.1, 04:19:56, Serial3/0
```

FIGURE IV.32. Routing table for R2

As observed from the routing tables, the router has learned the non-connected networks through OSPF, and there exists a route to every destination.

Finally, the ping results from the router to other destinations are tested, (the homeClient and the SERVERS).

```
IGEE#ping 192.168.30.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.30.2, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 24/277/1024 ms
IGEE#ping 192.168.30.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.30.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/27/36 ms
IGEE#ping 192.168.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/13/28 ms
IGEE#ping 192.168.3.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.3.2, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 8/13/20 ms
IGEE#ping 192.168.4.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.4.2, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 8/24/36 ms
IGEE#ping 192.168.5.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.5.2, timeout is 2 seconds:
.!!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 8/11/12 ms
```

FIGURE IV.33. Ping results from the Router to homeClient and the SERVERS

The pings are successful which shows that the Router is working as intended and that all destinations are reachable.

- **Reading the state of the switches:**

Check the VLAN table: the very first thing to test is whether the switches are working as planned. Verify the configured VLANS on all the switches and save for future-use.

```
#!/usr/bin/env python3
import getpass
from netmiko import ConnectHandler

print ("--- Retreiving Vlan information ----")
for n in range (6,8):
        Switch = {
                'device_type': 'cisco_ios',
                'ip': "192.168.1."+str(n),
                'username': 'bachir',
                'password': '12345pass',
                'secret': '12345pass',
                }


        net_connect = ConnectHandler(**Switch) # establishing an  SSH session by passing the ip , username and password .
        net_connect.enable()
        net_connect.find_prompt()  # to verify the established SSH session .

        output = net_connect.send_command('terminal length 0')
        output2 = net_connect.send_command('sh vlan-switch bri')

        save = open ("/home/bachir/Vlan_tables/SW"+str(n-1)+".txt","w")
        save.write(output2)
        save.write("\n_____\n")
        save.close()
print ("------------------------- DONE ---------------------------------------")
```

FIGURE IV.34. Python script showing VLANS configuration information

The VLAN tables are stored after the execution of this script.

```
GNU nano 2.9.3                  SW6.txt

VLAN Name                        Status    Ports
---- ----------------------------------------
1    default                     active    Fa1/2, Fa1/3, Fa1/4, Fa1/5, Fa1
2    VLAN0002                    active
3    VLAN0003                    active
4    VLAN0004                    active    Fa1/0
5    VLAN0005                    active    Fa1/1
1002 fddi-default                active
1003 token-ring-default          active
1004 fddinet-default             active
1005 trnet-default               active
```

FIGURE IV.35. VLAN Table of SW6

```
GNU nano 2.9.3                  SW5.txt

VLAN Name                        Status    Ports
---- ----------------------------------------
1    default                     active    Fa1/2, Fa1/3, Fa1/4, Fa1/5, Fa1
2    VLAN0002                    active    Fa1/0
3    VLAN0003                    active    Fa1/1
4    VLAN0004                    active
5    VLAN0005                    active
1002 fddi-default                active
1003 token-ring-default          active
1004 fddinet-default             active
1005 trnet-default               active
```

FIGURE IV.36. VLAN Table of SW7

Four VLANS are created on the switches, it can be seen that the correct interfaces are assigned to their correct VLANS.

Checking the STP to see if the spanning-tree is running as desired. First, get the information from the distribution layer switches by executing some show commands.

