University of Huddersfield

School of Computer Science

MSc in Computing

Individual Project Report
CMI3420-QGA-YEAR-2324

Title

Service Provider IP/MPLS Network Automation using Python

Student Name:Fahmida Hossain

Student ID: 2356516

Supervised by Honorable Dr. Ilias Tachmazidis

Project Declaration

I declare that this thesis has been prepared by myself and has been solely prepared for the degree of Master of Computing at the University of Huddersfield. All views are my own except where I demonstrate otherwise in this report.

I have permission to use this project in university purely for educational purposes to further improve my knowledge.

Service Provider IP/MPLS Network Automation using Python

Table of Contents

List of Figures and Table

Dedication

I would like to praise Allah who is the Almighty and the Most Merciful for His blessing given to me during my education and completing this thesis.

I would like to dedicate this project to my Parents. My late father who has been an inspiration for me throughout my academic career and my mum who has provided me priceless emotional support and raised me with patience. Also to my beloved husband, dearest one year young boy and In-laws Family as they supported me the academic year. I am also grateful to Dr. **Ilias Tachmazidis** for providing me with valuable support me throughout this semester.

Acknowledgement

Alhamdulillah, I have finally completed my MSc project. I praise the Almighty Allah (SWT), the Most Merciful, for allowing me to reach this stage. I am deeply grateful to Him for granting me the strength and perseverance to achieve this accomplishment.

Throughout this journey, I have been supported by numerous individuals, and their selfless contributions have been instrumental in turning my dream into a reality. I would like to extend my heartfelt thanks to my supervisor, whose expertise was invaluable for this topic. Your exceptional mentoring and guidance provided me with the tools and direction needed to complete this project successfully.

I would also like to express my deepest gratitude to my husband. You have been a constant source of strength and motivation throughout this entire semester. I am truly blessed to have you in my life, and it is with your unwavering support that I have been able to reach this milestone.

In addition, I would like to acknowledge Mr. Dewan Chowdhury and his wife Mrs. Naima Hassan for their significant influence in encouraging me to pursue a career in the networking field. Their advice and guidance have been pivotal in helping me navigate through this project. I am immensely thankful for their support and mentorship, which played a crucial role in my success.

Service Provider IP/MPLS Network Automation using Python

Abstract

This project is Service Provider IP/MPLS Network Automation using Python, which aims at developing the solution for the configuration and management of the large scale IP/MPLS networks. The primary goal was to eliminate human factor and increase the productivity by applying Python scripts for configuration of OSPF, BGP, MPLS protocols. The use of Netmiko library helped in minimizing the time taken to complete the process as compared with the manual configuration where the configurations were done on six routers within three minutes only. The project also depicted the ability of automation in bigger networks where it delivered standardization and reduced occurrence of mistakes. Although this phase was mainly based on CLI automation, the subsequent phase will incorporate other modern network automation protocol such as RESTCONF and NETCONF in order to achieve full automation and increased flexibility in management of dynamic and complex networks.

Key-Words: Network Automation, IP/MPLS, Python, OSPF, BGP, MPLS, Netmiko, CLI-Based Automation, Service Provider Networks, Network Configuration, Network Management, Network Efficiency, Automation Scripts.

1.0 Introduction

Network management has a major challenge especially for the service providers when it comes to managing large networks. As the need for faster and efficient transmission of data has become more critical, these networks have become more complex and therefore often needs updating and fine-tuning for better efficiency and this is well illustrated in the IP/MPLS networks which offer a convenient way of routing and transferring data through multiple paths. However, the process of managing such networks can be quite cumbersome and this is manifested by the fact that the networks usually require human intervention which results in instability of the networks and their reliability.

Reflecting on my experiences in network management, I have often witnessed how manual configuration errors can quickly lead to network downtime or unexpected traffic bottlenecks. Some of the issues that may be highlighted by such risks can be illustrated using the example of a real company, namely BT Group – one of the largest British telecommunications companies. BT for instance had a major network outage in 2016 [29] which led to the interruption of broadband and telephony services across the country. The incident which was due to a software problem and the lack of proper solutions pointed at the need for improved and more effective approaches. Therefore, BT Group started applying network automation to address the increasing network complexity and to fulfill the legal requirements and to decrease the operational costs that are linked with manual network management (see Figure 1. 1).

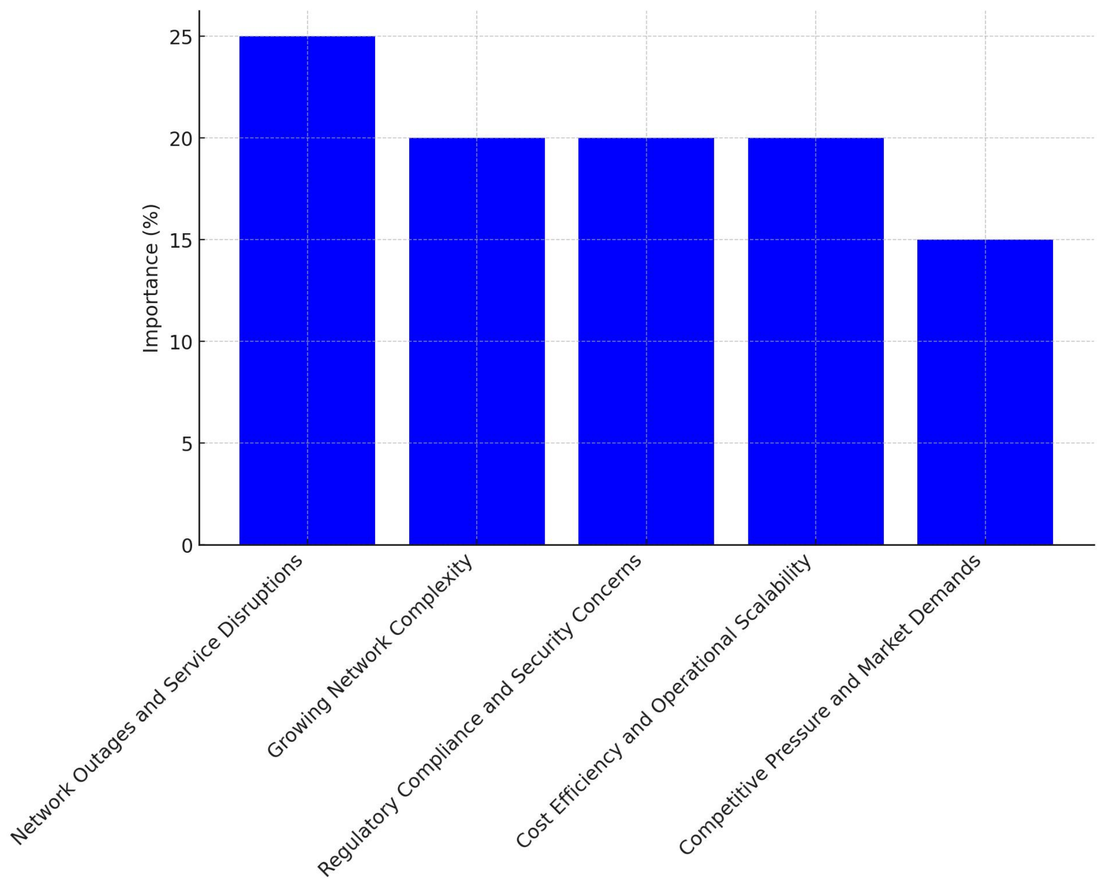


Figure 1. 1: BT Group's experience on the path to Network Automation [29]**.**

The following bar chart depicts the reasons that led to change in BT Group towards network automation. This can be seen from the chart which breaks down the different factors like network outages, complexity, regulatory, cost and competition pressures and how each of them affects the company's decision.

The beginning of the network automation for service providers at Cisco was in the early years of this decade with the launch of the Network Services Orchestrator (NSO) which offered end to end multivendor automation. Thus, the goals of the Cisco project were to enhance the network flexibility, cut down the operating expenses, and improve the service availability by integrating the automated workflows. Cisco has targeted to automate 60 percent of the data center networking configuration by 2023, which was only at 30 percent by the beginning of 2020, as well as engaging continuous integration of on-premises data center activities into CI/CD pipelines. Cisco's strategy was based on the transition towards the closed-loop, AI-based automation to address the increasing needs of efficiency and flexibility.[30] Other networking vendors such as Juniper Networks, Palo Alto Networks and Huawei also played part in the growth of network automation.

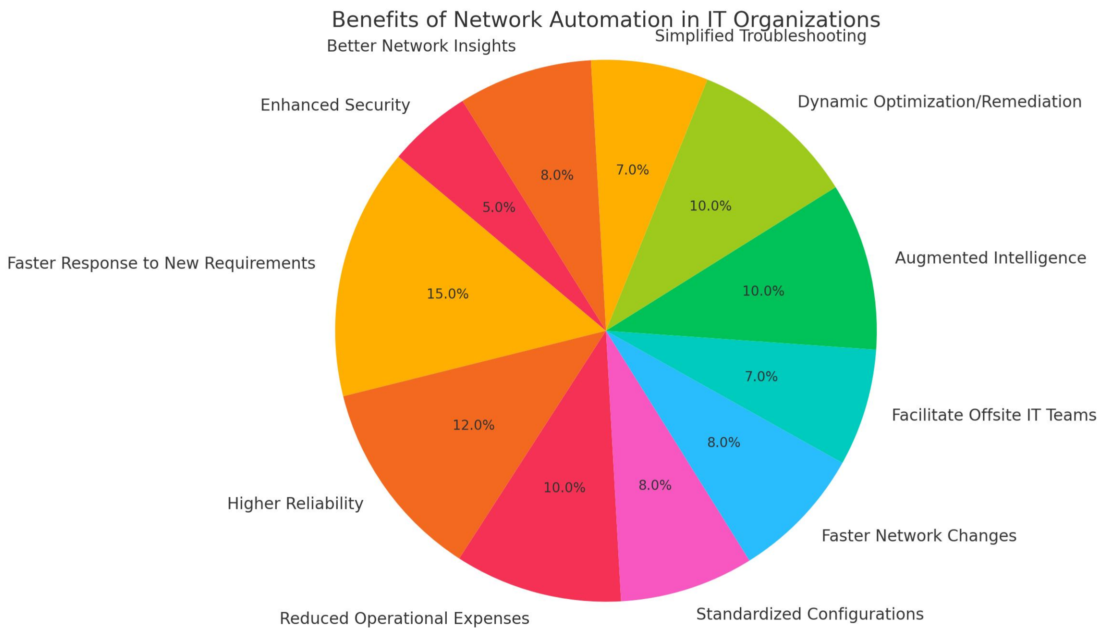


Figure 1.2: Benefits of Network Automation for IT Organisation [30]

This is the pie chart of the explanation of the advantages of network automation in the IT organizations like quick response to the new requirements, high reliability, fewer operations expenses, utilization of standard configurations, quick changes in the network, help to offsite IT teams, better intelligence, dynamic optimization/remediation, explanation of system problems easily, deep insight into the network and security, etc. , and the importance of these advantages at the same time.

Network automation today can be deployed across various platforms,[31] including:

Service Providers: In case of service providers, automation is key in order to manage large networks with millions of subscriber and devices that require tools such as Cisco's NSO, Juniper's NorthStar Controller and Huawei's Network Cloud Engine.

Enterprise Networking: Network automation also helps to cut and manage complexities of the network topology configurations in enterprises, as task including device on-boarding, software updates, and security policies, etc are also automated through the tools such as DNA Center from Cisco and Mist AI from Juniper.

Data Center Networking: Business continuity management in automated data centers is where the data center automation is designed to support business at layers of virtual infrastructures in the handling of dynamic work loads and resource optimization inherent in particular application features such as Cisco's ACI and VMware's NSX.

Relatively, the early beginning of the automation of networks can be attributed to Cisco starting around the early period of the 2010s Before the establishment of other players in the market selling their solutions such as Juniper, Huawei and Palo Alto Networks among others. Nowadays, it is accepted and organizations of various fields employ network automation in the hope for better operational enhancement, reduction of cost and certainly overall better security in the networks. Based on these industry changes and realizing that it was becoming increasingly difficult to manage large scale IP/MPLS networks services through manual techniques as highlighted from my experience, I pursued this project because it aims to automate networks using Python, such a network environment is typical in a service provider setting, where OSPF, BGP, MPLS among other important protocols are paramount in supporting service delivery.

1. 1 Overview of the Main aim of the Project

This work namely encompasses the deployment of an intelligent and autonomous tool for the configuration and management of IP/MPLS network in order to minimize the occurrence of error and time wastage as well as to improve the availability and dependability of the network. For these solutions, the project shall employ Python in order to control relevant fundamental protocols in the network including OSPF, BGP and MPLS. In doing so, the project shall seek to show how network automation can be utilized in the reduction of configuration time while at the same time trying to ensure that the devices within service provider environment are standardized.

1.2 Project Aims

To achieve this goal, I have set several specific objectives:In a bid to meet this goal, the following objectives are hereby specific and achievable;

1. Write Python application to configure OSPF, BGP and MPLS in the simulated environment of IP/MPLS networks.

2. Prove that configuration management of the network can be done far much faster in this method compared to the traditional techniques.

3. Explain how implementing of automation aids in avoiding of configuration mistakes that may occur when configuring various devices.

4. As far as network operations are concerned, comparative evaluation of the automation impact on network performance and reliability must reflect parameters such as convergence times and packet loss.

5. Test the automation approach thoroughly so as to determine their effectiveness.

1.3 Novelty

That is why this project is special and differs from the previous ones; as it covers the whole network automation with Python starting from the network box configuration to the dynamic routing protocols management in Python. In contrast to many of the current approaches, which are mostly based on a set of scripts for configuring one protocol or operating within the frame of a single network setup, this project encompasses several tools and libraries to solve the tasks that are actual for service providers. For instance, with a sample network, one can simulate various hypothetical scenarios and at the same time be in a position to appreciate some of the real life applications of concepts in Python based network automation.

1.4 Legal and ethical factors

It can also be said that the present project does not contain any of the data related to the real companies/ organisations and therefore; it does not contain any violations of the legal and ethical rules such as the GDPR. All data in the project is created either dynamically in the virtual machines via internal cloud NAT or is set and configured to have a static value without any input from a human. The tools that are used, like network emulators and ‘Microsoft’s Visual Studio Code’ to name a few, are either Open-source tools or free tools hence reducing the risk involved. With these standards, the project also entails that there are no risks that are paraded with data privacy or security breaks.

1.5 Key Research Questions

To guide my research, I'm asking several key questions to myself:To guide my research, I'm asking several key questions to myself:

1. What needs to be done via Python to be able to claim to autoconfigure an IP/MPLS network?
2. What benefits can the use of Python bring to network automation; particularly in a context of minimizing error rate and maximizing effectiveness?
3. In what ways does the automation of an MPLS/ IP network affect the performance as well as its reliability?
4. Network automation is a great way to go about implementing automation of a network but what problems might possibly arise when undertaking it? How can these problems be solved?

1.6 Challenges and Risks

It is however important to also note that there has always been some challenges or risks associated with network automation especially when it comes to implementation. However there are few issues with it, one of which is that there is no regularity to address the support for the automation process based on the network device that is being used. Indeed, the automation scripts may also fail because of bugs or other network conditions that were not specifically foreseen, this may occasion network fault or error. The other significant risk relates to the scripts, which are used to control the network configurations; if these scripts are unchecked they may pose a serious security insecurity to the network. To minimize these risks, I will start running tests of the automation scripts on the staged environment; this will help test its operation in different conditions. Also, there will be excellent error-handling mechanisms whereby any issues that may arise concerning the network will be well identified and solved.

1.7 Security Considerations

This makes security a top notch consideration in the context of network automation. In this project, I've made sure that the scripts are designed to handle communications securely using methods like SSH, and that they protect sensitive data, such as passwords, from exposure. I am also doing a lot of things to minimize the possibility of the scripts opening up what I would like to refer to as back doors and subsequent unauthorized access to the network.

1.8 Commercial and Economic Benefits

From an organisation's perspective, achieving automation in network setups has some advantages especially in commercial business. This means that by sparing the amount of time that is usually taken in manual most configurations, the service providers are in a position to reduce the costs incurred in operations and control instances where customers experience lengthy downtimes due to wrong settings made by individuals. These create an improvement in service providers, thus serving customers to ensure market advantage is achieved. Furthermore, through automation, it is easier for the service providers to expand their networks and facilitate growth of their business without having to seek for more resources.

In particular, this project seeks to argue that it is possible to use Python in a way that helps solve the problem of managing networks in a more effective manner. Therefore when explaining the concept of automation, I should point out such positive impacts that may be experienced by the service providers including reduced number of errors, more time gained, and improved performance of the network in this case. I think the strategy that I chose would be quite insightful in showing how contemporary technologies can be applied in networking.

1.9 Structure of the Report

Chapter 2: Literature Review
Recapitulates concepts of IP/MPLS and network automation in order to set the theoretical context which follow.

Chapter 3: Methodology
Describes how the automation environment and the python script files had been set up.

Chapter 4: Results and Findings
Provides the results and the evaluation of the automated network configurations.

Chapter 5: Future Work and Conclusion
This paper presents the results of the conducted analysis and discusses the further prospects for automating networks.

Chapter 6: Appendices
Includes additional information, for example, code snippets, and test outcome.

2.0 Literature Review

The paper in [1] describes a platform for automating the simulation of MPLS VPNs using OPNET Modeler in order to address issues such as intricate scenario configurations, incorrect configurations, lengthy simulation times and other chores. As the proposed solution, there is the three-tiered structure that aims at the control, management, and the layer containing the data for further creation of the scenarios. The platform able to accommodating of many parameter changes (for example, the number of sites, clients, routing protocols) and include web-based control interfaces Jrn. This was achieved in less than a minute and the findings revealed that the platform has the capability of producing up to 800 scenarios which is way better than the OPNET manual means and other existing OPNET tools. Saves time in setting up the simulation and also light reduces human interferences in the simulation outcomes. For the further deployment of the above proposed methods, the authors outlined future developments such as QoS and security. This research is related to my work as it shows the possibility of using automation in networks, which may help to improve time and error factors, both of which are the goals of the Python automation solution in the IP/MPLS networks.

In the paper [2] the author investigates the possibility of using virtualization on the real IP/MPLS core network developed for the network with many constraints and focusing on latency. Control and data plane that is also known as SDN has been depicted as virtualization that simplifies networks' scalability, security and dynamic adaptation. This approach is intended to solve issues that are inherent to the IP/MPLS networks, including high latency and inept utilization of resource. The study also shows how virtualization can trim down the latency on a telematics network of the Autonomous Port of Kribi in Cameroon from a high of above a 100ms to sub-1ms. The authors underline that it means that the code of the network must be modified continuously in order to retain and enhance the network performance. They also advise the incorporation of other layers of security and backup in order to counter threats such as DDoS. This paper I find useful for my project because it provides information on SDN implementation for improving IP/MPLS networks, specifically in terms of removing latency and improving the networks' performance, which is aimed at accomplishing my Python-based automation project.

The paper in [3] address the problem of how to improve BGP convergence and proximity to full security of prefixes in IP/MPLS networks. It introduces the BGP PIC to prescriptively cut down the time for the BGP route convergence with the ultimate goal of strengthening highly reliable networks against possible faults. This work also discusses a proactive and reactive approach in containing secured BGP prefixes through Resource Public Key Infrastructure (RPKI) and the Artemis BGP monitoring tool. The authors explain that it is possible to implement these technologies to greatly reduce convergence delays and prevent such attacks as prefix hijacking Moreover it highlights future usage of BGPsec as a more secure version of BGP since it is a trust based routing protocol. This research is related to my work since it provides some of the measures that may be implemented towards enhancing the BGP stability and security within the IP/MPLS networks that can complement my idea of enhancing the reliability and security of networks through the use of automation technologies.

In the paper [4], the author pays much attention to the often used DevOps tools, namely, GitLab and Ansible for MPLS service automation in the context of SDN. The proposed system incorporates the IM-PMS-SDN that utilizes the management plane southbound interface MPSI in end-to-end MPLS provisioning with possibility for preliminary service tests, label assignment, and validity examination. The authors are able to show an example where GitLab CI/CD pipelines and Ansible scripts can deploy network infrastructure as well as configure, test services automatically rather than taking days. The study aims at showing the benefits of integrating SDN controllers with DevOps tools to fight human-factor-induced errors, minimize service outages and improve the flexibility of managing the networks. This aligns with the goals of my project because it demonstrates how automation can be utilised so as to increase the efficiency in matters concerning the network. This paper is of significance with my project as it discusses the integration of CI/CD pipelines for network automation hence related to my automation of IP/MPLS network using the Python language.

The paper in [5], provides the details of the next generation network and the association of cloud computing is mentioned there. These include virtualisation, software defined networking (SDN), and network function virtualisation (NFV) to make a network elastically adaptable, malleable, and programmable The authors proceed to explain how these technologies help attain the optimal utilisation of available network resources and also the mechanism for generalising service delivery. The paper also describes the process of creating an SDN testbed using GNS3 and Docker to illustrate the application of network virtualization, SDN and NNFV as improved system with low operational costs and efficient management of migration to the 5G network. This paper is useful to my study because it establishes the manner in which virtualization and SDN mechanisms operate in network management and ways in which similar mechanisms may be incorporated in mypython-based automation framework for IP/MPLS networks.

The paper in [6], the author mainly describes the MPLS networks, and also gives new progress in TE and DiffServ, MPLS-TP. Some of the benefits of MPLS for the network usage, according to the authors, are; efficient use of bandwidth, lesser delay time, and better QoS control. It also examines challenges which organizations face when deploying

MPLS network such as security and flexibility when migrating to MPLS- TP. This work is centered on the fact that there will always be demand for new technologies due to facts such as the rising bandwidth demand and the quest to achieve optimum networking. From the given review, MPLS is still viewed as a possibly future network solution in which research should be continuously carried out to meet issues to do with security and network dynamism. This paper is useful in my project because it provides information regarding the current trends and issues concerning MPLS networks, which may be handy in an endeavour to compare the real-life impacts of using Python in automating MPLS IP networks.

The paper in [7] provides an approach towards the automatic configuration management of service for different IP/MPLS networks based on user goals. Namely, it employs automated planning paradigms that enable it to create sequences of configurations that transform existing network services from one state to another state that we define as the 'target state'. The approach is used to reduce the complexity and errors related to manual configuration processes through the use of YANG and YAML modes in the definition of services models and configurations. To illustrate the efficacy of this method, the study includes a prototype that verifies numerous service configuration scenarios, including the feasibility of automating configurations and decreasing the time to deployment in this simulation exercise. Therefore, the paper concludes about the handling of versatile network services with the help of the proposed approach on the other hand the more advanced application of the proposed framework of service creation and deployment is possible in the respect of supporting more complex requirement. This research is applicable to my project for the given study explores on auto-mated planning and declarative modeling for IP/MPLS networks which is similar to what I aim at implementing using Python for similar tasks.

