Automated Network Planning and Configuration Tool

Cameron Finlay

BSc Ethical Hacking (Hons), 2024-2025

Supervised by

Mr David McLuskie

School of Design and Informatics

Abertay University

[Table of Figures 6](#_Toc196872436)

[Acknowledgements 8](#_Toc196872437)

[Abstract 9](#_Toc196872438)

[1. Introduction 11](#_Toc196872439)

[1.1 Background 11](#_Toc196872440)

[1.2 Aim 13](#_Toc196872441)

[1.3 Research Question 13](#_Toc196872442)

[1.4 Objectives 14](#_Toc196872443)

[1.5 Structure 14](#_Toc196872444)

[2. Literature Review 15](#_Toc196872445)

[2.1 Automated Network Configuration 16](#_Toc196872446)

[2.2 Intent Based vs Rule Based Configuration 17](#_Toc196872447)

[2.3 Network Planning 18](#_Toc196872448)

[2.4 Network Mapping 18](#_Toc196872449)

[2.5 Router Configuration 19](#_Toc196872450)

[2.6 Switch Configuration 20](#_Toc196872451)

[2.7 Firewall Configuration 20](#_Toc196872452)

[2.8 Existing Solutions 20](#_Toc196872453)

[3. Methodology 22](#_Toc196872454)

[3.1 High Level Manual set up Guide 22](#_Toc196872455)

[3.2 Graphic User Interface 23](#_Toc196872456)

[3.3 Topology Questionnaire Script 25](#_Toc196872457)

[3.3.1 Designing the Topology Questionnaire script 25](#_Toc196872458)

[3.3.1.1 Number of devices 26](#_Toc196872459)

[3.3.1.2 Scalability 29](#_Toc196872460)

[3.3.1.3 Redundancy 30](#_Toc196872461)

[30](#_Toc196872462)

[3.3.1.4 Security 30](#_Toc196872463)

[3.3.1.5 Services 31](#_Toc196872464)

[3.3.1.6 Architecture Choice 32](#_Toc196872465)

[3.4 Network Virtualisation 33](#_Toc196872466)

[3.5 Topology Generation Implementation 33](#_Toc196872467)

[3.5.1 Generating a Mesh Topology 35](#_Toc196872468)

[3.5.2 Generating a Peer-to-Peer Topology 40](#_Toc196872469)

[3.5.3 Generating a Centralised Hub and Client Server Topology 41](#_Toc196872470)

[3.5.4 Peer-to-Peer 42](#_Toc196872471)

[3.5.5 Client-Server Architecture 42](#_Toc196872472)

[3.5.6 Centralised Network Architecture 42](#_Toc196872473)

[3.5.7 Mesh Network Architecture 42](#_Toc196872474)

[3.5.8 Omitted Architectures 42](#_Toc196872475)

[3.6 Network mapping 43](#_Toc196872476)

[3.7 Connection Implementation 45](#_Toc196872477)

[3.8 General Device Configuration 46](#_Toc196872478)

[3.9 Device Configuration for Routers 47](#_Toc196872479)

[3.10 Device Configuration for Switches 49](#_Toc196872480)

[3.11 Configuration for Firewalls 50](#_Toc196872481)

[3.12 Testing 51](#_Toc196872482)

[3.12.1 Test Case 1 51](#_Toc196872483)

[3.12.2 Test Case 2 51](#_Toc196872484)

[3.12.3 Test Case 3 52](#_Toc196872485)

[3.12.4 Test Case 4 52](#_Toc196872486)

[3.12.5 Test Case 5 52](#_Toc196872487)

[3.13 Survey 52](#_Toc196872488)

[3.14 Proposed languages and tools 54](#_Toc196872489)

[4. Results 56](#_Toc196872490)

[4.1 Topology Generation 56](#_Toc196872491)

[4.1.1 Test Case 1 56](#_Toc196872492)

[4.1.2 Test Case 2 57](#_Toc196872493)

[4.1.3 Test Case 3 59](#_Toc196872494)

[4.1.4 Test Case 4 61](#_Toc196872495)

[4.2 Network Mapping 62](#_Toc196872496)

[4.2.1 Manual IP Mode 62](#_Toc196872497)

[4.2.2 Subnet Discovery Mode 64](#_Toc196872498)

[4.3 Network configuration 65](#_Toc196872499)

[4.3.1 Router 65](#_Toc196872500)

[4.3.2 Switch 66](#_Toc196872501)

[4.3.3 Firewall 66](#_Toc196872502)

[4.4 Survey Results 67](#_Toc196872503)

[4.4.1 Question 1 67](#_Toc196872504)

[4.4.3 Question 3 68](#_Toc196872505)

[4.4.4 Question 4 68](#_Toc196872506)

[4.4.5 Question 5 69](#_Toc196872507)

[5 Discussion 69](#_Toc196872508)

[5.1 To design and implement a tool that uses users input to recommend an appropriate network architecture 70](#_Toc196872509)

[5.2 To identify standard configuration practices for routers and switches 72](#_Toc196872510)

[5.3 To explore and implement automated deployment of internal firewalls 74](#_Toc196872511)

[5.4 To outline the essential steps for enabling remote configuration 77](#_Toc196872512)

[6. Conclusion and Future Work 79](#_Toc196872513)

[7. References 81](#_Toc196872514)

[8. Appendices 86](#_Toc196872515)

[8.1 Appendix A - Script Code 86](#_Toc196872516)

[Appendix B - Low level Manual Set up Guide 120](#_Toc196872517)

[**Abstract** 121](#_Toc196872518)

[**1.**  **Hardware** 122](#_Toc196872519)

[**1.1**  **Appropriate Hardware** 122](#_Toc196872520)

[**1.2**  **Cabling** 123](#_Toc196872521)

[**1.2.1**  **Power Supply** 123](#_Toc196872522)

[**1.2.2**  **Ethernet Cables** 124](#_Toc196872523)

[**1.2.3**  **Console Cable** 124](#_Toc196872524)

[**2.**  **Software** 124](#_Toc196872525)

[**2.1**  **Manual configuration** 124](#_Toc196872526)

[**References** 125](#_Toc196872527)

[Appendix C - Survey Question 4 and 5 Results 126](#_Toc196872528)

# Table of Figures

Figure 1 Intent lifecycle within IBN(Kim et al., 2024) 9

Figure 2 NETCONF Protocol Layers(Choi, Choi and Hong, 2004) 14

Figure 3 GUI homepage 21

Figure 4 Code for Homepage 21

Figure 5 Workstation Question 24

Figure 6 Build user interface Code 24

Figure 7 Display question Code 25

Figure 8 Clear input Code 26

Figure 9 Scalability Question 26

Figure 10 Scalability Question 27

Figure 11 Security Question 28

Figure 12 Services Question 29

Figure 13 Calculating number of Switches and Routers 31

Figure 14 Topology generation Mesh If statement 31

Figure 15 Topology generating Peer-To-Peer if statement 31

Figure 16 Processing Routers 32

Figure 17 Processing Switches 33

Figure 18 Connecting Workstations to Switches 34

Figure 19 Connecting Workstations to each other 34

Figure 20 Connection with firewall 35

Figure 21 Direct Router to switch connection 36

Figure 22 Network Mapping Interface 40

Figure 23 Subnet Discovery Code 41

Figure 24 Scanning Specific Hosts code 42

Figure 25 Telnet connection code 43

Figure 26 SSH connection Code 43

Figure 27 Telnet Configuration code 45

Figure 28 SSH Config Code 46

Figure 29 Basic firewall rules 47

Figure 30 Stricter Firewall rules 48

Figure 31 Survey Question 1 and 2 50

Figure 32 Survey Questions 3, 4 and 5 51

Figure 33 Topology Diagram 1 53

Figure 34 .txt Output file 1 54

Figure 35 Topology Diagram 2 54

Figure 36 .txt Output file 2 55

Figure 37 Topology Diagram 3 56

Figure 38 .txt Output File 3 57

Figure 39 Topology Diagram 4 57

Figure 40 .txt Output File 4 58

Figure 41 Manual IP mode single IP address 59

Figure 42 Single manual IP address scan results 59

Figure 43 Multiple Specific IP address scan 60

Figure 44 Multiple manual IP address scan results 60

Figure 45 Host discovery 61

Figure 46 Configuration results 62

Figure 47 Question 1 Pie chart4.4.2 Question 2 63

Figure 48 Question 2 Pie chart 64

Figure 49 Question 3 Pie chart 64

# Acknowledgements

I would like to thank my phenomenal supervisor David McLuskie for his neverending support and advice in the past year and for igniting my passion for networking in the first place. The amazing course he taught will shape my future and I know it will be for the better. The feedback provided by David has made the process of writing and developing my dissertation far easier and with far less issues than what would have occurred had I been attempting such a project by myself.

I would also like to thank my friends for supporting and listening to my problems and anxieties over the course of the year as I pondered the never ending problems I faced when writing this dissertation.

Finally I would like to thank my Dad for getting me through the final few weeks of my dissertation. I would not have been able to achieve anywhere near what I have without his love and support all these years.

# Abstract

The increasing complexity of network configuration presents a significant challenge for non-technical users, who often lack the expertise to design and deploy secure, functional networks. While commercial solutions like Cisco DNA Center leverage artificial intelligence to simplify network planning through intent-based policies, these tools often prioritize business needs over security, require substantial financial investment, and may produce unintended configurations due to AI-driven automation. This dissertation addresses these limitations by developing an open-source, script-based tool that guides non-technical users through network planning, topology generation, and device configuration while enforcing security best practices by default. The tool employs a rule-based methodology, ensuring compatibility with a wide range of devices—including legacy hardware—without relying on complex protocols like NETCONF or YANG, which often require specialized knowledge or modern hardware support.

The study explores four key research questions: how user input can be translated into network architecture recommendations, what configurations are essential for routers and switches, how internal firewalls can be automatically deployed, and how remote configuration steps can be effectively taught to non-technical users. A questionnaire-driven interface simplifies the planning process, allowing users to define high-level requirements such as scalability, redundancy, security, and services without needing granular technical knowledge. The tool then generates visual network topologies and automates device configurations, applying security measures such as disabling unused services, restricting remote access to SSH, and implementing firewall rules based on user-defined security levels. Testing was conducted using GNS3 for network simulation, though limitations in emulating servers and workstations highlighted the need for future real-world or hybrid testing approaches.

A user study involving 14 participants evaluated the tool’s effectiveness, with results indicating strong usability for non-technical users. The topology diagrams received high ratings for clarity, though some users noted that complex networks became visually cluttered. The accompanying manual setup guide, designed using contextual learning principles, successfully conveyed essential setup steps, though abstract concepts like validation testing were less consistently retained. While the tool demonstrated success in automating secure configurations, its static rule-based approach lacked the dynamic adaptability of commercial solutions, and the absence of real-time validation posed potential deployment risks.

Future work should focus on improving topology visualization for large networks, integrating real-time configuration validation, and expanding the tool’s adaptability through community-driven development. Enhancements such as interactive topology elements, firewall rule previews, and video-based tutorials could further bridge the gap between usability and technical precision. By maintaining an open-source framework, this project provides a foundation for democratizing network configuration, offering a viable alternative to proprietary tools while balancing accessibility, security, and functionality. The findings contribute to broader efforts in network automation, demonstrating that simplified, user-centric approaches can coexist with robust security practices.

# 1. Introduction

## 1.1 Background

The increasing complexity of modern computer networks has made network planning and configuration a challenging task, particularly for non-technical users who lack specialized knowledge in networking. Traditional network configuration often requires expertise in command-line interfaces, protocols, and security best practices, creating a steep learning curve for beginners. While commercial solutions like Cisco DNA Center (Dna et al.) leverage artificial intelligence to simplify network design through intent-based policies, these tools often prioritize business needs over security and may not be accessible to smaller organizations or individuals due to cost and proprietary restrictions. Furthermore, AI-driven automation can sometimes lead to unintended configurations, raising concerns about reliability and security(Tu et al., 2025);(Kim et al., 2024). The intent based configuration lifecycle is shown below to illustrate current approaches to address the challenges faced.

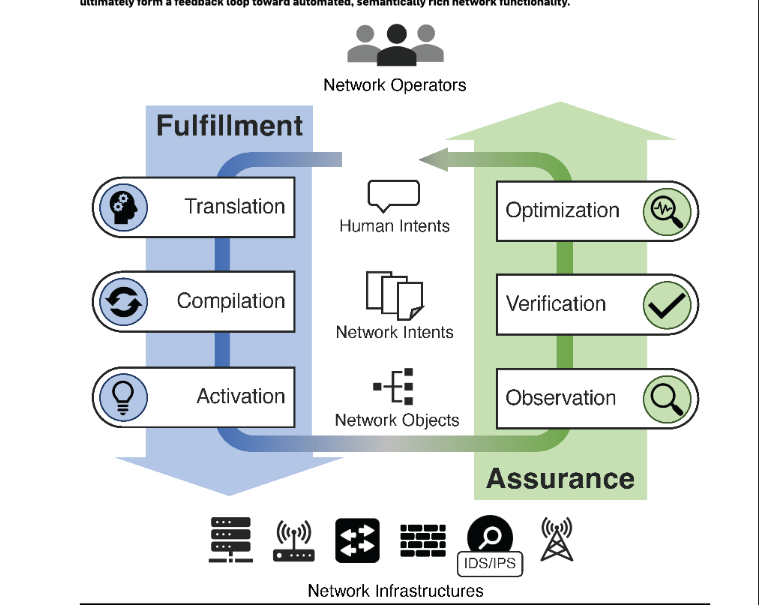


Figure 1 Intent lifecycle within IBN(Kim et al., 2024)

To address these challenges, this project explores an alternative approach: an open-source, script-based tool that guides non-technical users through network planning, device selection, and secure configuration. Unlike AI-driven solutions, this tool employs a rule-based methodology, ensuring that security best practices are enforced by default while still accommodating user requirements. The research builds upon existing literature in network automation (Bringhenti et al., 2023; Leivadeas & Falkner, 2023), intent-based networking(Tu et al., 2025; Zeydan & Turk), and configuration management (Ji et al., 2009-03; Granderath and Schönwälder , 2023), while also considering the limitations of legacy systems that may not support modern protocols like NETCONF and YANG (Hussain et al.; Ji et al., 2009-03)

A key challenge in network automation is balancing usability with security. While intent-based systems allow users to define high-level requirements without technical details, they risk introducing vulnerabilities if security is not a foundational consideration (Kim et al., 2024). Conversely, traditional rule-based approaches (e.g., SNMP, NETCONF) provide precise control but require deeper technical knowledge(Case, 1988); Granderath and Schönwälder , 2023). This project seeks a middle ground by automating best-practice configurations for routers and switches (*Cisco 800M Series Integrated Services Routers Software Configuration Guide,* 2015; Alicea & Alsmadi, 2021) while using simplified user inputs to determine network structure and security policies. Additionally, the study examines how visual network topologies (Sankpal et al., 2022; Shu et al. 2024) and step-by-step guides (De La Puente & Perez 2023; Widmer 1996) can improve user comprehension, particularly for those with limited networking experience.

The research also confronts practical limitations in testing network configurations. While tools like GNS3 are valuable for simulating routers and switches, they struggle to accurately emulate servers and workstations, restricting full-scale validation of automated configurations. This gap suggests that future iterations of the tool may require real-world device testing or integration with virtual machines to ensure comprehensive functionality. By addressing these challenges, this project contributes to the broader goal of democratizing network setup, making it more accessible while maintaining robust security and performance standards.

## 1.2 Aim

This project aims to design and implement a scripted tool that can assist non-technical users with planning, testing and configuring a customisable computer network and evaluate its effectiveness.

## 1.3 Research Question

* How can a tool be used to take user input and recommend network architecture and devices.
* What configurations should be used and how would they be implemented on routers and switches.
* How can internal firewalls and other security measures be automatically deployed on a router or switch.
* What steps must be taken to allow for remote configuration using the script and how can they be taught to users.

