OULU UNIVERSITY OF
APPLIED SCIENCES

Khoa Dang

NETWORK AUTOMATION WITH PYTHON

NETWORK AUTOMATION WITH PYTHON

Khoa Dang
Bachelor's Thesis
2023
Degree of Information technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme, Option

Author(s): Dang Ngoc An Khoa
Title of the bachelor's thesis: Network Automation with Python
Supervisor(s): Teemu Leppänen
Term and year of completion: 2023 Number of pages: 44

This thesis aims to overview how to use Python and its variety library to automate network system operations. The idea will contain theoretically explained and practically demonstrated how the Python scripts will help automate the network tasks and make it easier for the network engineers to keep track of and manage the network system.

With Python, network administrators and engineers can create scripts and programs to automate various network operations, such as configuring routers and switches, monitoring network performance, and collecting network data. Python libraries such as Netmiko, Napalm, and Paramiko provide interfaces to network devices and allow automation scripts to interact with network devices programmatically.

Keywords: Network, Automation, Python, Script, Configuration, Device.

CONTENTS

VOCABULARY

OS – Operation System

SSH – Secure Shell

SFTP – Simple File Transfer Protocol

RSAkey - Rivest–Shamir–Adleman Key

DSSKey - Direct Station Selection Key

ECDSAKey - Elliptic Curve Digital Signature Algorithm Key

OSPF – Open Shortest Path First

GNS3 - Graphical Network Simulator-3

VTY - Virtual Teletype

NFV - Network functions virtualization

SDN – Software-Defined Networking

YAML - Yet Another Markup Language

SDA – Software-Defined Access

CLI – Command-Line Interface

NVRAM – Non-Volatile Random Access Memory

SCP – Secure Copy Protocol

ACL – Access Control List

VLAN – Virtual Local Area Network

IDE - Integrated Development Environment

1 INTRODUCTION

Network automation has become increasingly important today as networks have become more complex and critical to the success of businesses and companies. Automating everyday network tasks can help network administrators and engineers to reduce errors, improve productivity, and scale their infrastructure with ease. Python is an essential programming language that has gained significant popularity in the networking industry due to its simplicity, versatility, and extensive library of modules.

This thesis explores Python for network automation and its impact on network management, configuration, and monitoring.

This thesis is intended for network administrators, engineers, and developers who want to explore the benefits of using Python for network automation. The thesis assumes that readers already have some basic knowledge of networking concepts and some experience with Python programming. However, the thesis can be an excellent introduction to those new to networking or Python.

The contribution of this thesis lies in providing a comprehensive understanding of Python-based network automation, its benefits, challenges, and potential impact on network management. The thesis aims to provide insights into how Python-based network automation can help businesses and organizations to manage and scale their network infrastructure more effectively.

The objective of this thesis is to use several Python libraries to automate the networking device's operations, from using SSH protocol which is a network protocol that provides a secure way to access and communicate with remote systems to establish the SSH connections to configuring and managing the networking devices.

Moreover, this thesis will give me fantastic opportunities to improve my programming with Python skills and extend my knowledge of the network system. Setting up and managing the hypervisor software on my PC and the virtual machines is also an essential objective of this thesis.

While doing this thesis, there will be a lot of errors and obstacles waiting for me ahead, and it is inevitable. I hope to achieve all the objectives.

2 TOOLS

Instead of using real networking devices for conducting the network automation tasks in this thesis, I installed a layer two hypervisor software, Virtual Box, to host a GNS3 virtual machine since it is unnecessary and will take a lot of money if I decide to use the real one.

It is easy to download the networking device's IOUs, which are network simulators that allow users to simulate Cisco routers and switches on their computers. It is a powerful tool that can be used for learning and practicing networking concepts, testing network configurations, and creating virtual network topologies.

You can import the IOUs into the GNS3's GUI, which is a Graphical User Interface, the GNS3 software is the primary interface through which you interact with the software and the networking devices which runs on the GNS3 virtual machine hosted by Virtual Box, I will give you a specific explanation of GNS3 in the next section.

The next step is to build up your network topology, which includes routers, switches, firewalls, etc. Of course, configuring the system is a crucial stage. If needed, you must configure the subnets and inter-Vlan routing, configuring the communication between virtual local area networks. Still, in this case, IP addresses and VTY lines for SSH connections are more than enough since I should leave room for illustrating other network automation tasks, IP stands for Internet Protocol, and VTY stands for Virtual Teletype, which is used to allow connecting to the devices using telnet or Secure Shell (SSH).

2.1 GNS3

Gns3 Is an open-source network simulation tool that allows users to simulate complex network topologies on their computers. Network engineers and students widely use it to learn and practice networking concepts, test network configurations, and create virtual network environments.

GNS3 supports various network devices, such as routers, switches, and firewalls from multiple vendors, including Cisco, Juniper, and others. It allows users to simulate different network scenarios and configurations and test the behavior of network devices in a controlled environment.

2.2 PyCharm

PyCharm is an integrated development environment (IDE) for Python programming language that JetBrains developed. It is a powerful and popular tool among Python developers that provides advanced code editing, debugging, testing, and deployment features.

PyCharm provides a user-friendly and customizable interface that makes writing and managing Python code easy, including code completion and suggestion, debugging, code Analysis, and deployment.

You do not necessarily have to use PyCharm to code network automation tasks with Python, but it can make the process easier and more efficient.

PyCharm is a powerful IDE for Python development that offers many features and tools to help you write, test, and debug your code.

Moreover, using PyCharm to code Python does offer several benefits, including Intelligent code completion, which is indispensable and can save time and effort. At the same time, coding and built-in unit testing is another essential benefit of PyCharm that makes it easy to write and run tests for your code. This can help ensure that your code works as intended and can catch issues early in development.

PyCharm can be a powerful tool for Python developers, offering a range of features and tools that can save time, increase productivity, and help you write better code.

2.3 Cisco IOUs

IOU stands for IOS on Unix is a virtualized version of Cisco's IOS software that can be used for network simulation and testing purposes. It allows network engineers to create virtual network topologies and practice various network tasks, such as configuring routers and switches, without needing physical hardware.

Cisco IOU is often used in conjunction with network simulation software like GNS3 or EVE-NG, which are the network virtualization platform that allows you to create and manage virtual network environments for testing and learning purposes, which provide a graphical user interface for building and managing virtual network topologies. IOU images can be loaded into these simulation tools to create virtual Cisco devices that can be configured and tested like physical network devices.

2.4 Python Libraries

Netmiko

Netmiko is a multi-vendor Python library that simplifies network device configuration and management automation. It provides a consistent and easy-to-use interface for managing network devices across multiple vendors, such as routers, switches, and firewalls.

Netmiko uses the Secure Shell (SSH) protocol to connect to network devices and provides functions and methods to execute commands on the devices. It supports many network device vendors, including Cisco, Juniper, Arista, and Huawei.

With Netmiko, network administrators and engineers can automate common networking tasks, such as device configuration, firmware upgrades, backup and restore operations, and network monitoring. Netmiko allows users to interact with devices programmatically, saving time and reducing the likelihood of errors or misconfigurations.

Netmiko is built on top of the Paramiko SSH library and provides an extended set of methods and features specifically designed for network automation. It also supports using custom SSH keys, which can help improve security and compliance with organizational policies.

Since Netmiko is one of Python's proprietary libraries, we do not need to install it separately. We can install Netmiko directly from Pycharm's toolbar with only a few clicks.

Paramiko

Paramiko is a Python library that implements the SSH (Secure Shell) protocol, allowing Python applications to connect to remote servers and execute commands securely. It enables Python applications to connect to remote servers using the SSH protocol and provides a secure channel for communication between the client and server. (2).

Paramiko provides classes and functions that enable Python applications to establish SSH connections, authenticate with remote servers using various authentication methods (such as passwords, SSH keys, or certificates), and execute commands on remote servers.

Paramiko is widely used in network automation, where it is often used in conjunction with other libraries, such as Netmiko or Napalm, to automate network device configuration and management tasks. It can also be used in other areas where secure remote access is required, such as system administration, cloud computing, and cybersecurity.

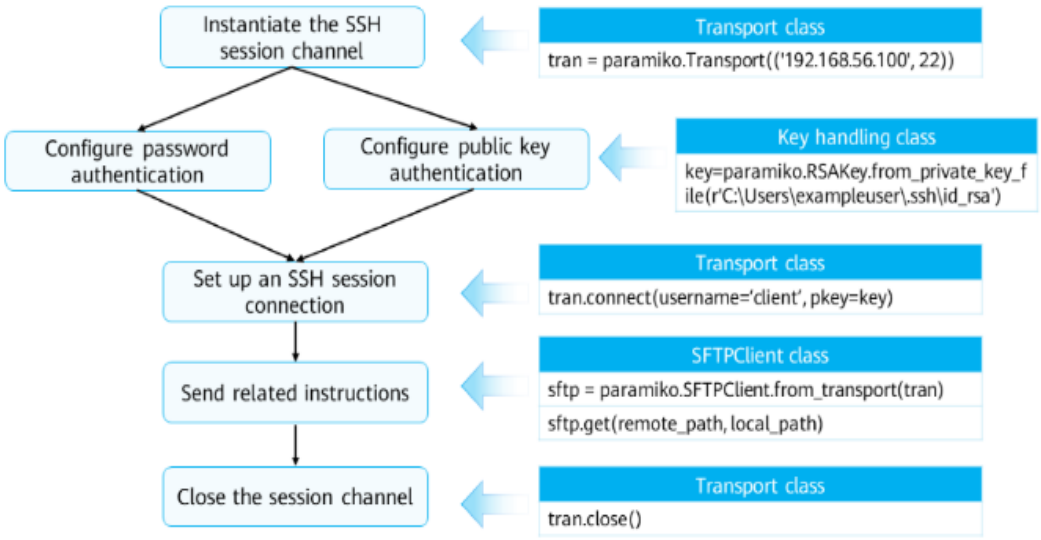


FIGURE 1. Paramiko operations (2).