The paper in [8] discusses the utilization of the "hardened pipes" in the IP/MPLS networks: the concept that was described in the IETF RFC 7625 providing end-to-end bandwidth guarantee, low delay, and high reliability of the transmission for the critical applications. The authors review the utilization of hard pipes with normal IP/MPLS traffic while providing logical partition and prioritized treatment of such sensitive services as TDM-like leased commercial lines. They are bidirectional LSP for traffic engineering in addition to QoS because this research also discusses service reliability as well as timing synchronization for efficient traffic flow. Thus, the paper suggests further investigation on bandwidth utilization and obtaining possible service level agreements (SLAs) focus on critical user applications, there is a clear need for better traffic engineering techniques for failure-sensitive services. The following research is related to my project since it covers issues on how to maintain networks performance and reliability in IP/MPLS, which will go in line with my goal of using Python in automating the configuration of IP/MPLS network.

The paper in [9] focuses on the use of private-cloud based Python based network automation programs for managing data center networks and its development. What is also measured in the study is how these automation programs perform when executed on the cloud infrastructure as opposed to a small physical server. Performance criteria used are time used in carrying out network related activities like reading and writing configurations and the time taken to recover from network failure There is a positive shift that shows that cloud implementation is even more reliable and constant, with a 22% improvement in execution times for some operations under the worst case scenario. Further, the cloud setup gives improved recovery time because AZs are designed with high availability in mind which reduce the recovery time greatly. This work is relevant to my project as it show the benefits of implementing cloud based infrastructure for automation of network by focusing on the performance and reliability which is befitting for the proposed automation of the IP/MPLS network with Python.

The paper in [10] the author sought to compare the efficiency of network automation particularly through scripting in Python as against manual configuration on Cisco devices. For this study, 36 Cisco devices were configured from scratch using manual settings as well as python scripts, in a simulated lab environment. Concerning the outcome, it was found that a time of 5797 seconds was taken while using the manual approach, while only 120 seconds was taken while using the automation implying 99% time savings and zero error while the manual method had two errors. From the findings, it is clear that Python is effective in automating networks' functioning, as it increases the speed, efficiency and productivity of the networks system. The research also a note that automation is cheaper in terms of reducing of human interferences and operational cost other than other methods suitable for network management responsibilities. As the name suggests, this research is related to my project as it discusses about the benefits of Python based automation of IP/MPLS networks which is in line with my objective to manage the network through scripting.
The paper in [11] outlines λBGP a modular and extensible form of BGP that is implemented in Haskell with the aim of improving programmability in BGP routing. The authors also describe the disadvantages of the current BGP settings that can be barely switched and adapted to the new network environment. λBGP provides a reactive control API which supports route processing policies to be defined by the network managers in terms of a high order programming language. This approach enhances the scalability as well as confgienability of BGP speaks whilst at the same time enduring compliance to existing BGP norms. The evaluation shows that λBGP can reach the similar performance to original BGP deployments while offering the additional control over routes to be advertised or chosen. According to the authors, the proposed λBGP can become the basis for subsequent research aimed at designing programmable BGP system in the future, which should facilitate the more efficient and less reactive management of the network. This research is close to my project as it sets out different strategies for improving the programmability of BGP that would fit my Python automation system for dynamic MPLS/IP policies.
The paper in [12], aims at developing an application for automated network management tasks such as configuration backup, restore and OSPF enable on different device using the python programming language. With an aim to work well in multi-vendor networks, the application makes use of libraries such as Paramiko, Netmiko and NAPALM and can connect through secure method like SSH. Demonstrate how thread is used through examples to speed up the time and reduce the time taken by the program to run together with the use of GUI to make the software more interactive. The results show that flexibility of the Python language and the use of different libraries help to perform complicated operations much quicker and with less mistakes in comparison with manual types of configurations. The paper also outlines future developments, which include; recognition of multiple vendors' devices and improved input validation. This paper is related to my project as it discusses methods and tools used in network configuration automation, which is in line with my intentions of improving the MPLS/IP networks management through Python scripting.
The paper in [13], the authors described Python scripting to perform scripts for configuration changes on parallel third party devices-Cisco, Juniper and Huawei. To compare the effectiveness of manual and automated configurations the authors of the study use the PNETLab emulator assuming the network with 30 multi-vendor devices. The outcome of the analysis reveals that the time taken to configure has been dramatically brought down through automation and errors have been removed unlike when it was done

manually where three mistakes were noted. This analysis shows how effective Python programming language is and how threading allows setting several gadgets at the same time to achieve higher velocity. Therefore, the study brings out the fact that automation is not only time effective but also efficient in ensuring that the management of networks is effectively conducted. This paper is useful in my work description, as it establishes how scripting in Python can help in the automation of multi-vendor network settings – a goal that relates to my focus of using Python for the improvement of IP/MPLS network administrative assignments.

The paper in [14] gives elaborate understanding of network management using the programming language Python where it configures and gains control on routers and switches, in addition to, monitor and obtain data on the network performance. It shows how Python libraries such as Netmiko, Paramiko, as well as Napalm can be used to communicate with different network devices from several vendors in a secure manner. The study also encourages the use of Integrated Development Environment (IDEs) such as PyCharm to Write and Debug scripts. The paper identifies the practical example of how the task automation looks like, like for instance allowing protocols SSH, configuration of IPs, and management of network devices with the integrated Python code along with the network simulation software known as GNS3. As such, the author opines that Python-based network automation makes complex activities easier, minimises human intervened errors, and increases operational performance, making it a crucial skill area for network engineers. My project objectives will be served by this research since it presents various techniques and approaches in Python automation especially as it applies to typical IP/MPLS networks.

The paper in [15] also brings a contribution to the automation of the management of the CE routers in MPLS-VPN networks with the help of the route leaking techniques. This research also proves that Configuration Made Easy through CLI terminals are slow, time-consuming, and involving a lot of errors especially in large networks. To address these challenges the authors suggest adopting RL to ease routing table management and otherwise interact with CE routers via a Web interface. The study also reveals that the proposed automation program of this research developed using Python libraries, NAPALM and Paramiko within the Django framework reduces configuration time by about 40 sec for each device compared to manual CLI methods. At the end the researcher established that utilizing MPLS Route leaking alongside automation tools is a feasible solution in managing large and complicated config values in MPLS-VPN contexts. This research is useful for my project since it provides working knowledge in the automation of managing CE routers using Python, an area I wish to improve on in IP/MPLS networks.

The paper in [16], the author describes the automation in respect of the conventional non SDN network, the focus is made on the prospect and challenge of the given approach. The authors of the paper in question expand on such matters as the rigid hierarchical network and, as a matter, manual configuration in particular. This survey establishes how network automation has become more prevalent because of the devices and the proliferation of network devices; which makes the operation and human errors cost go up even higher. This paper does an analysis of different strategies and tools that are used in network automation that include the python libraries (Paramiko, Netmiko), and models like REST API, Ansible among others as taking lesser time, and effort and having reduced errors. Automating the networks' management, as concluded by the authors, enhances efficiency and scalability of the networks implying that the traditional networks are likely to derive benefits from the incorporation of the automated solutions. This work is pertinently related to my study as it outlines the current position and future possibility of the network automation which supports the assertion that Python-based automation tools applicable for IP/MPLS network management are crucial.

The paper in [17] propose the Software Defined Network Service Deployment and Control (SDNSDC) to realize the deployment and control of the services of networks in NFV areas. The authors weigh on the challenges that prevail in the configuration of network services whereby the services are usually configured manually, and this usually comes with several complications like mistakes and it is time consuming. The SDNSDC mechanism is based on the provision of the formal description files in order to automatically DU, manage the operational behavior of VNFs. A prototype system incorporating Linux containers (LXC) has been chosen for its successful implementation to validate the feasibility of the proposed solution which necessarily reveals a significant improvement in speed and correctness relative to the previous approaches. However, the study suggests its potential in other more general use cases involving dynamic large scale NFV, which require quick instantiation and versatility. This research is relevant to my project in that it gives information regarding the application of automated control mechanisms for management of network services which can be integrated when automating IP/MPLS networks utilizing Python.
The paper in [18] puts forward a centralized model driven traceroute for TCP/IP routers based on NETCONF API and YANG data model. This approach is a step forward from the SNMP based network management approaches because it allows the traceroute utility to be invoked remotely hence doing away with having to go to each router terminal individually. The tool being proposed here, named as “Trace,” aims at using an NM station to send traceroute commands, trace BGP AS paths, measure RTT, and generally collect responses in a centralized manner. The study shows that when using YANG-based data models with NETCONF API, traceroute commands are done with the same syntax across TCP/IP devices irrespective of the operating systems. This also enhances the manner in which troubleshooting is carried out and is compatible with most YANG based network management schemes. This research is related to my project since it discusses the utilization of programable APIs to improve the formation of the network, which is essentially the goal of my project that involves using Python to automate MPLS/ IP Network formation.
The paper in [19] examined the properties of developer contributions to the FOSS projects that fall under the domain of network automation. It concentrates on the first-time contributors who usually do not have standard software development experience. From 81 projects of GitHub, including 71 projects of network automation and 10 most popular projects of other domains, the authors investigate pull requests and issues to identify the differences between the first-time contributions and contributions from active and regular users. The study shows that the pull requests were more accepted in network automation projects (88% of pull requests were merged in network automation projects out of 357 pull requests as compared to 72% of pull requests of the top ten merged projects) but pull request latency was also higher and there were larger and complex contributions. In this regard, the study noticed that whereas network automation projects seem to be more inclusive, they pose different challenges compared to conventional LAMPS in terms of quality assurance and collaboration. This paper is useful to my study because it offers concepts regarding the FOSS contributions in network automation: onboarding process and community participation, both of which are connected to the collaborative elements of my Python-based automation toolbox for IP/MPLS networks.
The paper in [20] the author describes the advantages and issues arising from automating the network systems configuration management using Python and the Netmiko library. It covers the creation of scripts to perform task concerning the network state or network configuration, as well as checking on the general state of the network. In the study, the automated configuration processes are compared with the manual ones utilizing Cisco devices in Packet Tracer and GNS3 with considerable time differences and low errors ratio. The findings of the study are that with the help of Python, it is possible to automate the

process with the emphasis on Netmiko that helps to avoid numerous mistakes and expenses compared to manual configuration. It also recommends that future improvements can be made with advanced libraries and artificial intelligence for more automation coverage as well as network control. This paper is relevant to my project as it elaborates on the use of Python scripting and particular libraries for automating the existing network functions/ tasks; which is in line with this project's objective of automating IP/MPLS network management.
The paper in [21] describes a work that aims at creating a Python library named PyMIP that acts as an API to interact with Visual TeMIP C++ API that is utilized in management of alarm systems in telecom networks. The main objective was aimed at providing application framework and modules for TeMIPs development with the use of Python scripting language which would help to reduce the time and efforts needed for developing applications and modules. The authors emphasize that doing away with compilation times and using Python as a language is advantageous, which is especially important in an iterative process used in the network management. It also presents a development of a lightweight web server and client for interacting with PyMIP library to demonstrate how python programming language can be used in development of internet base network management tools. 21. This research has shown it is true that Python provides another cogent solution to the traditional utilization of C++ in addressing networking protocols and the Python programming language is flexible, takes less time to develop and therefore easy as an option to C++. This study is quite useful to my work as it shows how Python programming language can be used in network management which is important given my aim to improve IP/MPLS network automation with Python scripting.
The paper in [22] discusses about efficient networking automations through automation based on the application stack which involves Jinja template engine, Nornir inventory management, and FastAPI. It drew the attention toward increasing role of network administrators as technologies are continually advancing and demanding strong automated solutions for maintaining healthy and secure networks. The authors employ their own created questionnaires and interviews with the administrators of the computer network, in order to gain both quantitative and qualitative data, regarding their attitude to the network automation. Literature review indicates that more automation within a network enhances its efficiency as well as reducing chances of risks and mistakes done by people. The Jinja template engine makes simple work of Network management settings, while Nornir is a more all-encompassing way of automating networks. To solve the problem, the study suggests companies should incorporate training to apply automation technology into their priorities, as well as developing best practices to follow. This paper is pertinent to my project since it gives outcomes of implementing Python implemented structures like Nornir and Jinja for network automation, specifically for my endeavor to minimize the management of IP/MPLS networks.
The paper in [23] offers a paper based on the authors' practice of migrating an orchestrated firewall service to a multi-tenant datacentre with help of SDN. The project involves using the POX SDN controller along with Python API to deploy firewalls and also to control the traffic flow of different tenants in a data center environment. The implementation employs the use of Mininet to create the network environment as well as illustrate the installation of firewall rules through Pyton based automation in accordance with specific set parameters. Exploring features of SDN for the data centre automation, the study finds that it is possible to get even more procurement, more centralized management and dynamic policy for more security. On that premise, the study establishes a strong argument that implementing SDN controllers to automate the firewall services is efficient in managing the current data centers since it reduces errors associated with manual configuration. This work is useful for my study as it shows how Python scripting

and SDN can be adopted to perform the management of networks, which are similar to my goal of automating IP/MPLS networks for optimal configuration.
In the paper [24] the author discusses the design and construction of an MPLS based telecommunication network for today's services such as gaming, video/audio streaming, and enterprise access. It also leads the reader through MPLS, VPLS, DWDM, BGP which are technologies to use in constructing a strong and efficient network to meet the customers' demand of high speed, low latency and reliability. This paper focuses on how to come up with a hierarchical network model of a core distribution and access layer and this by effectively using MPLS to the traffic. According to the research, MPLS poses the advantages of lower delays, proper utilization of bandwidth and traffic engineering choices. This paper also take a sample of some of the challenges that needs to be addressed in MPLS before it can deliver effectively which includes; network diplomacy and security, some protocols such as BGP and OSPF are recommended to stabilize the MPLS network. This research is germane to my project as it offers the broad outlook of MPLS deployment trends, technologies, and practices that are in consonance with my objective of automating IP/MPLS network configurations using Python.
The paper in [25] presents the Python-based Active Router (PyBAR) system with a lightweight active networking architecture that is intended to facilitate network management and control operations. The system also utilizes Python in order to simple and fast create new network services and easily expand existing ones by adding functions right into network devices. Other related tasks like quality of service management, active monitoring as well as automatic tunnel establishment are also supported by the PyBAR platform with the flexibility of Python scripts and efficient native C modules. The study demonstrates how the configuration and control could be implemented with Python which would be more effective than the current approach. The architecture of the PyBAR system means platform independence, direct interaction with network components, perfect control of packet processing, and an opportunity for secure administration. This research is useful to my work since it presents active network management using python scripting hence fulfilling my aim of automation in ip/mlps network for enhancement of efficiency and control.

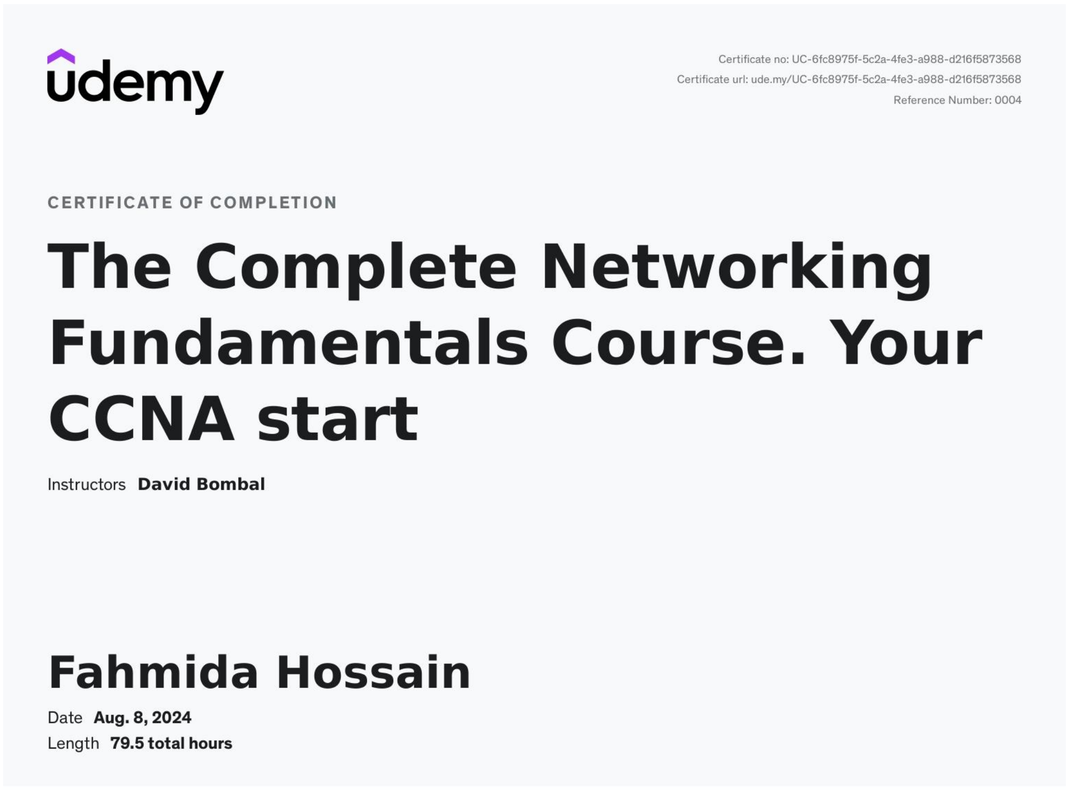
2. 1 Critical Analysis

From all the papers in question, it is possible to identify the general appreciation of the possibility of employing Python and similar technological platforms in automating the network management processes that allow for cutting the time spent on configurations, making fewer mistakes because of relying on the computer program, as well as improving the effectiveness of the firm. A number of case studies provide qualitative gains, including various aspects where it proves to be much faster, and where the overall network performance is boosted dramatically. Nevertheless, there are certain drawbacks: lack of balance in the security aspect; the issue with the approach of integrating individual tools and technologies; basic issues with scaling up of solutions in massive and diverse networks. However, there are negative effects inherent in the use of utilities based on Python in the application of the solution to complex multivendor networks, as well as issues with further support for open source tools and their security. Future work can then aim at overcoming these limitations and come up with better and fortified architectures, other features such as artificial intelligence for predictive management and guarantee compatibility across various network devices. This critical stance assists in positioning my project as an effort in an attempt to narrow these gaps especially where Python is used to attain extensive IP/MPLS network automation.

3. 0. 0 Methodology

The process of work to be explained in this section concerns programming-related tasks, tools and techniques employed in this project, which include data processing, network topology configurations, process automation, secure shell connection and source code editor (Visual Studio Code). This project entails configuring a network emulation scenario through the EVE-NG Community Edition (Version 6. 2. 0-4) that requires a virtual machine from VMware Workstation Pro 17. The specifics of the software tools used in the project as well as configurations necessary to work on the project are also described here.

For further reinforcement on networking, I took the "The Complete Networking Fundamentals Course" on Udemy by David Bombal to get a firm grasp of the basics of routing, switching and IP addressing among others. It was important to do this for a preparation for Cisco Certified Network Associate (CCNA) certification (see the Certificate of Completion in the Figure 3.1) also took "Python Learning for Network Engineers", which is a series of videos on the NetworkEvolution YouTube channel[26] that focuses on the usage of Python language for automation of network engineering tasks such as connecting to the network devices, running commands, and obtaining data.


udemy

Certificate no: UC-6fc8975f-5c2a-4fe3-a988-d216f5873568
Certificate url: ude.my/UC-6fc8975f-5c2a-4fe3-a988-d216f5873568
Reference Number: 0004

CERTIFICATE OF COMPLETION

The Complete Networking Fundamentals Course. Your CCNA start

Instructors **David Bombal**

Fahmida Hossain

Date **Aug. 8, 2024**
Length **79.5 total hours**

Figure 3. 1: Completion Certificate of Cisco Certified Network Associate Course

3. 0. 1 Justification

To provide a network automation environment and prove the research objectives and the project's goals are crucial. This methodology was selected for several reasons:This methodology was selected for several reasons:

Virtual Device Images: Cisco VDI was used because these are near real systems emulations and ideal for study and testing. The above images are officially endorsed by the EVE-NG emulator for network automation work as well as stability on the platform.

Network Emulator (EVE-NG): That is why, utilizing the EVE-NG Community Edition, it is possible to practice and improve the skills in emulating various network topologies. It also has support for a variety of network devices which include Cisco IOS and Virtual IO Service also known as VIOS to mimic network emulation in a virtual environment.

Programming Language (Python): There is a lot of coding with Python inherent in this project because it is a simpl Python is integral to this project because of its simplicity and a rich network automation library support. It is widely used in the industry and only offers the functionalities that are needed for the scripting of automation tasks such as configuring devices, connecting and getting data.

Visual Studio Code: The choice of this open-source platform as the principal development environment was guided by the fact that it offers support for the programming language in Python and integrated support for SSH Version 2 for communication with Cisco devices. Debugging is also possible in it along with other features for effective management and controlling of the scripts.

This methodology will benefit the work to be done in this project hence the objectives of automating the IP/MPLS network management for efficiency and effectiveness by minimizing errors encountered during the changes.

3. 0. 2 Design Methodology

The architecture of this project was modular which was very extendable and could also be easily tested. The network automation solution was recommended and improved in cycles. Initially, the idea was to automate simple tasks (e. g. , interfaces configuration), while in the next stages OSPF, BGP, and MPLS configurations were the major concerns. During the process, an iterative development approach was used and the automation scripts have been improved and modified until they are developed in several forms, and the functionality of the scripts has been tested frequently through out the stages. This also made it easier to modify the project as I progressed because the solution could transform appropriately in regards to the goals of the project.

Additional alterations were made to the original idea in the process of enlarging a program when several other routines, such as error control, source IP address assignment depending on the traffic type, and compatibility with different devices, are also to be solved.