```
#!/usr/bin/env python3
import getpass
from netmiko import ConnectHandler

print ("--- Retreiving STP information ----")
for n in range (2,6):
        Switch = {
                'device_type': 'cisco_ios',
                'ip': "192.168.1."+str(n),
                'username': 'bachir',
                'password': '12345pass',
                'secret': '12345pass',
                }

        net_connect = ConnectHandler(**Switch) # establishing an  SSH session by passing the ip , username and password .
        net_connect.enable()
        net_connect.find_prompt()  # to verify the established SSH session .

        output = net_connect.send_command('terminal length 0')
        save = open ("/home/bachir/STP/info.txt","w")
        if n  == 2 :
                output2 = net_connect.send_command('sh spanning-tree root')
                save.write("-------STP INFO OF SW2----------------")
                save.write(output2)
        if n == 4 :
                output4 = net_connect.send_command('sh spanning-tree brief')
                save.write("-------STP INFO OF SW4----------------")
                save.write(output4)
        save.write("\n_____________________________________________________________\n")
        save.close()
print ("------------------------- DONE ---------------------------------------")
```

FIGURE IV.37. Python script that shows STP configuration information

From SW1:

```
VLAN1
  Root ID     Priority    8191
              Address     c203.3350.0000
              This bridge is the root
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
VLAN2
  Root ID     Priority    8191
              Address     c203.3350.0001
              This bridge is the root
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
VLAN3
  Root ID     Priority    8191
              Address     c203.3350.0002
              This bridge is the root
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
VLAN4
  Root ID     Priority    8192
              Address     c204.2a18.0003
              Cost        19
              Port        53 (FastEthernet1/12)
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
VLAN5
  Root ID     Priority    8192
              Address     c204.2a18.0004
              Cost        19
              Port        53 (FastEthernet1/12)
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
```

FIGURE IV.38. SW1 STP root bridge information

Clearly, switch SW1 is the root bridge for the VLAN 1,2,3 as planned.

From SW3:

```
VLAN4
  Spanning tree enabled protocol ieee
  Root ID     Priority    8192
              Address     c204.2a18.0003
              Cost        19
              Port        52 (FastEthernet1/11)
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID   Priority    32768
              Address     c205.1e24.0003
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
              Aging Time 300

Interface                                    Designated
Name                 Port ID Prio Cost  Sts Cost  Bridge ID            Port ID
-------------------- ------- ---- ----- --- ----- -------------------- -------
FastEthernet1/11     128.52   128    19 FWD     0  8192 c204.2a18.0003 128.52
FastEthernet1/13     128.54   128    19 FWD    19 32768 c205.1e24.0003 128.54
FastEthernet1/14     128.55   128    19 BLK    19 16384 c203.3350.0003 128.55
FastEthernet1/15     128.56   128    19 FWD    19 32768 c205.1e24.0003 128.56
```

FIGURE IV.39. SW3 STP port states

From this output, observe that spanning-tree protocol (STP) is blocking f1/14 link, which is connected to SW1, and it is forwarding by f1/13 and f1/11 which are connected to SW4 and SW2 respectively.

This is expected because the SW2 is the root bridge for VLAN 4.

IV.2 Network Automation with ansible tool and YAML playbook:

Ansible is a radically simple IT automation tool that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs. It is open source and runs in many operating systems such as Windows and Linux, it connects remotely to devices using SSH protocol [42].

Playbooks are YAML files that describe configurations and deployment in ansible and how ansible manages devices [43]. Each playbook manages a set of devices with a set of ansible tasks.

YAML is a human-readable, data-serialization language. It is used for configuration files and in applications where data is being stored or transmitted.

IV.2.1 Installing Ansible on the controller:

Before we start using ansible, it first must be presented and installed on the controller.

So first we update the references with “sudo apt-get update”, and then install ansible for python3 using the command “pip3 apt-get install ansible”. Once finished we verify by the command “ansible –version”.

```
bachir@MintVM:~$ ansible --version
ansible 2.10.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/bachir/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /home/bachir/.local/lib/python3.6/site-packages/ansible
  executable location = /home/bachir/.local/bin/ansible
  python version = 3.6.9 (default, Jul 17 2020, 12:50:27) [GCC 8.4.0]
bachir@MintVM:~$
```

FIGURE IV.40. Verifying ansible version

IV.2.2 Backup of the running configuration with ansible:

The following YAML playbook will be loaded and run by ansible against the hosts file to retrieve the running-configuration and store it.