Napalm

Napalm is a Python library that provides a vendor-agnostic API (Application Programming Interface) for working with network devices. It is designed to simplify network automation and management by abstracting away the underlying vendor-specific details and providing a consistent interface across different network devices.

Napalm allows network engineers and administrators to automate common network management tasks such as configuration, provisioning, monitoring, and troubleshooting. It supports many network device vendors, including Cisco, Juniper, Arista, and Huawei.

Napalm provides a set of joint operations that can be performed on network devices, such as retrieving configuration information, applying configuration changes, checking interface statistics, and gathering network topology information also provides features such as rollback support, validation of configuration changes, and comparison of configuration differences between devices. (11).

Napalm can be combined with Python libraries like Netmiko, Paramiko, and Ansible to build complex network automation workflows. It can also be integrated with popular network monitoring tools such as Prometheus and Grafana to monitor real-time network performance.

Async I/O

Async IO stands for Asynchronous Input/Output, which allows multiples IO operations to be executed simultaneously without colliding with each other, and this means that instead of waiting for the currently executed network operations to complete before moving on to the following tasks, the script will continue performing other tasks while the previous functions are still in process. (8).

Async IO can improve the performance of scripts that interact with network devices, resulting in faster execution times and more efficient use of system resources.

Someone will ask why we must use the Async IO model in performing network automation tasks.

Async IO helps improve the performance of the scripts, which we use to conduct the automation of the networking devices since it helps our code to run asynchronously, and this can dramatically reduce the time for waiting for the other I/O operations to complete.

Moreover, Async IO can simplify your code by eliminating the need for callbacks or explicit threading. This can make your code more readable and easier to maintain.

One of the upsides when using the Async I/O model is to reduce resource usage by allowing it to process I/O operations with fewer threads or processes. This can be particularly important when working with networking devices with limited resources.

Why does Async I/O benefit network automation? To answer this question, we must dig deeper into the differences between Concurrency and Parallelism.

The figure below shows how the processor solves the task using Concurrency and Parallelism methodologies (3).

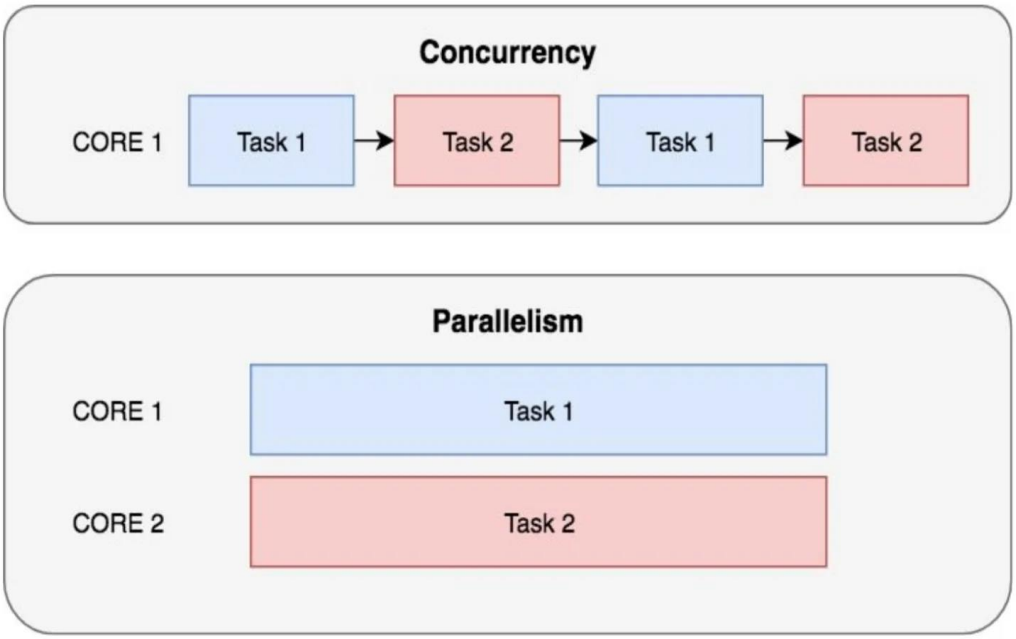


FIGURE 2. Concurrency vs. Parallelism (3).

Concurrency

Concurrency means being able to handle multiple jobs at once, and those jobs do not have to happen at the same time. It is clearly explained through the following example:

A boy is jogging in the morning. In the process of running, he discovers that he has forgotten to tie his shoes. Now he must stop and start relacing before he can continue running.

This is a basic example of concurrency. The guy above can handle both running and lacing at the same time, meaning he can handle multiple tasks at the same time.

Parallelism

Parallelism means being able to handle multiple jobs at the same time. It sounds like concurrency, but it's not; back to the example of the jogger friend from earlier. Suppose he goes for a run and listens to music simultaneously. In this case, he can do two jobs simultaneously running and listening to music. That is, these two actions happen in parallel.

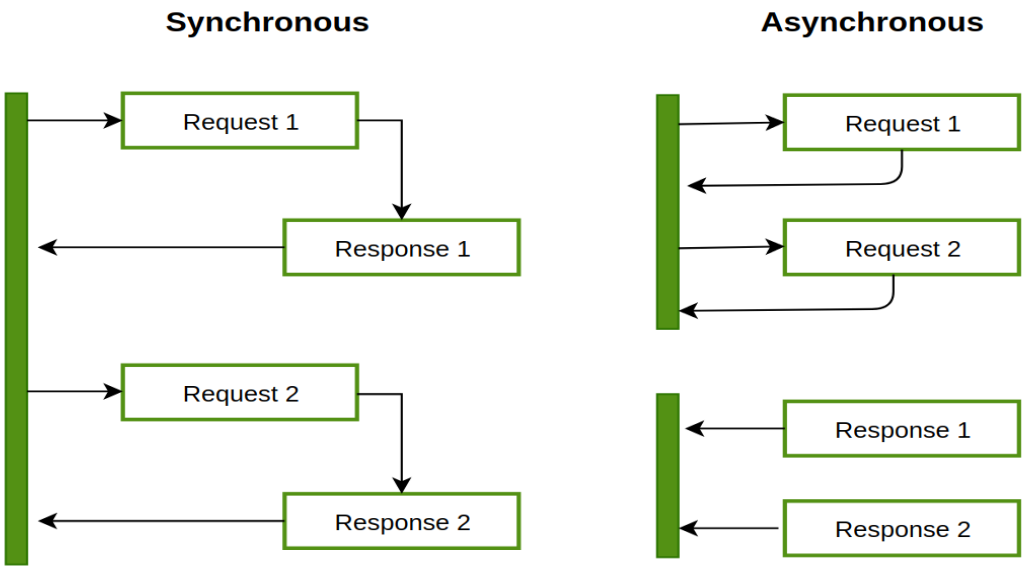


FIGURE 3. Synchronous vs. Asynchronous (4).

The synchronous model runs with a single thread and returns the result individually. At the same time, the Asynchronous starts another thread and wait for the result to come back and resynchronizes to combine it. (4).

3 SETTING UP THE SYSTEM

The working environment required for this thesis includes a network system that can be used to simulate network automation operations. As I mentioned above, GNS3 had been chosen as the network simulator. Cisco IOUs should be the best choice among networking devices vendors since other networking devices' OS (Operation System) hardly rely on Cisco's innovations.

To set up the system correctly, first, you must install the GNS3 software, download the Cisco IOUs, and then import the IOU images into the GNS3. You must specify what kind of devices the IOU images belong to, and you will not want to create a switch but use a router's image.

The networking devices running on the GNS3 machine must be pre-configured, for example, login credentials, IP address, SSH protocol, etc.

I will show you step-by-step how to set up a networking system used in this thesis to illustrate the network system's operations in the upcoming pages.

3.1 How to import Cisco IOUs into GNS3

3.1.1 Download the IOU images

Firstly, you need to download the specific version of Cisco IOUs images. However, remember that Cisco IOU is not officially available for download from Cisco's website and is intended for use only in non-production environments with proper licensing and permission. That is why you need to find those from other sources, and it may cost you a little time to search. I download those IOUs for free from "mega.nz".

3.1.2 Import the IOU images into GNS3

The next step is to import the IOU images into GNS3, and I assume that you already installed the GNS3 virtual machine and linked it to the GNS3 software, then you need to open the GNS3 software's GUI and follow step by step as I show it below.

- Open GNS3 and go to Edit -> Preferences -> IOU Devices, Figure 4.