3.0.3 Why Python Chosen

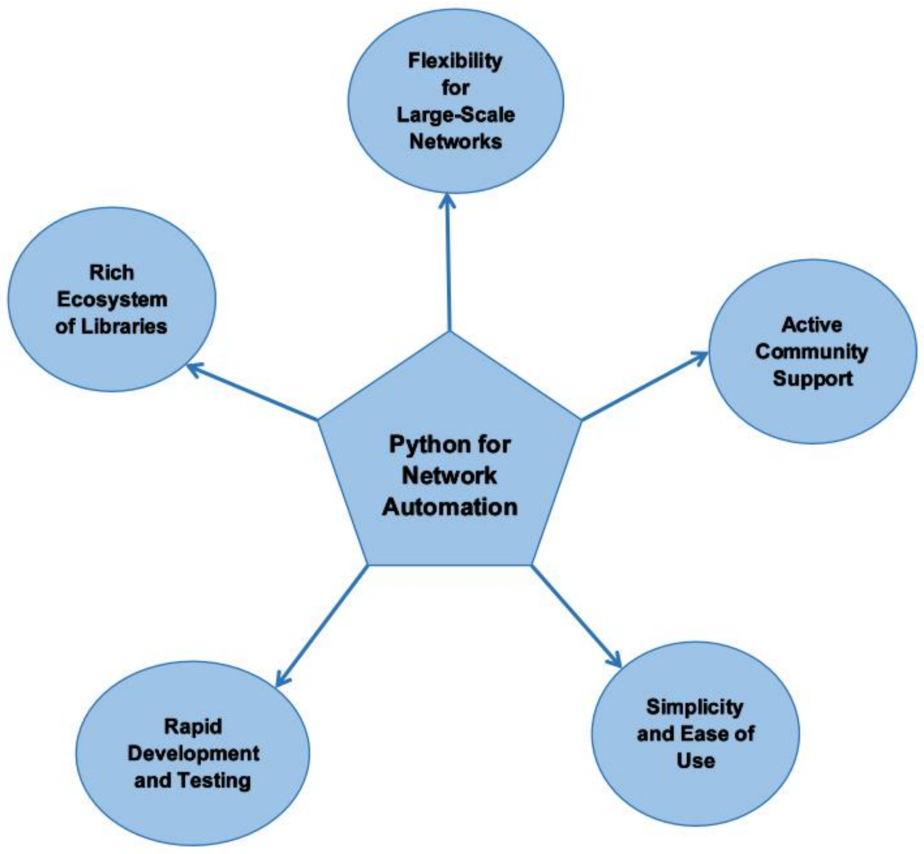

Flexibility for Large-Scale Networks
Rich Ecosystem of Libraries
Active Community Support
Python for Network Automation
Rapid Development and Testing
Simplicity and Ease of Use

Figure 3. 2: Potential Criteria to Choose Python for Network Automation

The reason why I selected Python is that it has features of being versatile and it is rapidly embraced by many organizations in network automation From other programming languages, Python stands to benefit from aspects like simplicity and easy learning to help the network engineers who may not be so good in programming to perform simple operation. Its endowing libraries including Netmiko, NAPALM, Paramiko and others are more focused on networking automation which makes interaction with network devices, configuration changes as well as secure data extraction possible.

Python has more flexibility than script languages that are ubiquitous in service providers because networks are huge, multi-protocol including OSPF, BGP and MPLS. These environments require properly configured, and accurately replicable across potentially hundreds or thousands of devices. Python does not apply the regular language where there are so many downtimes and operational risks when developing and testing. Furthermore, its community support means updated resources, practices and problem solving tips are available and hence the best to use for this project.

3. 1. 0 Emulator Overview

EVE-NG is a powerful tool for designing the scenario of the network where one can create the network environment and test it in the absence of the physical equipments. This emulator can allow one to integrate routers, switches, and other network devices and thus making it be among the best tools that can be used in network automation process.

EVE-NG is equipped with browser-based Graphical User Interface (GUI) which allows creation and operation of networks in very simple ways within a topology. The program gives users the opportunity to drag and drop the network nodes from the built-in and download templates which make the easy assembly and configuration of the complex network. For this project the Community Version EVE-NG version 6. 2. 0-4 was chosen since it supports most of the network devices images such as Cisco IOS, VIOS in order to test the IP/MPLS configurations needed.

System Requirements:
To make the functioning of the EVE-NG reliable, there are several minimum system requirements that need to be fulfilled:

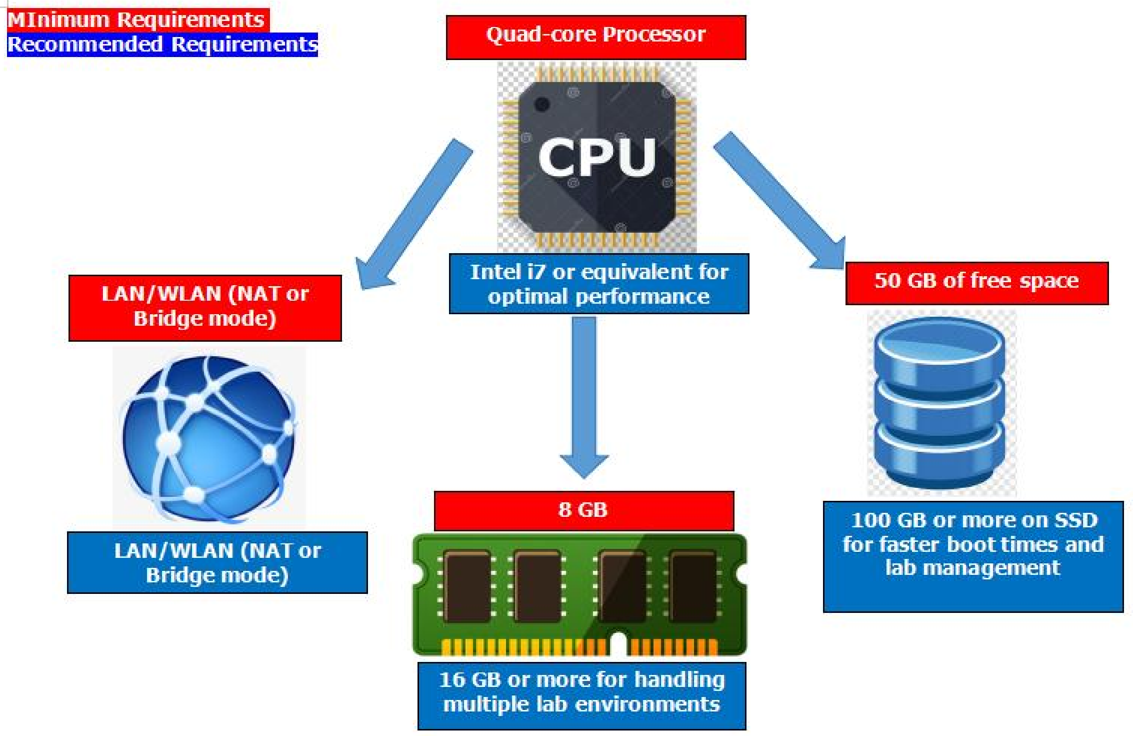

MInimum Requirements
Recommended Requirements
Quad-core Processor
CPU
Intel i7 or equivalent for optimal performance
LAN/WLAN (NAT or Bridge mode)
50 GB of free space
LAN/WLAN (NAT or Bridge mode)
8 GB
100 GB or more on SSD for faster boot times and lab management
16 GB or more for handling multiple lab environments

Figure 3.3: EVE-NG Installation Requirements
CPU: A minimum of 4GB RAM, preferably in dual channel with core i5/i7 Intel or Ryzen 5/7 AMd processor with VT-x EPT disabled/enabled in BIOS preferred.
RAM: To accommodate multiple virtual devices at least eight gigabytes are necessary.
Storage: An Operating system with at least 200 MB of RAM, A free Hard disk drive or SDD with space of at least 40 GB to store Virtual images and configurations files.
Network: A LAN or WLAN connection which is set at NAT or Bridge mode to allow the connection of the above virtual devices and other outside networks.

These specifications ensure that the emulator is in a position to run effectively especially when there are many virtual devices to control at any given period of time. As of now, the emulator is run on VMware Workstation Pro 17 which will enable use to use EVE-NG and manage virtual devices.

This project can then build on EVE-NG to emulate an authentic and practical network environment for purposes of testing and verifying automation scripts before actualization of the deployment process can be done, without having to undertake the time-consuming and costly process of setting up a physical lab.

3.2.0 License Agreement

In the construction of this network environment, I used EVE-NG community edition version which is a free variation for this project.

3.3.0 Types of Automation

Regarding this project, I had a interest in CLI-Based Automation using the aid of the Python tool named Netmiko. This approach let me demystify my network tasks such as starting an SSH session with the devices and then type commands on their command line interface. I also scripted simple administrative tasks such as configuring OSPF for internal routing, BGP for external routing and MPLS in an IP/MPLS with the help of Netmiko.

The reason for choosing CLI-based automation is because the tool can support all common network devices for most brand such as, Cisco router and switch and IOS, IOS-XE, VIOS and Juniper Junos. They also incorporated this method to make sure that the devices were configured in one fashion andRULE eliminated variability that could be expected when dealing with different devices manually as well as it had an added advantage of taking time as compared to when configuring the devices individually.

3.4.0 Specific Progression

While considering different types of automation, I categorized my work into three parts where each part focuses on one type of automation only. The idea behind undertaking such an approach was to establish a number of efficiencies and feasibility of all the methods in a setting which was planned with the aim of establishing the most effective techniques in handling an IP/MPLS network.

CLI-Based Automation: The initial idea of the CLI automation collection project was to use Python programming language and Netmiko library. This method was preferred for its widest compatibility and simplicity, especially in implementing in network devices that are comparatively old to support higher level automation profocal like RESTCONF or NETCONF. Autonomous configuration tools I created using python scripting language included the aspects like IP configuring, routing protocols (OSPF, BGP) initializing and MPLS enabling in the several devices of the network. At this phase, there were some difficulties that I encountered, for instance, while debugging the scripts there were errors that I had to solve: To overcome these challenges I had to go through a cycle of making changes to the scripts, use the resources from Github and get assistance from the online community. This aspect greatly helped in reducing the time taken in manual configuration, minimise and reduce the effects of errors greatly showed the benefits of automating a network in its management.

RESTCONF Automation (Planned Future Extension): While I was unable to fully integrate RESTCONF in this project I started my research and script drafting for this automation tool. RESTCONF was attractive as it allowed controlling network devices via RESTful APIs, which are becoming the standard in today’s networks. I wrote scripts with Python libraries including the requests library to invoke HTTP-based commands on devices that support RESTCONF. While recognising that it was difficult to master this approach within the time frame of the project, I plan to apply RESTCONF-based automation in the future because of its scalability and flexibility.

NETCONF Automation (Planned Future Extension): Thus, it is with theoretical interest that I considered using not only RESTCONF, but also NETCONF for network automation. NETCONF also provides greater manageability of the network devices since it can now support the management of configuration changes as well as the proper error reporting mechanism. I looked into Python connections such as ncclient that allows for NETCONF communications and considered its use in networks. Despite the fact that this method was not used in the current project, the knowledge of its possibilities allowed creating a basis for further testing and projects. NETCONF is a chance to increase the level of automation in the management of networks in managers that will be supported by this high-level protocol.

The segmentation of the project helped me to dedicate considerable efforts in CLI based automation while laying a groundwork for work on RESTCONF and NETCONF. Ideally, each automation type has its benefits and drawbacks, and such distinctions will be important as I further refine my experience in network automation.

3.5.0 CLI-Based Automation

When it comes to application of CLI-based automation in this project, the Netmiko Python library has been used. Netmiko is for smooth SSH connection towards network device and it manages other specific details of vendors in order perform configuration commands. While spokes like the standard SSH library, Paramiko, Netmiko has some extra features to deal with the network vendors needs, which is why it is more preferable for automating network tasks.

The Netmiko library was installed with the help of pip install netmiko command. Among the functions of Netmiko, it is worth mentioning the ability to connect via securely encrypted SSH, determine the device and vendor’s prompt, perform the command(s) issued, control the handling of the result, and other fine details such as terminal paging and width adjustment.

The primary tool used in this work was Netmiko but for the next job in the automation line I considered using NAPALM; Network Automation and Programmability Abstraction Layer with Multivendor support. NAPALM is multi-purpose and can be used for many network automation tasks, including: Collecting device configs, managing interfaces, query data for protocols – BGP, OSPF MPLS and much more. It supports multiple network devices such as Cisco and Juniper and offers outputs in different structured formats including JSON that makes it a perfect tool for monitoring and analysis.

3. 5. 1 Topology Overview

The topology for CLI-based network automation in this project is as shown below in figure 3. 5. 1, was implemented employing the EVE-NG Community Edition to build an accurate emulation of an IP/MPLS network infrastructure. These features include routers which are Cisco IOS 7206VXR (Dynamips) and switches used in VIOS which create a realistic environment of the service provider network.

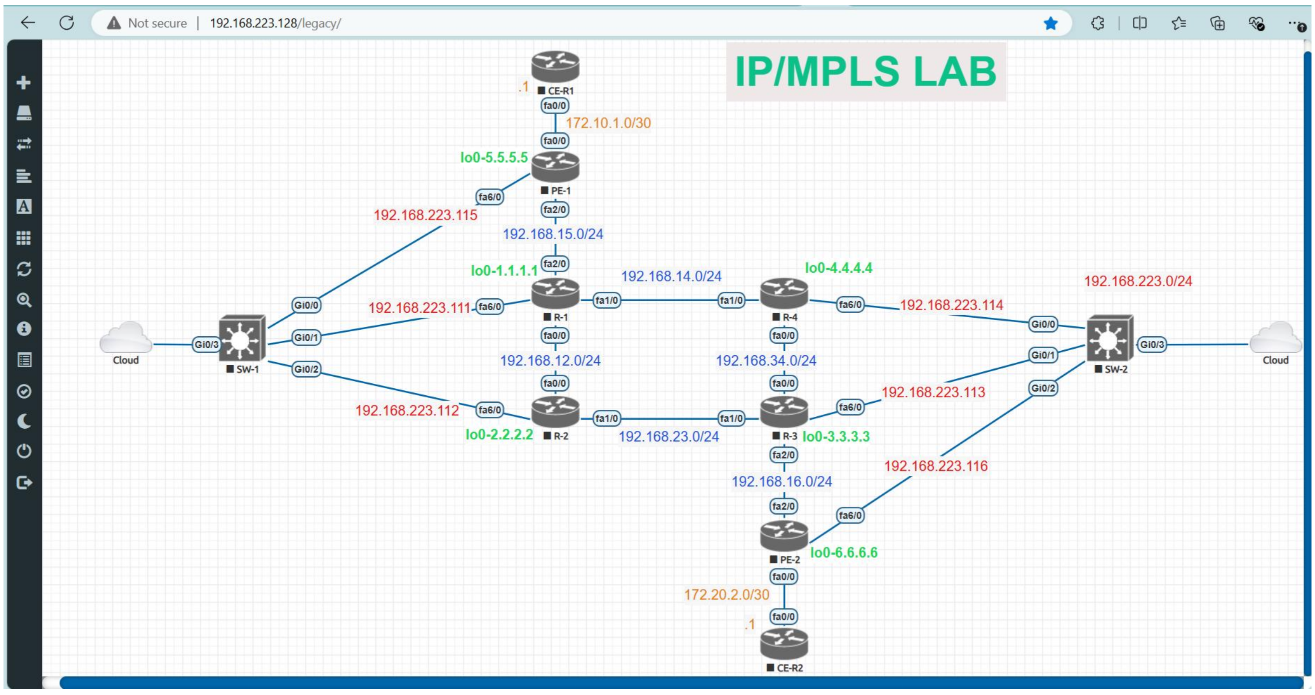


Figure 3. 5. 1 Unl Topology of IP/MPLS Lab

The network topology is made of routers R-1, R-2, R-3, R-4 are with cisco IOS which support dynamic routing protocol OSPF, BGP and MPLS. The providers involved are the PE-1 and PE-2 routers, providing connection to two CE routers which are CE-R1 and CE-R2 and simulate different customer settings. These routers are interconnected with core switches (SW-1 And SW-2) that offer connection to other networks through clouds links.

In the topology all the devices were programmed and automated with the help of Python scripts while utilizing Netmiko to initiate an SSH connection. This automation also included the main tasks like OSPF for internal routing, BGP for external routing as well as MPLS for transmission. Devices also had separate credentials for SSH so as to secure the devices while automation scripts could allocate IP addresses, configure the routing protocols as well as manage the devices' settings.

The loopback interfaces were configured on each router with the stable IP address to differentiate it from the active interface and to make it to be active once the device is powered On. These interfaces are necessarily unique within the network to act as endpoints for routing protocols such as OSPF and BGP east-west protocols and constantly monitor, manage and troubleshoot an MPLS VPN even if the physical interfaces are administratively down.

3.5.1.1 Rationale for Protocol Choices

The selection of OSPF, BGP, and MPLS for this project was based on their suitability for service provider networks and their ability to address the specific requirements of large-scale, dynamic environments:The specification of OSPF, BGP and MPLS for this project was made depending on their applicability to service provider networks and their effectiveness in managing large complex networks with highly dynamic environment as follows:

OSPF (Open Shortest Path First): Chosen because of, comparatively fast convergence speed, and scalability and interoperation flexibility for mixed IP environment. OSPFIGP favors Variable Length Subnet Masking that is VLSM and it also favors the frequent updates which is very useful in a service provider networks where the routing information has to be updated frequently.

BGP (Border Gateway Protocol): Selected in reference to its capacity to perform policy-based routing decisions and its capacity in handling with the routing tables. As compared to IGP, BGP is more controlling in the routing choices and is a requirement in the used topologies where traffic has to be properly routed across various network domains.

MPLS (Multiprotocol Label Switching): Chosen for its potential to increase the speed and efficiency of data delivery, by showing packets the way. MPLS offers the element of flexibility, low latency, as well as the network QoS necessary in line with traffic priority such as in service providers' networks.

These protocols explain why Python-base automation is possible in addressing various network configurations that can be expected of a service provider for downloadable purposes. This approach demonstrates one of the key issues associated with manual approaches, including longer time taken when performing the task, a higher likelihood of errors, and inconsistent work across an operational multi-protocol network, all of which can be improved by automating with the help of Netmiko.

3.5.2 Key Objectives of CLI Based Network Automation

This was a major achievement since we were able to prove that it is possible to use Python scripts to configure more than one network device through the CLI at once. In practice of this project, OSPF, BGP and MPLS dynamic routing protocols configured in the network on routers and switches of the simulated IP/MPLS network environment with the help of the Netmiko library were used.

The project shown that even if network have many protocols it would take 1-2 minutes to configure all protocols using script written in python while it could take much more time than one engineer. The use of the script also negates the need for human intervention thus minimizing on errors, one network engineer is able to handle the entire process of configuration efficiently.

The proactive strategy aims at answering the growing expectations on improved timely and effective network management in responsive environments. The project demonstrates the ability of CLI automation using Python scripting to address the industrial demand, having a uniform and protected configuration for all devices at a much faster and efficient way.

3.5.3 Data Sets

In order to achieve the right identification of the depth level of the network connection and to secure each device, unique data were attributed to each device and these are as follows: IP address for each device, unique username, passwords, and other credentials needed to perform a secure SSH connection for further configuration and monitoring of the network connection in this project. The IP addresses of the devices were obtained in Dynamic Host Configuration Protocol (DHCP) manner through the home router in order to facilitate easy management of the IP addresses.

Device Name	Device Type	Host IP Address	Username	Password	Port	Secret
R1	cisco_ios	192.168.223.111	admin	cisco	22	cisco
R2	cisco_ios	192.168.223.112	admin	cisco	22	cisco
R3	cisco_ios	192.168.223.113	admin	cisco	22	cisco
R4	cisco_ios	192.168.223.114	admin	cisco	22	cisco
PE-1	cisco_ios	192.168.223.115	admin	cisco	22	cisco
PE-2	cisco_ios	192.168.223.116	admin	cisco	22	cisco

Table 3.2.1: Data Sets of Network Topology

The table above presents the data sets employed in CLI-based network automation in this project. Each device was connected with a unique IP address received from home router dynamically using DHCP protocol, which was efficient in handling and minimized the work of manually configuring each. The SSH authentication such as username, password, and port number, as well as secret was hard-coded in the automation scripts to ensure that there was that there was proper authorization with reduced chances of exposure.

It is very important to make sure these data points are properly set within the Python scripts before one starts running any SSH connections otherwise one will fail. The dynamic IP address allocation is quite realistic in situations when some devices are assigned new addresses according to the network they operate in, at the same time, using static credentials ensures that only authorized personnel can access the devices for automation purposes.

The subsequent subtopics will elaborate on the data processing techniques applied in the processes and the procedures of assigning dynamic IP addresses and handling SSH streaming in the automation scripts and applications.

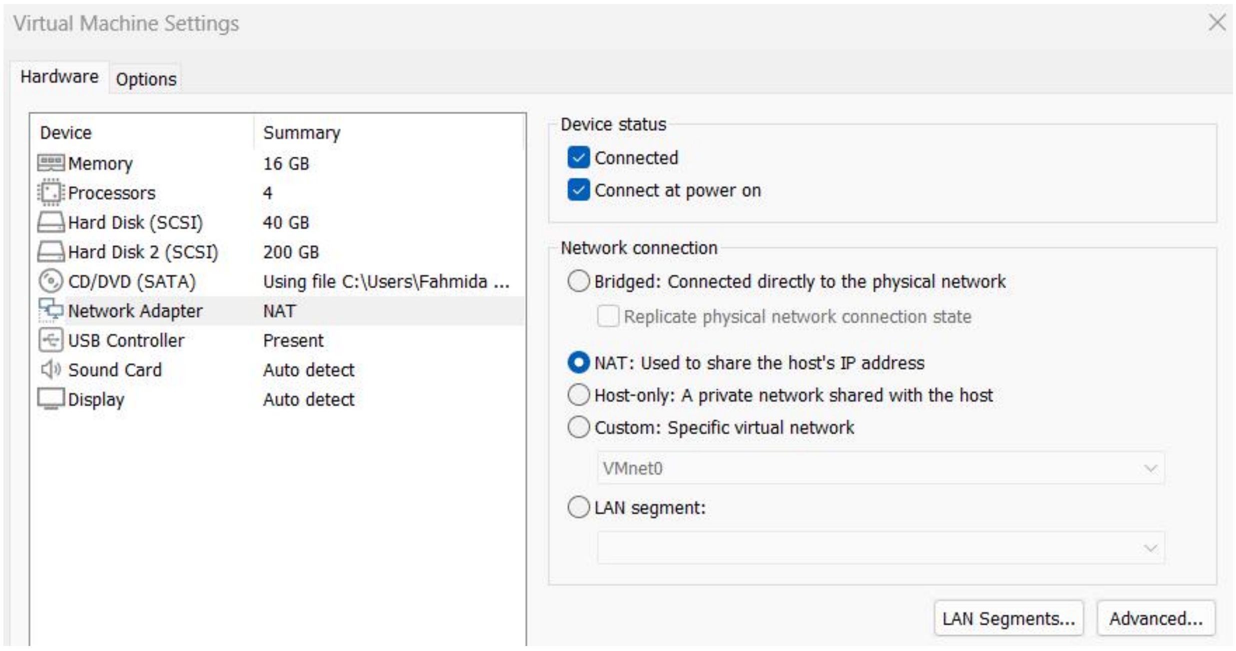
3.5.4 Data Collection

On this project, Dynamic IP addressing was conducted on all the networks taking the IP from the home router's DHCP server. It offered the convenience to manage configurations of the devices which are in the EVE-NG virtual network easily. Each component within the consisting network as indicated the following table. 2. 1, were provided with specific credential to authenticate the SSH for access and automation with the help of Python script Netmiko.