```
---
- name : BACKUP CONFIGURATION
  hosts: all
  gather_facts: false
  connection: network_cli

  tasks:
   - name: GETTING CONFIG
     ios_command:
       commands:
         - show run
     register : out
   - name : SAVE CONFIG
     copy :
       content: "{{out.stdout[0]}}"
       dest : "/home/bachir/Network-programmability/Backups/{{ inventory_hostname }}.txt"
   - name : REMOVE NON CONFIG LINES
     lineinfile:
       path: "/home/bachir/Network-programmability/Backups/{{ inventory_hostname }}.txt"
       line : "Building configuration..."
       state : absent
   - name : REMOVE LINE 2
     lineinfile:
       path: "/home/bachir/Network-programmability/Backups/{{ inventory_hostname }}.txt"
       regexp : "Current configuration.*"
       state : absent
```

FIGURE IV.41. YAML playbook to backup configuration

```
[routers]
R1 ansible_host=192.168.244.130
R2 ansible_host=192.168.10.2
[switches]
ESW1 ansible_host=192.168.1.2
ESW2 ansible_host=192.168.1.3
ESW3 ansible_host=192.168.1.4
ESW4 ansible_host=192.168.1.5
ESW5 ansible_host=192.168.1.6
ESW6 ansible_host=192.168.1.7
[all:vars]
ansible_connection=network_cli
ansible_ssh_user=bachir
ansible_ssh_pass=12345pass
ansible_network_os=ios
ansible_become=yes
ansible_become_methos=enable
ansible_become_user=bachir
ansible_become_pass=12345pass
```

FIGURE IV.42. Hosts file