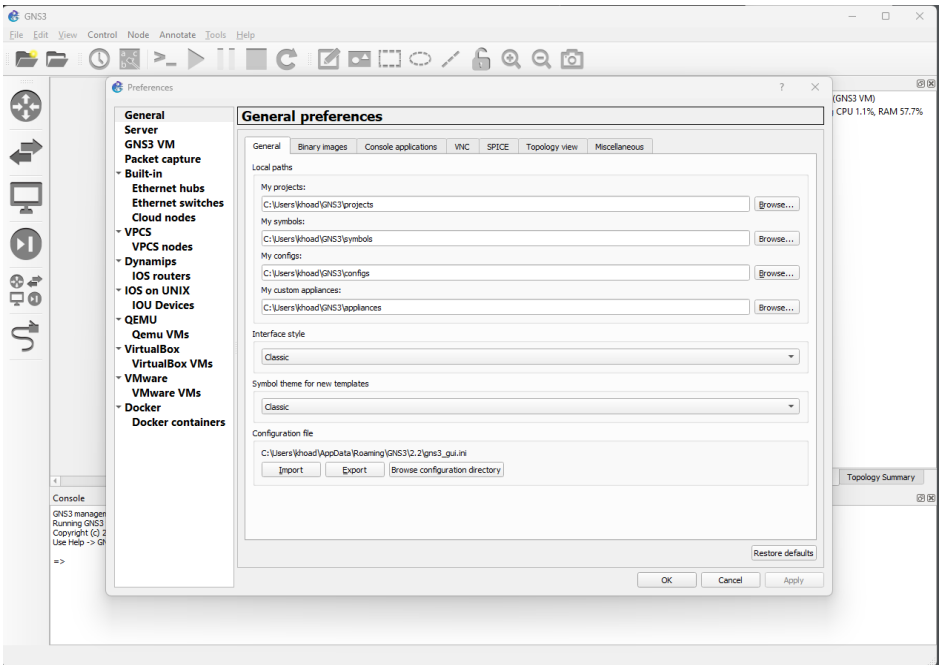


FIGURE 4. Gns3 Preferences

- Click on the "New" button to add a new IOU device, Figure 5.

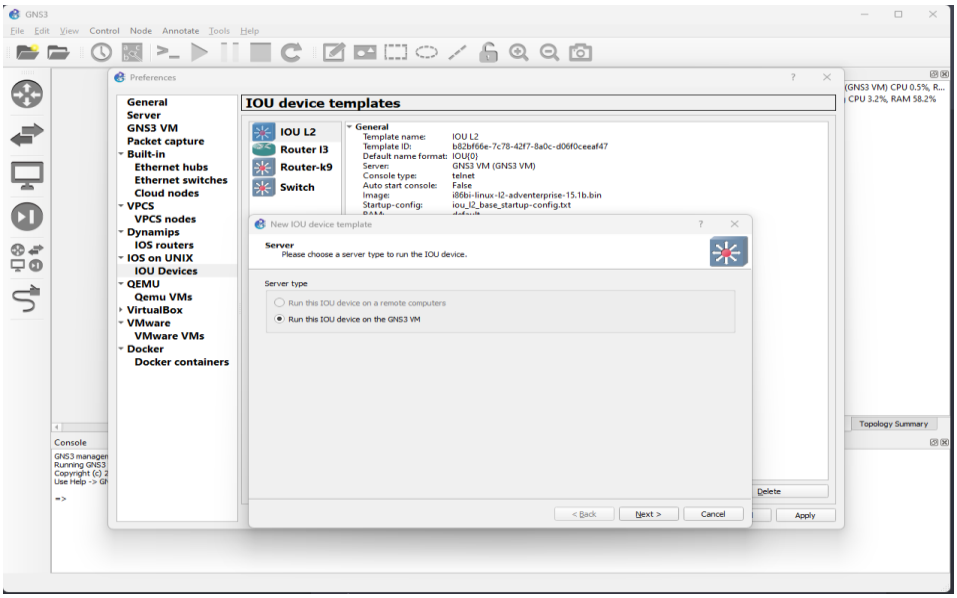


FIGURE 5. Creating new device

- Name your IOU device and select the IOU image you downloaded in Step 1.
- Click on the "Finish" button to save your IOU device settings, Figure 6.

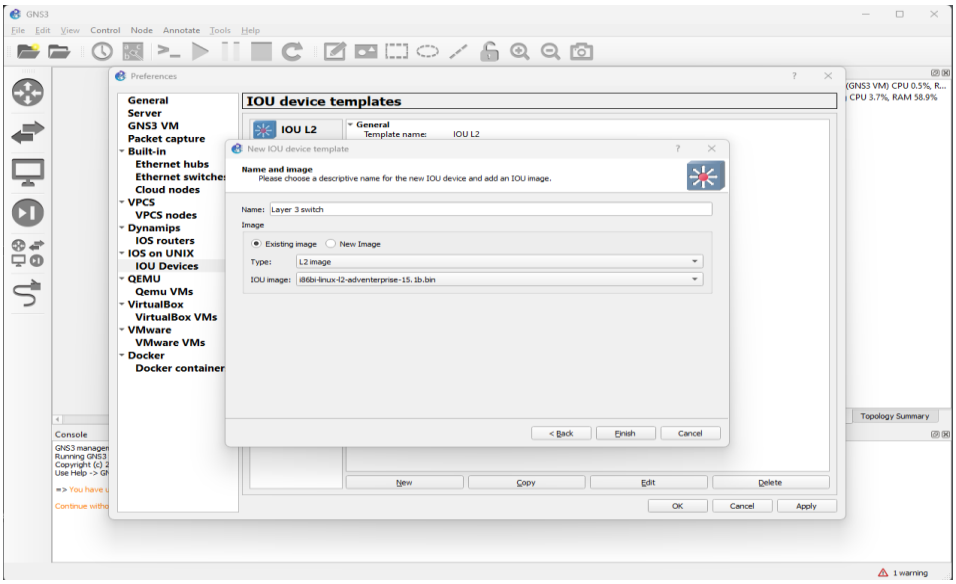


FIGURE 6. Creating device name

- Drag and drop the IOU device into your GNS3 topology.

After finishing all the steps above, you can start building and configuring your network topology.

Remember that when dragging and dropping the devices in a GNS3 project, you must decide whether the devices will run on the GNS3 virtual machine or the host OS, which is your computer.

It is recommended to create a network adapter to establish the connectivity between the GNS3 virtual machine and the host OS, and it could be a Microsoft KM-TEST loopback interface. It depends on your host OS.

3.2 Pre-configuring devices

The networking devices should be pre-configured with the IP address so they can be reached. In addition, Login credentials should be applied to the devices since it is the first security layer that protects them from being accessed by unauthorized users.

SSH protocol is the backbone and helps transport the data between the control node and managed devices, which is indispensable.

In the next section, the pre-configuring process will be presented step-by-step.

3.2.1 IP address

The next step is configuring the interface's IP address directly connected to the layer-3 switch. I entered the interface Ethernet0/0 of the router one and typed in *"ip address 10.1.1.10 255.255.255.0"* 255.255.255.0 is the netmask of the /24 subnet, and "no shutdown" to enable the interface since the router's interfaces are in down status by default.

After the configuration had been applied, I accessed the running configuration, and the result is shown in Figure 7 below.

```
interface Ethernet0/0
 ip address 10.1.1.10 255.255.255.0
```

FIGURE 7. IP address

The exact configuration had been applied to the other two routers since some network automation tasks require the Python scripts to configure several devices simultaneously.

3.2.2 Enable SSH protocol on the devices

It is required to create a domain name first for the networking devices to be allowed to generate an RSA key based on the RSA algorithm, which is potential for the SSH connections to be established. (5).

I was typing in "ip domain-name" to create the domain name.

Typing in "crypto key generates rsa" to create an RSA key (5).

Press OK to confirm the configuration.

After applying the configuration to the devices, logs show the recently enabled SSH protocol status, as shown in Figure 8.

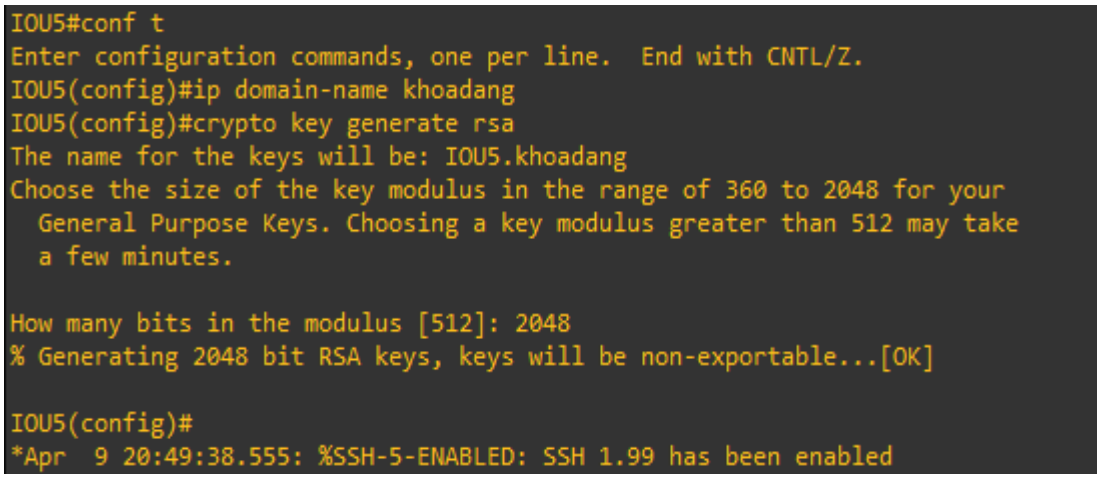


FIGURE 8. Enable SSH protocol

3.2.3 Configuring SSH on networking devices

As you already know that for the Python scripts to be able to conduct the network automation tasks by pushing the configuration files or commands to the networking devices, then facilitate the machines to run the configuration by themselves, there must be SSH connections between the control node, in this case, the scripts and the devices.

Networking devices use line VTY lines which are virtual interfaces on a router or switch that allow remote users to access the device's command-line interface (CLI) using Telnet, SSH, or other remote access protocols. VTY lines allow multiple users to access and configure a device simultaneously while maintaining security and control.