## 1.4 Objectives

* To design and implement a tool that uses user input to recommend an appropriate network architecture and determine the required devices, such as routers, switches, server and workstations
* To identify standard configurations practices for routers and switches and develop a method for automating their implementation
* To explore and implement automated deployment of internal firewalls and other security measures within router and switch configuration
* To outline the essential steps for enabling remote configuration via scripting and to develop a method for effectively communicating these steps to non-technical users
* Evaluate the effectiveness of the tool and relate the findings to previous work found within the literature review

## 1.5 Structure

The next section in this dissertation is a literature review, which provides a background and critical analysis of previous and currently used technologies in the field of automated network configuration, planning and mapping. The methodology section focuses on the explanations and justifications for why each artefact was created and the methods used to develop the artefacts. The Results are presented in the section after the methodology and align with the project's aims. The discussion section analyses and evaluates the approach, results and its significance in relation to the literature review. The conclusion and future work section is the final section and outlines the implications of the results and how the project could be continued in the future.

# 2. Literature Review

Individuals and businesses that utilise networks should always ensure that they are secure, maintained and fit for purpose. This however due to ever increasing size and variety of devices has made manual configuration no longer feasible. This section will cover the literature that is used as the backbone of the paper and is structure as follows:

Automated Network Configuration refers to the process of using software tools and scripts to configure and manage network devices. This approach improves efficiency, reduces human error.

Intent-Based Networking focuses on defining high-level business goals, allowing the network to automatically adjust itself to meet these objectives. Rule-Based Networking uses predefined static rules, requiring administrators to manually configure the network's behaviour.

Network planning is designing and optimising a network’s architecture to ensure performance, security, and scalability. It includes assessing bandwidth needs, redundancy strategies, and selecting appropriate hardware and software to prevent bottlenecks and vulnerabilities.

Network mapping is the process of identifying and documenting all network components, such as routers, switches, firewalls, and connected devices.

Routers require configuration settings such as IP addressing, routing protocols and security policies.

Switches need configurations related to VLANs, Spanning Tree Protocol and port security. These settings help in managing network segmentation, preventing loops, and enhancing security by controlling device connections.

Firewalls require configurations that define security policies, access control lists, intrusion prevention rules, and traffic filtering. Proper firewall configuration is essential for blocking unauthorised access, detecting threats, and enforcing network security policies.

Various existing solutions offer automated network configuration and management. These solutions help streamline operations, reduce downtime, and improve overall network security and performance.

# 2.1 Automated Network Configuration

Automated Network Configuration is a process by which a program will go through either a list of devices or a specified area of digital space and configure all the devices that it can access. This configuration process is to adapt systems to be more secure without losing functionality.

This for many people is done manually as they do not have access to a specialist tool whereby, they must learn how to update devices software, disabling features and processes that may be vulnerable.

A reason for misconfigurations is human error that takes the form of mistakes or oversight during the configuration process. These errors are significant in Firewalls and Network access points because of the important role that they play in protecting networks(Alicea and Alsmadi, 2021).

The major point of misconfiguration is in roles which can be generated by next generation firewalls automatically. However, it is Manual configuration that causes more errors in configuring network devices and takes longer which is a significant concern when deciding the approach as the time difference is significant(Mazin *et al.,* 2021-04-03). This time concern is because of scalability issues with manual configuration.

Studies show that configuring devices is a device specific task and as such can be complex to automate. This is because the network operator community wants to be able to control specific details of devices whilst being able to do network wide configurations. It then states that most configurations to currently working networks are small and implement incremental changes (Alicea and Alsmadi, 2021).

One of the major problems of the simple network management protocol (SNMP)l is that it performs poorly at configuration management (Choi, Choi and Hong, 2004). The Internet Engineering Task Force found that an XML approach is better when developing Network Configuration protocol (NETCONF). The NETCONF using XML can be conceptually partitioned into four layers as shown below in Figure 2.

A close-up of a computer code

AI-generated content may be incorrect.

Figure 2 NETCONF Protocol Layers(Choi, Choi and Hong, 2004)

XML is a platform-neutral data representation configuration language and parser. However, XML was not developed for this purpose but has been found to be exceptional in its ability to be used for configuration management. Below is an example of a Software Architectural Analysis Method (SAAM). It is a good example of the complexity of design and configuration that XML can manage automatically.

# 2.2 Intent Based vs Rule Based Configuration

Intent based configuration is a high-level abstraction of a description of network service along with some attributes (Leivadeas and Falkner, 2023) (Zeydan and Turk, 2020)This abstraction would mean that network configuration is based on features and functionality. This approach to configuration is beneficial because it provides a better end-user experience however there are drawbacks. This approach is now unable to solve how to implement significant configurations to networks whilst maintaining security and it cannot be used as an initial set up as it has very little ability to implement security measures as many of these measures would get in the way of features.

Current technology is unable to accurately and securely implement features on networks using the abstraction that intent-based approaches utilise (Bringhenti *et al.,* 2023). They go on to mention that intent-based approaches are not at the stage where they are faultlessly configuring networks to allow the main functions that the intent states they are improving and are most likely the future of Automated network configuration.

Rule based approaches to automated network configuration are more capable of securely and configuring networks. The RESTCONF approach to configuring networks is far more scalable than manual configuration and when set up correctly produces configuration results (Granderath and Schönwälder , 2023).

# 2.3 Network Planning

The challenges of not planning for the demands that will be placed on your network are that the network must be able to handle the expected peaks of usage on the physical and medium access layer (Wu, 2005). The major points to consider are does information need to be bidirectionally transported around any parts of the network. The second is how robust must connections be i.e. do devices need multiple ways to connect to the network or is one wire enough. The third thing to consider is the device or physical constraints. This means if employing topologies like star or mesh does the device have enough ethernet ports to be able to connect to every other device (Sankpal *et al.,* 2022).

Planning your computer network in topologies is important because topologies defining the layout, virtual shape and structure of a network both physically and logically will when planned effectively at the beginning allow the network's size and complexity to grow steadily and manageably. A well-defined physical and logical network will inform how different systems and nodes are connected and communicate with each other (Singh and Kumar , 2018).

# 2.4 Network Mapping

A network mapping algorithm defines a system of components into two different types of nodes. The first is Hosts and the second is switches. This mapping algorithm works by sending probes of increasing length into the network. The algorithm looks for non-null responses to build a new model vertex. These model vertices are then searched for duplicates because switches show up multiple times and must be detected as switches to build an accurate model of the network (Mainwaring *et al., 1997*).

The importance of having a mapping algorithm to be able to map virtual networks is because of the use of cloud networks which have become far more common therefore mapping algorithms used by companies should be able to work on virtual networks and physical ones seamlessly. The difficulties with this are when virtual links, virtual nodes or a shared substructure have constraints. Therefore, mapping algorithms should include a method such as a heuristics mapping approach that can handle online virtual network requests (Hsu and Shieh, 2013).

The challenges in assigning virtual networks to underlying physical networks in an efficient manner. The proposed solution is to use a distributed algorithm with load balancing (Houidi, Louati and Zeghlache, 2008). This approach optimises for efficiency when maintaining up-to-date information about the virtual network.

# 2.5 Router Configuration

Routers are the backbone of the internet. Their job is to forward traffic between networks and deal with issues such as scheduling support for differential services (Desig, Keshav and Sharma, 1998). Routers must be capable of being secured against many different forms of attack but also must be capable of forwarding lots of different types of traffic to enable all the different types of services that are wanted on a network

(*Cisco 800M Series Integrated Services Routers Software Configuration Guide*2015). However, router configuration must be done in a way that all routers, even legacy ones are able to be configured. This is important because many of these systems are still fit for purpose if they can be configured properly (Hussain *et al., 2017*)

# 2.6 Switch Configuration

Switches are vital parts of networks. Switches transfer data packets between its multiple inputs and output ports. However, switches unlike many devices have automatic processes that cannot be changed. It depends on the switch, but some have features like medium access control address learning, forwarding and filtering of packets, loop prevention, Auto-Negotiation and Broadcast and Multicast handling. However, it is possible to get more sophisticated switches that can change some of these features or don’t possess vulnerable versions (Wang *et al., 2013*)

# 2.7 Firewall Configuration

The first line of defence within a network is the firewall because it acts as the access control to the rest of the network. Firewalls are prone to misconfiguration which can lead to a false sense of security for the network behind it. This false sense of security can lead to networks being configured for convenience instead of security. However, more recent advances with Next Generation firewalls can be auto generated based on the networks and threats present (Alicea and Alsmadi, 2021). This approach to auto generating firewalls based on the network and threats will reduce the chance of human error which is the most common cause of a cyber-attack.

# 2.8 Existing Solutions

There are various existing solutions to the problems that will be addressed within this paper. One such is the SNMP which allows for easy administrative access to the devices in a network. However, SNMP is mainly used for reading information about a device such as performance metrics and device health and status. SNMP has limited control functionalities used for configuring devices (Case, 1988).

Then NETCONF was created to fix the shortcoming for the SNMP. NETCONF had stronger authentication and encryption, more granular access control and better structured representation (Schönwälder et al., 2010) (Ji *et al.,* 2009-03). However, NETCONF has limited support across vendors and isn't enabled on the legacy hardware that couldn't handle the newer versions of operating software.

Intent based network configuration is at the moment not widely used within industry because it lacks refinement and the same capabilities that rule based approaches have. The proposed approach using a large language model would translate high level natural language into low level configurations and would employ dynamic in-context learning (Tu, Nam and Hong, 2025)(Sreerangapuri, 2023). The approach shown in this paper is promising as to the accuracy of the translation and swift processing. However, it fails to touch on the security of the configurations. The challenges mainly stem from the fact that intent based still leads to security misconfigurations. Furthermore, it introduces a new problem where Semantic Gap Vulnerabilities can be introduced where the program can misinterpret the high level command it is given and configure devices to behave in unintended ways (Kim et al., 2024). There is currently an internet based solution that assists users with planning configuring and can verify that devices on the network are functional. Cisco DNA Center is a network management and automation platform that centralises control and management of Cisco digital network architecture(Dna et al.). This solution leverages artificial intelligence to convert the high level user input into low level commands. It does so by asking how the user would like the network to act and then making rules that will allow the network to do so. Cisco however attempts to solve the problem of security by providing visibility and metrics for network health and security. Cisco Further leverages Ai to provide users with insights and analysis of the network based off of the data collected.

Mapping computer networks can be very difficult depending on the scale and the variety of services provided on the network. There are various methods to be able to efficiently map a network. One approach is to use the kth-Hop Traceroute method. This method uses the Traceroute tool which is commonly used for diagnostics and uses it to find the path that the packet takes from the destination IP address to the target IP. This method allows a user to see all devices on a network and how they all connect (Afshar *et al., 2022*).

# 3. Methodology

The methodology begins with the High-Level Manual Setup Guide, detailing physical device connections and initial remote access configuration for non-technical users. Next, the Graphic User Interface (GUI) design is explained, emphasizing usability through simplified input methods like dropdown menus and buttons. The Topology Questionnaire Script section outlines the interactive process for gathering user requirements and generating network architectures, followed by the Topology Generation Implementation, which describes the algorithmic translation of user inputs into visual network diagrams. Network Virtualization covers the use of GNS3 and VMware for testing, while Network Mapping explains IP scanning and host discovery. The Connection Implementation and Device Configuration sections detail secure remote access (SSH/Telnet) and automated router/switch/firewall setups using Python. Finally, Testing validates functionality across different network architectures, and a User Survey assesses the tool’s accessibility for non-experts. Each phase integrates academic literature to justify design choices and ensure alignment with networking best practices.

## 3.1 High Level Manual set up Guide

The manual set-up guide is to inform users of how to configure networking devices so that they are able to fulfil the needs of the network topology that is recommended to them by the script(*Cisco 800M Series Integrated Services Routers Software Configuration Guide*2015). Devices such as routers, switches, firewalls, servers and workstations then need to be physically connected using cabling either to each other or to a power source. The guide will explain different types of cables used and what to plug them into(Sankpal *et al.,* 2022).

The initial configuration must then cover how to prepare devices to allow initial remote access for the script(Sankpal *et al.,* 2022; Ji *et al.,* 2009-03). This will require an IP address to be created and the connection interface to be configured to allow traffic to and from the device. Furthermore, Open Shortest Path First(OSPF) should be configured on all network hardware to route the traffic around the network in an efficient manner. This in turn will allow the automated setup script to actually interface with intended devices and allows users with little to no technical knowledge to still be able to use the tool(Sankpal *et al.,* 2022; Ji *et al.,* 2009-03; Leyk, Mcinvale and Chen).

As this section does not require coded integration with the script it can be done as a .txt or .pdf

document using text editors such as notepad, google docs or word.

## 3.2 Graphic User Interface

To improve usability a graphic user interface is implemented with a help page to explain what each button does(Shu *et al.*) To improve usability the number of choices a user is able to make is limited so that as few questions as possible are not open-ended. This way of objectively questioning the user with well defined questions and accompanying explanations will allow users with no prior knowledge to use this tool to plan ,map and configure a network(Widmer , 1996). To achieve this a GUI is the best solution that will allow the usage of buttons, dropdown menus and tick boxes will significantly improve the usability when compared with alternatives such as the command line interface (Kumar and Dahiya, 2017).

The GUI for the network set-up and mapping tool is designed to be simplistic and ensure that the user is not overwhelmed with information at any stage. This home page is the place where all three different parts of the tool can be launched from with the push of a button. As shown below in figure 3. This GUI was made using the tkinter library for python 3 the code for which is shown below in figure 4.

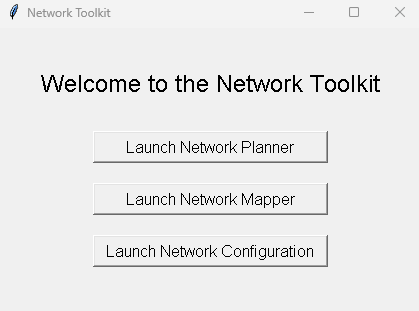


Figure 3 GUI homepage



Figure 4 Code for Homepage

## 3.3 Topology Questionnaire Script

Network architecture refers to the layout of interconnected devices (Singh and Kumar, 2018) A well designed network architecture has the benefits of improving the efficiency, scalability, reliability, automation and security of the services provided by the network and data held on it (Wu, 2005). The script described will be an interactive and user-friendly way of having the user think about what the network’s needs are and then storing them to be used later in the configuring of the network (Leivadeas and Falkner, 2023). Because this project is aimed at non-technical users or companies without dedicated network staff the number of architectures suggested will be limited to four because they are easier for less skilled users to understand and configure (De La Puente and Perez, 2023).

Questions posed to the user will be asked in a non-technical way. This means that a user with no prior knowledge will be able to fully understand what the question is asking and then be able to answer accurately what they want.. This script must cover the pertinent information such as expected size, scalability, fault tolerance, and type of data being shared. The script will then generate a topology that will be used to inform the configuration portion of the script. The questions, answers and meanings are shown below in Appendix A 1 and if the answers don't unanimously lead to a given architecture then the answer that has the most correlation with the answers will be recommended.

This part of the script will be multiple choice and has error handling to mitigate the chance for mistakes by the user. However this part of the script is low complexity to program and is therefore programmed in python which is the same language that is being used for the topology generator.

### 3.3.1 Designing the Topology Questionnaire script

To generate a network topology that accurately reflects user requirements, a structured questionnaire-based approach was adopted (Bringhenti et al., 2023). The script begins by asking five targeted questions: the number of workstations, the desired level of scalability, fault tolerance, security, and the required network services (Alicea and Alsmadi, 2021). Each question is carefully designed to correspond to key aspects of network planning (Wang et al., 2013), ensuring the resulting topology is not only technically viable but also aligned with the user’s specific goals (Hussain et al., 2017). By starting with the number of workstations, the script establishes the size of the network, which directly influences infrastructure needs. Scalability helps determine how easily the network can grow in the future, while fault tolerance informs the level of redundancy necessary to ensure continued operation during hardware failures.