3.5.4.1 System Requirements for Data Collection

Two key configurations were necessary to facilitate dynamic IP assignment and data collection:To support dynamic IP assignment and data collection two major configurations were required:

1. **NAT Configuration on VMware:** This is because the project was conducted on a laptop with internet connection through Wi-Fi and hence the NAT mode was active in VMware Workstation Pro. This allowed the EVE-NG virtual machines to communicate with other outside networks through the Virtual machine's NIC.



Virtual Machine Settings

Hardware | Options

Device	Summary
Memory	16 GB
Processors	4
Hard Disk (SCSI)	40 GB
Hard Disk 2 (SCSI)	200 GB
CD/DVD (SATA)	Using file C:\Users\Fahmida ...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Display	Auto detect

Device status

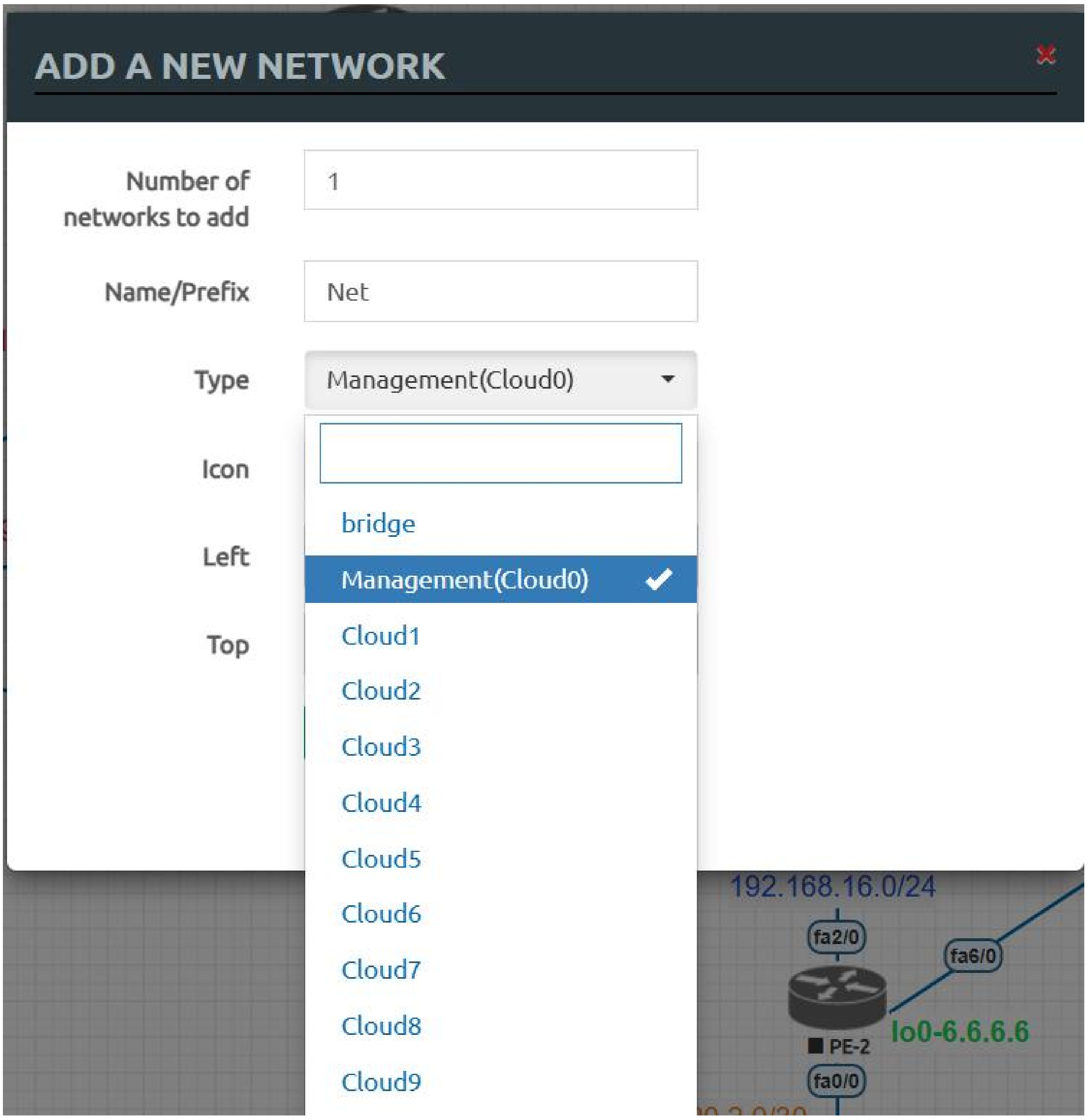
- [✓] Connected
- [✓] Connect at power on

Network connection

- ○ Bridged: Connected directly to the physical network
  - ☐ Replicate physical network connection state
- ● NAT: Used to share the host's IP address
- ○ Host-only: A private network shared with the host
- ○ Custom: Specific virtual network
  - VMnet0
- ○ LAN segment:

Figure 3.5.4.1: It shows the Network Adapter Setting

2. Cloud NAT in EVE-NG: To achieve this in EVE-NG, the devices were directly connected to Cloud NAT (Management Cloud) and got the IP assigned by home router's DHCP. This configuration was necessary after each device is allowed to obtain an IP address dynamically and participate in simulation of the networks. .



ADD A NEW NETWORK

Number of networks to add: 1

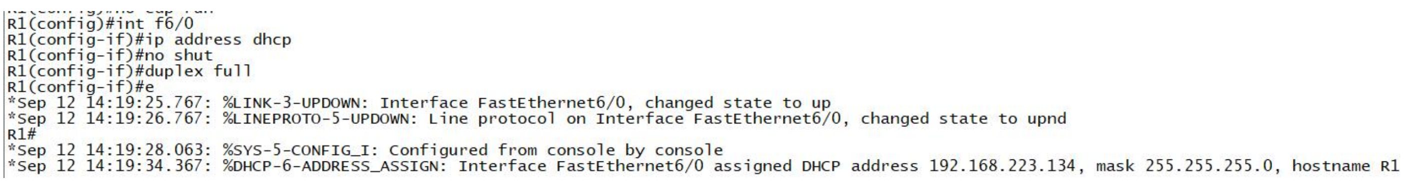
Name/Prefix: Net

Type: Management(Cloud0)

Figure 3.5.4.2 Shows Management cloud

3.5.4.2 Management Interface and DHCP Configuration

Each device was connected with a Management Interface assigned to GigabitEthernet 0/0 so each device’s IP address would be obtained through DHCP dynamically. This allowed the automation scripts to be able to communicate with the devices because of the dynamically provided IP addresses. The following configuration was applied to each device:The following configuration was made on each of the equipments:



```
R1(config)#int f6/0
R1(config-if)#ip address dhcp
R1(config-if)#no shut
R1(config-if)#duplex full
R1(config-if)#e
*Sep 12 14:19:25.767: %LINK-3-UPDOWN: Interface FastEthernet6/0, changed state to up
*Sep 12 14:19:26.767: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet6/0, changed state to upnd
R1#
*Sep 12 14:19:28.063: %SYS-5-CONFIG_I: Configured from console by console
*Sep 12 14:19:34.367: %DHCP-6-ADDRESS_ASSIGN: Interface FastEthernet6/0 assigned DHCP address 192.168.223.134, mask 255.255.255.0, hostname R1
```

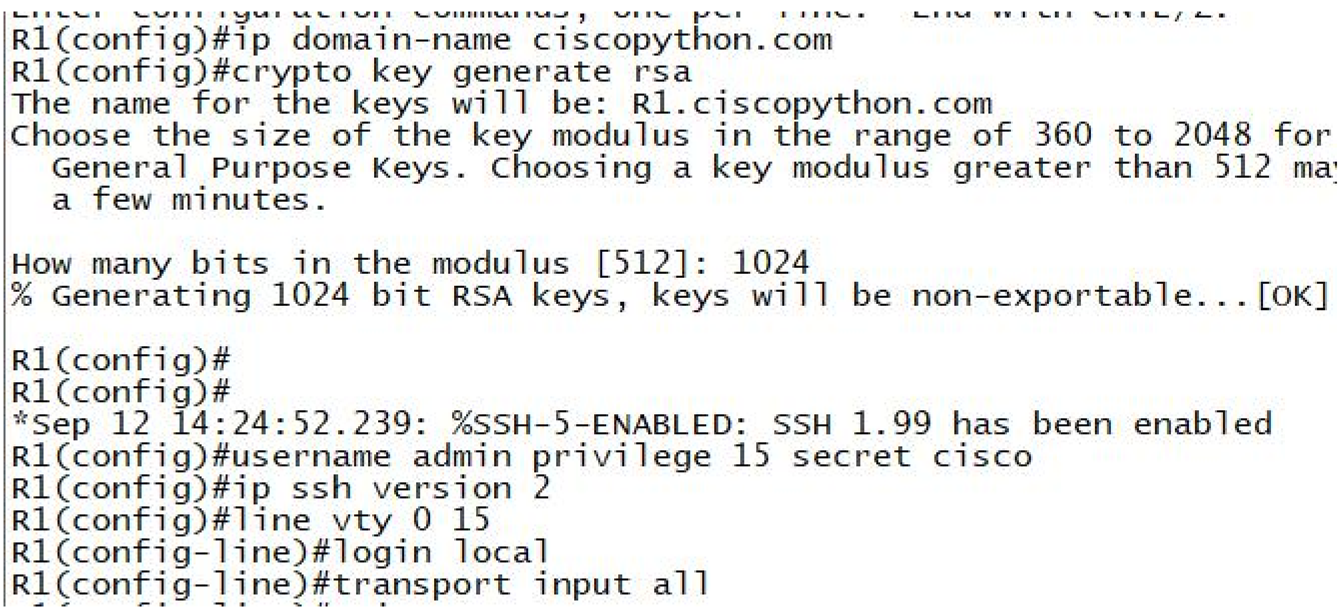
Figure 3.5.4.3: DHCP configuration

This scenario ensured that each device was provided with a dynamic IP address from the DHCP in such a manner that the automation scripts could utilize the IP addresses to make contact and also set the devices remotely.

3.5.4.3 SSH Configuration and Secure Access

All these devices before automating had Secure Shell (SSH) set up for security and the purpose of remote access by hand. In contrast to Telnet application, the SSH had no encryption facilities but had the Secure and encrypted access to the devices and thus avoided unauthorized access. Other real life IT applications that were done include PuTTY and Secure CRT which were used for the actual manual ssh as and when required for diagnostic and monitoring purposes only in case of necessity.

In Python, the most efficient way of automation of the SSH connections was using the Netmiko library due to the capability of establishing secure communication between the scripts and the devices. The following SSH configuration commands were used to establish secure access:The following did uphold the SSH configuration commands that were used:



```
R1(config)#ip domain-name ciscopython.com
R1(config)#crypto key generate rsa
The name for the keys will be: R1.ciscopython.com
Choose the size of the key modulus in the range of 360 to 2048 for
  General Purpose Keys. Choosing a key modulus greater than 512 ma
  a few minutes.

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

R1(config)#
R1(config)#
*Sep 12 14:24:52.239: %SSH-5-ENABLED: SSH 1.99 has been enabled
R1(config)#username admin privilege 15 secret cisco
R1(config)#ip ssh version 2
R1(config)#line vty 0 15
R1(config-line)#login local
R1(config-line)#transport input all
```

Figure 3.5.4.4: Establishing secure access

SSH is important in the project since it provided a high level of security when applying the configurations thereby avoiding cases of people who may try to access the systems with the intent of making some changes that may not be beneficial or applying some unwanted configurations on the systems. PuTTY and SecureCRT were used for the manual access and monitoring in case of necessity.

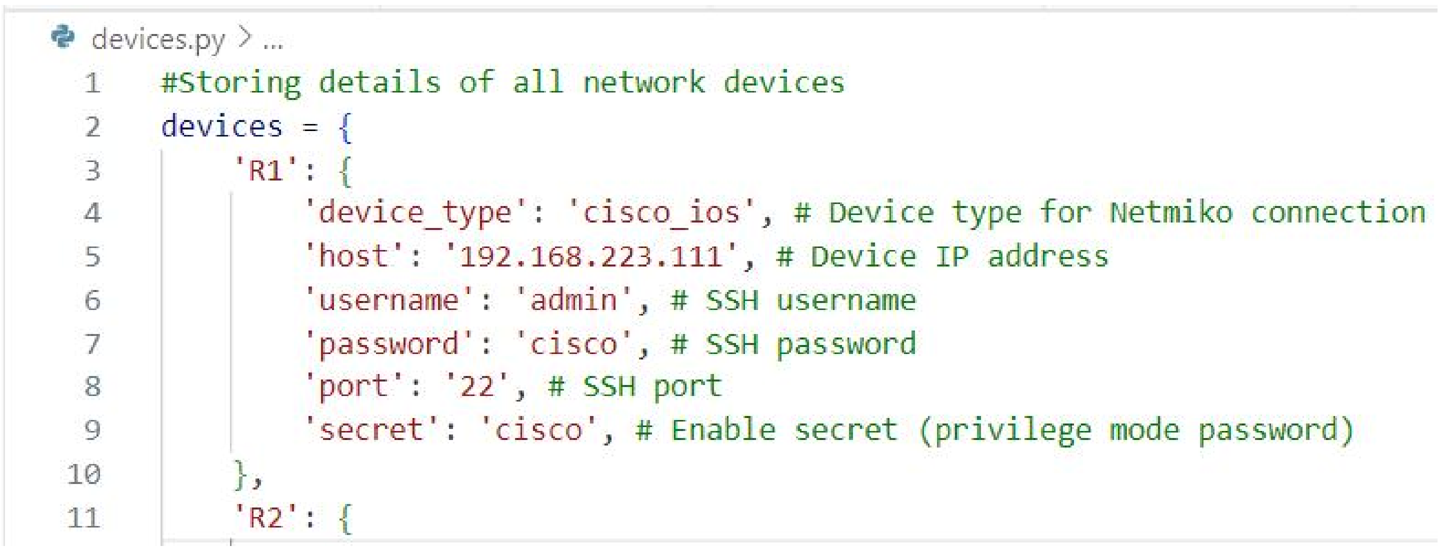
3.5.5 Data Store

For automation of configuration, credentialing and network parameters such as MTU in this project, storage of network device information was a serious factor. Static and dynamic data as base for automation scripts were used and updated when the script was running to provide reliable results.

3.5.5.1 Storage of Device Information

General information that does not change with time, for instance, device IP addresses, SSH logins, configuration commands among others, were hard coded within the Python scripts. This would mean that data about each of the networks devices including the routers and switches would be available for automatic configuration. The devices also contained data concerning the type of the device, its host IP, username, password and the port in a database. py file.

The following Python dictionary was used to store this device information:



devices.py > ...

```python
#Storing details of all network devices
devices = {
    'R1': {
        'device_type': 'cisco_ios', # Device type for Netmiko connection
        'host': '192.168.223.111', # Device IP address
        'username': 'admin', # SSH username
        'password': 'cisco', # SSH password
        'port': '22', # SSH port
        'secret': 'cisco', # Enable secret (privilege mode password)
    },
    'R2': {
```

Figure 3.5.4.5: Storing Device Information

The feature described in this static method of storing device information enabled the automation scripts to easily pull the creditentials and commands that are relevant at the point of running the script.

3.5.5.2 Dynamic Data Management

Other types of data, for example the IP addresses obtained by DHCP from the home router, were dynamically acquired, attributed and applied to all the devices in the EVE-NG network environment. While the IP addresses were dynamically assigned, the credentials and the configuration parameters are fixed so as to have a secure connection with each device.

The ‘Netmiko’ library in Python aided the automation of SSH connections with the use of the stored credentials; On the other hand, an observation of the IP addresses assigned dynamically by the DHCP server helped in identifying the devices on the network.

3.5.5.3 Security Considerations in Data Storage

As for the stored data, special attention was paid to security, since the content of the data was such important information as usernames, passwords, and SSH secrets. Although in this simulated environment the credentials were simply stored in plain text, it is crucial to stress out that for real-life applications, it is advisable to employ encrypted storage or use environment variables for this purpose. Options such as Ansible Vault or, for instance, HashiCorp Vault may be used for handling production-environment credentials securely.

This project was capable of having a secure and an efficient manner of administration of the device configurations and credentials while at the same time following the protocols of use like the SSHs and separation of statis and the dynamic data.
The accessibility of the configurations of devices is well optimized with the help of the Python dictionaries and at the same time very much secured. This method minimizes the possibilities of having to manual first OSPF, MPLS, and BGP among so many devices and thus facilitating their implementation.

3.5.6 Setup of the Network Automation Environment and Experimentation

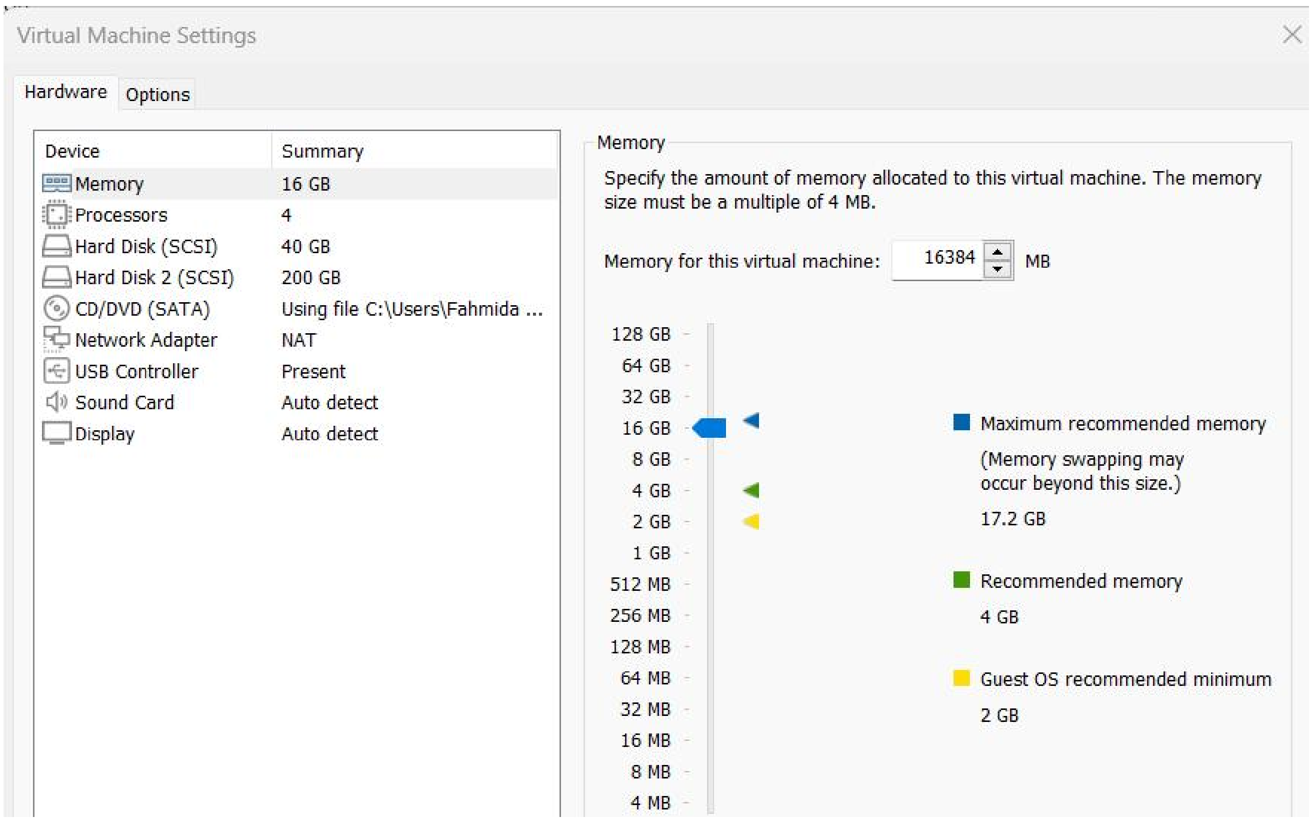
The setup of the network automation environment was one of the primary stages of this project: it was necessary in order to present framework for running scripts written in Python language for autoconfiguration of network for several devices. The environment for this kind of experiment was created using VMware Workstation Pro 17 (Free version), and with respect to the emulation of a network, the EVE-NG platform was used. The following sections give a guide on how to set up the automation environment as the following.