Once VTY lines are configured, remote users can access the device's CLI by connecting to the device's IP address or hostname using the specified access protocol (e.g., Telnet or SSH). The device will prompt the user for authentication credentials. If the credentials are valid, the user will be granted access to the device's CLI. (5).

To enable SSH functionality on a networking device, the first step is to access the device’s global config mode by typing in “configure terminal” from privileged config mode, shown in Figure 9 below.

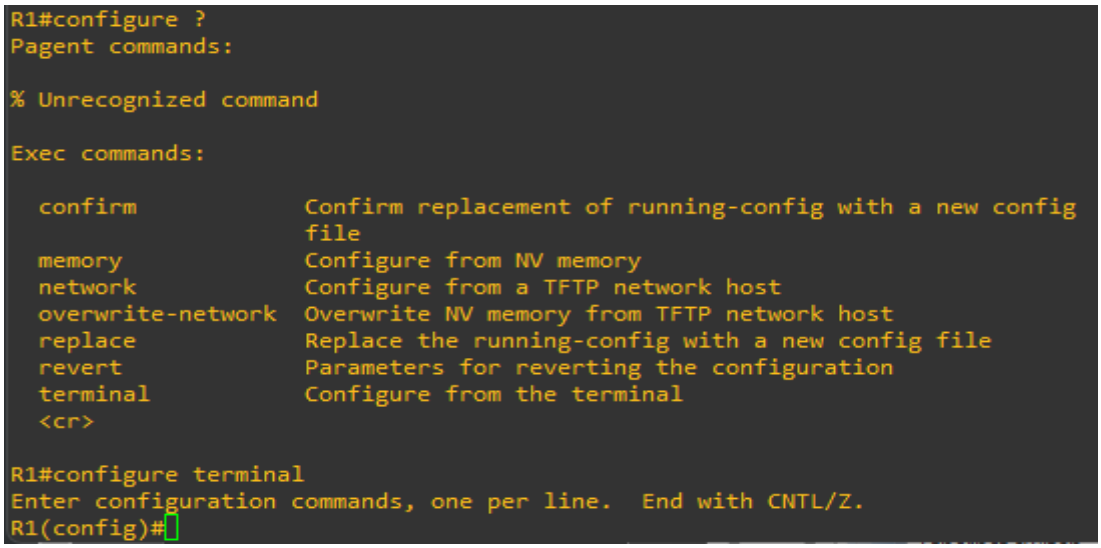


FIGURE 9. Global configuration mode

The next step is to configure the VTY lines by typing in “line vty 0 4” to enter line VTY’s config mode, 0 4 means there are 5 VTY lines from 0 to 4 that will be configured.

The next step is to configure the VTY lines to require the user to use the local login credentials to gain access to the device’s CLI by typing “login local”, the login local credentials had been pre-configured.

An ACL is an Access Control List and is needed to limit the users, only the authorized IP addresses or sub-nets will be able to gain access to the devices’ CLI “ip access-list 1 permit 10.1.1.0 0.0.0.255”, 0.0.0.255 is the wildcard mask of the /24 subnet since it requires the wildcard mask to configure an ACL. (6).

Apply the ACL to the VTY lines by “access-class number in/out”.

Setting the aging time by “exec-timeout minutes”

Using “transport input ssh” to allow only SSH connections to be enabled on VTY lines, by default, Cisco devices allow Telnet connections to VTY lines for remote management, which is a less secure protocol than SSH. By using "transport input ssh" command, the device is configured to allow only SSH connections to VTY lines, which provides enhanced security for the remote management of the device shown in Figure 10 below.

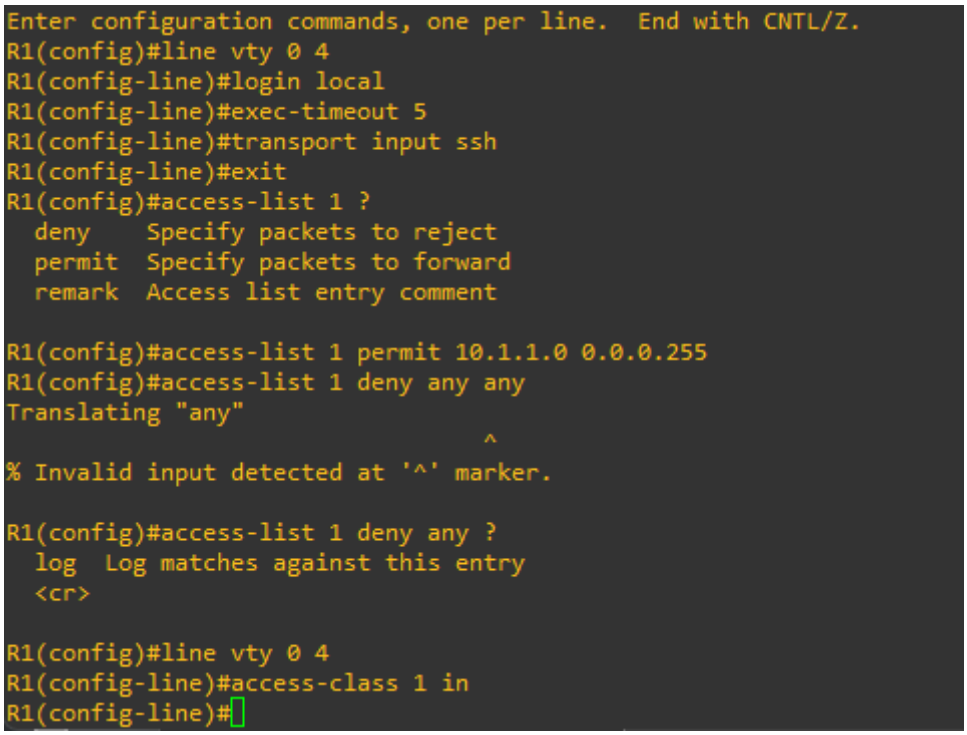


FIGURE 10. VTY line configuration

VTY lines are an essential feature of network devices as they allow remote management and configuration of the device, which is especially useful for devices that are not physically accessible or are in remote locations. However, securing VTY lines and ensuring that only authorized users have access to the device's CLI is important to prevent unauthorized access and security breaches.

Many more commands can be applied to the routers to exploit the potential of the VTY lines. However, within the scope of this thesis, those are unnecessary.

After applying all those configurations to the router, the “show running-config” is below, shown in Figure 11.

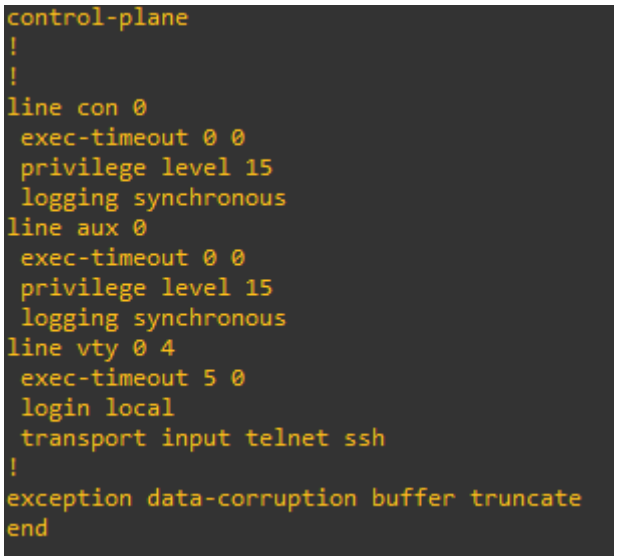


FIGURE 11. Running configuration

The running configuration on a Cisco device is the currently active configuration used to operate and function. It contains all the current settings, parameters, and commands that have been entered into the device's configuration. The running configuration is stored in the device's RAM, which is random access memory, which means it is volatile and will be lost if the device is powered off or reset.

That is why after applying all the configurations to the devices, we should execute the command “do write” and press ok. The "do write" command is used in the command-line interface (CLI) to save any changes made to the device's running configuration to non-volatile memory (NVRAM), which will make it the startup configuration that the device will use when it is powered on or rebooted.