This approach was chosen because it offers a balance between simplicity and customizability. Rather than overwhelming users with low-level configuration details, the script abstracts complex design choices into five core criteria that most stakeholders—technical or not—can understand and respond to. This design promotes usability and encourages informed decision-making by focusing on high-impact network characteristics. Furthermore, these questions map cleanly to algorithmic decisions within the script, enabling efficient generation of network topologies that are tailored, practical, and adaptive. This modular input system makes the tool scalable and extensible for future enhancements, such as adding more granular service configurations or policy-based routing features. The following sections 3.3.1.1-6 will break down further each part of the script and how it functions

### 3.3.1.1 Number of devices

The number of devices recommended by the script is based on three factors: the first is the type of architecture that is best suited for the network's needs. This affects how many servers are needed. Second is the amount of scalability that is needed on the server this will affect how many routers, switches and servers are needed as well. Lastly is how many workstations are required. This along with scalability and services gives a complete idea of how many devices are needed. For the script to know how many devices the user wants it prompts the user to enter the number when they press the Generate Topology button on the home screen. The script will then show the screen below in figure 5.

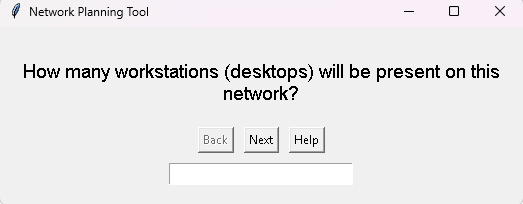


Figure 5 Workstation Question

This portion of the user interface has three buttons and a description at the top of the page. These buttons are common among all pages of the generate topology questionnaire. The first the back button will take a user back to the previous page to change their answer to a question. The Next button will save the user's answer and present them with the next question. The final button “Help” will cause the script to make a pop up text that gives more details on what the question is asking and how to answer it. Below is the code used to build the user interface shown in figure 6.

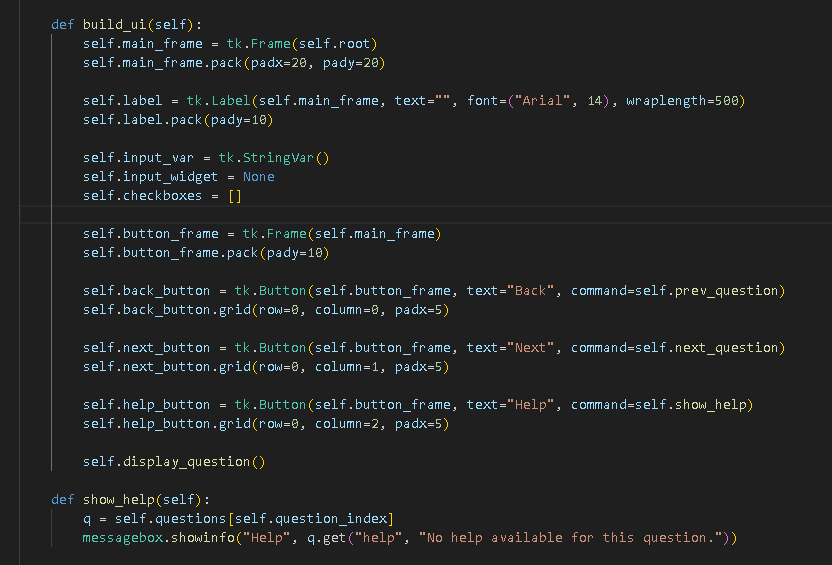


Figure 6 Build user interface Code

Next how each type of question is displayed as there are three different types of questions so they all require different ways of answering them. First there is the “Entry” type which is used for entering the number of user devices. Second is the “Options” type which is used for Scalability, Redundancy and Security. This type has a drop down menu which will be shown later in figure 9, 10 and 11. Finally there is the “checklist” type that allows a user to check as many of the options as they want instead of just one which is shown below in figure 7. The code for how each of these is shown below in figure 8.

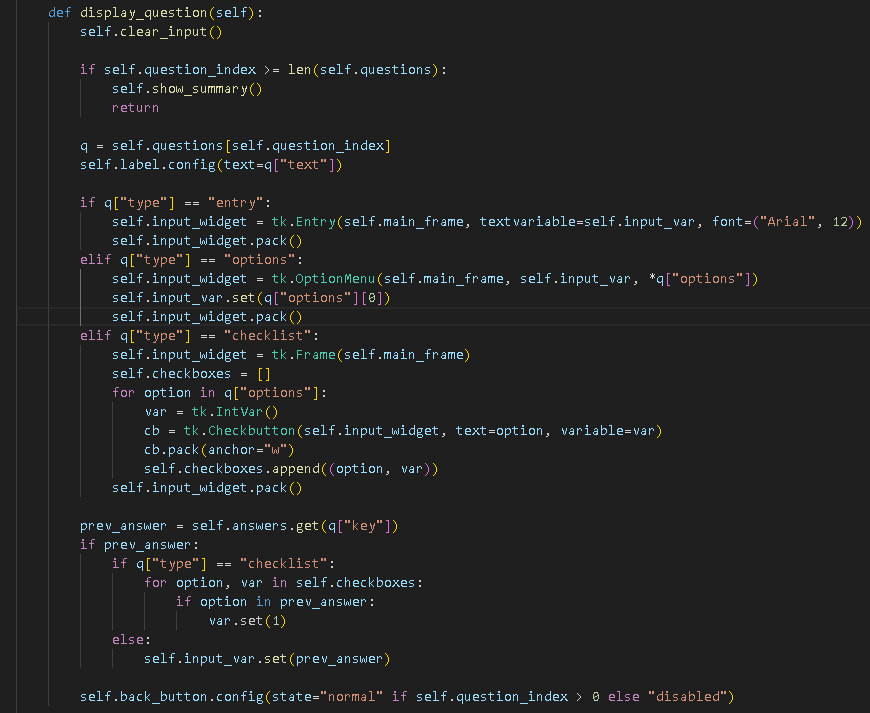


Figure 7 Display question Code

Once the question has been answered appropriately and the user presses the “Next” button the script will determine what the next type of question will be and populate it with options before displaying it. The first part of displaying the question is ensuring that the screen and inputs are clear of any residue from the previous question or screen. This is done using the “clear\_input” function which is shown below in figure 8.

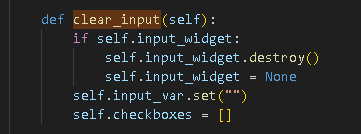


Figure 8 Clear input Code

### 3.3.1.2 Scalability

The second factor in generating a topology diagram for the user and planning their network is the amount of scalability that would be required on the network. This has three options: Low/None, a moderate amount and a lot. Each of which has its own effect on the ratio of workstations to switches. At low to none there will be 6 workstations to a switch, 3 switches to a router and 2 routers to a server assuming there are any servers. If there are more than 1 server then they will all be connected to a router to allow traffic to travel securely in between each server. If there are no servers then all the routers can either be connected together in a mesh architecture where every router is connected to every other router or in a peer to peer architecture where all routers are connected together in series. As the amount of scalability increases the number of workstations to a switch reduces from 6 to 1 to 4 to 1 and then when lots of scalability is needed it becomes 2 to 1 of workstation to switch. The user interface for the scalability question is shown below in figure 9.

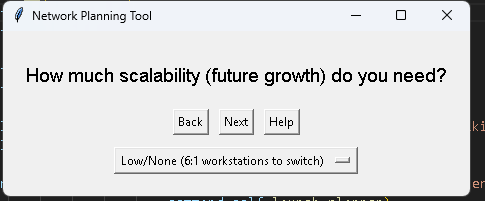


Figure 9 Scalability Question

### 3.3.1.3 Redundancy

The third question in generating a topology is how much fault tolerance the network should have. This means how different paths can any given device on the network use to route traffic to any other given device on the network. The lowest example of fault tolerance is where every device only has one path it can take to get to another device. So if any connection along that route fails then communication breaks down. To increase the fault tolerance there should instead be more path that traffic can take such as having the router connected together so that if a switch's connection to a router fails then it can re-route traffic through another switch maintaining functionality of the network. This is an example of moderate fault tolerance. For high amounts of fault tolerance as mentioned earlier in some cases routers can be connected together and workstations that are connected to the same switch can all be connected together if they have enough ports. The user interface for this question is shown below in figure 10.

### 

Figure 10 Scalability Question

### 3.3.1.4 Security

The fourth question is security and how extreme does the user want it to be on the network. At the lowest level of security only incoming traffic from outside the network should be filtered and logged. At the moderate level all traffic traveling through the router i.e. between the server or from switch to another should be filtered. At the highest level all traffic traveling through all routers and switches on the network will be filtered and logged ensuring maximum security. Furthermore as part of the basic configuration portion of the script all devices will have various services disabled if they are enabled and shouldn't be. The user interface for this question is shown below in figure 11.

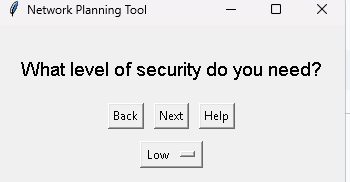


Figure 11 Security Question

### 3.3.1.5 Services

The fifth question is the services that the user wants to have available on the network. There are three services that this script checks for the first is file sharing which allows for devices on the network to share files to other computers. The file sharing ports are 139 and 445 which is commonly used by server message blocks. Another way would be using the file transfer protocol which uses ports 20 and 21. The second service is database hosting service in which different types of databases use different ports but SQL server uses 1433, Oracle uses 1521, MySQL uses 3306 and PostgreSQL uses 5432. The use of this is if a network needs to host a database like the back end of a website or other general information that needs to be queried frequently and efficiently. The final service is website hosting which is for users that want to host their own website and either allow users on the network to access or if the server where the website is being hosted is connected to a router that is connected to the internet allow users from all over to access the website. For the sake of security the only type of website port that was considered was the hyper-text-transfer-protocol -secure on port 443. Shown below in figure 12 is the service question.

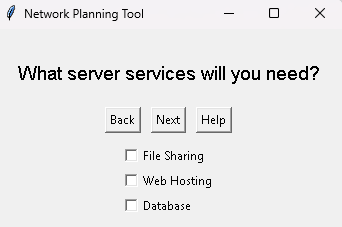


Figure 12 Services Question

### 3.3.1.6 Architecture Choice

Finally the questionnaire uses the users answer to all the questions and makes a decision about what architecture would best fit the users needs. The first architecture Mesh requires that there only be a total of 18 switches, no database or website hosting needs and that the user wants high redundancy. In the mesh architecture all switches are connected to all the other switches that are connected to the same router. All routers are connected to all other routers and all workstations are connected to all other workstations that are connected to the same switch.

The second type of architecture peer-to-peer is created when the users ask for low to moderate redundancy and no database or website hosting. This architecture has all routers connected together in series. It also has up to three switches per router depending on redundancy up to 6 workstations per switch.

The third architecture type is a centralised hub. This type of architecture has one server where all the services for the network are hosted. This architecture is chosen when there are less than three routers needed and database or website hosting are needed.

The final architecture is a client to server architecture where all servers are connected to a router to allow traffic to travel from one server to another.

## 3.4 Network Virtualisation

The test environment must be able to simulate small to large device networks as well as the functionality of each device be simulated realistically  (Hsu and Shieh, 2013). Furthermore the environment must be able to be connected to the developers computer so that the set-up script can be deployed without needing to code it from scratch every time the network is restarted (Houidi, Louati and Zeghlache, 2008). As such the tools GNS3 and VMWare will be used as this combination of tools allow for a very accurate virtual version of small to large networks and their functionality (Mainwaring et al., 1997).

The test environment GNS can accurately simulate various types of devices with a plethora of operating systems. On top of this GNS is capable of handling large and small networks and in real time simulates what each device would do.

## 3.5 Topology Generation Implementation

The Topology Generation will include the ability for four different topologies to be generated out of the seven considered (Sankpal et al., 2022); these four were chosen because they are the simplest and cover all the needs that a basic network has  (Zeydan and Turk, 2020). The Topology generation will allow users to tweak any configuration before the user begins the device configuration process portion of the script (Choi, Choi and Hong, 2004). The user may choose to add additional devices such as more workstations or routers that were not originally planned by the Topology generation script (Afshar et al., 2022)..

The algorithm takes inputs such as the number of workstations, scalability requirements, redundancy levels, security preferences, and server needs, then dynamically generates an optimised network topology. The process involves evaluating these parameters, calculating device quantities, determining physical placements, establishing connections, and assigning IP addresses—all while adhering to predefined rules that ensure the network meets the specified functional requirements. The script calculates the number of switches and routers based on the number of workstations and scalability that has been specified by the user, the code for which is shown below in figure 13. Furthermore math.ceil is used when calculating to ensure that the calculation always rounds up and comes out to a whole integer.

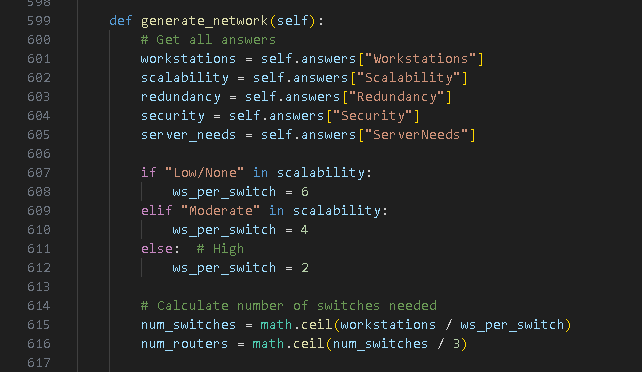


Figure 13 Calculating number of Switches and Routers

The algorithm begins by assessing the input parameters to select an appropriate topology type. Four primary types are considered but broken down into these types by two if statements shown below in figure 14 and 15. These if statements check for redundancy, server needs number of devices present on the network to divide up the architectures.

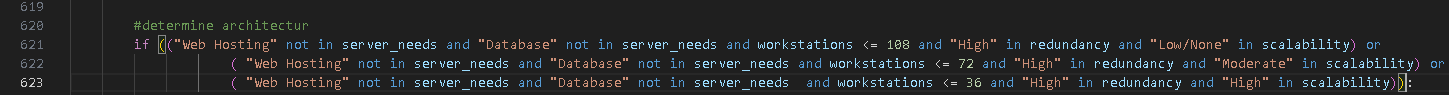


Figure 14 Topology generation Mesh If statement



Figure 15 Topology generating Peer-To-Peer if statement

### 3.5.1 Generating a Mesh Topology

The mesh topology generation process creates a highly interconnected network structure designed for maximum redundancy and fault tolerance. This implementation specifically generates a hexagonal mesh configuration when high redundancy requirements are detected in combination with no server needs. The topology features routers arranged in a circular pattern with multiple interconnected paths between devices, ensuring alternative routes exist in case of link failures. The routers are processed sequentially meaning one router then a switch connected to the router then the workstation connected to the switch and so on. This is shown below in figures 16, 17, 18 and 19 .