3.5.6.1 VMware Installation and Configuration

The automation server was hosted on a virtual machine which was developed using VMware – Workstation Pro 17. Therefore, and for the purpose of experimenting the proposed protocols, a Windows 11 laptop was set up with VMware to make some networks simulations. The virtual machine was provisioned with sufficient resources that enable the end to end management and operation of the EVE-NG platform and all the network automation functionalities. The following specifications were set for the virtual machine:The following specifications was then set for the virtual machine:

- **Memory**: 16 GB
- **Processors**: 4 cores
- **Hard Disk**: 200 GB
- **Network Adapter**: NAT mode (to dynamically assign IP addresses to virtual devices via the home router's DHCP server)
- **Operating System**: Ubuntu 64-bit (EVE-NG)

Refer to **Figure 3.5.6.1** and the image below for the VMware virtual machine settings, which highlight the allocation of resources and network configurations:



Virtual Machine Settings

Hardware | Options

Device	Summary
Memory	16 GB
Processors	4
Hard Disk (SCSI)	40 GB
Hard Disk 2 (SCSI)	200 GB
CD/DVD (SATA)	Using file C:\Users\Fahmida ...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Display	Auto detect

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 16384 MB

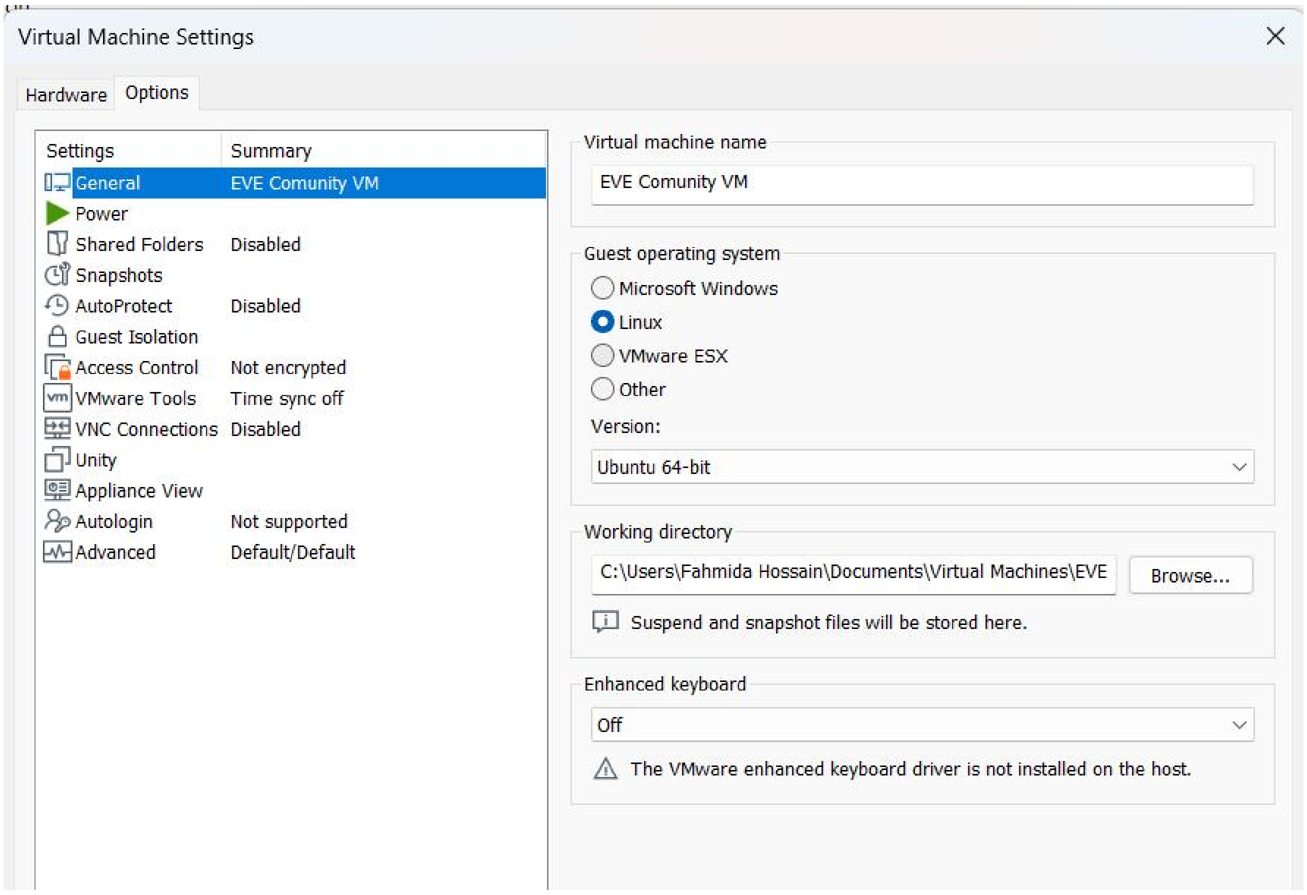
🟦 Maximum recommended memory
(Memory swapping may occur beyond this size.)
17.2 GB

🟩 Recommended memory
4 GB

🟨 Guest OS recommended minimum
2 GB

Figure 3.5.6.1: Virtual Machine Settings

3.5.6.2 Operating System and Python Installation



Virtual Machine Settings

Hardware | Options

Settings	Summary
General	EVE Comunity VM
Power	
Shared Folders	Disabled
Snapshots	
AutoProtect	Disabled
Guest Isolation	
Access Control	Not encrypted
VMware Tools	Time sync off
VNC Connections	Disabled
Unity	
Appliance View	
Autologin	Not supported
Advanced	Default/Default

Virtual machine name

EVE Comunity VM

Guest operating system

○ Microsoft Windows
● Linux
○ VMware ESX
○ Other

Version:

Ubuntu 64-bit

Working directory

C:\Users\Fahmida Hossain\Documents\Virtual Machines\EVE

Suspend and snapshot files will be stored here.

Enhanced keyboard

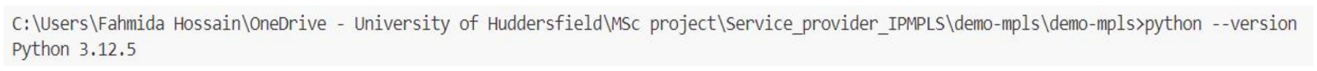
Off

The VMware enhanced keyboard driver is not installed on the host.

Figure 3.5.6.2: Operating Sytem for EVE-NG

From the Figure 3.5.6.2, the virtual machine was installed Ubuntu 64-bit hosting the EVE-NG network emulator. This was why Ubuntu was preferred as it was capable of supporting the automation tools to be used in this project. After installing the operating system, the following additional components were installed to support network automation:The opposite components were then, installed in the system in order to carry out the computer's network automation:

- **Python 3.12.5**: The latest version of Python was installed to run the automation scripts.



```
C:\Users\Fahmida Hossain\OneDrive - University of Huddersfield\MSc project\Service_provider_IPMPLS\demo-mpls\demo-mpls>python --version
Python 3.12.5
```

Figure 3.5.6.3 : Python Version Command

- **pip**: It is a tool for installation of the libraries which concern the automation of networks through Python.

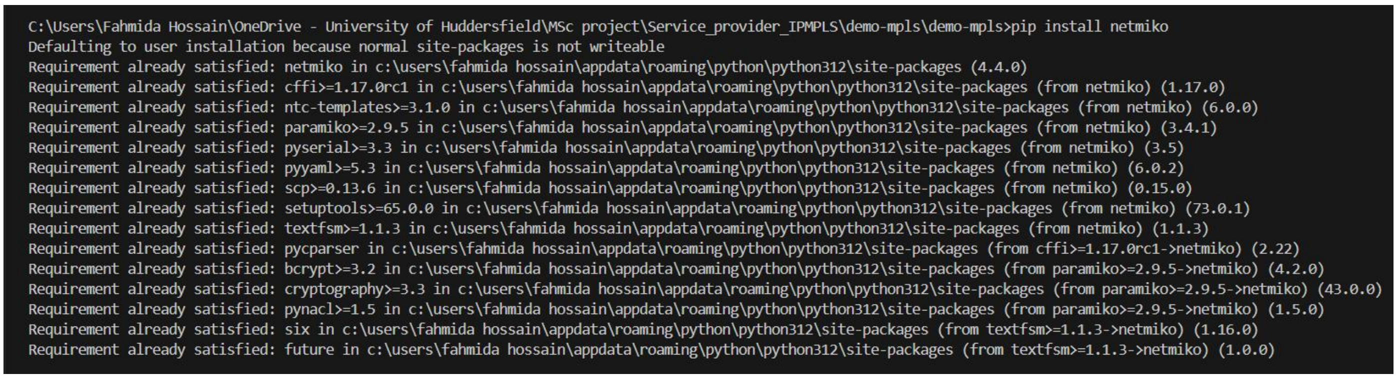
When using the python software there was a problem I for instance encountered a problem in that python was not responding well as soon as I installed it. In response to this I set some settings of the path on the system using the interpreter to be Python. This step was necessary in order to make the terminal recogniSe the existence of python for purposes of running automation scripts.

3.5.6.3.EVE-NG Installation

Running on the virtual machine was EVE-NG Community Edition (Version 6. 2. 0-4). EVE-NG is a very useful utility can be used to emulate a particular networking scenarios and help to run the automation scripts without actua hardware.
EVE-NG is connected with VMware Workstation Pro with connect virtual network via NAT. This enabled the virtual devices to obtain the IP addresses from the home router's DHCP server in a rather dynamic way. Additional tests where necessary were possible through communication with other networks while NAT mode was used for interactions between the virtual devices and the automation server.

3.5.6.4 Python Libraries and IDE Setup

After the success of the installation of the python software, it was time to install the libraries that are used in network automation. This project's key library was Netmiko, which, among other things, allows for automation of SSH connection to network devices for configuration. The library was installed using the following command:Here is the command with which the library was installed:



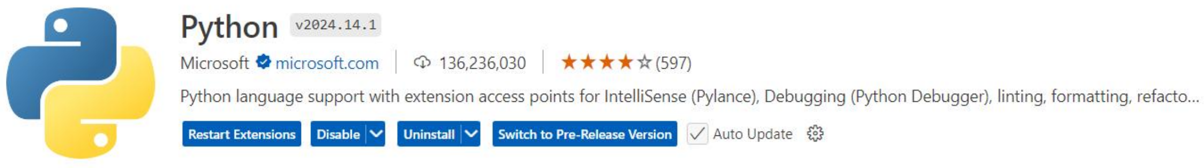
```
C:\Users\Fahmida Hossain\OneDrive - University of Huddersfield\MSc project\Service_provider_IPMPLS\demo-mpls\demo-mpls>pip install netmiko
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: netmiko in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (4.4.0)
Requirement already satisfied: cffi>=1.17.0rc1 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (1.17.0)
Requirement already satisfied: ntc-templates>=3.1.0 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (6.0.0)
Requirement already satisfied: paramiko>=2.9.5 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (3.4.1)
Requirement already satisfied: pyserial>=3.3 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (3.5)
Requirement already satisfied: pyyaml>=5.3 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (6.0.2)
Requirement already satisfied: scp>=0.13.6 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (0.15.0)
Requirement already satisfied: setuptools>=65.0.0 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (73.0.1)
Requirement already satisfied: textfsm>=1.1.3 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from netmiko) (1.1.3)
Requirement already satisfied: pycparser in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from cffi>=1.17.0rc1->netmiko) (2.22)
Requirement already satisfied: bcrypt>=3.2 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from paramiko>=2.9.5->netmiko) (4.2.0)
Requirement already satisfied: cryptography>=3.3 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from paramiko>=2.9.5->netmiko) (43.0.0)
Requirement already satisfied: pynacl>=1.5 in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from paramiko>=2.9.5->netmiko) (1.5.0)
Requirement already satisfied: six in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from textfsm>=1.1.3->netmiko) (1.16.0)
Requirement already satisfied: future in c:\users\fahmida hossain\appdata\roaming\python\python312\site-packages (from textfsm>=1.1.3->netmiko) (1.0.0)
```

Figure 3.5.6.4: Netmiko Instalation Command

To write, develop, execute and test the Python automation scripts the Visual Studio Code IDE was chosen to as the primary environment. Several extensions were installed within

VS Code to enhance the development environment:In order to improve the development environment several extensions were integrated into VS Code.

- **Python v2024.14.1**: Regarding the functionality it provided the basic necessities like syntax highlighting, autocompletion and debugging tools which came into handy when programming with python.



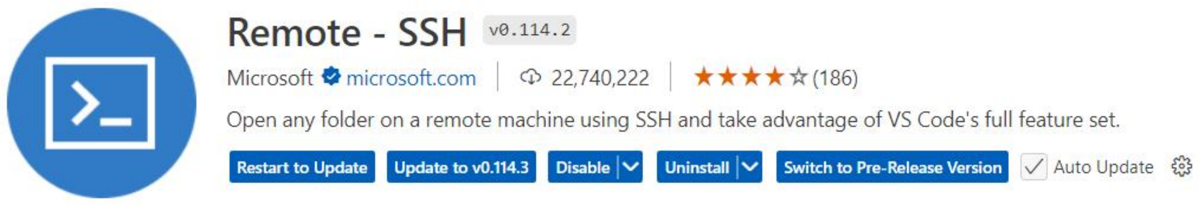
Python v2024.14.1

Microsoft microsoft.com | 136,236,030 | ★★★★☆ (597)

Python language support with extension access points for IntelliSense (Pylance), Debugging (Python Debugger), linting, formatting, refacto...

Figure 3.5.6.5:python Extension for VS Code

- **Remote - SSH v0.114.2**: In a nutshell, this extension allowed me to get the virtual network devices on EVE-NG and configure and adjust them within VS Code.



Remote - SSH v0.114.2

Microsoft microsoft.com | 22,740,222 | ★★★★☆ (186)

Open any folder on a remote machine using SSH and take advantage of VS Code's full feature set.

Figure 3.5.6.6 SSH Extension for VS Code

3.5.7 Automation Script Deployment

Once environment was fully set up, I made automations scripts like ospf_config and ran them. py, bgp_config. py, and mpls_config. py. These scripts were able to incorporate netmiko in controlling OSPF, BGP and also MPLS configuration in many devices.

These scripts were execute from the VS Code terminal based on the SSH connections created by Netmiko. This showed that most of the automation procedures work effectively by completing tasks, which would have taken much time when done manually, within a few minutes across the devices in the network.

3.5.7.1 Connection Establishment

To communicate with the network devices the Python script is developed with the help of Netmiko library. ConnectHandler is most crucial method which help to create connection for SSH.

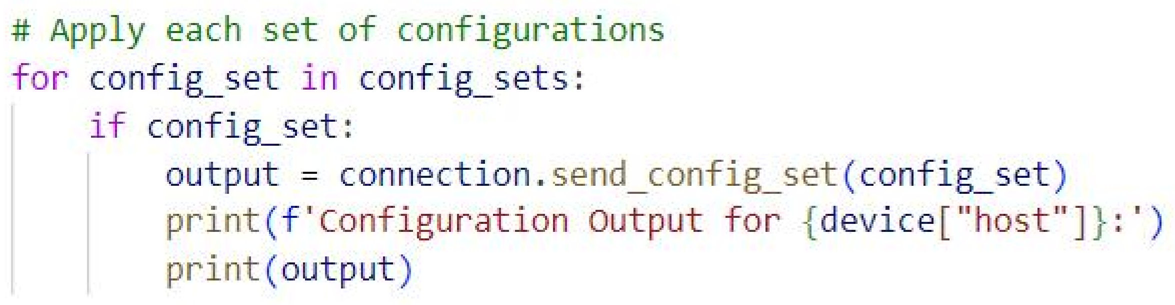
```
# Connect to the device
connection = ConnectHandler(**device)
```

The following line was used in the scripts to establish the connection. This method assist in making a secure connection with the help of SSH and credentials to the network device as per the description.

3.5.7.2 Sending Configuration Commands

Once a proper network connection has been formed with the network device through an SSH session it is then time to send in the commands that holds the desired configurations. It is here that a help arrives in the form of the send_config_set() method from the Netmiko library. Commands such as OSPF, BGP OR MPLS are transmitted in groups unlike when they have to be fed individually and this saves on time. For instance, the configuration of each of the routers are in dictionary format and before the script transmits it, it selects the appropriate one for the device.

The following code snippet demonstrates how this is achieved:The following code shows how this is done;

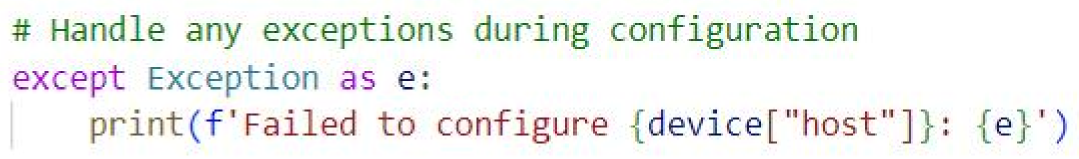


```
# Apply each set of configurations
for config_set in config_sets:
    if config_set:
        output = connection.send_config_set(config_set)
        print(f'Configuration Output for {device["host"]}:')
        print(output)
```

The command connection. When all configurations are applied at once in a single run, it will be hard to make a mistake and the configuration set from send_config_set(config_set) confirm that all the devices are well configured. It includes the status of the configuration after the update and the notification is sent out and printed to the console for the analysis by the Network administrators in real-time. This method saves time than having to perform the configurations manually while at the same time ensuring that complex configurations that involve a series of steps are done correctly.

3.5.7.3 Error Handling and Logging

In order to increase the reliability of the automation script there were frequent errors chances and log files injected in the script. However, whenever setting nets there are always issues such as connection faults or command execution issues. To combat this problem the script wraps such critical operations as SSH connection creation and command execution with try-except construct. In case something is wrong, for instance, if an SSH connection the is lost, or an invalid configuration command is issued, Netmiko shows an exception. The following code snippet demonstrates how errors are handled:



```
# Handle any exceptions during configuration
except Exception as e:
    print(f'Failed to configure {device["host"]}: {e}')
```

This makes certain that the automation process does not come to a stand still if one of the devices is faulty. Rather it jumps to the next device and writes the problems it encountered for usage in future for rectification. This is because by integrating logging, the administrator will be able to know which of the devices failed and why making it easier in the process of troubleshooting. This reduces the occurrence of script failure and the impact of this failure when in large complex network environment.

3.5.7.4 Testing and Validation

Subsequent to the automation script, a lot of testing was done within the EVE-NG environment to confirm whether the configurations where applied properly and correctly. The highly dynamic environment also provided the ability to test and validate the scripts since the solution emulated the network devices in real time.
Another execution application was PuTTY and SecureCRT for performing manual SSH sessions in the case of troubleshooting or monitoring all the network configurations to confirm their functionality.

3.5.8 Flow Chart of Python Script

Python scripting that has been used in this project enables the configuration of the network devices whereby tasks such as opening SSH sessions, interfacings, and protocols for instance OSPF, BGP, as well as MPLS are completed on multiple devices. To make a clear conception about structures and flow of the automating script, following flowchart is designed with the significant activities present in the process.

3.5.8.1 Flowchart Diagram

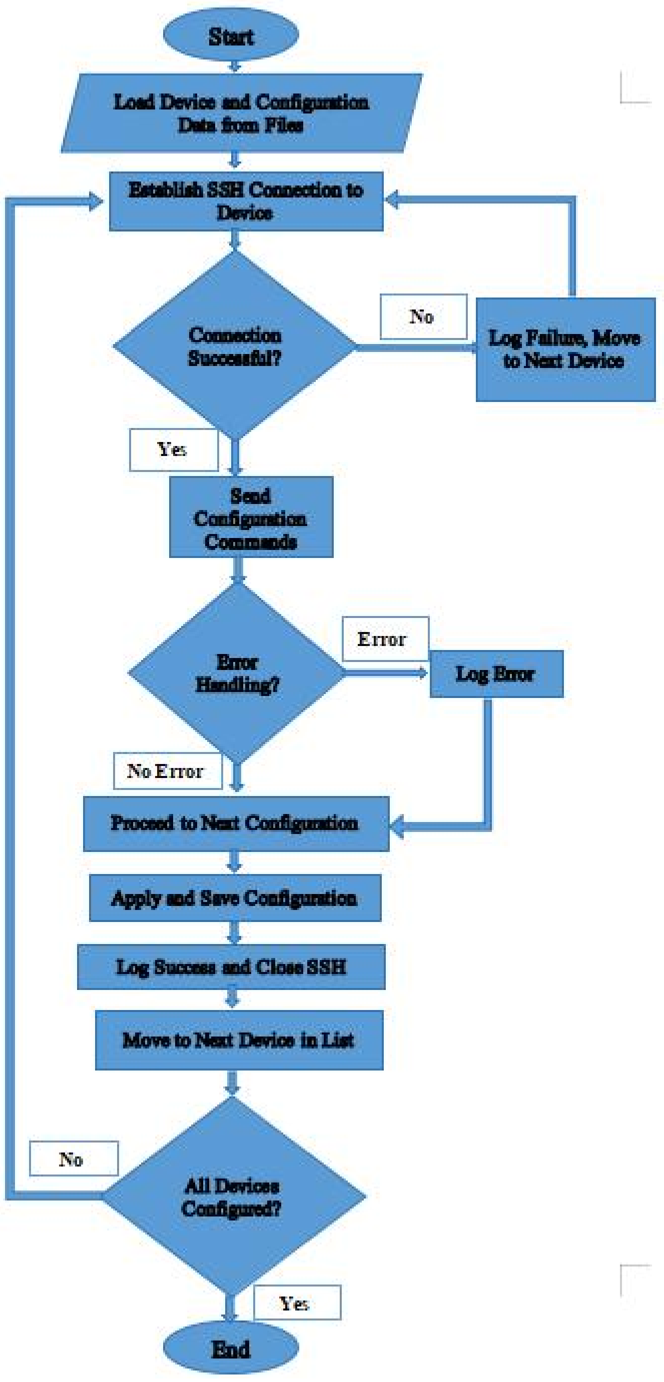

Start
Load Device and Configuration Data from Files
Establish SSH Connection to Device
Connection Successful?
No
Log Failure, Move to Next Device
Yes
Send Configuration Commands
Error Handling?
Error
Log Error
No Error
Proceed to Next Configuration
Apply and Save Configuration
Log Success and Close SSH
Move to Next Device in List
No
All Devices Configured?
Yes
End

Figure 3.5.8.1: Flow chart of Automation Process

3.5.8.2 Flowchart Description

Here is the flowchart of the process of automatic configuration of the network devices using a python script which in addition provides detail information about each of the steps involved:

1. Start:

- Automating the process starts by running the 'Python script'.

2. Load Device and Configuration Data from Files:
- Data is needed in the script for example; device details like IP address/credential; and configuration such as routing protocols information such as OSPF, BGP, MPLS amongst others; are retrieved from other files.

3. Establish SSH Connection to Device:
- The script attempts at establishing an SSH connection of the current network device using the loaded data.

4. Connection Successful?:
- The script first takes a check to make sure that the connection at SSH has been done so successfully.
- If the connection is established then the next functions from the program are to be executed, which are sending the configurations commands to the switch.
- If then the script fails to complete then it puts the failure and continues to the next device it has in the list.

5. Send Configuration Commands:
- It transmits the correct configuration command onto the network device that is connected. These commands may set OSPF, BGP, MPLS or interfaces of one network in another or many networks.

6. Error Handling?:
- If there is any mistake made on the certain command before executing it then the script will display the error on the terminal.
- If there is any error than it will just record the steps and then proceed to carry out the next step of installation.
- In case there is no error, the process proceeds the next configuration of a pattern item by incrementing the width of an example.

7. Proceed to Next Configuration:
- If there is no error then the script either applies the next set of commands that are in the script.

8. Apply and Save Configuration:
- This stage entails the making of the requisite configuration changes followed by writing these changes and making them ‘remembered’ on the device.

9. Log Success and Close SSH:
- After the above mentioned configuration has been effected the script then displays a success message and then terminates the ssh session to the device.

10. Move to Next Device in List:
- The script moves on to the next device in the list and repeats the process for that device.

11. All Devices Configured?:
- The script checks if all devices in the list have been configured.
- If all devices are configured, the process ends.
- If there are still devices left, the script loops back to the "Establish SSH Connection" step.

12. End:
- The script concludes after all devices have been successfully configured.

The following flowchart presents an efficient and automated process of setup of several network devices and focusing on security and speed of operations through SSH connection. Doing so, the script minimizes people’s mistakes when configuring the network and organizing OSPF, BGP, or MPLS schemes. It also includes error handling techniques which are used to be able to make the script run to other devices even if one of them fails to work. It should also be noted that the modularity and scalability, which means that the script can manage a greater number of nodes in a network compared to the ‘’manual’’ method; This makes the script especially helpful when configuring several devices while at the same time, it eliminates errors.

4.0 Results and Findings

This section descries the result of the network configuration process, which show the comparison between the manual and automated work along with the improvement that have been achieved with the help of automation. Further, we discuss the observations made from the analysis of the screenshots obtained to support the results of the automation process.