3.3 Network topology

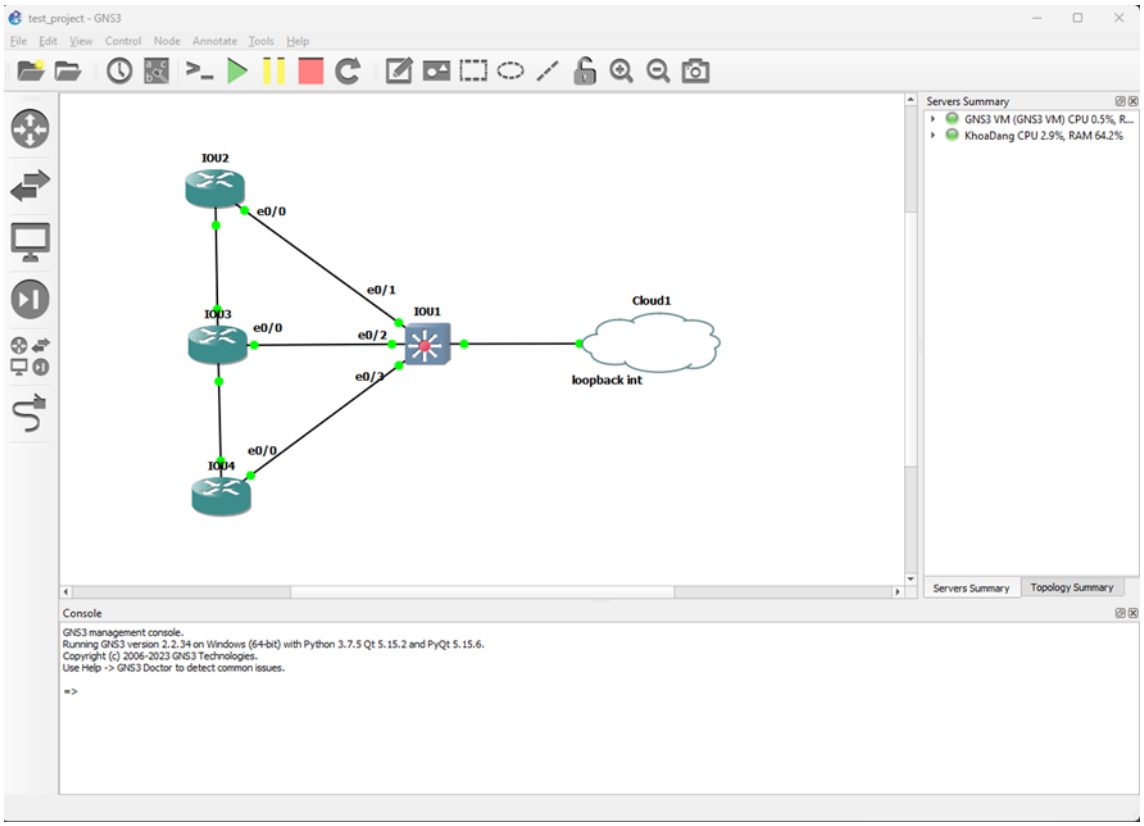


FIGURE 12. Network topology with GNS3

The network topology I created for this thesis is shown in Figure 12.

The network topology used for this thesis includes layer-3 or multilayer switches that can perform layer-3 OSI model operations, and routers each have one connection to a multilayer switch.

The layer-3 switch has a point-to-point connection to a network adapter which is created to establish the connectivity between the experiment networking system that runs on the GNS3 virtual machine and the host OS, in this case, my PC. I created a KM-test loopback interface adapter to establish the connectivity between nodes. (7).

4 DIARY ENTRIES

4.1 Week 1

In the first week, the main task was to set up the working environment, for example, downloading and installing Pycharm and GNS3 software.

Moreover, searching for the Cisco IOU images took a lot of work since those images are not free, and finding ones that work correctly seemed impossible. I overcame all those problems and finished setting up the GNS3 simulator.

Working with text files in Python is essential, and text files are a common way to store data. You can read from and write text files in Python to store and retrieve data. This is particularly useful for storing large amounts of data, such as logs or configuration files.

I practiced creating ACL configuration with text files, which will be later used to solve the task of configuring several routers using several text files.

I also spent time studying data serialization, and I mainly focused on JSON and YAML since both are useful for incoming tasks, for example, I will use Netmiko to read the configuration of the devices and retrieve and store those into JSON files.

YAML is used to create the playbook file in the network automation with Ansible. Even though I still need to finish the Ansible scripts, studying YAML did tremendously help me, and I will keep working on the Ansible tasks.

That was my first week, and everything did go well.

4.2 Week 2

This week, I started configuring the routers, for example, the subnet and the SVI for the layer three switches.

I got a big problem, the router IOU images were not working correctly, and several times stopped me from generating the RSA keys. The key is indispensable, and without the key, the device will not enable SSH protocol.

I spent hours searching and finally figured out why the key could not be generated. It was because the domain name had not been defined dynamically the same as the actual device.

It did not stop right there, it took me three more hours to find the K-9 version of Cisco IOUs, which supports the SSH v2.

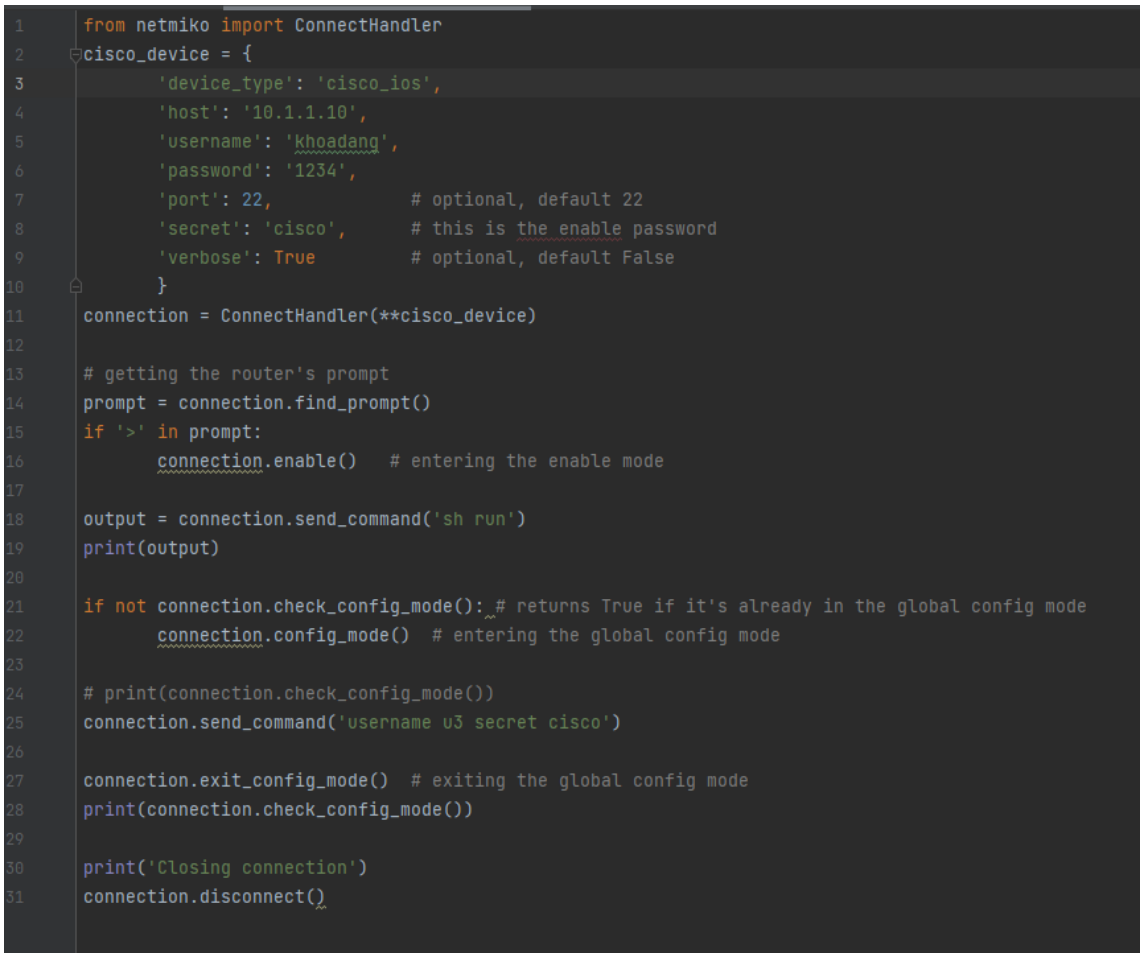
After configuring the SSH's VTY lines, I configured the port security and the trusted interfaces for the devices, for example, Dynamic Arp Inspection and DHCP snooping.

I spent two days this second-week studying programming with Python from Youtube videos and studying program flow and Python loops.

On Friday this week, I returned to the data serialization and created playbook files with YAML.

4.3 Week 3

This week, I started studying Netmiko and created a script to establish a simple SSH connection from the control node to a single router.



```python
from netmiko import ConnectHandler
cisco_device = {
        'device_type': 'cisco_ios',
        'host': '10.1.1.10',
        'username': 'khoadang',
        'password': '1234',
        'port': 22,             # optional, default 22
        'secret': 'cisco',      # this is the enable password
        'verbose': True         # optional, default False
        }
connection = ConnectHandler(**cisco_device)

# getting the router's prompt
prompt = connection.find_prompt()
if '>' in prompt:
        connection.enable()   # entering the enable mode

output = connection.send_command('sh run')
print(output)

if not connection.check_config_mode(): # returns True if it's already in the global config mode
        connection.config_mode()  # entering the global config mode

# print(connection.check_config_mode())
connection.send_command('username u3 secret cisco')

connection.exit_config_mode()  # exiting the global config mode
print(connection.check_config_mode())

print('Closing connection')
connection.disconnect()
```

FIGURE 13. Connect and show running configuration with the Netmiko script

Even though I had created the comments for the code in the script, I still decided to create a code explanation section to give the readers a better understanding. In the next section is the code explanation for Figure 13.

“connection = ConnectHandler(**cisco_device)” using Netmiko ‘s ConnectHandler method to establish an SSH connection to a Cisco network device.

“Cisco-device” is a dictionary containing the required parameters for the device you want to connect to, the ConnectHandler method will create an SSH connection to the device using the provided credentials. Once the connection is established, you can use Netmiko methods to send commands to the device, retrieve information, or configure the device. (1).

“if not connection.check_config_mode():” checks if the device is already in the configuration mode. If the device is not in configuration mode, it executes the code block following the if statement. (1).

“connection.send_command('username u3 secret cisco')” sends the username and secret password, which is formatted in MD5 to the device to enable the global config mode.

“output = connection.send_command('sh run')” sends the “sh run” command to the device to retrieve the running-config information of the device, “sh run” stands for “show running-config”, and the command should be executed in privileged exec mode.

The only problem was that the secret password to gain access to the privileged config mode was displayed in the plain text in the script, even though it is formatted in MD5 in the device’s running-config. It can be solved by using “getpass”.

I spent days this week studying the SDN concept, its control, and data plane and kept learning to program with Python, and this week was about sets, frozen sets, and dictionaries in Python.