```
[defaults]
inventory=hosts
host_key_checking = False
retry_files_enabled = False
deprecation_warnings=False
interpreter_python=/usr/bin/python3
[persistent_connection]
command_timeout=120
```

FIGURE IV.43. ansible configuration file

IV.2.3 Code explanation and execution:

Each YAML playbook starts with three dashes. It specifies the name of the playbook, the hosts that it executes against (in our case all the devices inside the host file) as well as the connection type which is CLI. Then the YAML playbook starts the tasks, each

task has a name and the set of commands to execute. All the tasks should be in the same level of indentation.

In our example, the first task named **GETTING CONFIG** which executes the "show run" command then saves the data in a variable **out** using the **register** module. The second task **SAVE CONFIG** uses the **copy** module to save the data in **out** to the backup folder in a text file named with the name specified in the hosts file.

The last two tasks delete the first two lines from the saved file by the module **lineinfile** and identify the state of the line as absent. They are deleted because they are not config lines.

The ansible.cfg file specifies the hosts file to use by inventory, the python version to use and a set of other parameters.

The hosts file specifies the device's IP address and a set of parameters regarding the device including: the device's OS type, the SSH user and password and the enable password.

The execution of this playbook is shown in the figure below:

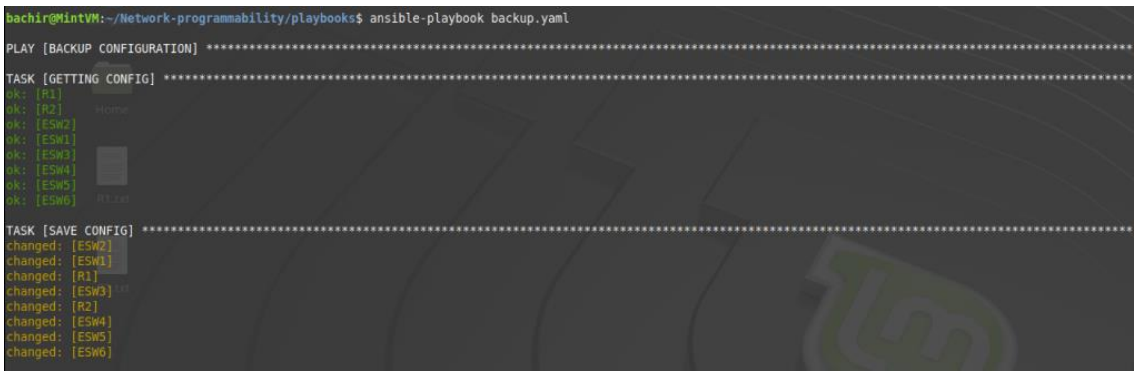

FIGURE IV.44. Ansible Playbook execution

The result:

After the execution is finished, we check the Backups folder and the result is shown below:

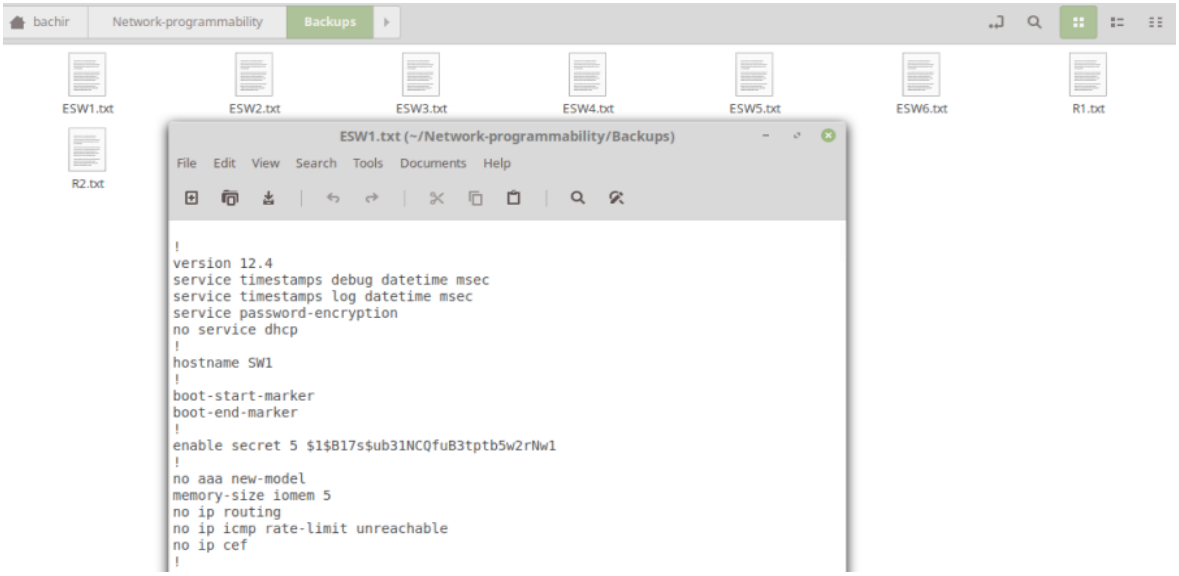

FIGURE IV.45. Ansible execution result

We can see a running-configuration file for each device stored in a text file that can be used in the future to restore the network to this state in case a failure occurs.

IV.3 APIC-EM controller vs Automation using Python in hybrid SDN:

After the demonstration of two examples in managing a network made of traditional devices, one using APIC-EM controller and the other using Python network automation. Similarities and differences were noticed:

- The installation process and the hardware requirement: it is clear that APIC-EM is very hardware demanding. In contrast, network programmability just needs relatively simple hardware.
- Both approaches can control the whole network from a single controller, hence Centralized control is established.
- Both methods use SSH and Telnet credentials to log and connect to devices, but APIC-EM can also use SNMP and CDP.
- In Both approaches, device information can be retrieved, and backup configuration can be made.
- Lowering operational expense by eliminating tedious and manual processes through automating tasks. This leads to minimal configuration errors as well as better time efficiency.
- Lower networking costs.
- Both approaches allow the discovery of network devices and hosts, but only APIC-EM draws a complete network topology.
- APIC-EM has a very organized and simple Graphical User interface (GUI). On the other hand, Network Programming uses CLI.
- APIC-EM has some advanced features and applications that allow the detection of errors and faults like Path trace and easy QoS.

The similarities and the differences between APIC-EM controller and Automation using Python in hybrid SDN are summarized in the following table:

TABLE IV.3. APIC-EM vs Python Automation