4.1 Manual Configuration Challenges

At the beginning, the setup of the routers was done in a static manner. While doing this, I came across a major challenge in which one of the routers had a wrong configuration and hence resulted in a connection loss. It also took considerable time to diagnose the problem and to eradicate it. I searched for the mistake in turning on lights to every router separately, and to the settings separating lines to pinpoint the mistake. What I discovered from this exercise is the fact that manual configurations are often subject to human error and that, correcting such errors can be tedious especially in cases of basic network applications.

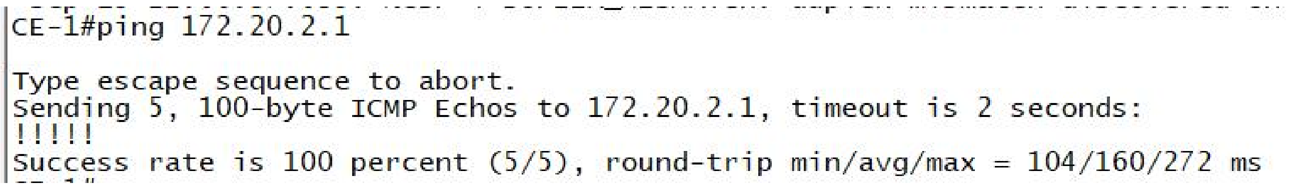
In a real world context this might mean that the protection scheme is blocking important traffic and hence contributing to the unavailability of the network, services or operations. Failure to realize that a configuration error cost quite some time to rectify is a good example that demonstrates how manual procedures can act as constraints in handling networks.

4.2 Automation Efficiency

For the second part of the project, the configuration was automated with the use of Python scripts. After the automation environment has been configured I was able to start the Python script to configure all the six routers within three minutes at most. Each of the configuration steps that could otherwise be configured manually, one at a time for each router, was configured automatically, and without any occurrences of error during the process.

4.2.1 Ping from CE-1 to CE-2 :

The ping response that has been obtained from CE-1 to CE-2 indicates that the automated configuration scripts ran properly. The interaction between the customer edge devices CE-1 and CE-2 shows that all the protocols and configurations like OSPF, BGP and MPLS were applied through the above mentioned Python automation scripts. This alarms checks to ensure that the network is running as it should.

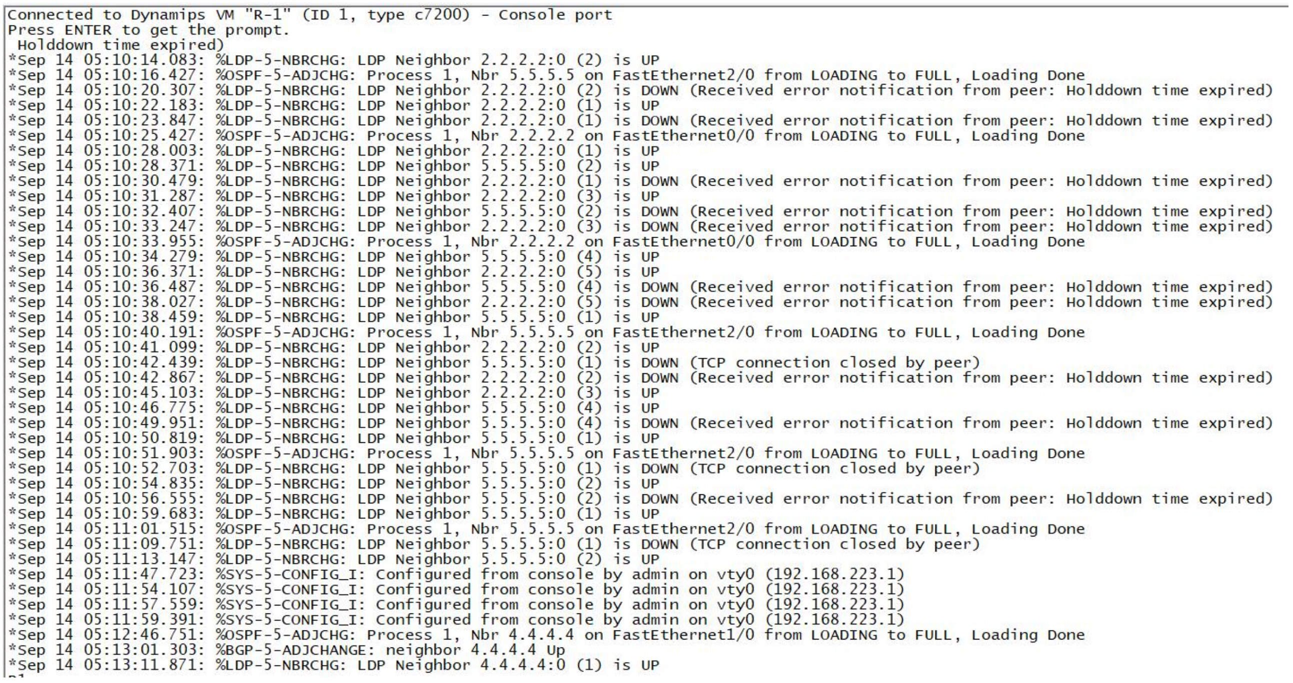


```
CE-1#ping 172.20.2.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.20.2.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 104/160/272 ms
```

Figure 4.2.1 : Getting ping from CE-1 to CE-2

4.2.2 Configuration Time :

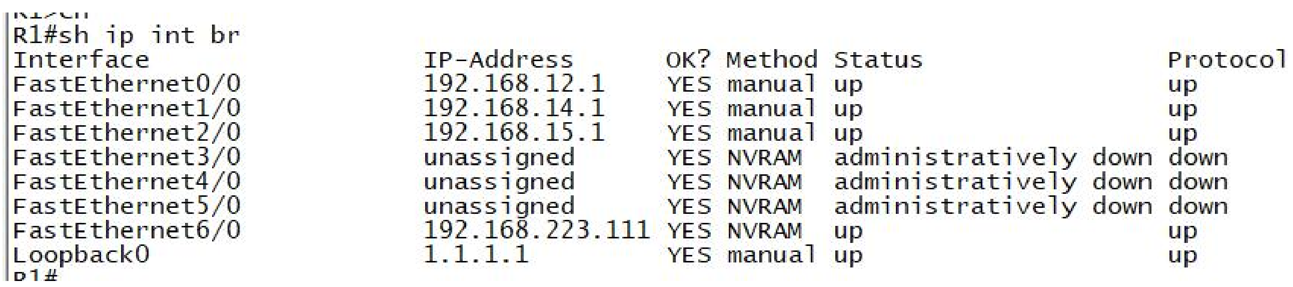


```
Connected to Dynamips VM "R-1" (ID 1, type c7200) - Console port
Press ENTER to get the prompt.
 Holddown time expired)
*Sep 14 05:10:14.083: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (2) is UP
*Sep 14 05:10:16.427: %OSPF-5-ADJCHG: Process 1, Nbr 5.5.5.5 on FastEthernet2/0 from LOADING to FULL, Loading Done
*Sep 14 05:10:20.307: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (2) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:22.183: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (1) is UP
*Sep 14 05:10:23.847: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (1) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:25.427: %OSPF-5-ADJCHG: Process 1, Nbr 2.2.2.2 on FastEthernet0/0 from LOADING to FULL, Loading Done
*Sep 14 05:10:28.003: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (1) is UP
*Sep 14 05:10:28.371: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (2) is UP
*Sep 14 05:10:30.479: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (1) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:31.287: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (3) is UP
*Sep 14 05:10:32.407: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (2) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:33.247: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (3) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:33.955: %OSPF-5-ADJCHG: Process 1, Nbr 2.2.2.2 on FastEthernet0/0 from LOADING to FULL, Loading Done
*Sep 14 05:10:34.279: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (4) is UP
*Sep 14 05:10:36.371: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (5) is UP
*Sep 14 05:10:36.487: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (4) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:38.027: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (5) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:38.459: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (1) is UP
*Sep 14 05:10:40.191: %OSPF-5-ADJCHG: Process 1, Nbr 5.5.5.5 on FastEthernet2/0 from LOADING to FULL, Loading Done
*Sep 14 05:10:41.099: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (2) is UP
*Sep 14 05:10:42.439: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (1) is DOWN (TCP connection closed by peer)
*Sep 14 05:10:42.867: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (2) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:45.103: %LDP-5-NBRCHG: LDP Neighbor 2.2.2.2:0 (3) is UP
*Sep 14 05:10:46.775: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (4) is UP
*Sep 14 05:10:49.951: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (4) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:50.819: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (1) is UP
*Sep 14 05:10:51.903: %OSPF-5-ADJCHG: Process 1, Nbr 5.5.5.5 on FastEthernet2/0 from LOADING to FULL, Loading Done
*Sep 14 05:10:52.703: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (1) is DOWN (TCP connection closed by peer)
*Sep 14 05:10:54.835: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (2) is UP
*Sep 14 05:10:56.555: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (2) is DOWN (Received error notification from peer: Holddown time expired)
*Sep 14 05:10:59.683: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (1) is UP
*Sep 14 05:11:01.515: %OSPF-5-ADJCHG: Process 1, Nbr 5.5.5.5 on FastEthernet2/0 from LOADING to FULL, Loading Done
*Sep 14 05:11:09.751: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (1) is DOWN (TCP connection closed by peer)
*Sep 14 05:11:13.147: %LDP-5-NBRCHG: LDP Neighbor 5.5.5.5:0 (2) is UP
*Sep 14 05:11:47.723: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (192.168.223.1)
*Sep 14 05:11:54.107: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (192.168.223.1)
*Sep 14 05:11:57.559: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (192.168.223.1)
*Sep 14 05:11:59.391: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (192.168.223.1)
*Sep 14 05:12:46.751: %OSPF-5-ADJCHG: Process 1, Nbr 4.4.4.4 on FastEthernet1/0 from LOADING to FULL, Loading Done
*Sep 14 05:13:01.303: %BGP-5-ADJCHANGE: neighbor 4.4.4.4 Up
*Sep 14 05:13:11.871: %LDP-5-NBRCHG: LDP Neighbor 4.4.4.4:0 (1) is UP
```

Figure 4.2.2: Consuming Configuration Time

From the logs it is evident that the whole automation process took roughly 3 minutes and no errors were reported while configuring the automation. This shows how the use of Python automation is effective in configuring number of devices and as well protocols. Unlike when it is done manually that might have been slower and prone to human errors if not well done, the automation script helps in getting the configurations done faster and at a go across the intended devices.

4.2.3 IP Address Configuration :

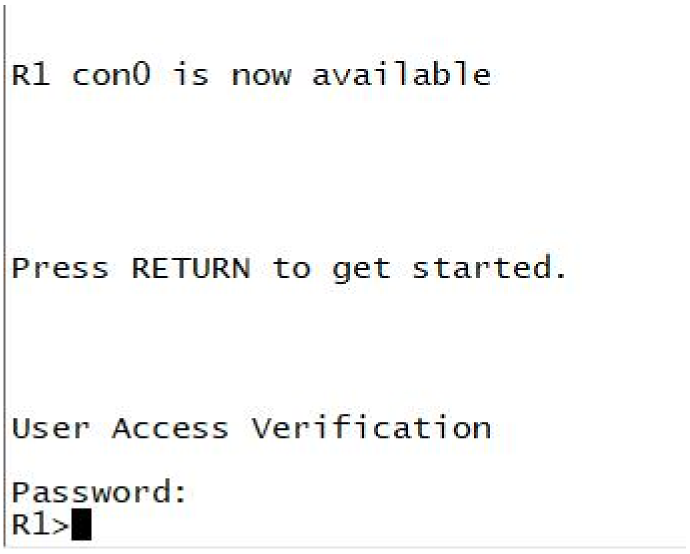


```
R1#sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            192.168.12.1    YES manual up                    up
FastEthernet1/0            192.168.14.1    YES manual up                    up
FastEthernet2/0            192.168.15.1    YES manual up                    up
FastEthernet3/0            unassigned      YES NVRAM  administratively down down
FastEthernet4/0            unassigned      YES NVRAM  administratively down down
FastEthernet5/0            unassigned      YES NVRAM  administratively down down
FastEthernet6/0            192.168.223.111 YES NVRAM  up                    up
Loopback0                  1.1.1.1         YES manual up                    up
```

Figure 4.2.3 Routing Table after Automation

It is also evident that the IP addresses assigned to the interfaces of R1 matches the specifications shown by the output of the show ip interface brief command in the show_py_script Rica. This confirms the validity of automation process as pertains to IP assignment hence establishing that the script was in fact effective in programing the desired network setting.

4.2.4 Password Configuration:



```
R1 con0 is now available



Press RETURN to get started.



User Access Verification

Password:
R1>
```

Figure 4.2.4: setting password through console

The console prompt also displays that the password for R1 has been successfully implemented, for the device asks for the encrypted password of ‘Maheer’ while entering the login details. This confirms that all aspects of the devices were made secure by passwords and other encryptions that were instated using the automation script.

These results give very high approval of the extent of the automation process. The automated configuration was much faster than the manual configuration, and the types of mistakes which could be made during the manual process were also avoided.

4.3 The Scalability of Automation

In fact, if one had to configure six routers by hand, while it certainly is more doable than configuring twenty routers, there efficiency of automation increases even more. If I had to set up a thousand routers individually I would be extremely slow in doing so and the chances of making mistakes would be very high. But it's a known fact that as the number of devices grows so does the chances of human mistake: configuration will be done wrong, connections will be unstable, and networks can fail.

On the other hand, the same automated script would manage 1, 000 routers with the same ease that was observed while managing six routers. The time required to configure all the routers would be directly proportional to the number of routers while the chances of a human error would reduce. This scalability makes it mandatory to automate some of the processes in the large network because this reduces time and effort taken in configuration and also minimize the occurrences of error.

4.4 Automation Saves Time and Reduces Errors

The project also shows that the degree of automation leads to a huge amount of time-saving. I was able to configure all the six routers within about three minutes in an automatic manner as opposed to when I was having to do it using the manual procedures, which would have taken a lot more time besides having to introduce some troubleshooting strategies in the process. In addition to saving time, the automation reduces prospects of making a mistake which was evident when manually configuring connections where a single mistake resulted in connection fail.

Automated scripts guarantee that properly working configurations are reproduced throughout the environment, which directly affects the network's stability in the RTC systems' operational environments.

5.0 Future Work

In my first project proposal, I was planning to work on the higher level of protocols, like RESTCONF and NETCONF. But because of time spent and the current state of my skills, I opted for CLI based automation using the Netmiko which provides a good ground on Python based automation for network configurations.

5.0.1 RESTCONF and NETCONF

NETCONF and RESTCONF are two other protocols that provide the modern and scalable way for configuration of networks using automations. Unlike CLI based manual script automation, these protocols utilize data models such as YANG and allows management though interfaces. It is used primarily for SDN and NFV as it provided even more versatility in the multi-supplier environment.
Although I could not implement these protocols in this project, my future goal is to:
Learn and Master RESTCONF and NETCONF: Learn what use of Python libraries such as requests for RESTCONF and ncclient for NETCONF entails.
Implementation: These Protocols into your existing project in order to expand the horizon well beyond levels of traditional CLI based automation.
Explore YANG Data Models: Keep the configuration and management of the networks looking more valued, structured based and easily scalable for large and complex structures.

5.0.2 Expanding Beyond CLI Automation

Although, CLI based automation with netmiko was quite useful, today's network requires feature such as RESTCONF, NETCONF etc. Specifically, the proportion of the configurations that will be required to handle via the APIs will grow as the network size expands. My future work will focus on transitioning to full automation by:My future work will focus on transitioning to full automation by:

Automating at Scale: NETCONF Y RESTCONF are more appropriated for the administration of large networks that include hundreds or thousands of devices.

Eliminating Manual Tasks: Towards achieving a fully hands-off means of network configuration, which would improve not only the speed, but also the accuracy.

5.1 Conclusion:

In this project, I looked into how Python can be used in automating configurations of IP/MPLS network especial with respect to core network protocol such as OSPF, BGP and MPLS. In this scenario, I was able to show that automation via CLI, using the Netmiko library, could drop the amount of configuration time drastically, reduce the potential for mistakes, and ensure that multiple devices are set up in the exact same manner. The automation configuration was done in the right manner as it only took three minutes to configure six routers while the manual method is tiresome and tends to involve several errors. Although I faced some problems with manual adjustments for example diagnosing errors that resulted into ceasing, the automation scripts reduced these problems, I was able to perform 100 % ping tests between CE-1 and CE-2 without packet loss. In the future, the project provides a solid basis for further development in network automation by examining RESTCONF and NETCONF, both of which are more manageable through API protocol. These modern methods are useful in controlling large networks that may consist of hundreds if not thousands of devices; hence Network Automation becomes a vital component of service providers. The experiences in this project will be looked at while working towards the optimal goal of having full automation in network, its efficiency and accuracy.

6. References:

1. Ayoub Bahnasse, Talea, M., Badri, A., & Fatima Ezzahraa Louhab. (2018). New smart platform for automating MPLS virtual private network simulation. *2018 International Conference on Advanced Communication Technologies and Networking (CommNet)*. https://doi.org/10.1109/commnet.2018.8360268
2. Dany, P., Alphonse Binele Abana, Tonye, E., Salomon, P., & Gilles. (2023). Optimizing Telematics Network Performance through Resource Virtualization in a Disruptive Environment: The Case of the IP/MPLS Core Network. *Network and Communication Technologies*, *8*(2), 1–1. https://doi.org/10.5539/nct.v8n2p1
3. Koskinen, J. (2021). *Optimization of BGP Convergence and Prefix Security in IP/MPLS Networks*. Utupub.fi. https://www.utupub.fi/handle/10024/152670

4. A. Seck, C. S. E. Bassene, S. E. Zabolo and S. Ouya,(2022). *Building an interactive Software Defined Network from the MPSI for MPLS Service provisioning with Gitlab and Ansible*. International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Maldives, Maldives, 2022, pp. 1-6, doi: 10.1109/ICECCME55909.2022.9988003

5. Διπλωματικές Εργασίες - Πατρικάκης Χαράλαμπος. (2021). A survey on next generation networks and cloud computing | Διπλωματικές Εργασίες - Πατρικάκης Χαράλαμπος. Uniwa.gr. https://greyliterature.consert.eee.uniwa.gr/project/cloud-computing-network-virtualization

6. Ridwan, M. A., Radzi, N. A. M., Wan Ahmad, W. S. M., Abdullah, F., Jamaludin, Md. Zaini., & Zakaria, M. (2020). *Recent trends in MPLS networks: technologies, applications and challenges*. IET Communications, 14(2), 177–185. https://doi.org/10.1049/iet-com.2018.6129

7. Gibadullina, E., Viskova, E., & Stepanov, S. (2022). *Automated Service Configuration Management in IP/MPLS Networks*. https://doi.org/10.1109/monetec55448.2022.9960765
8. Kamoun, F., & Outay, F. (2018). IP/MPLS networks with hardened pipes: service concepts, traffic engineering and design considerations. *Journal of Ambient Intelligence and Humanized Computing*, *10*(7), 2577–2584. https://doi.org/10.1007/s12652-018-0734-2
9. Monerah Al-Mekhlal, Abdulrahman AlYahya, Abdullah Aldhamin, & Khan, A. (2022). *Network Automation Python-based Application: The performance of a Multi-Layer Cloud Based Solution*. https://doi.org/10.1109/coins54846.2022.9854953
10. Mazin, A. M., Rahman, R. A., Kassim, M., & Mahmud, A. R. (2021, April 1). *Performance Analysis on Network Automation Interaction with Network Devices Using Python*. IEEE Xplore. https://doi.org/10.1109/ISCAIE51753.2021.9431823
11. Hart, N., Charalampos Rotsos, Vasileios Giotsas, Race, N., & Hutchison, D. (2020). λBGP: Rethinking BGP programmability. *Lancaster EPrints (Lancaster University)*. https://doi.org/10.1109/noms47738.2020.9110331
12. Milios, G. (2021). Network Automation using Python. *Ihu.edu.gr*. https://repository.ihu.edu.gr//xmlui/handle/11544/29802
13. Mihăilă, P., Bălan, T., Curpen, R., & Sandu, F. (2017). Network Automation and Abstraction using Python Programming Methods. *MACRo 2015*, *2*(1), 95–103. https://doi.org/10.1515/macro-2017-0011
14. An, N. (2023). Network Automation with Python. *Theseus.fi*. http://www.theseus.fi/handle/10024/802362
15. Mochamad Yazid Gupron, Umaisaroh Umaisaroh, & Agung Wicaksono. (2022). *Network Automation for CE Router with Route Leaking in MPLS-VPN Network*. https://doi.org/10.1109/isitdi55734.2022.9944460
16. Altalebi, O., & Abdullahi Abdu Ibrahim. (2024). *Automation of Traditional Networks: A Mini-Review*. *3*, 1–8. https://doi.org/10.1109/iccsc62074.2024.10616419
17. Chen, M., Zhang, S., Deng, H., Chen, B., Xing, C., & Xu, B. (2020). Automatic deployment and control of network services in NFV environments. *Journal of Network and Computer Applications*, *164*, 102677. https://doi.org/10.1016/j.jnca.2020.102677
18. Golmohammadi, Z. (2019). Centralized model driven trace route mechanism for TCP/IP routers : Remote traceroute invocation using NETCONF API and YANG data model. *Tuni.fi*. https://trepo.tuni.fi/handle/10024/117763
19. Anderson, J., Steinmacher, I., & Rodeghero, P. (2020). *Assessing the Characteristics of FOSS Contributions in Network Automation Projects*. https://doi.org/10.1109/icsme46990.2020.00039
20. Elezi, A., & Karras, D. (2023). On automating network systems configuration management. *CRJ*, 18–31. https://doi.org/10.59380/crj.v1i1.639
21. Andersson, M., & Wedin, R. (2006). Python scripting for network management: PyMIP-TeMIP made simple.diva-portal.org FULLTEXT01.pdf (diva-portal.org)
22. Muhammad, T., & Munir, M. (2023). Network Automation. *European Journal of Technology*, *7*(2), 23–42. https://doi.org/10.47672/ejt.1547
23. Saminderjit Singh Chahal. (2016). *SDN- Data Center Automation with Firewall as a Service*. https://doi.org/10.7939/r3-rwr8-8c98
24. Rodgers, C. (2018). Building a MPLS Based Telecommunication Network. Journal of Telecommunications System & Management, 07(03). https://doi.org/10.4172/2167-0919.1000175
25. Baumgartner, F., Braun, T., & Bhargava, B. (2002). Design and Implementation of a Python-Based Active Network Platform for Network Management and Control. Lecture Notes in Computer Science, 177–190. https://doi.org/10.1007/3-540-36199-5_14