4.4 Week 4

This week, I continued studying network automation with Netmiko and tried to fix the problem with the SSH key, when I established an SSH connection with the device, it returned “Authentication failed”, so I searched around the internet to find the way out for the error, finally Chat GPT did help me solving the error by adding “ssh_client.connect(**router, look_for_keys=False, al-low_agent=False)” to the script, and it will bypass the host key requirement stage every time the control node wants to initiate the remote connections the devices.

When everything seemed to be working perfectly, another obstacle got in the way, and I created a loopback interface adapter to create connectivity between the devices running on the virtual machine and the host OS. However, the connection was unstable, and many times got down for no apparent reason. Then, the Google search engine once again proved to be an indispensable tool for the

developers, it recommended me to uninstall the WinSCP and install WinPcap instead, and it worked.

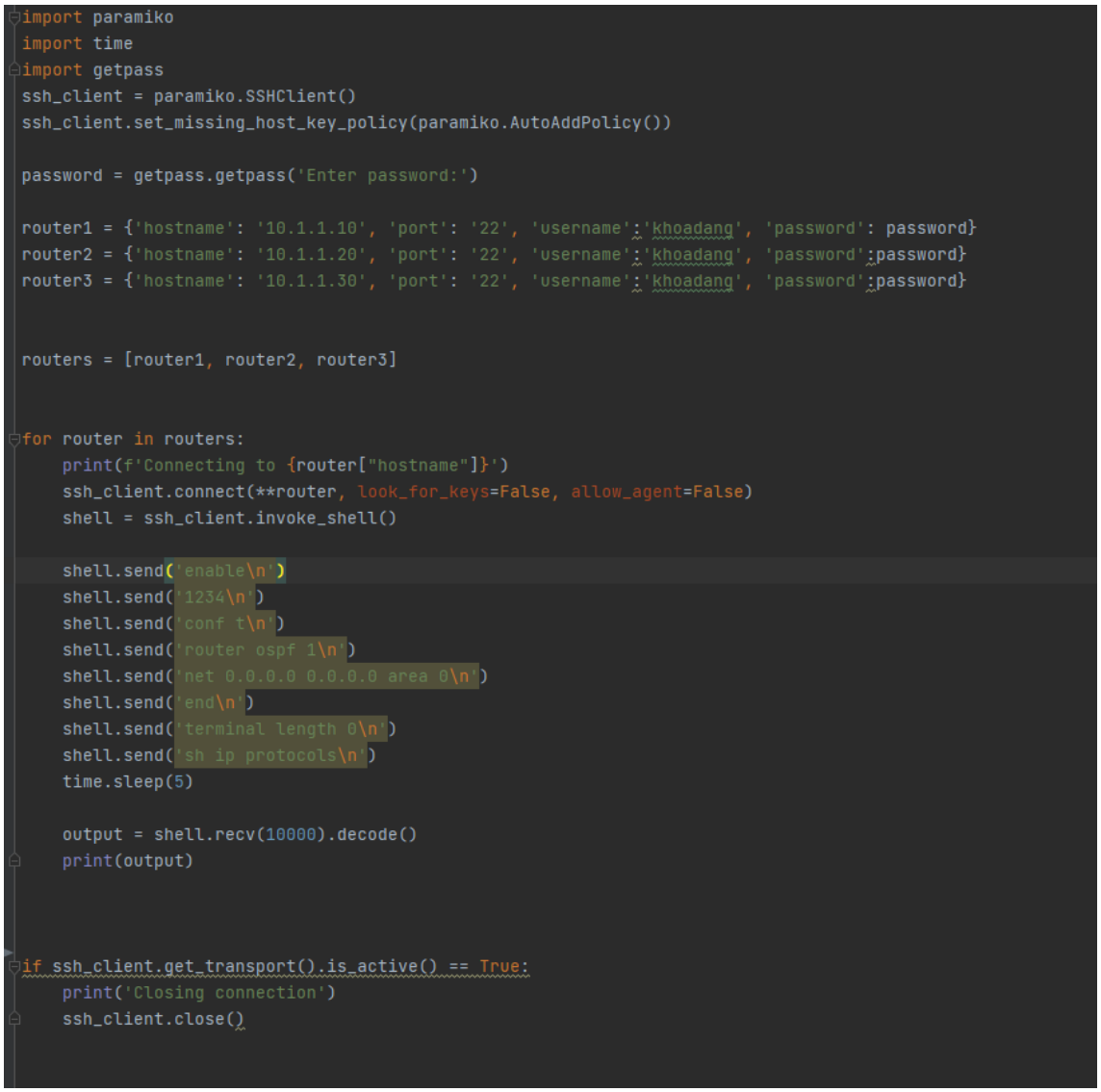
I spent the last few days of the week moving on to the Paramiko library, and I started coding a Python script using Paramiko to establish an SSH connection to a single network device and push the OSPF configuration commands to the device.

As I mentioned in week 1, I still need to implement the code so the important access credentials will not be displayed in plain text and be exploited by the attackers. This week I used “getpass” to actively hide the password and require the user to enter the password before gaining access to the device.

4.5 Week 5

This week, I kept coding the Paramiko script, studying programming with Python, and studying errors and exception handling with Python.

In the middle of the week, I finished coding and testing the script, and it did work properly.



```
import paramiko
import time
import getpass
ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

password = getpass.getpass('Enter password:')

router1 = {'hostname': '10.1.1.10', 'port': '22', 'username':'khoadang', 'password': password}
router2 = {'hostname': '10.1.1.20', 'port': '22', 'username':'khoadang', 'password':password}
router3 = {'hostname': '10.1.1.30', 'port': '22', 'username':'khoadang', 'password':password}


routers = [router1, router2, router3]


for router in routers:
    print(f'Connecting to {router["hostname"]}')
    ssh_client.connect(**router, look_for_keys=False, allow_agent=False)
    shell = ssh_client.invoke_shell()

    shell.send('enable\n')
    shell.send('1234\n')
    shell.send('conf t\n')
    shell.send('router ospf 1\n')
    shell.send('net 0.0.0.0 0.0.0.0 area 0\n')
    shell.send('end\n')
    shell.send('terminal length 0\n')
    shell.send('sh ip protocols\n')
    time.sleep(5)

    output = shell.recv(10000).decode()
    print(output)



if ssh_client.get_transport().is_active() == True:
    print('Closing connection')
    ssh_client.close()
```

FIGURE 14. Configuring OSPF several routers with Paramiko script

To give the readers a better understanding of the script and the OSPF's configuration, I placed the code explanations below for Figure 14.

"ssh_client.connect(**router, look_for_keys=False, allow_agent=False)" allows you to control the behavior of Paramiko when it comes to SSH key authentication. Sometimes, you may not want to use SSH keys or the SSH agent, so you can use these arguments to bypass these steps and use other authentication methods directly. Specifically, the look_for_keys=False argument tells Paramiko not to look for SSH keys on the local machine that could be used for authentication, while allow_agent=False tells it not to use the SSH agent.

After establishing SSH connections to the Routers, “shell = ssh_client.invoke_shell()” is used to create shell sessions over the established SSH connections, then sending the commands to the Router one by one as if the network engineer is directly interacting with the Routers’ CLI.

“enable” is used to enter the privilege-config-mode.

“Conf t” is used to enter the global-config mode.

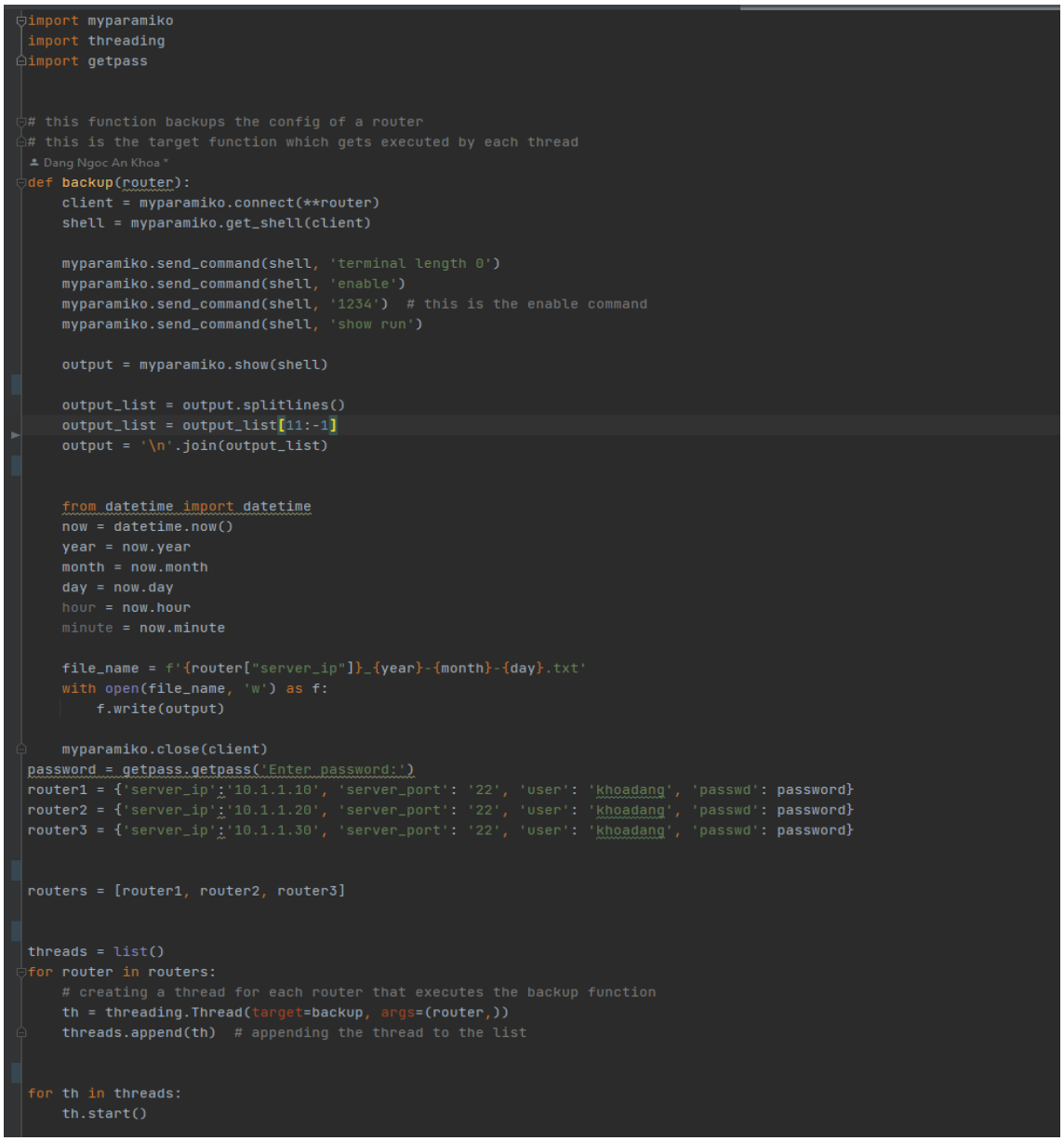
“router ospf 1” OSPF stands for Open Shortest Path First which is one of the Link-State dynamic routing protocols, and “1” is the OSPF process ID since the process ID does not have to be unique among the Routers so that we could set the process ID value the same for all the Routers. (9).