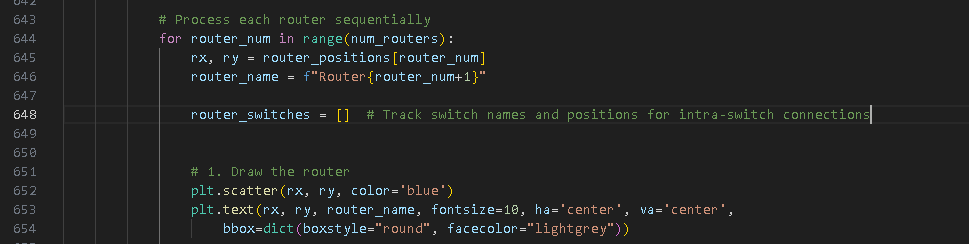


Figure 16 Processing Routers

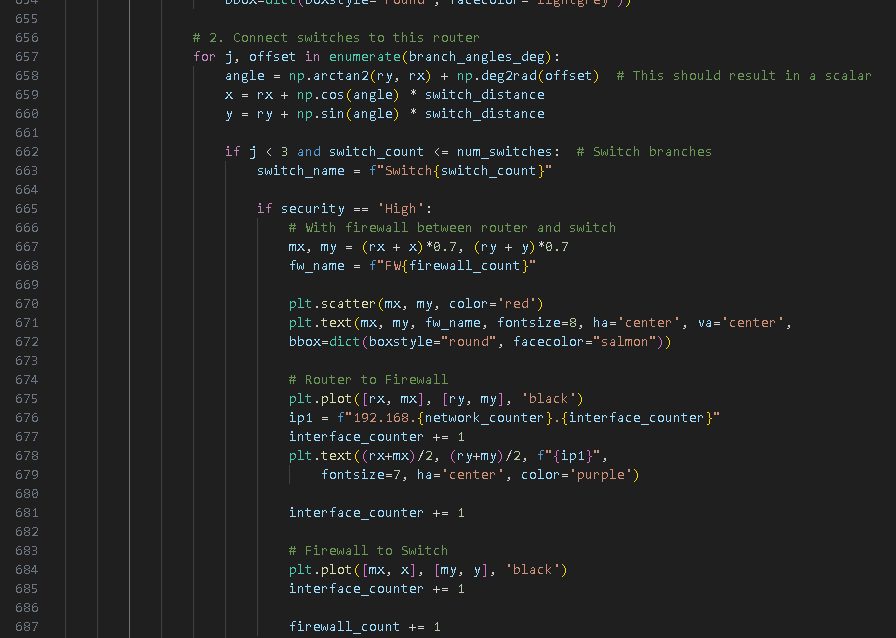


Figure 17 Processing Switches



Figure 18 Connecting Workstations to Switches

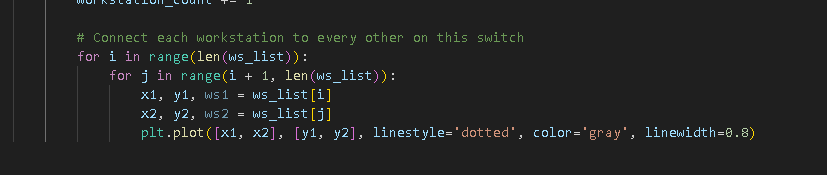


Figure 19 Connecting Workstations to each other

The generation process begins by calculating optimal device positions using geometric principles. Routers are positioned at equal angular intervals (60 degrees apart) along a circle with a predefined radius (50 units in this implementation). This creates the hexagonal base structure of the network. For each router position, switches are then placed at calculated offsets using four primary branch angles (-30, 0, 30, and 60 degrees) relative to the router's radial position. This arrangement creates a balanced distribution of network components while maintaining consistent spacing between devices.

Device connectivity follows a multi-layered approach to establish the mesh characteristics. Each router connects to its adjacent routers either directly or through intermediary firewalls when high security is specified. The difference between a direct connection and through a firewall when generating a topology is shown below in figures 20 and 21.

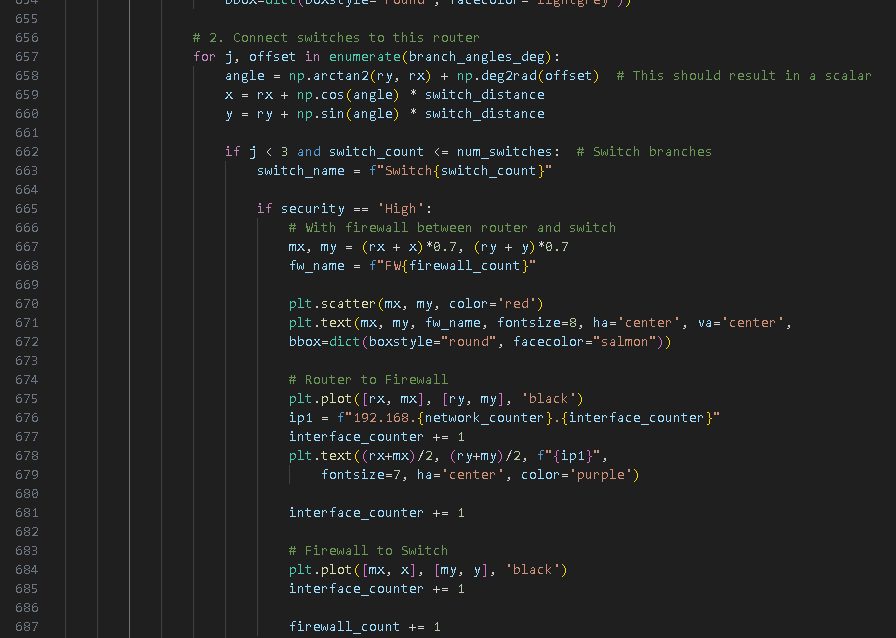


Figure 20 Connection with firewall

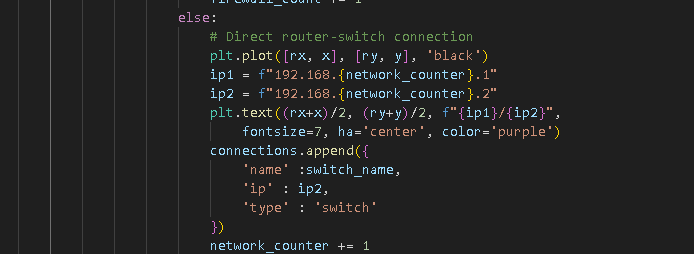


Figure 21 Direct Router to switch connection

The switches connected to each router are themselves interconnected in a partial mesh configuration, creating redundant paths at the switching layer. At the workstation level, all devices connected to the same switch are fully interconnected with dotted-line relationships, visually representing the mesh topology's redundant nature. This three-tiered mesh structure (router-to-router, switch-to-switch, and workstation-to-workstation) provides comprehensive path redundancy throughout the network hierarchy.

The implementation includes IP address management that automatically allocates addresses based on device type and network segment. Each router-switch pair receives its own /24 subnet, with interface addresses assigned sequentially. Workstations inherit addressing from their parent switch's subnet while maintaining unique host identifiers. The visualisation system employs color-coding (blue for routers, green for switches, orange for workstations, red for firewalls) and clear labeling to enhance diagram readability. Connection lines display interface IP addresses at their midpoints, and security devices are prominently highlighted when present. This combination of geometric precision, logical addressing, and visual clarity produces a complete mesh topology representation that meets specified requirements for redundancy, security, and scalability.

The drawing of the mesh generation follows a systematic process implemented through several key steps. First, Using numpy's linspace function, it computes six equally spaced angular positions around a circle (0°, 60°, 120°, etc.) but only activates the number needed based on the router count. For each active router position, the code then generates four potential connection branches (-30°, 0°, 30°, and 60° relative to the router's radial angle) where switches can be placed.

Security integration occurs at multiple levels in the mesh topology. When high security is specified, firewalls are automatically inserted between routers and switches, appearing at 70% of the connection distance. These security devices are visually distinct with red coloring and salmon-colored label backgrounds. Additional firewalls may be placed between interconnected routers depending on security settings. The workstation interconnection feature activates only when high redundancy is requested, creating a true mesh at the endpoint level by drawing dotted gray lines between all workstations sharing the same switch.

The IP addressing scheme maintains consistency across the mesh structure by incrementing network segments systematically. Each new router-switch pair advances the third octet (network\_counter), while interface addresses use sequential fourth octets (interface\_counter). The code carefully manages these counters to prevent overlap and maintain proper subnet relationships. Workstation connections display both interface IPs (switch and workstation) on their connecting lines, while router-to-router links show the transit IPs when firewalls are present.

Visual customisation parameters like router\_radius (50 units), switch\_distance (50 units), and workstation\_distance (10 units) ensure proper spacing between device tiers. The matplotlib figure size (12×12 inches) provides adequate canvas space for complex meshes, with automatic axis equalisation maintaining proper proportions. Final diagrams include comprehensive titles detailing the security level, redundancy setting, and device counts, providing immediate understanding of the topology's characteristics. This implementation balances algorithmic precision with flexible parameterisation, allowing generation of mesh topologies tailored to specific network requirements while maintaining visual clarity and accurate technical representation.

### 3.5.2 Generating a Peer-to-Peer Topology

The peer-to-peer topology generation creates a horizontal linear network structure optimised for straightforward deployments without server requirements. This implementation differs fundamentally from the mesh approach by employing a sequential router arrangement and modified connection patterns that emphasize direct linkages between adjacent devices.  
The topology begins by positioning routers along a straight horizontal axis at fixed intervals (15 units apart). Each router serves as a connection hub with four potential branches (-30°, 0°, 30°, and 60° relative to vertical) for switch placement. Unlike the radial mesh design, this linear approach creates a cascading network flow where data travels sequentially through router nodes. The 0° branch typically carries primary traffic, while the angled branches provide supplementary connections that can be activated based on device counts and redundancy requirements.

### 3.5.3 Generating a Centralised Hub and Client Server Topology

The centralised hub and Client server topology generated by this script are different only in the fact that one only has one server and the other has multiple servers that are connected together using a router. The generation algorithm constructs a hierarchical network architecture with servers as the focal point, designed for environments requiring dedicated server resources. The implementation positions servers along a horizontal axis below the primary router tier.  
The system begins by calculating the required number of servers based on router count (ceil(num\_routers/2)), then positions them with fixed vertical spacing (y = -8) and dynamic horizontal distribution. When multiple servers are present, a dedicated Server Router is automatically introduced above the server cluster to centralise connections. Servers are visually distinguished by teal-colored nodes with light cyan labels, while their connections to routers use sky blue dotted lines for immediate visual identification.  
Server connections incorporate configurable security measures, including optional firewall placement at 60% of connection paths when high security is specified. The algorithm enforces strict access policies between servers and routers, with firewall-protected links displaying IP addressing. For high-redundancy configurations, routers sharing the same server establish secondary connections, creating backup pathways that maintain network availability if primary links fail.  
While workstations maintain standard star-topology connections to their local switches, the overall network traffic patterns emphasize client-server communication. The IP addressing scheme preserves logical segmentation, with servers occupying dedicated subnets while workstations inherit addressing from their respective switch segments. This approach ensures clear traffic flow management while maintaining compatibility with the system's security and redundancy frameworks. In the following section 3.5.5-8 explains what each architecture chose to be generatable by the script and also explains why other architectures were not used.

### 3.5.4 Peer-to-Peer

All devices are considered equal and can act as both servers and clients. There’s no central server. The Use case this is for small home networks.

### 3.5.5 Client-Server Architecture

This architecture uses one or more central servers to provide resources and services to client devices.The use case for this kind of architecture is Web hosting and corporate internal networks.

### 3.5.6 Centralised Network Architecture

This is a classic centralised system where all network resources and data are stored in one server. The use case of architecture is a traditional mainframe system.

### 3.5.7 Mesh Network Architecture

This network architecture has every device connected to every other device in the network, directly or indirectly. The use case for this architecture are smart home systems and military communications.

### 3.5.8 Omitted Architectures

The following architectures have been omitted because they are more complex to configure as they include a more varied selection of devices and techniques such as cloud networking and having multiple software systems deployed across multiple devices.

* Distributed Network Architecture
* Cloud-Based Network Architecture
* Hybrid Network Architecture

## 3.6 Network mapping

The network scanning implementation follows a structured approach to identify and analyse hosts within specified IP ranges (Case, 1988). The system architecture employs a graphical user interface built using Tkinter, providing users with intuitive controls for host input, subnet discovery, and scan initiation (Schönwälder et al., 2010). The interface consists of a main application window containing input fields, action buttons, and a display area for listing target hosts (Tu et al., 2025). Users can manually enter individual IP addresses or entire subnets, with input validation ensuring proper formatting before processing (Kim et al., 2024) this is shown below in figure 22.

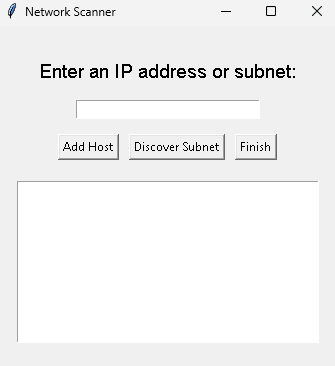


Figure 22 Network Mapping Interface

For subnet discovery, the system integrates the Nmap scanning tool through Python's nmap module. When users initiate a subnet scan, the application performs a ping sweep (-sn flag) to identify live hosts within the specified range. The discovery process captures all responsive hosts and adds them to the target list while filtering out duplicate entries. Error handling mechanisms manage potential scanning exceptions, providing users with descriptive feedback through message dialogs when issues occur. The code for discovery mode is shown below in figure 23.

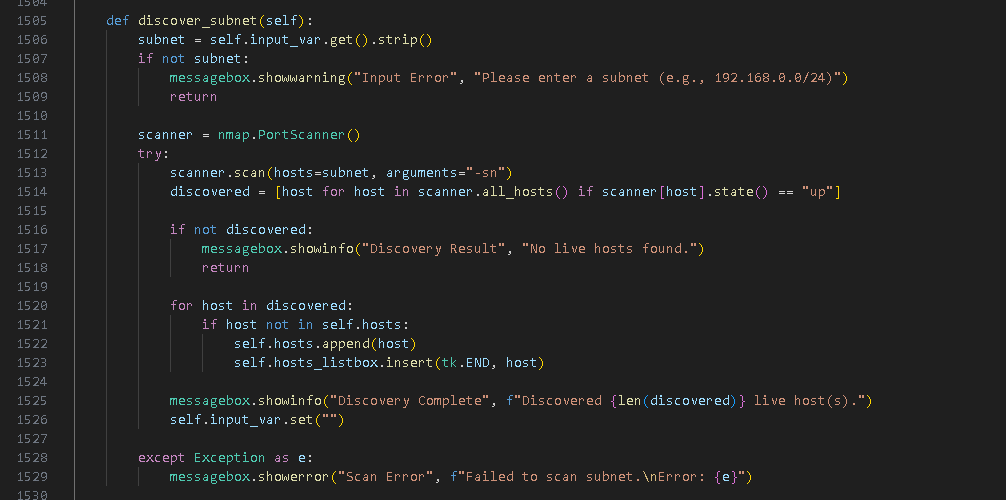


Figure 23 Subnet Discovery Code

The scanning methodology implements concurrent processing to efficiently handle multiple hosts. Upon scan initiation, the system first verifies host availability through a preliminary ping sweep before proceeding with detailed port scanning. A thread pool executor manages up to 10 simultaneous scan operations, optimising performance for networks containing numerous hosts. Each scan examines all available protocols and their associated ports, recording the state (open, closed, filtered) of each detected port.

Results processing incorporates comprehensive output generation, with scan details written to both the console and a persistent text file (scanresults.txt). The output format organises information by host, including protocol details and port statuses, facilitating analysis of network configurations. Error handling at multiple levels ensures recovery from network issues or scanning exceptions, with error messages preserved in the output for troubleshooting purposes.

Performance optimisations include intelligent thread management and selective scanning parameters that maintain thoroughness without unnecessary overhead. The system's modular design separates the user interface, scanning logic, and result handling components, enabling future expansion of functionality while maintaining current operational stability. The code for scanning specific hosts is shown below in figure 24.

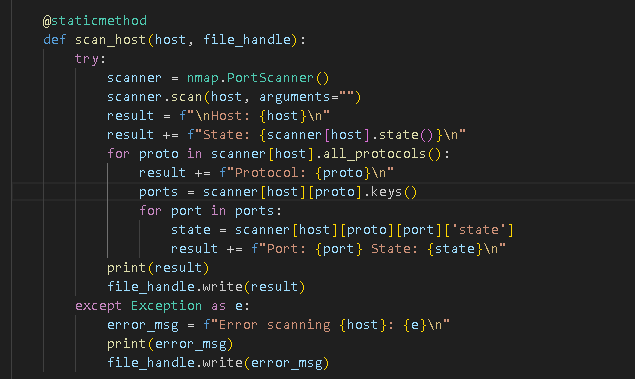


Figure 24 Scanning Specific Hosts code

## 3.7 Connection Implementation

The connection part of the script must allow the script and device that it is being launched from to connect to all devices on the network in the correct order to avoid errors or missed devices. This connection must create a privileged shell within the device that can then be used to configure the device according to what is suggested in the topology generation section of the script. This connection will be able to be made in multiple different ways but the connection will be changed to a secure method such as SSH. After this secure method of configuration has been enabled the device should then have the remaining configurations performed. The language that will be used to connect and configure will be Python as this language has good integration with many well maintained libraries which simplify the process of connecting securely to a device and then configuring it. Shown below in figure 25 and figure 26 is the code used to connect to the devices using both ssh and telnet depending on how the device is configured.