Features and functions	SDN with APIC-EM	Network Automation
Centralized control	Yes	Yes
Hardware requirement	Heavy	Relatively light
Discovery of devices	Yes	Yes
Connecting to network devices	Using CDP, SSH, TELNET and SNMP	Using Telnet and SSH
Configuration retrieval (VLANS, routing protocols, interface statistics ….)	Yes	Yes
Configuration update on devices	Easy	Change the command file
Drawing the network topology	Yes	No
Time efficiency (Saved time)	Save a huge amount of time	Save much time compared to regular configuring
Operational expenses	Low	Low
Ease of use	Graphical user interface (GUI) easy	Running the python codes from the terminal
ACL analysis and error detection.	Yes	No

The table yields a win for APIC-EM controller over Python programming overall. But it must be noted that Python programming still has its use cases, especially when using APIC-EM is not ideal/possible.

Conclusion:

The Cisco's APIC-EM and Automation using python scripts brought revolutionary changes to network operation in the industry.

As seen from the comparison, both have their strengths, but automating a network using python programming is lacking as an industrial solution overall when compared to APIC-EM. It is nonetheless a big improvement over traditional networking schemes. It helps saving time and establishing centralized control for small to medium networks.

It is also relevant to say that when the network does not involve Cisco devices, using APIC-EM becomes impossible, which leaves room for Python automation to take the lead.

GENERAL CONCLUSION

Conclusion

This project is a result of the demands and needs for a newer networking model that caused a movement from traditional networking towards a more simplified network management and control known as software-defined networking. SDN promises to transform today's static networks into flexible, scalable and programmable platforms with the ability to create network policies to allocate resources dynamically. This is accomplished by separating the control plane from the data plane.

The greatest advantage of SDN is allowing the creation of a framework to support more data-intensive applications like big data and virtualization. It also improves performance and applies security policies smoothly, hence it is considered to be the future of networking.

The problem with SDN comes from the total decoupling of control and data planes, which rises the need for new SDN network devices. APIC-EM can be used to configure and control traditional devices, but it is limited to Cisco technologies. The high hardware demands are also a problem. That's where network programmability and automation with python shows more flexibility and adaptability. Configuring and monitoring any device via automation, independent of vendors.

In this project, a presentation of SDN concepts, functionalities and advantages were held. Furthermore, the architectural components of SDN: Infrastructure layer, control layer and application layer were discussed and studied. Lastly the functionality of this SDN solution using APIC-EM as well as network automation with python were tested on different networks and a comparison between the two approaches was made to show the advantages of each solution.

This final year project consists mainly of implementing a network automation approach using python libraries and with ansible tool, to configure and maintain a network created in a simulation software (GNS3), and then compare the results to the APIC-EM approach.

As an improvement to this project and to ensure continuous evolution in the networking field, below are some additional works to be done:

- Using Network Programmability for managing real networks.
- Study of distributed SDN Controllers architecture.
- Study and implement an SDN network with OpenFlow and OpenDaylight Controller.

Bibliography

[1] Benzekki, Kamal; El Fergougui, Abdeslam; ElbelrhitiElalaoui, Abdelbaki (2016). "Software-defined networking (SDN): A survey". *Security and Communication Networks,* Issue Online: 30 March 2017, (page 38).

[2] Sunshine, Carl A. (1989). Computer Network Architectures and Protocols. Springer Science & Business Media, (page 10).

[3]http://www.tcpipguide.com/free/t_OverviewOfKeyRoutingProtocolConceptsArchitecturesP.htm, Version Date: September 20, 2005.(consulted 13-03-2020).

[4] https://searchnetworking.techtarget.com/definition/switch,(consulted 20-03-2020)

[5] by cisco press "*Introduction to Networks v6 Companion Guide"* nov 22, 2016 (consulted 22-03-2020).

[6] By Cisco Systems, Inc.. CCNA: Network Media Types Sample Chapter is provided courtesy of Cisco Press. Date: Mar 14, 2003. (consulted 24-03-2020).

[7] https://www.lifewire.com/introduction-to-network-cables-817868(consulted 21-03-2020).

[8] Guy Pujolle, Networks & Telecommunication_ Advanced Networks, Software Networks_ Virtualization, SDN, 5G, Security (2015, Wiley-ISTE), (page 52).

[9] https://www.sunbirddcim.com/what-is-data-center(consulted 23-03-2020)

[10] https://www.techopedia.com/definition/29883/data-center-virtualization(consulted 23-03-2020).

[11] Ken Gray and Thomas D. Nadeau "Software *Defined Networks: An Authoritative Review of Network Programmability Technologies"* August 8, 2013, (page 40).

[12] Mark Mitchiner and Reema Prasad "*Software-Defined Networking and Network Programmability: use cases for defence and intelligence communities", (page 2&3).*

[13] https://www.techopedia.com/definition/32318/data-plane(consulted 30-03-2020).

[14] concepts cisco SDN, François Goffinet, (section 1. IMPACT DE L'INFORMATIQUE EN NUAGE,1.5 SDN ET NFV)

[15] https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11-531522.html, May 7, 2009.(consulted 01-04-2020).

[16] Rajesh Kumar, Software Defined Networking - a definitive guide, 2013, (page 2).

[17] https://www.sdxcentral.com/networking/sdn/definitions/southbound-interface-api/ (consulted 03-04-2020).

[18] https://www.sdxcentral.com/networking/sdn/definitions/north-bound-interfaces-api/(consulted 03-04-2020).

[19] https://searchnetworking.techtarget.com/definition/SDN-controller-software-defined-networking-controller(consulted 05-04-2020).

[20] William Stallings, Foundations of Modern Networking_ SDN, NFV, QoE, IoT, and Cloud (2015, Addison-Wesley Professional) (page 164)

[21] http://netlab.boun.edu.tr/WiSe/doku.php/research:sdn, section: Distributed Controller Mechanism for SDN-based Campus Networks, (consulted 10/04/2020)

[22] Esmaeil Amiri,Emad Alizadeh,Khalilollah Raeisi "*An Efficient Hierarchical Distributed SDN Controller Model "(page 64).*

[23] Sriram Subramanian, "*Software-Defined Networking (SDN) with OpenStack" October 28, 2016.(page 78).*

[24] Sriram Subramanian, "*Software-Defined Networking (SDN) with OpenStack" October 28, 2016.(page 114).*

[25] https://networklessons.com/cisco/ccna-routing-switching-icnd2-200-105/introduction-to-apic-em, (consulted 20/04/2020)

[26] Cisco Network Visibility Application on APIC-EM User Guide, Release 1.6.0.x, September 4, 2018. (consulted 10-05-2020).

[27] A. Hussein, Louma Chadad, Nareg Adalian, Ali Chehab, Imad H. Elhajj and Ayman Kayssi *"Software-Defined Networking (SDN): the security review".* Published online: 08 Aug 2019.