26. Python learning for Network Engineers | Part 01 | Introduction | Python for Network Automation (youtube.com)
27. Python for Network Engineers | Free Scripts & Training | Python Network Automation Network Engineer Tutorial (rogerperkin.co.uk)
28. netmiko · PyPI
29. https://www.ip6net.net/power-outage-affects-large-parts-of-bts-broadband-network/
30. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/network-automation-strategy-wp.html
31. https://www.cisco.com/c/en_uk/solutions/automation/network-automation.html

7. Appendix:

1.0 Code for Python-Based Network Automation and Manual Configuration

1.1 Interface Configuration (interface_config.py)

```
interface_configs = {
    'R1': [
        'interface f0/0',
        'ip address 192.168.12.1 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f1/0',
        'ip address 192.168.14.1 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f2/0',
        'ip address 192.168.15.1 255.255.255.0',
        'no shutdown',
        'exit',
        'interface lo0',
        'ip address 1.1.1.1 255.255.255.255',
        'exit',
    ],
    'R2': [
        'interface f0/0',
        'ip address 192.168.12.2 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f1/0',
        'ip address 192.168.23.2 255.255.255.0',
        'no shutdown',
        'exit',
        'interface lo0',
        'ip address 2.2.2.2 255.255.255.255',
        'exit',

    ],
    'R3': [
        'interface f0/0',
        'ip address 192.168.34.3 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f1/0',
        'ip address 192.168.23.3 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f2/0',
        'ip address 192.168.16.3 255.255.255.0',
        'no shutdown',
        'exit',
        'interface lo0',
        'ip address 3.3.3.3 255.255.255.255',
        'exit',
    ],
    'R4': [
        'interface f0/0',
        'ip address 192.168.34.4 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f1/0',
        'ip address 192.168.14.4 255.255.255.0',
        'no shutdown',
        'exit',
        'interface lo0',
        'ip address 4.4.4.4 255.255.255.255',
        'exit',
    ],

    'PE-1': [
        'interface f0/0',
        'ip address 172.10.1.2 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f2/0',
        'ip address 192.168.15.2 255.255.255.0',
        'no shutdown',
        'exit',
        'interface lo0',
        'ip address 5.5.5.5 255.255.255.255',
        'exit',
    ],
    'PE-2': [
        'interface f0/0',
        'ip address 172.20.2.2 255.255.255.0',
        'no shutdown',
        'exit',
        'interface f2/0',
        'ip address 192.168.16.2 255.255.255.0',
        'no shutdown',
        'exit',
        'interface lo0',
        'ip address 6.6.6.6 255.255.255.255',
        'exit',
    ],

}
```

1.2 BGP Configuration (bgp_config.py)

```
bgp_configs = {
    'R1': [
        'router bgp 1',
        'bgp router-id 1.1.1.1',
        'neighbor 4.4.4.4 remote-as 1',
        'neighbor 4.4.4.4 update-source Loopback0',
        'address-family vpnv4',
        'neighbor 4.4.4.4 activate',
        'neighbor 4.4.4.4 send-community extended',
        'exit-address-family',
    ],
    'R2': [
        'router bgp 1',
        'bgp router-id 2.2.2.2',
        'neighbor 4.4.4.4 remote-as 1',
        'neighbor 4.4.4.4 update-source Loopback0',
        'address-family vpnv4',
        'neighbor 4.4.4.4 activate',
        'neighbor 4.4.4.4 send-community extended',
        'exit-address-family',
    ],
    'R3': [
        'router bgp 1',
        'bgp router-id 3.3.3.3',
        'neighbor 4.4.4.4 remote-as 1',
        'neighbor 4.4.4.4 update-source Loopback0',
        'address-family vpnv4',
        'neighbor 4.4.4.4 activate',
        'neighbor 4.4.4.4 send-community extended',
        'exit-address-family',
    ],
    'R4': [
        'router bgp 1',
        'bgp router-id 4.4.4.4',

        'neighbor 1.1.1.1 remote-as 1',
        'neighbor 1.1.1.1 update-source Loopback0',
        'neighbor 2.2.2.2 remote-as 1',
        'neighbor 2.2.2.2 update-source Loopback0',
        'neighbor 3.3.3.3 remote-as 1',
        'neighbor 3.3.3.3 update-source Loopback0',
        'neighbor 5.5.5.5 remote-as 1',
        'neighbor 5.5.5.5 update-source Loopback0',
        'neighbor 6.6.6.6 remote-as 1',
        'neighbor 6.6.6.6 update-source Loopback0',

        'address-family ipv4',
        'neighbor 1.1.1.1 activate',
        'neighbor 2.2.2.2 activate',
        'neighbor 3.3.3.3 activate',
        'neighbor 5.5.5.5 activate',
        'neighbor 6.6.6.6 activate',
        'exit-address-family',

        'address-family vpnv4',
        'neighbor 1.1.1.1 activate',
        'neighbor 1.1.1.1 send-community extended',
        'neighbor 1.1.1.1 route-reflector-client',
        'neighbor 2.2.2.2 activate',
        'neighbor 2.2.2.2 send-community extended',
        'neighbor 2.2.2.2 route-reflector-client',
        'neighbor 3.3.3.3 activate',
        'neighbor 3.3.3.3 send-community extended',
        'neighbor 3.3.3.3 route-reflector-client',
        'neighbor 5.5.5.5 activate',
        'neighbor 5.5.5.5 send-community extended',
        'neighbor 5.5.5.5 route-reflector-client',
        'neighbor 6.6.6.6 activate',
        'neighbor 6.6.6.6 send-community extended',
        'neighbor 6.6.6.6 route-reflector-client',
        'exit-address-family',
    ],
    'PE-1': [
        'router bgp 1',
        'bgp router-id 5.5.5.5',
        'neighbor 4.4.4.4 remote-as 1',
        'neighbor 4.4.4.4 update-source Loopback0',
        'address-family vpnv4',
        'neighbor 4.4.4.4 activate',
        'neighbor 4.4.4.4 send-community extended',
        'exit-address-family',
    ],
    'PE-2': [
        'router bgp 1',
        'bgp router-id 6.6.6.6',
        'neighbor 4.4.4.4 remote-as 1',
        'neighbor 4.4.4.4 update-source Loopback0',
        'address-family vpnv4',
        'neighbor 4.4.4.4 activate',
        'neighbor 4.4.4.4 send-community extended',
        'exit-address-family',
    ],

}
```

1.3 OSPF Configuration (ospf_config.py)

```
ospf_configs = {
    'R1': [
        'router ospf 1',
        'router-id 1.1.1.1',
        'passive-interface Loopback0',
        'int f0/0',
        'ip ospf 1 area 0',
        'int f1/0',
        'ip ospf 1 area 0',
        'int f2/0',
        'ip ospf 1 area 0',
        'int lo0',
        'ip ospf 1 area 0',
        'exit',
    ],
    'R2': [
        'router ospf 1',
        'router-id 2.2.2.2',
        'passive-interface Loopback0',
        'int f0/0',
        'ip ospf 1 area 0',
        'int f1/0',
        'ip ospf 1 area 0',
        'int lo0',
        'ip ospf 1 area 0',
        'exit',
    ],
    'R3': [
        'router ospf 1',
        'router-id 3.3.3.3',
        'passive-interface Loopback0',
        'int f0/0',
        'ip ospf 1 area 0',
        'int f1/0',
        'ip ospf 1 area 0',
        'int f2/0',
        'ip ospf 1 area 0',
        'int lo0',
        'ip ospf 1 area 0',
        'exit',
    ],
    'R4': [
        'router ospf 1',
        'router-id 4.4.4.4',
        'passive-interface Loopback0',
        'int f0/0',
        'ip ospf 1 area 0',
        'int f1/0',
        'ip ospf 1 area 0',
        'int lo0',
        'ip ospf 1 area 0',
        'exit',
    ],
    'PE-1': [
        'router ospf 1',
        'router-id 5.5.5.5',
        'passive-interface Loopback0',
        'int f2/0',
        'ip ospf 1 area 0',
        'int lo0',
        'ip ospf 1 area 0',
        'exit',
    ],
    'PE-2': [
        'router ospf 1',
        'router-id 6.6.6.6',
        'passive-interface Loopback0',
        'int f2/0',
        'ip ospf 1 area 0',
        'int lo0',
        'ip ospf 1 area 0',
        'exit',
    ],
}
```

1.4 MPLS Configuration (mpls_config.py)

```
mpls_configs = {
    'R1': [
        'int f0/0',
        'mpls ip',
        'int f1/0',
        'mpls ip',
        'int f2/0',
        'mpls ip',
        'exit',
    ],
    'R2': [
        'int f0/0',
        'mpls ip',
        'int f1/0',
        'mpls ip',
        'exit',
    ],
    'R3': [
        'int f0/0',
        'mpls ip',
        'int f1/0',
        'mpls ip',
        'int f2/0',
        'mpls ip',
        'exit',
    ],
    'R4': [
        'int f0/0',
        'mpls ip',
        'int f1/0',
        'mpls ip',
        'exit',
    ],
    'PE-1': [
        'int f2/0',
        'mpls ip',
        'exit',
    ],
    'PE-2': [
        'int f2/0',
        'mpls ip',
        'exit',
    ],
}
```

1.5 Network Automation Orchestration (config_network.py)

```
# configure_network.py

# Import necessary libraries and configurations
from netmiko import ConnectHandler
from devices import devices
from interface_config import interface_configs
from ospf_config import ospf_configs
from bgp_config import bgp_configs
from mpls_config import mpls_configs

# Function to configure a device
def configure_device(device, config_sets):
    try:
        # Connect to the device
        connection = ConnectHandler(**device)

        # Enter enable mode
        connection.enable()

        # Apply each set of configurations
        for config_set in config_sets:
            if config_set:
                output = connection.send_config_set(config_set)
                print(f'Configuration Output for {device["host"]}:')
                print(output)

        # Disconnect from the device
        connection.disconnect()

    # Handle any exceptions during configuration
    except Exception as e:
        print(f'Failed to configure {device["host"]}: {e}')

# Main function to iterate over all devices and apply configurations
def main():
    for router_name, device in devices.items():
        print(f'Configuring {router_name} ({device["host"]})...')
        # Gather config sets for each device
        config_sets = [
            interface_configs.get(router_name),
            ospf_configs.get(router_name),
            bgp_configs.get(router_name),
            mpls_configs.get(router_name),
        ]

        # Configure the device
        configure_device(device, config_sets)

# Start the configuration process
if __name__ == "__main__":
    main()
```

1.6 Device Information (devices.py)

```
#Storing details of all network devices
devices = {
    'R1': {
        'device_type': 'cisco_ios', # Device type for Netmiko connection
        'host': '192.168.223.111', # Device IP address
        'username': 'admin', # SSH username
        'password': 'cisco', # SSH password
        'port': '22', # SSH port
        'secret': 'cisco', # Enable secret (privilege mode password)
    },
    'R2': {
        'device_type': 'cisco_ios',
        'host': '192.168.223.112',
        'username': 'admin',
        'password': 'cisco',
        'port': '22',
        'secret': 'cisco',
    },
    'R3': {
        'device_type': 'cisco_ios',
        'host': '192.168.223.113',
        'username': 'admin',
        'password': 'cisco',
        'port': '22',
        'secret': 'cisco',
    },
    'R4': {
        'device_type': 'cisco_ios',
        'host': '192.168.223.114',
        'username': 'admin',
        'password': 'cisco',
        'port': '22',
        'secret': 'cisco',
    },
    'PE-1': {
        'device_type': 'cisco_ios',
        'host': '192.168.223.115',
        'username': 'admin',
        'password': 'cisco',
        'port': '22',
        'secret': 'cisco',
    },
    'PE-2': {
        'device_type': 'cisco_ios',
        'host': '192.168.223.116',
        'username': 'admin',
        'password': 'cisco',
        'port': '22',
        'secret': 'cisco',
    },

}
```

Manual configuration:

1.7 Management Script

The manual configurations which I did manually back of the automation :

```
1. Management Script
====================
-------------
R1# For getting IP from my home internet
-------------
en
conf t
hostname R1
int f0/0  # Configuring interface f0/0
ip address dhcp  # Set IP address via DHCP
no shut  # Enable interface
end
-------------

Configurations
====================
-------------
R1#
-------------
```

```
en
conf t
hostname R1  # Setting hostname to R1
no ip domain-lookup  # Disabling domain lookup
no cdp run  # Disabling Cisco Discovery Protocol

!
# Configuring interface f6/0 with static IP
int f6/0
ip address 192.168.223.111 255.255.255.0  # Assigning static IP
no shut  # Enabling interface
duplex full  # Setting full duplex
!
!
# Configuring SSH for secure access
ip domain-name ciscopython.com  # Setting domain name for SSH
crypto key generate rsa # Generating RSA keys for encryption
1024
!
username admin privilege 15 secret cisco  # Creating user with privileged access
ip ssh version 2  # Enabling SSH version 2
!
# Configuring vty lines for remote access
line vty 0 15
login local
transport input all
exit
!
end
wr # Save the configuration
!

-------------
R2#
-------------
en
conf t
hostname R2
no ip domain-lookup
!
int f6/0
ip address 192.168.223.112 255.255.255.0
no shut
duplex full
!
!
ip domain-name ciscopython.com
crypto key generate rsa
1024
!
username admin privilege 15 secret cisco
ip ssh version 2
!
line vty 0 15
login local
transport input all
exit
!
end
wr
!

-------------
R3#
-------------
en
conf t
hostname R3
no ip domain-lookup
!
int f6/0
ip address 192.168.223.113 255.255.255.0
no shut
duplex full
!
!
ip domain-name ciscopython.com
crypto key generate rsa
```

```
1024
!
username admin privilege 15 secret cisco
ip ssh version 2
!
line vty 0 15
login local
transport input all
exit
!
end
wr
!

-------------
R4#
-------------
en
conf t
hostname R4
no ip domain-lookup
!
int f6/0
ip address 192.168.223.114 255.255.255.0
no shut
duplex full
!
!
ip domain-name ciscopython.com
crypto key generate rsa
1024
!
username admin privilege 15 secret cisco
ip ssh version 2
!
line vty 0 15
login local
transport input all
exit
!
end
wr
!

-------------
PE-1#
-------------
en
conf t
hostname PE-1
no ip domain-lookup
!
int f6/0
ip address 192.168.223.115 255.255.255.0
no shut
duplex full
!
!
ip domain-name ciscopython.com
crypto key generate rsa
1024
!
username admin privilege 15 secret cisco
ip ssh version 2
!
line vty 0 15
login local
transport input all
exit
!
end
wr
!

-------------
PE-2#
-------------
en
```

```
conf t
hostname PE-2
no ip domain-lookup
!
int f6/0
ip address 192.168.223.116 255.255.255.0
no shut
duplex full
!
!
ip domain-name ciscopython.com
crypto key generate rsa
1024
!
username admin privilege 15 secret cisco
ip ssh version 2
!
line vty 0 15
login local
transport input all
exit
!
end
wr
!

=================================
For all routers
=================================
en
conf t
!
line console 0
password Maheer
login
exit
!
exit
wr
!
exit
!
====================
```

1.8 Client Script

The configurations for Client which configured manually

```
5. CE-to-PE Connections
=========================
------------------
CE-1
------------
en
conf t
host CE-1
!
int f0/0
ip address 172.10.1.1 255.255.255.252
no shut
!
exit
ip route 0.0.0.0 0.0.0.0 172.10.1.2
!
end
!
wr
!

------------------
CE-2
------------
en
conf t
host CE-2
!
```

```
int f0/0
ip address 172.20.2.1 255.255.255.252
no shut
!
exit
ip route 0.0.0.0 0.0.0.0 172.20.2.2
!
end
!
wr
!


------------------
PE-1
------------
en
conf t
ip vrf CUST_VRF
 rd 1:10
 route-target export 1:10
 route-target import 1:10
exit

int f0/0
ip vrf forwarding CUST_VRF
ip address 172.10.1.2 255.255.255.252
no shut
!
router bgp 1
  address-family ipv4 vrf CUST_VRF
    redistribute connected
  exit-address-family
!
!
end
!
wr
!

------------------
PE-2
------------
en
conf t
ip vrf CUST_VRF
 rd 1:10
 route-target export 1:10
 route-target import 1:10
exit
!
int f0/0
ip vrf forwarding CUST_VRF
ip address 172.20.2.2 255.255.255.252
no shut
!
router bgp 1
  address-family ipv4 vrf CUST_VRF
    redistribute connected
  exit-address-family
!
!
end
!
wr
!
```