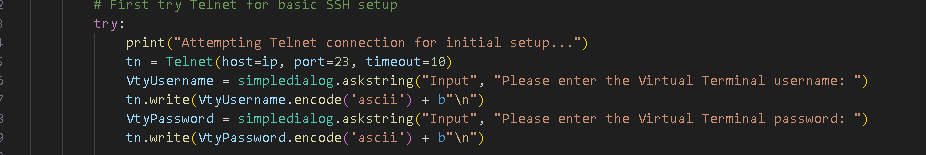


Figure 25 Telnet connection code

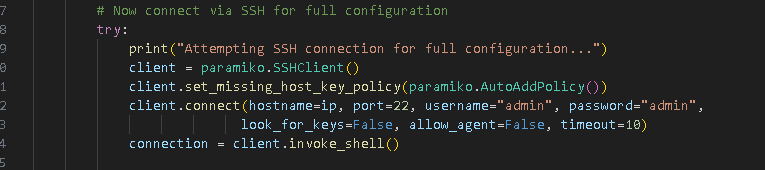


Figure 26 SSH connection Code

## 3.8 General Device Configuration

Device configuration must be done in a way which is secure and has good error handling. The security must come in the form of adhering to best security practices. Such as ensuring that logging and monitoring is enabled. Furthermore the devices visibility of the devices status must be made in such a way so that only users that should have access for the purpose of monitoring and maintaining are able to check it.

For communication the devices should have SSH enabled as the only method whereby a user can remotely access the device. The Devices should have several users with different privileges levels so that even if a lower level user is compromised then only the part that that user was able to modify is at risk. Furthermore the user accounts present on the devices should have complicated passwords that cannot be easily brute forced and should be changed regularly and stored in a secure manner.

For these configurations to be successful on as many devices as possible it is important to consider legacy devices that do not have modern configuration capabilities.

## 3.9 Device Configuration for Routers

Routers require a few basic configurations that are not unique to it but are not required unilaterally by devices on a network. Such as Disabling Domain Name System(DNS) Lookup, Hostname, Secret password so that it is secure and a login Banner. All of which will need to be specifically configured to the device and adhering to the generated topology.

Furthermore for the actual interfacing and routing that the router does its local area network and wide area network interfaces must have the correct IP address and subnets assigned to each. A default gateway is required to ensure proper packet handling and finally a Network Address Translation(NAT) should be configured to again ensure proper packet handling and interfacing with the internet if the devices supported by the router require it.

Security on a router is very similar to that of the general best practices. Meaning SSH only for remote access and all unused services are disabled. Finally routers can be used to implement a Access control List which enforces a set of rules that determine which users or processes have access to the resources and operations that the router guards.

The routers can also be configured to have system logs and send those logs to a central logging server which will depend on the network and security requirements stated by the user in the Topology script. The code for configuring the router is shown below in figure 27 and 28.

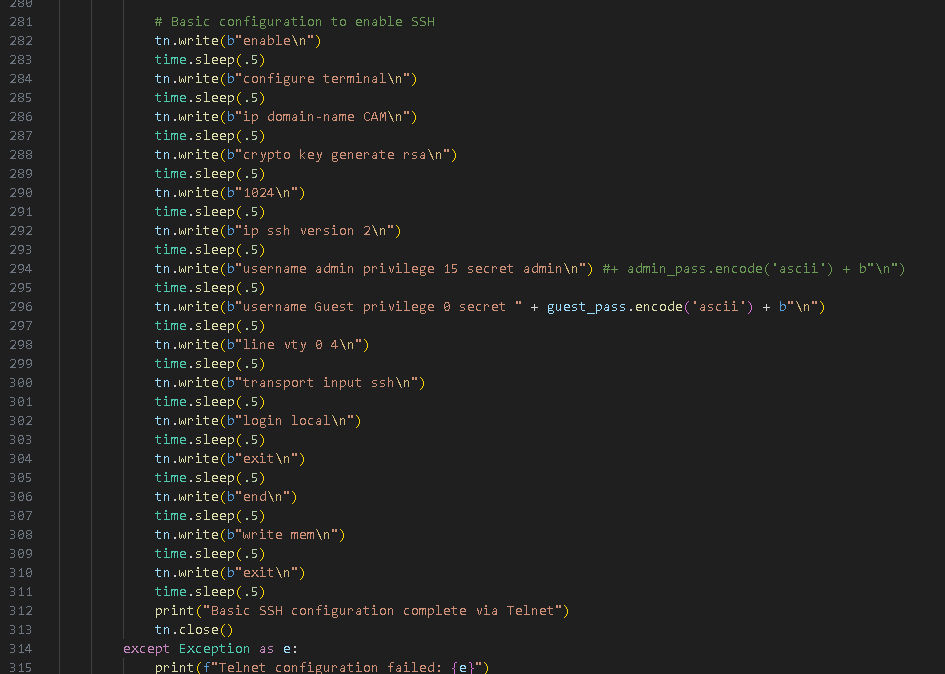


Figure 27 Telnet Configuration code



Figure 28 SSH Config Code

## 3.10 Device Configuration for Switches

Switches also require very similar configurations as the router. All the same basic configurations such as DNS Lookup, Hostname, Secret password and login Banner. The security configurations are also very similar as both devices use the same operating system which is Cisco IOS.

## 3.11 Configuration for Firewalls

Firewalls will be configured within key network devices such as switches and routers. Firewalls present on devices such as routers and switches mainly consist of access control lists and other such rules that either allow certain types of traffic to travel through the device. These rules can limit traffic coming from certain locations or going to certain locations or traffic that is attempting things such as accessing HTTPS on port 443 when the network does not have any database hosted on it. The specifics of what the firewall is configured to allow depends on the level of security, type of device that is configured on and what services are needed on the network.

The firewall and access control lists will be configured to allow all outbound traffic and block all incoming traffic that isn't ssh or ping protocol. Just like all other devices on the network the only secure method for securely remotely connecting to a firewall and by extension the only method that will be enabled is SSH. Again system logs will be stored and then configured to be stored temporarily on the device and then if there is a logging server send the logs to that server for more permanent storage. The code for configuring firewalls is shown below in figures 29 and 30.

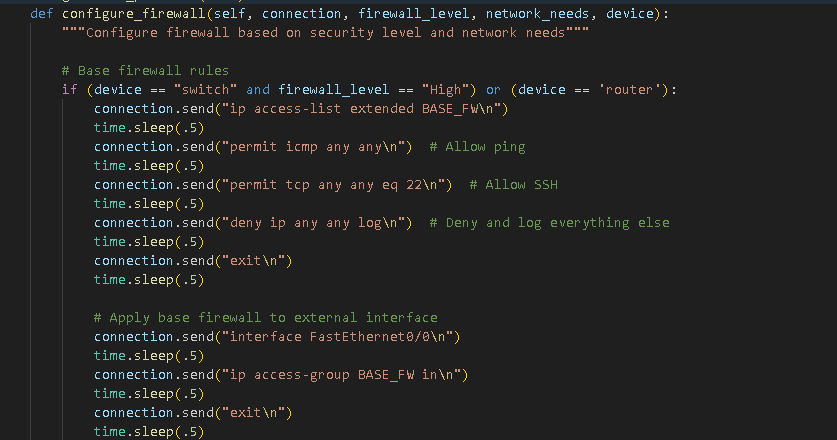


Figure 29 Basic firewall rules

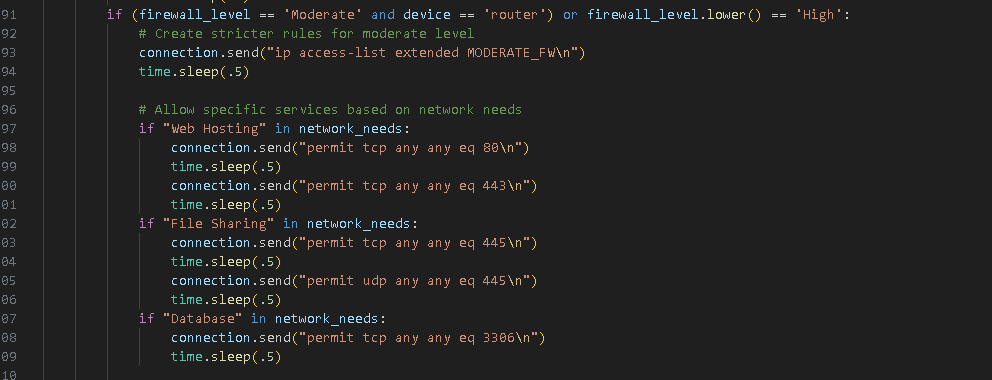


Figure 30 Stricter Firewall rules

## 3.12 Testing

To test the various features of the script there should be multiple test networks configured with varying qualities to test all the different capabilities and fringe cases.

### 3.12.1 Test Case 1

The first test case should be a peer to peer network with various services enabled that should not be. Some of the interfaces should be correctly configured to allow traffic to pass whereas others should be left incorrectly configured to force the script to configure the interfaces itself. Along with all this a loose description of the network requirements should be provided using the GUI it should then accurately generate a topology that matches the requirements. This will be a peer to peer architecture.

### 3.12.2 Test Case 2

The second test case will have a Client-Server architecture; this network. The server's ACL’s and firewalls will not allow any of the services that the network can be configured to allow. The script will have to change these devices and services to allow them. This case should also show off how to generate topologies with multiple servers.

### 3.12.3 Test Case 3

The third test case will be a Centralised Network which will showcase the maximum security mode having the maximum number of firewalls ensuring that the network is as segmented and protected as possible.

### 3.12.4 Test Case 4

The fourth test case will be the mesh topology where the high redundancy setting will be tested and the file sharing feature. This means the ACL’s should allow file sharing but nothing else.

### 3.12.5 Test Case 5

The Final test is having the network mapping feature accurately map all previous networks that are shown off in the test cases and checking that all summaries generated by it for the network are accurate.

## 3.13 Survey

To test whether or not the script and manual configuration guide proposed and developed in this paper are suitable to be used by non-technical users (Leyk et al., 2017). A survey of non-technical users who have used the tool and read the manual configuration guide should be conducted (Widmer, 1996). This survey should include asking the users how easily they understood the user interface, what it was trying to get them to do and how well they understood the output De La Puente and Perez, 2023). It should also test how much information the users were able to absorb from reading the high level manual set up guide as their understanding of that is paramount to effectively mapping and configurning portions of the script. The questions are shown below in figures 31 and 32.

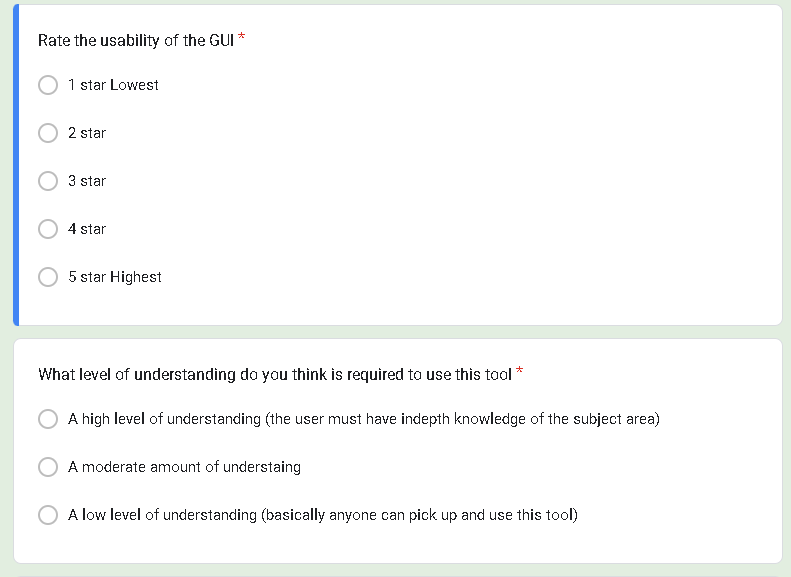


Figure 31 Survey Question 1 and 2

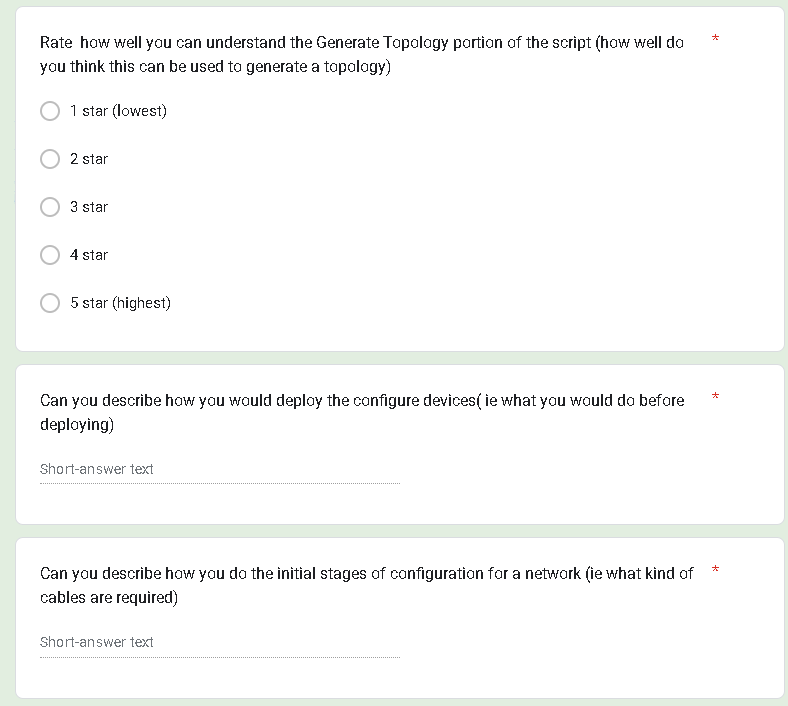


Figure 32 Survey Questions 3, 4 and 5

## 3.14 Proposed languages and tools

|  |  |
| --- | --- |
| Tools and languages | Benefits |
| Python | * Integrates with XML for device configuration and connection * Developer is experienced using this language for similar scripts |
| VScode | * Powerful Integrated development environment(IDE) * Developer is experienced using this IDE for similar scripts |
| GNS3 | * Developer is experienced using this virtualisation tool for similar scripts |
| VMWare | * Developer is experienced using this virtualisation tool for similar scripts |
| Google Docs | * This text editor has autosave functionality and spell checker * Developer is experienced using this text editing tool |

Table 3 Table of Proposed languages and tools

# 4. Results

This study investigated four key components: topology generation, network mapping, network configuration, and user survey responses. The topology generation algorithm produced four distinct network layouts. Network mapping was able to identify all devices on the network when in discovery mode]. The configuration module successfully deployed protocols security configuration and access control list protocols across all intended devices. Finally, survey responses with 14 participants revealed user perceptions of usability and previous technical knowledge requirements. The following sections detail these results.

## 4.1 Topology Generation

### 4.1.1 Test Case 1

The first test case should be a peer to peer network. It will have file sharing enabled. Furthermore it has 6 workstations, low redundancy, low security and low scalability. This causes the script to generate a very minimalistic topology diagram shown below in figure 33.