[28] Cisco Application Policy Infrastructure Controller Enterprise Module – Release 1.5 Data Sheet, June 29, 2017. (consulted 14-05-2020).

[29] Cisco Application Policy Infrastructure Controller Enterprise Module Hardware Installation Guide. (page 3).

[30] APIC-EM v1.x Developer Docs<APIC-EM REST API> (consulted 15-05-2020).

[31] APIC-EM v1.x Developer Docs<Automated Discovery> (consulted 15-05-2020).

[32] APIC-EM v1.x Developer Docs<Discovery jobs and scanning credentials>(consulted 15-05-2020).

[33] APIC-EM v1.x Developer Docs<Device Inventory and host inventory >(consulted 15-05-2020).

[34] https://solutionpartner.cisco.com/media/apic-em-getting-started-2-10-16/c_topology_getstart_apic_em.html (consulted 16-05-2020).

[35] Cisco Path Trace Application on APIC-EM User Guide, Release 1.6.0.x, October,23,2017.

[36] Cisco EasyQoS Application for APIC-EM User Guide, Release 1.6.0.x

[37] Cisco IWAN Application on APIC-EM User Guide, Releases 1.6.0

[38] Design and Implementation of an SDN Network and an API. (page 42 to 50).

[39] David Bombal, Network Automation and Programmability with Python, (consulted 15 – 18 September 2020)

[40] Onkar Kubade | Jul 29, 2019,Network Automation using Python Programming (consulted 18 september 2020).

[41] David Bombal , 2018, Python NAPALM Network Automation (20-06-2020)

[42] Staff writer. "Overview - How Ansible Works". *ansible.com*. Red Hat, Inc. p. 1. Retrieved December 7, 2016. (consulted 12-08-2020).

[43] Ansible Community. "Playbooks". *Docs.ansible.com. Ansible Documentation. Red Hat, Inc. p. 1. Retrieved April 26, 2014. (consulted 19-08-2020).*

[44] https://pypi.org/project/lanscan/ (consulted 02-11-2020).

[45] https://docs.debookee.com/en/latest/lanscan.html (consulted 02-11-2020).

FIGURE REFERENCES

FIGURE I.1 https://docs.microsoft.com/en-us/windows-hardware/drivers/network/windows-network-architecture-and-the-osi-model

FIGURE I.2. https://www.distrelec.de/en/router-cisco-small-business-rv082-eu/p/12511712

FIGURE I.3. https://www.tonitrus.com/en/networking/cisco/switch/cisco-sg550x-switch/10128003-014-cisco-sg550x-24-k9-eu-small-business-sg550x-24-switch-10.000-mbps-24-port-rack-modul/

FIGURE I.4. https://rfshop.com.au/wp-content/uploads/2018/04/LMR400-Coaxial-cable.jpg

FIGURE I.5. https://s3-us-west-1.amazonaws.com/foscoshopify/graphics/uploads/2011/01/UTP-Cable-icture_thumb.jpg

FIGURE I.6. https://5.imimg.com/data5/NI/ND/MY-8593783/fiber-optic-cable-500x500.jpg

FIGURE I.8. https://www.virtual-serial-port.org/images/upload/products/vspd/articles/what/3.jpg

FIGURE I.9. https://upload.wikimedia.org/wikipedia/commons/9/96/Vergleich_2von2_Crossoverkabel.jpg

FIGURE I.10. https://www.clubic.com/energie-renouvelable/actualite-887264-data-centers-gouffres-energetiques-pensait.html

FIGURE I.11. https://techfactss.com/vmware-vs-virtualbox/

FIGURE I.12 and FIGURE I.13. Mark Mitchiner and Reema Prasad "*Software-Defined Networking and Network Programmability: use cases for defence and intelligence communities*

FIGURE I.14. https://www.citypassenger.com/en/what-is-an-sdn/

FIGURE I.15. https://ipcisco.com/tag/nfv/

FIGURE I.16. https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11-531522.pdf

FIGURE II.1. FIGURE II.2. https://www.researchgate.net/figure/SDN-Northbound-and-Southbound-Interfaces-Hong-2014_fig29_325988616

FIGURE II.3. https://kemaza.co.za/blog/2019/01/05/software-defined-networking-sdn/

FIGURE II.4. https://honim.typepad.com/files/etw-net-pdf-jan18.pdf