“net 0.0.0.0 0.0.0.0 area 0” to specify the network range that we want the Routers to enable OSPF on, in this case, the Router will enable OSPF on the net-work range from 0.0.0.0 to 255.255.255.255, the second 0.0.0.0 is the wildcard mask that is opposite to the netmask, area 0 is the backbone area. (9).

Since I planned to implement a function that could be used to send the configuration text files directly from the control node to the devices, I tried studying the copying files tool SCP.

4.6 Week 6

The script below is used to access the running configuration of devices and create the backup files. I already made the comments on the script in Figure 15 for the readers to easier understand.



```python
import myparamiko
import threading
import getpass


# this function backups the config of a router
# this is the target function which gets executed by each thread
 Dang Ngoc An Khoa *
def backup(router):
    client = myparamiko.connect(**router)
    shell = myparamiko.get_shell(client)

    myparamiko.send_command(shell, 'terminal length 0')
    myparamiko.send_command(shell, 'enable')
    myparamiko.send_command(shell, '1234')  # this is the enable command
    myparamiko.send_command(shell, 'show run')

    output = myparamiko.show(shell)

    output_list = output.splitlines()
    output_list = output_list[11:-1]
    output = '\n'.join(output_list)


    from datetime import datetime
    now = datetime.now()
    year = now.year
    month = now.month
    day = now.day
    hour = now.hour
    minute = now.minute

    file_name = f'{router["server_ip"]}_{year}-{month}-{day}.txt'
    with open(file_name, 'w') as f:
        f.write(output)

    myparamiko.close(client)
password = getpass.getpass('Enter password:')
router1 = {'server_ip':'10.1.1.10', 'server_port': '22', 'user': 'khoadang', 'passwd': password}
router2 = {'server_ip':'10.1.1.20', 'server_port': '22', 'user': 'khoadang', 'passwd': password}
router3 = {'server_ip':'10.1.1.30', 'server_port': '22', 'user': 'khoadang', 'passwd': password}


routers = [router1, router2, router3]


threads = list()
for router in routers:
    # creating a thread for each router that executes the backup function
    th = threading.Thread(target=backup, args=(router,))
    threads.append(th)  # appending the thread to the list


for th in threads:
    th.start()
```

FIGURE 15. Retrieve and backup running configuration with the Paramiko script

This week, I implemented another Paramiko script that could configure several devices at once since it is one of the primary purposes when doing this network automation thesis.

The script can send the commands to the networking devices to read the running configuration and then retrieve that information to the control node and display it.

Since the job of a network administrator is to keep track of the system, implementing a function that will help back up the system status and logs is crucially important (10). That is why I decided to implement one for my Paramiko script.

It did work perfectly and saved the backup files in the text file format. The date, month, and year of the time the files were recorded were used to name the files.

I also implemented the threading ability for the script, and Python has a built-in threading module that provides a simple way to implement threading. It allows you to create multiple threads that can run in parallel, enabling you to execute multiple tasks simultaneously.

I created a thread for each router that executes the backup function.

I spent the last day of the week studying network automation with serial connections, and it is an essential concept that every network engineer needs to be familiar with.

4.7 Week 7

This week, I studied the Napalm library and tried implementing the script using Napalm to conduct configuration rollback. However, I found out that the IOU images that I used in this thesis did not interact smoothly with the Napalm. Therefore, I decided to stop working with Napalm tasks temporarily and left them untouched till I find the solution to this problem.

I returned to the Netmiko library since I could still implement some excellent scripts with Netmiko's functionality, and it came out that I had made the right decision. After spending several hours digging deep into and searching around the internet, I realized I could do much more with this Netmiko.

After implementing the Netmiko script, I moved to the Asyn I/O implementation tasks since it helps the scripts run more efficiently and reduce resource usage.

I spent two days studying about Async I/O concept, its benefits, and the differences between a script running synchronously and asynchronously. Then I got to code a script to compare between asynchronous and synchronous.

After finishing the testing script and figuring out the benefits of Async I/O. I decided to dig deeper into it by implementing and testing an Asynchronous Web Scraper.

I spent the last day of the week studying JINJA2, which creates a template file to conduct network automation with Ansible.

I implemented a script using Netmiko to configure several networking devices using several configuration files and placed the script below.

Since this script's code had not been commented on, I placed the code explanation on the next page for the readers to easier understand Figure 16.

```
from netmiko import ConnectHandler



with open('devices.txt') as f:
    devices = f.read().splitlines()

device_list = list()

for ip in devices:
    cisco_device = {
            'device_type': 'cisco_ios',
            'host': ip,
            'username': 'khoadang',
            'password': '1234',
            'port': 22,
            'secret': '1234',
            'verbose': True
            }
    device_list.append(cisco_device)

# print(device_list)

for device in device_list:
    connection = ConnectHandler(**device)

    print('Entering the enable mode ...')
    connection.enable()


    file = input(f'Enter a configuration file (use a valid path) for {device["host"]}:')

    print(f'Running commands from file: {file} on device: {device["host"]}')
    output = connection.send_config_from_file(file)
    print(output)

    print(f'Closing connection to {cisco_device["host"]}')
    connection.disconnect()

    print('#' * 30)
```

FIGURE 16. Configuring multiple devices from multiple files script

“with open('devices.txt') as f: devices = f.read().splitlines()” to read the devices IP address from a file into a list (each IP on its own line)

“for ip in devices: cisco_device = { device_list.append(cisco_device)” to iterate over the list with the devices in IP addresses.

“print('Entering the enable mode ...') connection.enable()” is used to print out the device_list.

From “file = input(f'Enter a configuration file (use a valid path) for {device["host"]}:')” to the end of the line is used to prompt the user for a configuration file.

After executing the script, the user will be required to enter the configuration file for each device, and each configuration file will be executed separately one by one.

In this script, three routers will be configured from three configuration files.

I could use the “getpass” to hide the login credentials in this script.

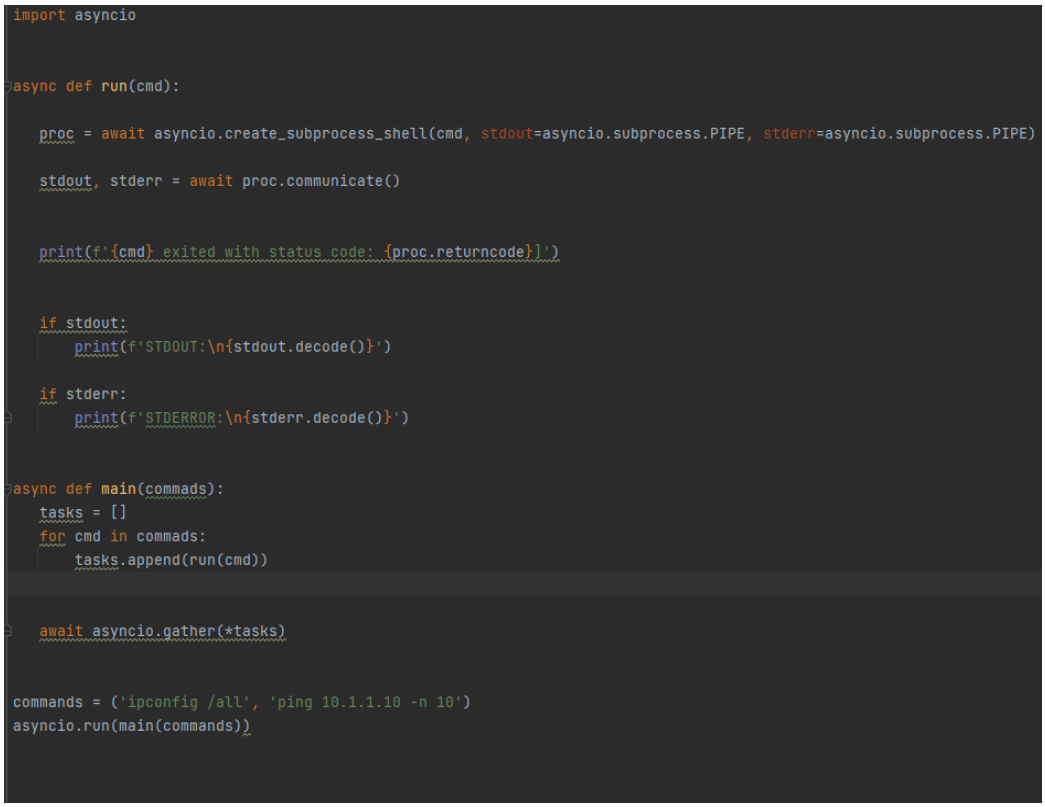
4.8 Week 8

This week, I installed a Linux Ubuntu virtual machine to help work with the network automation with Ansible tasks.