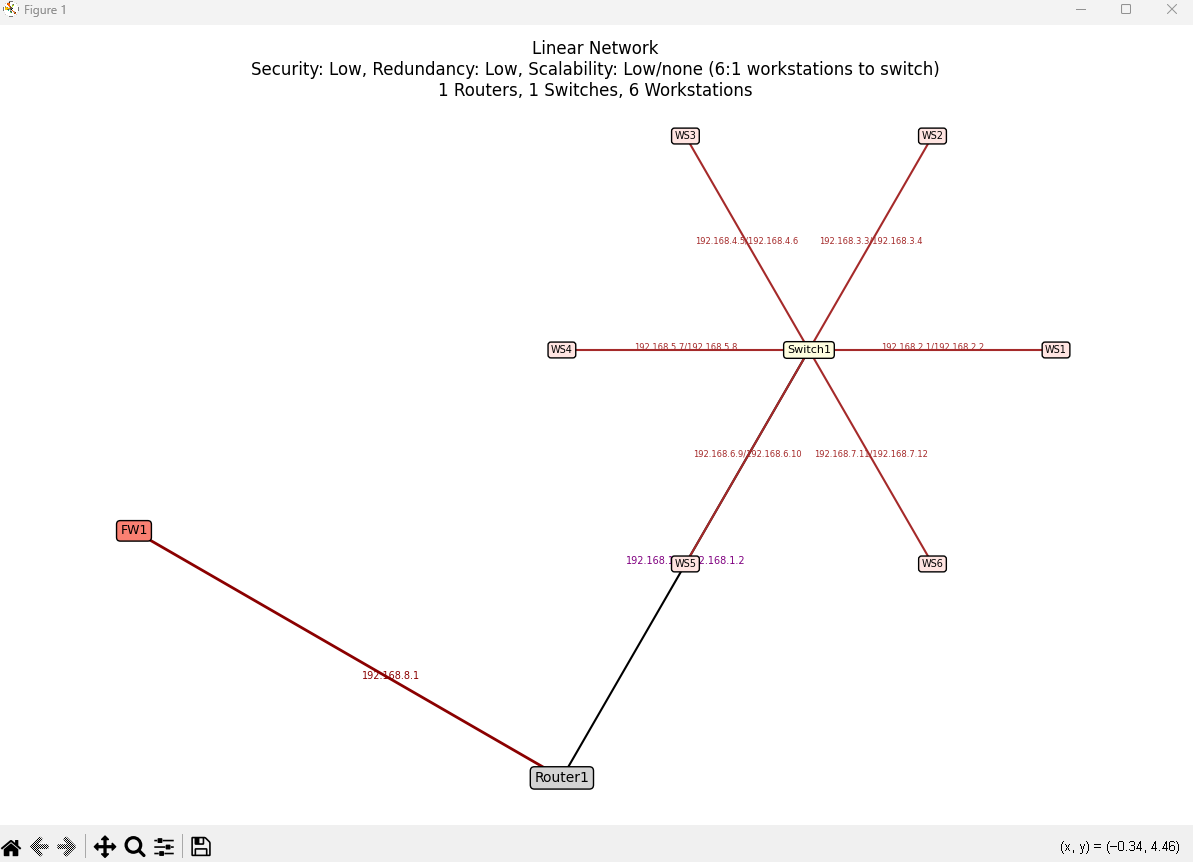


Figure 33 Topology Diagram 1

Furthermore after generating the topology diagram the script creates a .txt file where an IP address, device type and name are stored for every device that the diagram generates to be used by the configuration script. The content of this .txt file is shown below in figure 34.

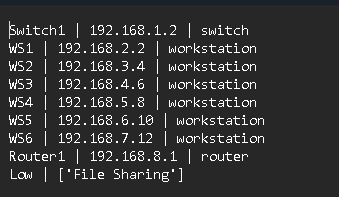


Figure 34 .txt Output file 1

### 4.1.2 Test Case 2

The second test case will have a Client-Server architecture; this network will have 12 workstations, maximum scalability, moderate security, and minimal redundancy. This means the network will have 2 servers with both database and website hosting capabilities. The network will also have 3 routers and 6 switches. Shown below in figures 36 and 37 is the topology diagram generated by the script and .txt output file.

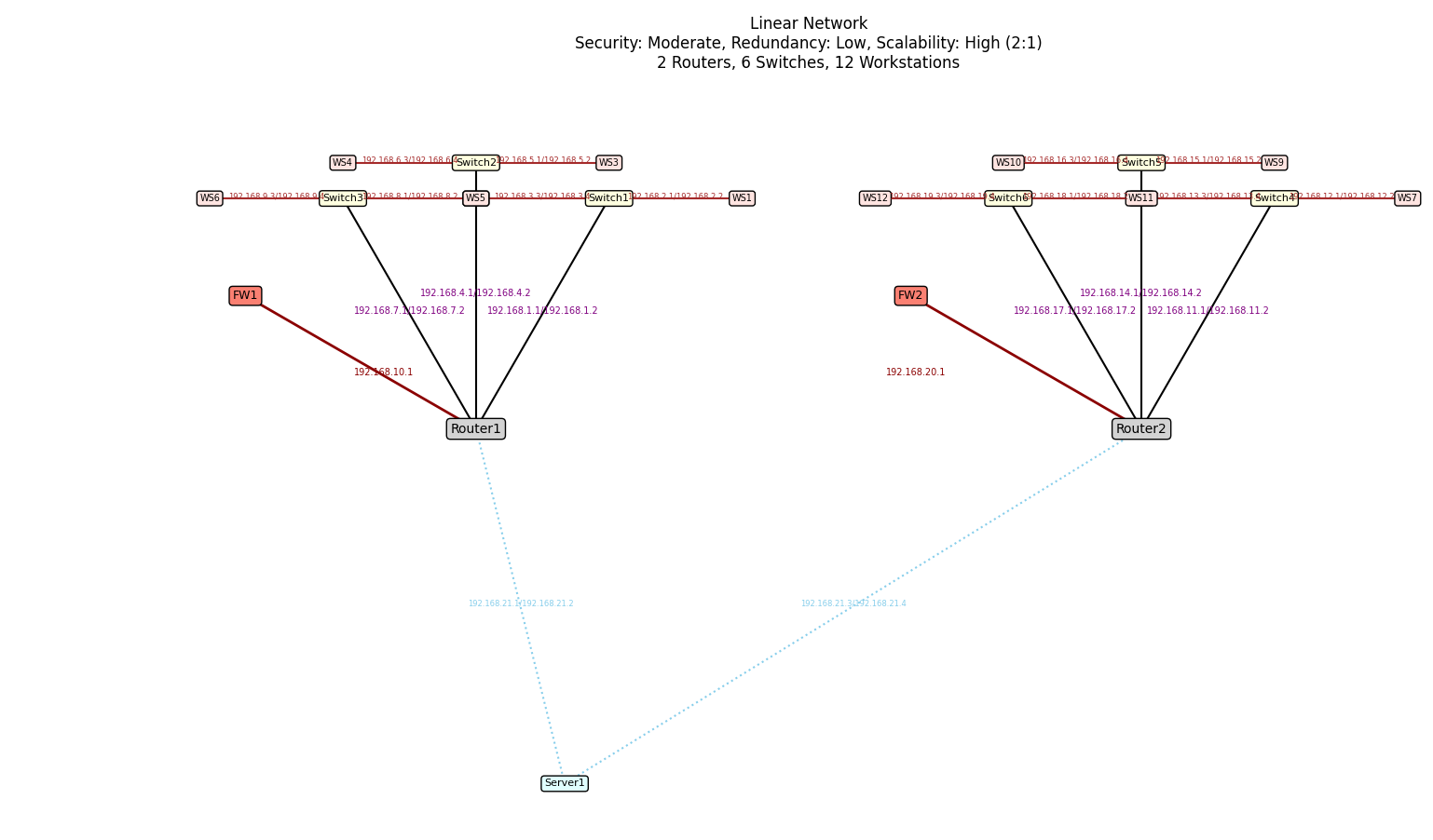


Figure 35 Topology Diagram 2

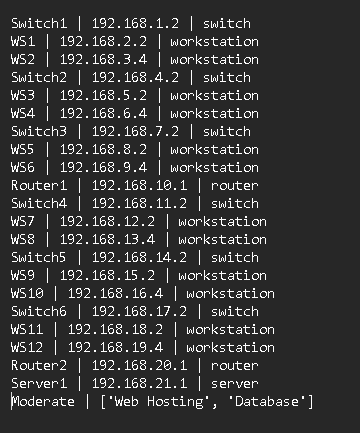


Figure 36 .txt Output file 2

### 4.1.3 Test Case 3

The third test case will be a Centralised Network with 6 workstations. This network will have maximum security and minimal scalability and redundancy. This means it has 1 server, 1 router and 1 switch. The central server should be configured with website and database hosting services. The Topology diagram and outputted .txt file are shown below in figures 37 and 38.

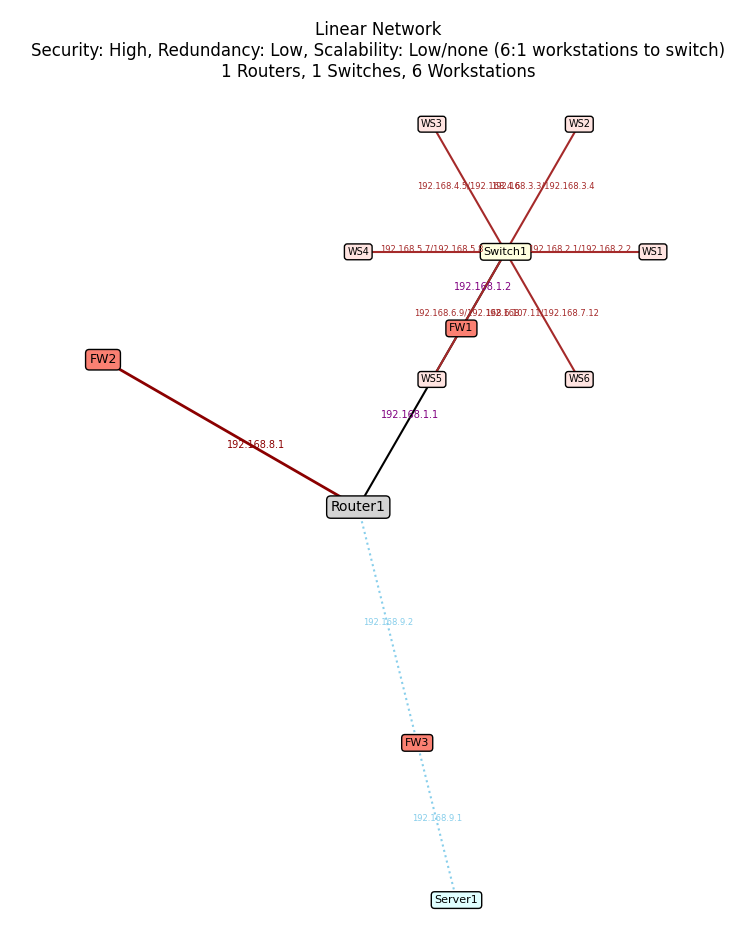


Figure 37 Topology Diagram 3

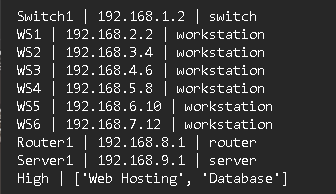


Figure 38 .txt Output File 3

### 4.1.4 Test Case 4

The fourth test will be a mesh topology with 8 workstations, minimal security, minimal scalability and maximum redundancy. This means there will be 2 routers, 4 switches. This network should be configured to allow file sharing. The Topology diagram and outputted .txt file are shown below in figures 39 and 40.

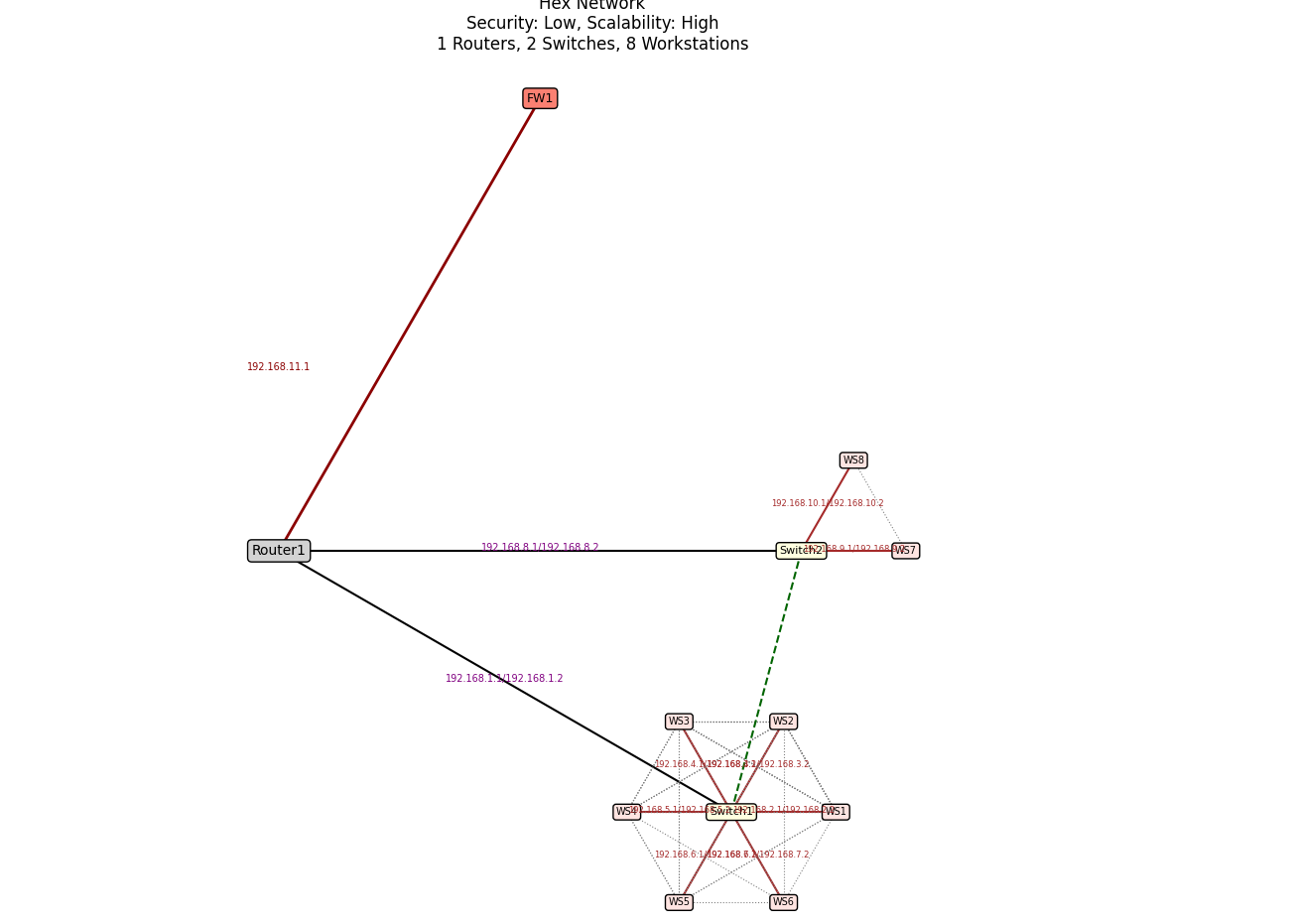


Figure 39 Topology Diagram 4

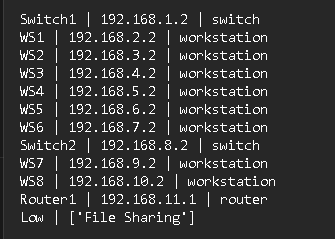


Figure 40 .txt Output File 4

## 4.2 Network Mapping

The network mapping functionality was tested in two distinct modes: manual IP input for targeted scanning and subnet discovery for automated range-based detection. Both modes leveraged Nmap’s scanning capabilities but differed in scope and use case.

### 4.2.1 Manual IP Mode

In this mode, the user manually entered specific IP addresses for scanning, allowing for precise control over which devices were analysed. The script performed a scan on each input IP. Testing confirmed a 100% accuracy rate in device detection and scan results. Shown below in figure 41 is Manual IP address mode being used on one host and below in figure 42 is the results which have been outputted to a .txt file for storage.