1.1 Visual Studio Command prompt details

```
R1(config)#interface f2/0
R1(config-if)#ip address 192.168.15.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface lo0
R1(config-if)#ip address 1.1.1.1 255.255.255.255
R1(config-if)#exit
```

```
R1(config)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router ospf 1
R1(config-router)#router-id 1.1.1.1
R1(config-router)#passive-interface Loopback0
R1(config-router)#int f0/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int f1/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int f2/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int lo0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#exit
R1(config)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router bgp 1
R1(config-router)#bgp router-id 1.1.1.1
R1(config-router)#neighbor 4.4.4.4 remote-as 1
R1(config-router)#neighbor 4.4.4.4 update-source Loopback0
R1(config-router)#address-family vpnv4
R1(config-router-af)#neighbor 4.4.4.4 activate
R1(config-router-af)#neighbor 4.4.4.4 send-community extended
R1(config-router-af)#exit-address-family
R1(config-router)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#int f0/0
R1(config-if)#mpls ip
R1(config-if)#int f1/0
R1(config-if)#mpls ip
R1(config-if)#int f2/0
R1(config-if)#mpls ip
R1(config-if)#exit
R1(config)#end
R1#
Configuring R2 (192.168.223.112)...
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#interface f0/0
R2(config-if)#ip address 192.168.12.2 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
```

```
R2(config)#interface f1/0
R2(config-if)#ip address 192.168.23.2 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface lo0
R2(config-if)#ip address 2.2.2.2 255.255.255.255
R2(config-if)#exit
R2(config)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router ospf 1
R2(config-router)#router-id 2.2.2.2
R2(config-router)#passive-interface Loopback0
R2(config-router)#int f0/0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#int f1/0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#int lo0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#exit
R2(config)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router bgp 1
R2(config-router)#bgp router-id 2.2.2.2
R2(config-router)#neighbor 4.4.4.4 remote-as 1
R2(config-router)#neighbor 4.4.4.4 update-source Loopback0
R2(config-router)#address-family vpnv4
R2(config-router-af)#neighbor 4.4.4.4 activate
R2(config-router-af)#neighbor 4.4.4.4 send-community extended
R2(config-router-af)#exit-address-family
R2(config-router)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#int f0/0
R2(config-if)#mpls ip
R2(config-if)#int f1/0
R2(config-if)#mpls ip
R2(config-if)#exit
R2(config)#end
R2#
Configuring R3 (192.168.223.113)...
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#interface f0/0
```

```
R3(config-if)#ip address 192.168.34.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface f1/0
R3(config-if)#ip address 192.168.23.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface f2/0
R3(config-if)#ip address 192.168.16.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface lo0
R3(config-if)#ip address 3.3.3.3 255.255.255.255
R3(config-if)#exit
R3(config)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router ospf 1
R3(config-router)#router-id 3.3.3.3
R3(config-router)#passive-interface Loopback0
R3(config-router)#int f0/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int f1/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int f2/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int lo0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#exit
R3(config)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router bgp 1
R3(config-router)#bgp router-id 3.3.3.3
R3(config-router)#neighbor 4.4.4.4 remote-as 1
R3(config-router)#neighbor 4.4.4.4 update-source Loopback0
R3(config-router)#address-family vpnv4
R3(config-router-af)#neighbor 4.4.4.4 activate
R3(config-router-af)#neighbor 4.4.4.4 send-community extended
R3(config-router-af)#exit-address-family
R3(config-router)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#int f0/0
R3(config-if)#mpls ip
R3(config-if)#int f1/0
```

```
R3(config-if)#mpls ip
R3(config-if)#int f2/0
R3(config-if)#mpls ip
R3(config-if)#exit
R3(config)#end
R3#
Configuring R4 (192.168.223.114)...
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface f0/0
R4(config-if)#ip address 192.168.34.4 255.255.255.0
R4(config-if)#no shutdown
R4(config-if)#exit
R4(config)#interface f1/0
R4(config-if)#ip address 192.168.14.4 255.255.255.0
R4(config-if)#no shutdown
R4(config-if)#exit
R4(config)#interface lo0
R4(config-if)#ip address 4.4.4.4 255.255.255.255
R4(config-if)#exit
R4(config)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#router ospf 1
R4(config-router)#router-id 4.4.4.4
R4(config-router)#passive-interface Loopback0
R4(config-router)#int f0/0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#int f1/0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#int lo0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#exit
R4(config)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#router bgp 1
R4(config-router)#bgp router-id 4.4.4.4
R4(config-router)#neighbor 1.1.1.1 remote-as 1
R4(config-router)#neighbor 1.1.1.1 update-source Loopback0
R4(config-router)#neighbor 2.2.2.2 remote-as 1
R4(config-router)#neighbor 2.2.2.2 update-source Loopback0
R4(config-router)#neighbor 3.3.3.3 remote-as 1
R4(config-router)#neighbor 3.3.3.3 update-source Loopback0
R4(config-router)#neighbor 5.5.5.5 remote-as 1
R4(config-router)#neighbor 5.5.5.5 update-source Loopback0
R4(config-router)#neighbor 6.6.6.6 remote-as 1
```

```
R4(config-router)#neighbor 6.6.6.6 update-source Loopback0
R4(config-router)#address-family ipv4
R4(config-router-af)#neighbor 1.1.1.1 activate
R4(config-router-af)#neighbor 2.2.2.2 activate
R4(config-router-af)#neighbor 3.3.3.3 activate
R4(config-router-af)#neighbor 5.5.5.5 activate
R4(config-router-af)#neighbor 6.6.6.6 activate
R4(config-router-af)#exit-address-family
R4(config-router)#address-family vpnv4
R4(config-router-af)#neighbor 1.1.1.1 activate
R4(config-router-af)#neighbor 1.1.1.1 send-community extended
R4(config-router-af)#neighbor 1.1.1.1 route-reflector-client
R4(config-router-af)#neighbor 2.2.2.2 activate
R4(config-router-af)#neighbor 2.2.2.2 send-community extended
R4(config-router-af)#neighbor 2.2.2.2 route-reflector-client
R4(config-router-af)#neighbor 3.3.3.3 activate
R4(config-router-af)#neighbor 3.3.3.3 send-community extended
R4(config-router-af)#neighbor 3.3.3.3 route-reflector-client
R4(config-router-af)#neighbor 5.5.5.5 activate
R4(config-router-af)#neighbor 5.5.5.5 send-community extended
R4(config-router-af)#neighbor 5.5.5.5 route-reflector-client
R4(config-router-af)#neighbor 6.6.6.6 activate
R4(config-router-af)#neighbor 6.6.6.6 send-community extended
R4(config-router-af)#neighbor 6.6.6.6 route-reflector-client
R4(config-router-af)#exit-address-family
R4(config-router)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#int f0/0
R4(config-if)#mpls ip
R4(config-if)#int f1/0
R4(config-if)#mpls ip
R4(config-if)#exit
R4(config)#end
R4#
Configuring PE-1 (192.168.223.115)...
Failed to configure 192.168.223.115: TCP connection to device failed.

Common causes of this problem are:
1. Incorrect hostname or IP address.
2. Wrong TCP port.
3. Intermediate firewall blocking access.

Device settings: cisco_ios 192.168.223.115:22


Configuring PE-2 (192.168.223.116)...
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
```

```
PE-2(config)#interface f0/0
PE-2(config-if)#ip address 172.20.2.2 255.255.255.0
PE-2(config-if)#no shutdown
PE-2(config-if)#exit
PE-2(config)#interface f2/0
PE-2(config-if)#ip address 192.168.16.2 255.255.255.0
PE-2(config-if)#no shutdown
PE-2(config-if)#exit
PE-2(config)#interface lo0
PE-2(config-if)#ip address 6.6.6.6 255.255.255.255
PE-2(config-if)#exit
PE-2(config)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#router ospf 1
PE-2(config-router)#router-id 6.6.6.6
PE-2(config-router)#passive-interface Loopback0
PE-2(config-router)#int f2/0
PE-2(config-if)#ip ospf 1 area 0
PE-2(config-if)#int lo0
PE-2(config-if)#ip ospf 1 area 0
PE-2(config-if)#exit
PE-2(config)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#router bgp 1
PE-2(config-router)#bgp router-id 6.6.6.6
PE-2(config-router)#neighbor 4.4.4.4 remote-as 1
PE-2(config-router)#neighbor 4.4.4.4 update-source Loopback0
PE-2(config-router)#address-family vpnv4
PE-2(config-router-af)#neighbor 4.4.4.4 activate
PE-2(config-router-af)#neighbor 4.4.4.4 send-community extended
PE-2(config-router-af)#exit-address-family
PE-2(config-router)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#int f2/0
PE-2(config-if)#mpls ip
PE-2(config-if)#exit
PE-2(config)#end
PE-2#

C:\Users\Fahmida Hossain\OneDrive - University of Huddersfield\MSc project\Service_provider_IPMPLS\demo-mpls\demo-mpls>py config_network.py
Configuring R1 (192.168.223.111)...
Configuration Output for 192.168.223.111:
```

```
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#interface f0/0
R1(config-if)#ip address 192.168.12.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface f1/0
R1(config-if)#ip address 192.168.14.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface f2/0
R1(config-if)#ip address 192.168.15.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface lo0
R1(config-if)#ip address 1.1.1.1 255.255.255.255
R1(config-if)#exit
R1(config)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router ospf 1
R1(config-router)#router-id 1.1.1.1
R1(config-router)#passive-interface Loopback0
R1(config-router)#int f0/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int f1/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int f2/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int lo0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#exit
R1(config)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router bgp 1
R1(config-router)#bgp router-id 1.1.1.1
R1(config-router)#neighbor 4.4.4.4 remote-as 1
R1(config-router)#neighbor 4.4.4.4 update-source Loopback0
R1(config-router)#address-family vpnv4
R1(config-router-af)#neighbor 4.4.4.4 activate
R1(config-router-af)#neighbor 4.4.4.4 send-community extended
R1(config-router-af)#exit-address-family
R1(config-router)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
```

```
R1(config)#int f0/0
R1(config-if)#mpls ip
R1(config-if)#int f1/0
R1(config-if)#mpls ip
R1(config-if)#int f2/0
R1(config-if)#mpls ip
R1(config-if)#exit
R1(config)#end
R1#
Configuring R2 (192.168.223.112)...
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#interface f0/0
R2(config-if)#ip address 192.168.12.2 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface f1/0
R2(config-if)#ip address 192.168.23.2 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface lo0
R2(config-if)#ip address 2.2.2.2 255.255.255.255
R2(config-if)#exit
R2(config)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router ospf 1
R2(config-router)#router-id 2.2.2.2
R2(config-router)#passive-interface Loopback0
R2(config-router)#int f0/0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#int f1/0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#int lo0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#exit
R2(config)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router bgp 1
R2(config-router)#bgp router-id 2.2.2.2
R2(config-router)#neighbor 4.4.4.4 remote-as 1
R2(config-router)#neighbor 4.4.4.4 update-source Loopback0
R2(config-router)#address-family vpnv4
R2(config-router-af)#neighbor 4.4.4.4 activate
R2(config-router-af)#neighbor 4.4.4.4 send-community extended
R2(config-router-af)#exit-address-family
```

```
R2(config-router)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#int f0/0
R2(config-if)#mpls ip
R2(config-if)#int f1/0
R2(config-if)#mpls ip
R2(config-if)#exit
R2(config)#end
R2#
Configuring R3 (192.168.223.113)...
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#interface f0/0
R3(config-if)#ip address 192.168.34.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface f1/0
R3(config-if)#ip address 192.168.23.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface f2/0
R3(config-if)#ip address 192.168.16.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface lo0
R3(config-if)#ip address 3.3.3.3 255.255.255.255
R3(config-if)#exit
R3(config)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router ospf 1
R3(config-router)#router-id 3.3.3.3
R3(config-router)#passive-interface Loopback0
R3(config-router)#int f0/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int f1/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int f2/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int lo0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#exit
R3(config)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router bgp 1
R3(config-router)#bgp router-id 3.3.3.3
R3(config-router)#neighbor 4.4.4.4 remote-as 1
R3(config-router)#neighbor 4.4.4.4 update-source Loopback0
R3(config-router)#address-family vpnv4
R3(config-router-af)#neighbor 4.4.4.4 activate
R3(config-router-af)#neighbor 4.4.4.4 send-community extended
R3(config-router-af)#exit-address-family
R3(config-router)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#int f0/0
R3(config-if)#mpls ip
R3(config-if)#int f1/0
R3(config-if)#mpls ip
R3(config-if)#int f2/0
R3(config-if)#mpls ip
R3(config-if)#exit
R3(config)#end
R3#
Configuring R4 (192.168.223.114)...
Configuration Output for 192.168.223.114:

R4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface f0/0
R4(config-if)#ip address 192.168.34.4 255.255.255.0
R4(config-if)#no shutdown
R4(config-if)#exit
R4(config)#interface f1/0
R4(config-if)#ip address 192.168.14.4 255.255.255.0
R4(config-if)#no shutdown
R4(config-if)#exit
R4(config)#interface lo0
R4(config-if)#ip address 4.4.4.4 255.255.255.255
R4(config-if)#exit
R4(config)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#router ospf 1
R4(config-router)#router-id 4.4.4.4
R4(config-router)#passive-interface Loopback0
R4(config-router)#int f0/0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#int f1/0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#int lo0
```

```
R4(config-if)#ip ospf 1 area 0
R4(config-if)#exit
R4(config)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#router bgp 1
R4(config-router)#bgp router-id 4.4.4.4
R4(config-router)#neighbor 1.1.1.1 remote-as 1
R4(config-router)#neighbor 1.1.1.1 update-source Loopback0
R4(config-router)#neighbor 2.2.2.2 remote-as 1
R4(config-router)#neighbor 2.2.2.2 update-source Loopback0
R4(config-router)#neighbor 3.3.3.3 remote-as 1
R4(config-router)#neighbor 3.3.3.3 update-source Loopback0
R4(config-router)#neighbor 5.5.5.5 remote-as 1
R4(config-router)#neighbor 5.5.5.5 update-source Loopback0
R4(config-router)#neighbor 6.6.6.6 remote-as 1
R4(config-router)#neighbor 6.6.6.6 update-source Loopback0
R4(config-router)#address-family ipv4
R4(config-router-af)#neighbor 1.1.1.1 activate
R4(config-router-af)#neighbor 2.2.2.2 activate
R4(config-router-af)#neighbor 3.3.3.3 activate
R4(config-router-af)#neighbor 5.5.5.5 activate
R4(config-router-af)#neighbor 6.6.6.6 activate
R4(config-router-af)#exit-address-family
R4(config-router)#address-family vpnv4
R4(config-router-af)#neighbor 1.1.1.1 activate
R4(config-router-af)#neighbor 1.1.1.1 send-community extended
R4(config-router-af)#neighbor 1.1.1.1 route-reflector-client
R4(config-router-af)#neighbor 2.2.2.2 activate
R4(config-router-af)#neighbor 2.2.2.2 send-community extended
R4(config-router-af)#neighbor 2.2.2.2 route-reflector-client
R4(config-router-af)#neighbor 3.3.3.3 activate
R4(config-router-af)#neighbor 3.3.3.3 send-community extended
R4(config-router-af)#neighbor 3.3.3.3 route-reflector-client
R4(config-router-af)#neighbor 5.5.5.5 activate
R4(config-router-af)#neighbor 5.5.5.5 send-community extended
R4(config-router-af)#neighbor 5.5.5.5 route-reflector-client
R4(config-router-af)#neighbor 6.6.6.6 activate
R4(config-router-af)#neighbor 6.6.6.6 send-community extended
R4(config-router-af)#neighbor 6.6.6.6 route-reflector-client
R4(config-router-af)#exit-address-family
R4(config-router)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#int f0/0
R4(config-if)#mpls ip
R4(config-if)#int f1/0
R4(config-if)#mpls ip
```

```
R4(config-if)#exit
R4(config)#end
R4#
Configuring PE-1 (192.168.223.115)...
Failed to configure 192.168.223.115: TCP connection to device failed.

Common causes of this problem are:
1. Incorrect hostname or IP address.
2. Wrong TCP port.
3. Intermediate firewall blocking access.

Device settings: cisco_ios 192.168.223.115:22


Configuring PE-2 (192.168.223.116)...
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#interface f0/0
PE-2(config-if)#ip address 172.20.2.2 255.255.255.0
PE-2(config-if)#no shutdown
PE-2(config-if)#exit
PE-2(config)#interface f2/0
PE-2(config-if)#ip address 192.168.16.2 255.255.255.0
PE-2(config-if)#no shutdown
PE-2(config-if)#exit
PE-2(config)#interface lo0
PE-2(config-if)#ip address 6.6.6.6 255.255.255.255
PE-2(config-if)#exit
PE-2(config)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#router ospf 1
PE-2(config-router)#router-id 6.6.6.6
PE-2(config-router)#passive-interface Loopback0
PE-2(config-router)#int f2/0
PE-2(config-if)#ip ospf 1 area 0
PE-2(config-if)#int lo0
PE-2(config-if)#ip ospf 1 area 0
PE-2(config-if)#exit
PE-2(config)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#router bgp 1
PE-2(config-router)#bgp router-id 6.6.6.6
PE-2(config-router)#neighbor 4.4.4.4 remote-as 1
PE-2(config-router)#neighbor 4.4.4.4 update-source Loopback0
PE-2(config-router)#address-family vpnv4
```

```
PE-2(config-router-af)#neighbor 4.4.4.4 activate
PE-2(config-router-af)#neighbor 4.4.4.4 send-community extended
PE-2(config-router-af)#exit-address-family
PE-2(config-router)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#int f2/0
PE-2(config-if)#mpls ip
PE-2(config-if)#exit
PE-2(config)#end
PE-2#

C:\Users\Fahmida Hossain\OneDrive - University of Huddersfield\MSc project\Service_provider_IPMPLS\demo-mpls\demo-mpls>py config_network.py
Configuring R1 (192.168.223.111)...
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#interface f0/0
R1(config-if)#ip address 192.168.12.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface f1/0
R1(config-if)#ip address 192.168.14.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface f2/0
R1(config-if)#ip address 192.168.15.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface lo0
R1(config-if)#ip address 1.1.1.1 255.255.255.255
R1(config-if)#exit
R1(config)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router ospf 1
R1(config-router)#router-id 1.1.1.1
R1(config-router)#passive-interface Loopback0
R1(config-router)#int f0/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int f1/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int f2/0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#int lo0
R1(config-if)#ip ospf 1 area 0
R1(config-if)#exit
```

```
R1(config)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#router bgp 1
R1(config-router)#bgp router-id 1.1.1.1
R1(config-router)#neighbor 4.4.4.4 remote-as 1
R1(config-router)#neighbor 4.4.4.4 update-source Loopback0
R1(config-router)#address-family vpnv4
R1(config-router-af)#neighbor 4.4.4.4 activate
R1(config-router-af)#neighbor 4.4.4.4 send-community extended
R1(config-router-af)#exit-address-family
R1(config-router)#end
R1#
Configuration Output for 192.168.223.111:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#int f0/0
R1(config-if)#mpls ip
R1(config-if)#int f1/0
R1(config-if)#mpls ip
R1(config-if)#int f2/0
R1(config-if)#mpls ip
R1(config-if)#exit
R1(config)#end
R1#
Configuring R2 (192.168.223.112)...
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#interface f0/0
R2(config-if)#ip address 192.168.12.2 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface f1/0
R2(config-if)#ip address 192.168.23.2 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface lo0
R2(config-if)#ip address 2.2.2.2 255.255.255.255
R2(config-if)#exit
R2(config)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router ospf 1
R2(config-router)#router-id 2.2.2.2
R2(config-router)#passive-interface Loopback0
R2(config-router)#int f0/0
R2(config-if)#ip ospf 1 area 0
```

```
R2(config-if)#int f1/0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#int lo0
R2(config-if)#ip ospf 1 area 0
R2(config-if)#exit
R2(config)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#router bgp 1
R2(config-router)#bgp router-id 2.2.2.2
R2(config-router)#neighbor 4.4.4.4 remote-as 1
R2(config-router)#neighbor 4.4.4.4 update-source Loopback0
R2(config-router)#address-family vpnv4
R2(config-router-af)#neighbor 4.4.4.4 activate
R2(config-router-af)#neighbor 4.4.4.4 send-community extended
R2(config-router-af)#exit-address-family
R2(config-router)#end
R2#
Configuration Output for 192.168.223.112:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)#int f0/0
R2(config-if)#mpls ip
R2(config-if)#int f1/0
R2(config-if)#mpls ip
R2(config-if)#exit
R2(config)#end
R2#
Configuring R3 (192.168.223.113)...
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#interface f0/0
R3(config-if)#ip address 192.168.34.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface f1/0
R3(config-if)#ip address 192.168.23.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface f2/0
R3(config-if)#ip address 192.168.16.3 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface lo0
R3(config-if)#ip address 3.3.3.3 255.255.255.255
R3(config-if)#exit
R3(config)#end
R3#
Configuration Output for 192.168.223.113:
```

```
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router ospf 1
R3(config-router)#router-id 3.3.3.3
R3(config-router)#passive-interface Loopback0
R3(config-router)#int f0/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int f1/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int f2/0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#int lo0
R3(config-if)#ip ospf 1 area 0
R3(config-if)#exit
R3(config)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#router bgp 1
R3(config-router)#bgp router-id 3.3.3.3
R3(config-router)#neighbor 4.4.4.4 remote-as 1
R3(config-router)#neighbor 4.4.4.4 update-source Loopback0
R3(config-router)#address-family vpnv4
R3(config-router-af)#neighbor 4.4.4.4 activate
R3(config-router-af)#neighbor 4.4.4.4 send-community extended
R3(config-router-af)#exit-address-family
R3(config-router)#end
R3#
Configuration Output for 192.168.223.113:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#int f0/0
R3(config-if)#mpls ip
R3(config-if)#int f1/0
R3(config-if)#mpls ip
R3(config-if)#int f2/0
R3(config-if)#mpls ip
R3(config-if)#exit
R3(config)#end
R3#
Configuring R4 (192.168.223.114)...
Configuration Output for 192.168.223.114:

R4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#interface f0/0
R4(config-if)#ip address 192.168.34.4 255.255.255.0
R4(config-if)#no shutdown
R4(config-if)#exit
R4(config)#interface f1/0
R4(config-if)#ip address 192.168.14.4 255.255.255.0
```

```
R4(config-if)#no shutdown
R4(config-if)#exit
R4(config)#interface lo0
R4(config-if)#ip address 4.4.4.4 255.255.255.255
R4(config-if)#exit
R4(config)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#router ospf 1
R4(config-router)#router-id 4.4.4.4
R4(config-router)#passive-interface Loopback0
R4(config-router)#int f0/0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#int f1/0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#int lo0
R4(config-if)#ip ospf 1 area 0
R4(config-if)#exit
R4(config)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#router bgp 1
R4(config-router)#bgp router-id 4.4.4.4
R4(config-router)#neighbor 1.1.1.1 remote-as 1
R4(config-router)#neighbor 1.1.1.1 update-source Loopback0
R4(config-router)#neighbor 2.2.2.2 remote-as 1
R4(config-router)#neighbor 2.2.2.2 update-source Loopback0
R4(config-router)#neighbor 3.3.3.3 remote-as 1
R4(config-router)#neighbor 3.3.3.3 update-source Loopback0
R4(config-router)#neighbor 5.5.5.5 remote-as 1
R4(config-router)#neighbor 5.5.5.5 update-source Loopback0
R4(config-router)#neighbor 6.6.6.6 remote-as 1
R4(config-router)#neighbor 6.6.6.6 update-source Loopback0
R4(config-router)#address-family ipv4
R4(config-router-af)#neighbor 1.1.1.1 activate
R4(config-router-af)#neighbor 2.2.2.2 activate
R4(config-router-af)#neighbor 3.3.3.3 activate
R4(config-router-af)#neighbor 5.5.5.5 activate
R4(config-router-af)#neighbor 6.6.6.6 activate
R4(config-router-af)#exit-address-family
R4(config-router)#address-family vpnv4
R4(config-router-af)#neighbor 1.1.1.1 activate
R4(config-router-af)#neighbor 1.1.1.1 send-community extended
R4(config-router-af)#neighbor 1.1.1.1 route-reflector-client
R4(config-router-af)#neighbor 2.2.2.2 activate
R4(config-router-af)#neighbor 2.2.2.2 send-community extended
R4(config-router-af)#neighbor 2.2.2.2 route-reflector-client
R4(config-router-af)#neighbor 3.3.3.3 activate
```

```
R4(config-router-af)#neighbor 3.3.3.3 send-community extended
R4(config-router-af)#neighbor 3.3.3.3 route-reflector-client
R4(config-router-af)#neighbor 5.5.5.5 activate
R4(config-router-af)#neighbor 5.5.5.5 send-community extended
R4(config-router-af)#neighbor 5.5.5.5 route-reflector-client
R4(config-router-af)#neighbor 6.6.6.6 activate
R4(config-router-af)#neighbor 6.6.6.6 send-community extended
R4(config-router-af)#neighbor 6.6.6.6 route-reflector-client
R4(config-router-af)#exit-address-family
R4(config-router)#end
R4#
Configuration Output for 192.168.223.114:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R4(config)#int f0/0
R4(config-if)#mpls ip
R4(config-if)#int f1/0
R4(config-if)#mpls ip
R4(config-if)#exit
R4(config)#end
R4#
Configuring PE-1 (192.168.223.115)...
Configuration Output for 192.168.223.115:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-1(config)#interface f0/0
PE-1(config-if)#ip address 172.10.1.2 255.255.255.0
PE-1(config-if)#no shutdown
PE-1(config-if)#exit
PE-1(config)#interface f2/0
PE-1(config-if)#ip address 192.168.15.2 255.255.255.0
PE-1(config-if)#no shutdown
PE-1(config-if)#exit
PE-1(config)#interface lo0
PE-1(config-if)#ip address 5.5.5.5 255.255.255.255
PE-1(config-if)#exit
PE-1(config)#end
PE-1#
Configuration Output for 192.168.223.115:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-1(config)#router ospf 1
PE-1(config-router)#router-id 5.5.5.5
PE-1(config-router)#passive-interface Loopback0
PE-1(config-router)#int f2/0
PE-1(config-if)#ip ospf 1 area 0
PE-1(config-if)#int lo0
PE-1(config-if)#ip ospf 1 area 0
PE-1(config-if)#exit
PE-1(config)#end
PE-1#
Configuration Output for 192.168.223.115:
```

```
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-1(config)#router bgp 1
PE-1(config-router)#bgp router-id 5.5.5.5
PE-1(config-router)#neighbor 4.4.4.4 remote-as 1
PE-1(config-router)#neighbor 4.4.4.4 update-source Loopback0
PE-1(config-router)#address-family vpnv4
PE-1(config-router-af)#neighbor 4.4.4.4 activate
PE-1(config-router-af)#neighbor 4.4.4.4 send-community extended
PE-1(config-router-af)#exit-address-family
PE-1(config-router)#end
PE-1#
Configuration Output for 192.168.223.115:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-1(config)#int f2/0
PE-1(config-if)#mpls ip
PE-1(config-if)#exit
PE-1(config)#end
PE-1#
Configuring PE-2 (192.168.223.116)...
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#interface f0/0
PE-2(config-if)#ip address 172.20.2.2 255.255.255.0
PE-2(config-if)#no shutdown
PE-2(config-if)#exit
PE-2(config)#interface f2/0
PE-2(config-if)#ip address 192.168.16.2 255.255.255.0
PE-2(config-if)#no shutdown
PE-2(config-if)#exit
PE-2(config)#interface lo0
PE-2(config-if)#ip address 6.6.6.6 255.255.255.255
PE-2(config-if)#exit
PE-2(config)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#router ospf 1
PE-2(config-router)#router-id 6.6.6.6
PE-2(config-router)#passive-interface Loopback0
PE-2(config-router)#int f2/0
PE-2(config-if)#ip ospf 1 area 0
PE-2(config-if)#int lo0
PE-2(config-if)#ip ospf 1 area 0
PE-2(config-if)#exit
PE-2(config)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#router bgp 1
PE-2(config-router)#bgp router-id 6.6.6.6
PE-2(config-router)#neighbor 4.4.4.4 remote-as 1
PE-2(config-router)#neighbor 4.4.4.4 update-source Loopback0
PE-2(config-router)#address-family vpnv4
PE-2(config-router-af)#neighbor 4.4.4.4 activate
PE-2(config-router-af)#neighbor 4.4.4.4 send-community extended
PE-2(config-router-af)#exit-address-family
PE-2(config-router)#end
PE-2#
Configuration Output for 192.168.223.116:
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-2(config)#int f2/0
PE-2(config-if)#mpls ip
PE-2(config-if)#exit
PE-2(config)#end
PE-2#

C:\Users\Fahmida Hossain\OneDrive - University of Huddersfield\MSc project\Service_provider_IPMPLS\demo-mpls\demo-mpls>
```

1.2 Project code source

Github Link- fahmmi94/Service-Provider-IP-MPLS-Network-Automation-using-Python (github.com)