I created several Ansible playbook files and tested those.

However, after spending several days with Ansible, I realized that it is unnecessary now to focus on Ansible since Ansible is massive. It will take me several months to surf its knowledge, and my thesis deadline is about to come, so I returned to my Async I/O scripts.

I implemented an Async I/O script to run the commands Asynchronously, and I placed the script below.



```
import asyncio


async def run(cmd):

    proc = await asyncio.create_subprocess_shell(cmd, stdout=asyncio.subprocess.PIPE, stderr=asyncio.subprocess.PIPE)

    stdout, stderr = await proc.communicate()


    print(f'[{cmd} exited with status code: {proc.returncode}]')


    if stdout:
        print(f'STDOUT:\n{stdout.decode()}')

    if stderr:
        print(f'STDERROR:\n{stderr.decode()}')


async def main(commads):
    tasks = []
    for cmd in commads:
        tasks.append(run(cmd))


    await asyncio.gather(*tasks)


commands = ('ipconfig /all', 'ping 10.1.1.10 -n 10')
asyncio.run(main(commands))
```

FIGURE 17. Commands run Asynchronously script

Since this script's code had not been commented on, I decided to place the code explanation below for the readers to understand Figure 17 better.

"def sync_f()" to define a synchronous function.

"time.sleep(1)" I used this to simulate an expensive task as working with an external resource, I want it to wait for one second.

"async def async_f()" is to define a asynchronous function, I also want it to wait for one second with "await asyncio.sleep(1)".

"await asyncio.gather(*tasks)" I scheduled the coroutines to run as soon as possible by gathering the tasks.

"asyncio.run(main())" is the entrance point of any asyncio program, and "main()" is the top=level coroutine, "asyncio.run()" was introduced in Python 3.7, and calling it creates an event loop, and runs a coroutine on it for you, "run()" makes the event loop.

After executing the script, the result came out as I expected when the code which runs asynchronously returned the result faster than the synchronous one, as shown in Figure 18.

```
C:\Users\khoad\AppData\Local\Programs\Python\Python310\python.exe "C:
one one one two two two Execution time (ASYNC):1.0019962787628174


one two one two one two Execution time (SYNC):3.0008366107940674

Process finished with exit code 0
```

FIGURE 18. Asynchronous vs. Synchronous

4.9 Week 9

This week, I spent time diving deep into the Async I/O content since it could greatly help me in my future career. You can get back to Async I/O in the Python Libraries section to get more information about it.

4.10 Week 10

This week, I read the writing thesis instruction and tried to outline what I have done in the last several weeks.

After outlining all the things I have done recently, I got into writing the introduction and preface parts of the thesis. I am not going to lie; I am bad at writing, especially illustrating ideas and expressing the ideas I want to tell the reader.

I spent several days writing and fixing the Introduction part because I wanted to put less technical content in the introduction, then duplicate it in the other parts of the thesis.

I wrote network automation with Netmiko and tried to arrange the code explanation.

4.11 Week 11

This week, I wrote the Paramiko part in the definition section of the Paramiko library. I used other sources to give a better understanding to the readers. However, I fixed several pieces of the content to avoid copying all the author's content.

I also searched around to find the correct figures to describe the content.

Even though I decided temporarily to stop dealing with Napalm tasks, it is worth sharing the Napalm content. Therefore, I left a place for Napalm in this thesis.

4.12 Week 12

I finished writing my thesis and submitted a draft to Mr. Teemu, my supervisor. However, my thesis still got many things that need to be fixed.

I spent the whole week restructuring the thesis, removing, and adding some parts of it.

5 CONCLUSIONS

After two and a half months of working, studying, and coding this thesis, I am confident that I have finished most of the goals set at the beginning.

Firstly, working with Netmiko and Paramiko libraries was not kind of simple for me. It has been only four months since the first time I started studying programming with Python, and I encountered countless errors while setting up the working environment for the thesis. For example, I spent the whole day creating the network adapter with a loopback interface to establish the connectivity between the networking devices and the host OS.

In addition, looking for the correct version of Cisco IOUs is quite tricky since the IOUs are not publicly available for free, and even if you have paid for those, finding the K-9 version that is compatible with the GNS3 version I had been using for this thesis is also an obstacle which cost me much time to get through.

Learning programming with Python can be challenging for some individuals, especially those new to computer programming. While Python is a popular programming language widely used in many industries, including web development, data science, and artificial intelligence, it can be overwhelming for beginners to get started.

One of the main reasons why learning programming with Python can be complicated is that it requires a lot of time and effort to master. Python has a steep learning curve; understanding the syntax, data types, and control structures can take some time. Additionally, programming with Python requires a logical and analytical mindset involving problem-solving and critical thinking skills.

Another factor that can make learning programming with Python challenging is the need for more guidance and support. Without proper resources and guidance, learners may feel lost and frustrated, leading to a lack of motivation to continue

learning. While many online resources are available, it can take time to determine which ones are credible and provide quality content.

After all, I went through all that's obstacles to finish my thesis. I realized that overcoming those challenges gave me more joy and pushed my adrenaline to the highest level I have ever experienced.

While setting the network topology required me to study more about port security and dynamic routing protocols, which allowed me to expand my network system knowledge.

Troubleshooting the networking devices' configurations helped me review what I have learned.

Studying network automation with Python can be a game-changer for individuals in the networking field. As several organizations adopt software-defined networking (SDN) and network automation solutions, the demand for network engineers with automation skills is increasing rapidly. Python, a popular programming language with many libraries and tools, is becoming the go-to language for network automation.

By studying network automation with Python, I can improve my skills in several ways. First, it can help me automate repetitive network tasks, such as configuring devices, managing network infrastructure, and monitoring network performance. This saves time and reduces the risk of human error, leading to increased network reliability and uptime.

Second, studying network automation with Python can help me better understand network protocols and technologies. By developing scripts and automation tools, I will gain insight into how networks operate and the intricacies of various networking protocols.

Third, Python provides high flexibility and customization, allowing me to develop custom automation solutions tailored to your organization's needs. This requires a solid understanding of network infrastructure, protocols, and programming concepts such as functions, loops, and data structures.

Moreover, studying network automation with Python can help me stay up to date with the latest trends and technologies in the networking industry. As new technologies such as software-defined networking (SDN) and network function virtualization (NFV) emerge, the need for automation skills becomes even more critical. Python is an important language that can be used to develop automation solutions for both traditional and modern network infrastructures.

Finally, studying network automation with Python can lead to career growth opportunities. As organizations increasingly adopt network automation solutions, the demand for network engineers with automation skills is multiplying. By developing automation skills with Python, I can position myself as an asset to the organization and increase my chances of career advancement.

REFERENCES

1. Ktbyers, 2022. Automate Cisco switches config using Netmiko. Date of retrieval 28.01.2023. https://vinaysit.wordpress.com/2018/11/01/automate-cisco-switches-config-using-netmiko/
2. Knimi, 2022. Network Development using Python Module (Paramiko). Date of retrieval 10.02.2023. https://forum.huawei.com/enterprise/en/network-development-using-python-module-paramiko/thread/1017215-871
3. Andrea Benfatti, 2018. Async programming in Python with asyncio. Date of retrieval 17.02.2023. https://dev.to/welldone2094/async-programming-in-python-with-asyncio-12dl
4. Baeldung, 2023. The Difference Between Asynchronous and Multi-Threading. Date of retrieval 26.02.2023. https://www.baeldung.com/cs/async-vs-multi-threading
5. Jean-Francio Dal, 2022. Configure SSH on Routers and Switches. Date of retrieval 02.03.2023. https://www.cisco.com/c/en/us/support/docs/security-vpn/secure-shell-ssh/4145-ssh.html
6. Study-CCNA. Configuring Extended ACLs (Access Lists). Date of retrieval 02.03.2023. https://study-ccna.com/configuring-extended-acls/
7. HUAWEI MateStation X. Add a virtual network interface controller (Microsoft Loopback adapter). Date of retrieval 09.03.2023. https://consumer.huawei.com/en/support/content/en-us00693656/
8. Ezzeddin Abdullah. An introduction to asynchronous programming in Python with Async IO. Date of retrieval 20.03.2023. https://andela.com/insights/an-introduction-to-asynchronous-programming-in-python-with-async-io/

9. ReneMolenaar. Troubleshooting OSPF Neighbour Adjacency. Date of retrieval 25.03.2023. https://networklessons.com/ospf/troubleshooting-ospf-neighbor-adjacency
10. Gittysatyam. Automate backup with Python Script. Date of retrieval 27.03.2023. https://www.geeksforgeeks.org/automate-backup-with-python-script/
11. Coding Networks, 2020. NAPALM Network Automation Python: Working with Cisco IOS. Date of retrieval 28.03.2023. https://codingnetworks.blog/napalm-network-automation-python-working-with-cisco-ios-and-ios-xr/