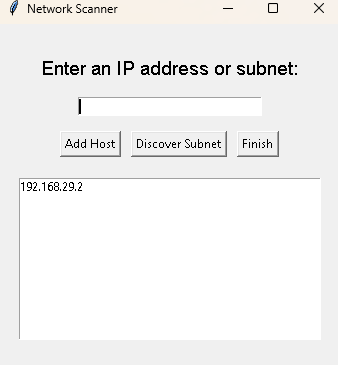


Figure 41 Manual IP mode single IP address



Figure 42 Single manual IP address scan results

Furthermore Manual IP mode can be used to scan multiple specific addresses back to back this allows a user to test long lists of addresses quickly and efficiently. Shown below in figures 43, 44 is the multiple inputted hosts and result .txt file

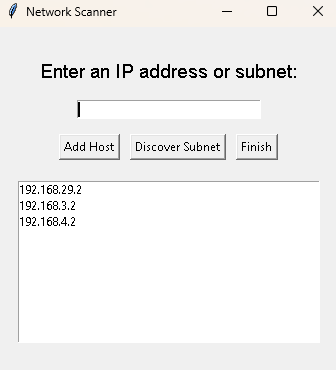


Figure 43 Multiple Specific IP address scan

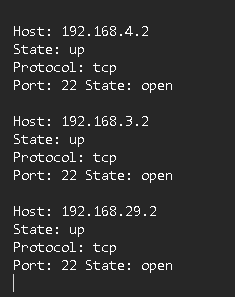


Figure 44 Multiple manual IP address scan results

### 4.2.2 Subnet Discovery Mode

For broader network exploration, the script accepted subnet inputs and performed an automated ping sweep to identify live hosts. These live hosts are then placed into the same section as hosts that have been inputted by the user this way; if the users wants they can take a note of them or scan them the same way they would hosts they entered themselves. Shown below in figure 45 is the result of scanning a network with devices like tiny linux workstations and cisco routes.

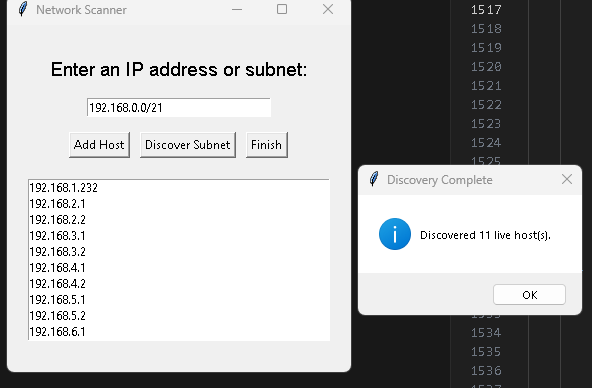


Figure 45 Host discovery

## 4.3 Network configuration

The network configuration process involved the automated setup of routers, switches, and firewalls based on security levels and network requirements that were stored from the topology generation portion of the script. The Python script successfully applied configurations to each device type, ensuring secure and efficient network operation.

### 4.3.1 Router

The router configuration process involved establishing both Telnet and SSH connections to apply security and routing settings. Key configurations included enabling SSH version 2, disabling insecure services such as tcp-small-servers, udp-small-servers, and finger, and implementing access control lists based on the specified firewall level. For high-security environments, the router enforced strict inbound traffic filtering, permitting only ICMP and SSH by default. Additional rules were dynamically applied based on network needs—such as allowing HTTPS for web hosting or SMB for file-sharing applications. Loopback interfaces were configured for management purposes, and all changes were securely logged. The script generated randomised administrative and guest credentials, enhancing security by preventing default credential exploitation. The passwords and accounts created for each device were stored along with the device name and IP address used to access the device. The device information is all stored within separate ..txt files one of which is shown below in figure 46.

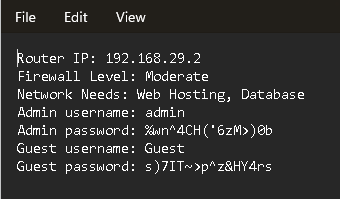


Figure 46 Configuration results

### 4.3.2 Switch

Switch configurations followed a similar security-first approach, with initial Telnet setup transitioning to SSH for secure management. The script disabled unnecessary services and implemented role-based access control by creating privileged and restricted user accounts with auto-generated passwords. VLAN configurations were not explicitly defined in this implementation, but the framework supports future expansion for segmentation. The switch’s security posture was reinforced by applying firewall-like ACLs when operating in high-security mode, restricting traffic to only essential services.

### 4.3.3 Firewall

The firewall configuration was dynamically adjusted based on the device type and the security level. High-security policies defaulted to a deny-all approach, permitting only ICMP and SSH, while moderate-level rules incorporated exceptions for services like web traffic or database access. The script denied packets for auditing, and rules were applied to the external facing interfaces to filter inbound traffic. For routers, firewall rules were extended to all interfaces, ensuring comprehensive protection.

Overall, the automated configuration tool successfully enforced consistent security policies across devices while adapting to varying requirements. The generated configuration files provided audit trails, and the use of randomised credentials mitigated credential-based attacks.

## 4.4 Survey Results

The Survey was filled out by 14 participants. These participants were given 15 minutes to use the script's features and become familiar with how the script functions. In this time they were directed to attempt to use all of the features of the script. After this every participant read through the high level manual set up guide. Finally the participants filled out the form based on how they felt and what they remembered and understood from the guide.

### 4.4.1 Question 1

The first question is “Rate the usability of the GUI” this question shows how the participant felt the usability of the user interface was. A pie chart breakdown is shown below in figure 47.

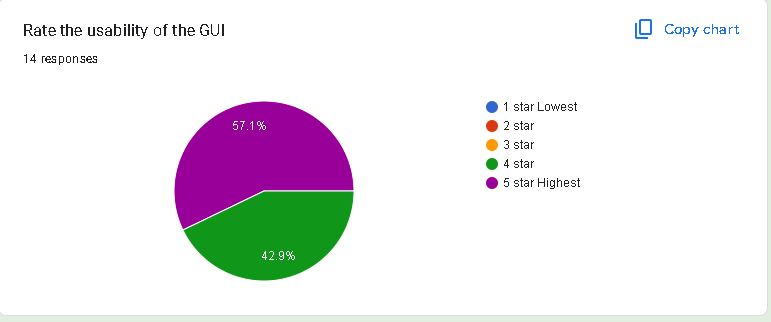


Figure 47 Question 1 Pie chart4.4.2 Question 2

The Second question is “What level of understanding do you think is required to use this tool” . This question shows how the participant felt the technical requirement to be able to proficiently use the tool is. A pie chart breakdown is shown below in figure 48.

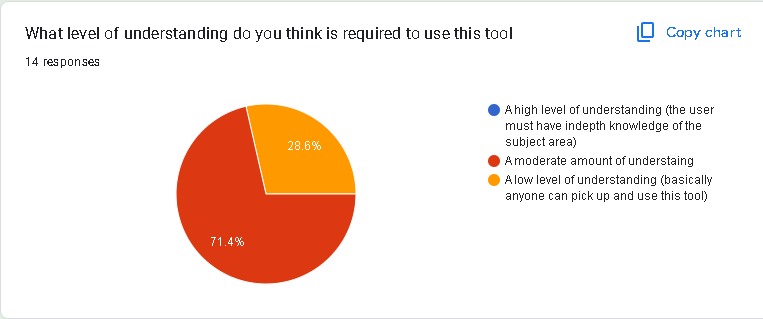


Figure 48 Question 2 Pie chart

### 4.4.3 Question 3

The Third question is “Rate how well you can understand the Generate Topology portion of the script(how well do you think this can be used to generate a topology” . This question shows how the participant felt about the readability and clarity of the diagrams generated by the script. A pie chart breakdown is shown below in figure 52.

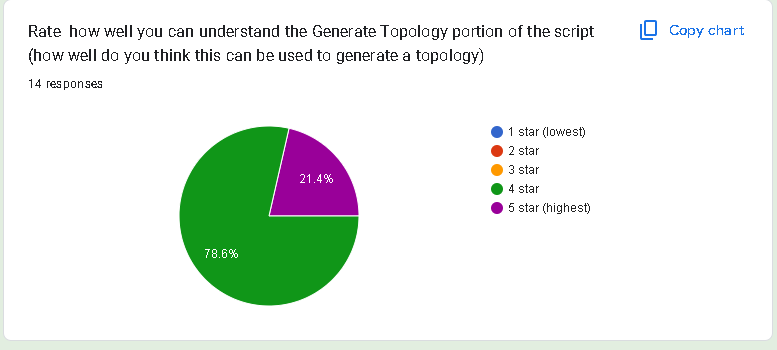


Figure 49 Question 3 Pie chart

### 4.4.4 Question 4

The fourth question is “Can you describe how you would deploy the configured devices (i.e. what you would do before deploying)”. This question gauges how well the user understood and retained the information provided to them from the high level manual set up guide about the planning requirements of using the generated topology feature of the script as well as what information was retained. The results are shown below in Table \4 found in Appendix C. It shows that 75% of users identified that they would need to understand how many devices or hardware they would need. 50% identified that they would need to know what kind of service requirements the network would need and that they would need to consider cabling and physical setup. 25% of participants talked about conducting pre-deployment configuration to the devices they would have on the network.

### 4.4.5 Question 5

The fourth question is “ can you describe how you do the initial stages of configuration for a network (i.e. what kind of cables are required)”. This question gauges how well the user understood and retained the information provided to them from the high level manual set up guide about what kind of hardware they would need to purchase before when setting up a network. The results are shown below in Table 5 found in Appendix C. It shows that 100% of users Identified cables as required. 64.3% of users identified manual command line interface configuration as part of the initial stage. 35.7% identified connecting the power supply to the devices as part of the initial set up stage. Furthermore 14.3% of users identified that testing\ verification of functionality would be needed.

# 5 Discussion

The aim of this project was to design and implement a scripted tool that can assist users with no technical knowledge with planning, testing and configuring a customisable computer network. Furthermore this project aims to evaluate its effectiveness based on previous literature and alternative planning and configuration tools. Four Research questions and a survey were proposed as part of the process to achieve the aim and conclude whether the tool was effective.

## 5.1 To design and implement a tool that uses users input to recommend an appropriate network architecture

The study's first research question aimed to determine how current tools use user input to recommend network architecture choices and determine the types and quantity of devices such as routers, switches, servers and workstations. As mentioned in the literature review, Cisco DNA Center(Dna et al. 2019) allows organisations to plan and configure networks using any of Cisco's range of network hardware. It does so by using AI to convert high level intent from users into policies that can be deployed to the network. It does this by first detecting what devices are present and then designing a network layout and pushing configurations to create this layout. This can be extremely useful for users looking to configure or redesign their existing network. Cisco DNA Center achieves this outcome by having the user intialy define how they want the network to act and recommending policies that will achieve the users desired outcome.

Based on features available in Cisco DNA Center, the developed network planning tool features a simplistic questionnaire that makes a user think about what they want the network to be able to do without asking granular questions on specific policies the network should employ. Cisco DNA Center focuses on a business needs first approach when configuring a network. The tool also focuses on allowing the pertinent services such as a hosted website or database to be freely accessed throughout the network. Furthermore the tool has the ability to visually represent the network it is planning to allow users to see how all the devices should connect together. This has the benefit of improving the understanding of a user who is unfamiliar with the topic (Shu et al. 2024).

While the methodology of using a questionnaire-driven input system and auto-generating visual network topologies offers a clear advantage in terms of accessibility and user-friendliness, it also presents several limitations. One significant challenge is the trade-off between abstraction and specificity—while the tool simplifies the planning process for users with limited technical expertise, it may not account for nuanced network requirements such as protocol preferences, specific IP addressing schemes, or hardware compatibility constraints. Additionally, by relying on a simplified intent model, there is a risk that the resulting topology may not fully align with the operational or security policies of more complex environments. Furthermore, although visualisation enhances user understanding, as supported by (Shu et al. 2024), the accuracy and clarity of the diagram depend heavily on the underlying logic of the generator. Without comprehensive validation against real-world network designs or expert reviews, there remains a potential gap between the intended and actual outcomes of the generated configurations.

The results of the study suggest that the questionnaire-based input system successfully guided users through the network planning process. Participants were able to define high-level requirements such as desired services, security level, and redundancy without needing to understand low-level configuration details. This supports the study's objective of taking user input and recommending network architecture and devices.. The resulting topologies generated by the tool were well understood by the user as shown by the results of the questionnaire receiving mostly 5 star ratings on the understanding of the generated topology.

However, a deeper evaluation of the results also reveals important limitations. While the tool accurately generated appropriate devices and services for simple to moderately complex networks, the diagrams it produced began to become deformed with high volumes of devices. Furthermore, despite the benefits of visual representation and help buttons with high level explanations a significant number of participants reported that a certain level of technical understanding was still required to fully interpret and use the generated topologies feature. This suggests that the tool’s accessibility could be improved through added guidance or interactive explanations. Additionally, the limited ability to modify or iterate on the auto-generated topology may hinder its use in more dynamic or advanced planning scenarios. These findings indicate that while the tool succeeds in lowering the barrier to entry for non-experts, further development is needed to enhance usability, especially through more intuitive visual elements and user support features.

Unlike Cisco DNA Center the script developed for this project is open source which allows users to personalise the features and functionality of the framework to meet their requirements. It is built using Python which is the fifth rated programming language in the world (Kumar & Dahiya 2017). Furthermore unlike Cisco DNA Center this project's script does not use AI because despite how useful AI it can still lead to anomalous or insecure deployed configurations(Leivadeas & Falkner, 2023; Tu et al., 2025; Zeydan & Turk).

## 5.2 To identify standard configuration practices for routers and switches

Another Question aimed to understand what kind of configurations would need to be made to routers and switches to allow them function securely and efficiently. The objective of conducting research into the basic best practice configurations present on all routers and switches it to create a baseline of security regardless of how the user wishes the network function. With tools that use intent based business requirements first approaches like Cisco DNA Center (Dna et al. 2019) security problems can arise but are quickly identified using proactive AI analysis. In the industry of intent based network configuration and planning using AI security is always placed second to the business needs. This is because if it was not then the AI may choose to not allow any traffic on or off the network as this would be the securest approach but would make the network pointless (Leivadeas & Falkner, 2023; Tu et al., 2025; Zeydan & Turk 2025). However Rule based approaches like NETCONF and SNMP require a more technical understanding to be able to implement(*Network\_configuration\_management\_using\_NETCONF\_and\_YANG*; Ji et al., 2009-03).

A solution to bridge the gap between the two approaches and have the effective security from the start that rule based configuration provides with the ease of use and low level technical knowledge requirements that intent based approaches like Cisco DNA Center. This project created a baseline of configurations that could be applied to all routers or switches that close all the features and services that a non-technical user would not use. To do so the official Cisco documentation and papers on configuring routers and switches were analysed to find what configurations were common across all such devices (*Network\_configuration\_management\_using\_NETCONF\_and\_YANG*; *Cisco 800M Series Integrated Services Routers Software Configuration Guide,* 2015; Bringhenti et al., 2023; Desig et al.; Lee et al.; Wang et al.). These configurations were then implemented using python but the implementation could not leverage modern solutions such as NETCONF and Yang which provide powerful frameworks for configuring devices standardising and simplifying the configuration process. Although useful, NETCONF and YANG require that the device being configured is set up to understand the protocols which many legacy systems are not capable of doing because of their hardware restraints or manufacturers no longer producing updates for them.

The methodology of defining a baseline set of router and switch configurations based on best practices from industry documentation was effective in providing a security-first foundation. This approach avoided relying on complex, protocol-dependent frameworks like NETCONF and YANG, making the implementation more broadly compatible with a variety of hardware, including older or unsupported devices (*Network\_configuration\_management\_using\_NETCONF\_and\_YANG 2009*; Ji et al., 2003). However, the methodology also had limitations. By manually coding the configurations in Python without real-time validation or device feedback, the implementation risked inaccuracies or partial applicability depending on the specific device models. While the approach succeeded in simplifying configuration for non-expert users, it did not have the same level of dynamic changes to the policies typically offered by industry tools.

The results indicate that the tool was able to generate consistent and functional configuration scripts for routers and switches, applying essential security practices such as disabling unused services, restricting remote access, and enabling secure protocols. The static nature of the configurations also meant that the scripts were predictable and easily reviewable however highlights a gap between generic security practices and tailored solutions. The results further showed that while the tool did reduce the risk of insecure defaults, it is still unable to dynamically enable and disable many services that users may want on their network.

While the outcomes demonstrate that the tool met its objective of delivering configurations to routers and switches, there were notable limitations. Although the tool reduced configuration errors and enforced best practices, it did not provide real-time feedback or error checking, which may lead to issues during deployment on physical devices. The inability to dynamically adjust configurations based on live device responses also limited the tool’s adaptability compared to industry solutions like Cisco DNA Center (Dna et al. 2019).

One significant limitation encountered during this study was the inability to fully simulate servers and workstations in GNS3, which restricted the testing of automated configurations for these devices. While GNS3 effectively emulated routers and switches, its limited support for end-host devices meant that critical aspects of network functionality—such as service hosting (e.g., web servers, databases) and workstation policies—could not be validated in a simulated environment. This gap could have been mitigated by incorporating real-world hardware for testing, allowing for a more comprehensive evaluation of the script’s ability to configure an entire network, not just its core infrastructure. Future iterations of this project should consider hybrid testing approaches, combining GNS3 for router and switch emulation with physical or cloud-based servers and workstations to ensure full functionality.

## 5.3 To explore and implement automated deployment of internal firewalls

The third objective aims to analyse how internal firewalls can be dynamically configured and deployed to the routers and switches of a network. By introducing firewalls to routers and switches the project's script can significantly increase the security on the network it is deployed to configure(Alicea & Alsmadi, 2021). Cisco DNA Center offers configuration options and policy management for access control lists on devices and internal firewalls but is not capable of managing them directly. It does so by setting IP-based access control policies and group based access control policies which either allow or filter out traffic based on the rules of the policy.

CIsco DNA Center, NETCONF and YANG all offer ways of configuring firewalls and ACL’s. To avoid the need for physical firewall devices and simplify the manual section of the configuration and set up process the firewalls are configured as part of the router and switches. It does so dynamically based on the user's inputs to the questionnaire. Initially only filtering certain types of traffic travelling through external facing interfaces on routers. Then if the users want to further increase the security the tool can be set to include more interfaces and configure switches to filter traffic as well. However all networks require that certain types of communications protocols are allowed to travel around, to and from the network. But to ensure that only the services that are intended on the network are allowed the internal firewalls are configured by default to drop all packets except ping which is used for testing and verification and SSH which is used for configuration. Then dynamically based on the user's answers more protocols are allowed to travel throughout the network.

The approach of embedding internal firewall rules directly into router and switch configurations based on user input provides a practical and user-friendly method of improving network security without requiring additional hardware. This methodology aligns with the goal of simplifying the configuration process for non technical users while still addressing key security concerns. By defaulting to a deny-all policy and only allowing essential traffic such as SSH and ping, the tool enforces a strong security baseline. The dynamic expansion of allowed protocols based on user selections ensures that only required services are enabled. However, this approach has limitations in granularity. Since the firewall rules are not context-aware or informed by real-time traffic monitoring, they may either over-restrict or insufficiently filter traffic in more complex environments. Furthermore, the tool’s decision-making relies entirely on static questionnaire responses, which, while accessible, may not capture nuanced user requirements or evolving network states.

The firewall implementation within the tool successfully enforced a secure-by-default posture by initially allowing only essential traffic (SSH and ping) and expanding permissions based on specified service needs. This approach ensured that networks were not left with overly permissive configurations and reduced the risk of unintended exposure. The dynamic adjustment of firewall rules based on high-level user input proved to be an effective mechanism for aligning access control with network functionality, especially in scenarios involving internal service access such as database or web hosting. However, the static rule sets and lack of contextual awareness limited the adaptability of the firewall configurations to more complex or evolving environments. The tool did not support granular per-device policy variation or time-based rules, which could be important in enterprise networks. Nonetheless, within its design scope, the tool met its objective of embedding internal firewalls in a streamlined, automated way, offering a practical level of security for basic and moderately complex network scenarios.

While the tool's firewall implementation improved network security and required minimal user input, its reliance on a static, rule-based model limited its adaptability. Some participants noted that they would have preferred a preview of the actual rules being applied or an explanation of their effects. This points to a need for better transparency and user education, especially for those with intermediate technical knowledge seeking more control. Additionally, the current design assumes that predefined protocol allowances are sufficient for most network needs, which may not always be the case in more specialized deployments. Despite these drawbacks, the tool effectively demonstrated that a dynamic yet simplified approach to internal firewall deployment is feasible and beneficial in educational or small-scale environments. The feedback highlights both the value of such a feature and areas for enhancement, particularly around user confidence and clarity.

Another challenge arose when attempting to test firewall rules that depended on interactions between servers, workstations, and network devices. Since GNS3 could not accurately simulate end hosts, certain security policies—such as access control lists (ACLs) governing server access—could not be fully verified. This limitation meant that while the firewall rules were theoretically sound, their real-world effectiveness remained partially untested. A practical solution would have been to deploy the configurations on a physical lab setup with real servers and clients, enabling end-to-end traffic testing. Additionally, integrating a lightweight virtual machine (VM) solution, such as VirtualBox or Docker containers, into GNS3 could have provided a workaround for simulating basic server and workstation behaviors.

## 5.4 To outline the essential steps for enabling remote configuration

The fourth objective is to determine how to efficiently convey the information that is required for a user of the project's script to complete the initial manual set and configuration portion that is required before the script can configure the remaining parts. To create the high level manual set up guide a review of papers explaining how to teach technical skills to people without any technical skills in that area was done (Leyk et al.; Widmer). This found that teaching people skills within context significantly improves learners' abilities to retain and understand the skills and information they are being taught. Furthermore applications like Brilliant are centered around teaching people skills like maths and science and utilise contextual learning (De La Puente & Perez).

Based on the features and style that is used in the studies and the brilliant application the high level manual set up guide attempts to go through the manual set up in the same order that a user would do the tasks and only give the user information that is pertinent to the task they are doing. The guide features pictures of different cables and the plug sockets to give context to the user of how they should be setting up the devices that have been planned for them by the topology generation script. The guide does not feature in depth information about how the systems that the user is setting up function, only how to put it together; this ensures that a user is not overloaded with information and has a better chance of following the instructions correctly.

The methodology used to assess the effectiveness of the high-level manual setup guide involved a user study featuring open-ended questions that tested knowledge retention and understanding after interacting with the guide. This approach was appropriate for gauging comprehension of procedural knowledge without requiring advanced technical understanding. However, the open-ended nature of the questions posed challenges for objective analysis, as participants' responses had to be qualitatively interpreted and manually categorized into themes. This introduces a potential for subjective bias in interpretation. Additionally, while the guide was influenced by pedagogical strategies such as contextual learning—shown to improve outcomes in teaching technical subjects to non-technical users (Widmer, 1996; Leyk et al., 2017)—there was no control group to compare learning outcomes with or without the guide, limiting the ability to isolate its effectiveness definitively.

The survey results demonstrate that the majority of participants grasped the foundational steps required for deployment and initial network configuration. For Question 4, 75% of users identified the need to understand hardware requirements prior to deployment, and 50% correctly noted the need to consider service requirements and cabling—both key takeaways from the guide. Furthermore, 25% mentioned pre-deployment configuration, indicating that some users retained higher-order procedural steps beyond physical setup. In response to Question 5, all participants correctly identified that cables were required, and 64.3% included command-line interface configuration as part of the process, suggesting strong retention of the setup sequence described in the guide. The fact that only 14.3% of users mentioned testing or verification suggests that while physical and CLI setup steps were retained, post-setup validation steps may require more emphasis or clearer framing in the guide.

The results suggest that the high-level manual setup guide was effective at communicating basic setup tasks to users with limited technical backgrounds. The high rates of correct identification of cables and configuration steps support the guide’s use of contextual visuals and step-by-step structure. However, the lower recognition of actions like device testing and pre-deployment configuration implies that either these concepts were not presented clearly enough or that users deprioritized them during recall. This highlights a possible limitation in how the guide presents more abstract or non-physical steps, such as validation or system readiness checks. Enhancing these sections with more concrete examples or visual cues could improve retention and user action. Overall, while the guide successfully conveyed essential setup tasks, further refinement is needed to ensure complete procedural understanding, especially in areas beyond the purely physical assembly.

The manual setup guide was designed under the assumption that users would be configuring real devices, but since GNS3 could not properly simulate workstations and servers, some steps (such as verifying service accessibility from end-user machines) could not be demonstrated effectively. This limitation may have contributed to the lower recall rates for post-setup validation steps in user testing. To address this, future versions of the guide should include alternative testing methods, such as using virtual machines or cloud instances to simulate real-world deployment scenarios. Additionally, providing video demonstrations of physical setups (e.g., connecting real workstations to configured routers) could help bridge the gap between simulation and actual implementation, ensuring users have a clearer understanding of the entire process.

# 6. Conclusion and Future Work

This dissertation set out to explore how network planning and configuration could be simplified for users with limited technical expertise while maintaining security and functionality. The study addressed four key research questions, each contributing to the development of a network planning tool that bridges the gap between intent-based and rule-based configuration approaches. The findings demonstrate that a questionnaire-driven input system, combined with automated topology generation and baseline security configurations, can effectively guide users through the network planning process. Participants were able to define high-level requirements without delving into granular technical details, and the resulting network diagrams were generally well understood. However, the study also revealed limitations, particularly in handling complex network scenarios, where the visual representation became less effective, and users still required some technical knowledge to fully interpret the output.

The tool’s approach to router and switch configurations successfully enforced security best practices by default, closing unnecessary services and restricting access to essential protocols. This methodology provided a strong security foundation without relying on advanced protocols like NETCONF and YANG, making it compatible with a broader range of devices, including legacy hardware. However, the static nature of these configurations limited adaptability, as the tool could not dynamically adjust rules based on real-time network conditions or more nuanced user requirements. Similarly, the implementation of internal firewalls within routers and switches proved effective in enforcing a secure-by-default posture, dynamically adjusting rules based on user input. While this approach simplified security deployment, it lacked granularity, particularly in enterprise environments where more sophisticated rule sets might be necessary.

The high-level manual setup guide, designed to assist users in the initial physical configuration of devices, was successful in conveying essential setup steps through contextual learning. Participants retained key procedural knowledge, particularly for physical tasks such as cabling and basic Command Line Interface configuration. However, the guide’s effectiveness diminished when explaining more abstract concepts, such as pre-deployment checks and post-setup validation, indicating a need for clearer instruction in these areas. Overall, the study confirms that a simplified, user-centric approach to network planning and configuration is feasible, though further refinements are necessary to enhance usability, adaptability, and educational clarity.

Several areas for future development have been identified through this research. First, the visual topology generator could be improved to better handle larger and more complex networks. Currently, the diagrams become distorted when representing high device counts, reducing their clarity. Implementing a more sophisticated layout algorithm or allowing users to manually adjust node placement could mitigate this issue. Additionally, integrating interactive elements into the topology—such as clickable nodes that display device details or configuration snippets—could enhance user understanding and engagement.

Another key area for improvement lies in the configuration generation process. While the current approach enforces strong security defaults, it lacks real-time validation and feedback mechanisms. Future iterations could incorporate device compatibility checks or simulate configuration deployment to identify potential issues before they arise in a live environment. Furthermore, expanding the tool’s ability to support dynamic rule adjustments—such as time-based access controls or context-aware firewall policies—would make it more adaptable to enterprise use cases. Exploring lightweight implementations of NETCONF or RESTCONF for supported devices could also provide a middle ground between full automation and manual configuration.

The firewall implementation could benefit from greater granularity and user control. Introducing a rule preview feature, where users can review and modify auto-generated firewall policies before deployment, would increase transparency and confidence. Additionally, incorporating basic traffic analysis to suggest rule optimizations could further enhance security without overburdening the user.

Finally, the high-level manual setup guide could be expanded into a more comprehensive learning resource. Incorporating short video demonstrations or interactive simulations for key setup tasks—such as cabling or initial CLI access—could improve knowledge retention. Adding troubleshooting sections or frequently asked questions (FAQs) would also help users resolve common issues independently. Future studies could compare learning outcomes between different instructional formats (e.g., text-based vs. video-based guides) to determine the most effective teaching methods for non-technical users.

Beyond these technical enhancements, the tool’s open-source nature presents opportunities for community-driven development. Encouraging contributions from network professionals could lead to broader hardware compatibility, additional feature sets, and localized documentation. Ultimately, the goal of this project—to make network planning and configuration more accessible—can be further realized through iterative improvements, user feedback, and collaborative innovation.

# 7. References

[1] Lee, S., Wong, T. and Kim, H.S. (2024 *To Automate or Not to Automate: On the Complexity of Network Configuration.*

[2] Choi, M., Choi, H. and Hong, J.W. (2004) *XML-Based Config ration Ma nag em en t for IP Network Devices*.

[4] Leivadeas, A. and Falkner, M. (2023) 'A Survey on Intent-Based Networking', *IEEE Communications Surveys &amp; Tutorials,* 25(1), pp. 625 Available at:<https://doi.org/10.1109/comst.2022.3215919>.

[5] Zeydan, E. and Turk, Y. *Recent Advances in Intent-Based Networking: A Survey.*

[6] Bringhenti, D., Marchetto, G., Sisto, R. and Valenza, F. (2023) 'Automation for Network Security Configuration: State of the Art and Research Trends', *ACM Computing Surveys,* 56(3), pp. 1 Available at:<https://doi.org/10.1145/3616401>.

[7] Granderath, M. and Schönwälder, J. *A Resource Efficient Implementation of the RESTCONF Protocol for OpenWrt Systems.* Available at:(Accessed: .

[8] Wu, Y. (2005) 'Network planning in wireless ad hoc networks: a cross-Layer approach', *IEEE Journal on Selected Areas in Communications,* 23(1), pp. 136 Available at:<https://doi.org/10.1109/jsac.2004.837362>.

[9] Sankpal, M.M.N., Chinchwade, M.P.V., Kognole, M.R.R. and Rode, M.K.N. (2022) 'Computer Network Topologies', *International Journal for Research in Applied Science and Engineering Technology,* 10(4), pp. 1508 Available at:<https://doi.org/10.22214/ijraset.2022.41553>.

[10] Singh, J. and Kumar, R., 2018 *A Review Paper on Networking Topologies*.

[11] Mainwaring, A.M., Chun, B.N., Schleimer, S. and Wilkerson, D.S. (1997) *System Area Network Mapping.*

[12] Hsu, W. and Shieh, Y. (2013) 'Virtual network mapping algorithm in the cloud infrastructure', *Journal of Network and Computer Applications,* 36(6), pp. 1724 Available at:<https://doi.org/10.1016/j.jnca.2013.02.028>.

[13] Houidi, I., Louati, W. and Zeghlache, D. (2008) *A Distributed Virtual Network Mapping Algorithm.*

[14] *Cisco 800M Series Integrated Services Routers Software Configuration Guide* (2015) .

[15] Desig, R., Keshav, S. and Sharma, R. (2008) 'Issues and Trends in',

[16] Hussain, S.M., Bhatti, S.N., Faizan, M. and Rasool, U. (2017) Legacy *System and Ways of its Evolution.*

[17] Wang, Z., Shaked, O., Cassuto, Y. and Bruck, J. (2013) *Codes for Network Switches.*

[18] Alicea, M. and Alsmadi, I. (2021) 'Misconfiguration in Firewalls and Network Access Controls: Literature Review', *Future Internet,* 13(11) Available at:<https://doi.org/10.3390/fi13110283>.

[19] Case, J. (1988) *Network Working Group*.

[20] Jürgen Schönwälder, Martin Björklund, Phil Shafer. (2010) *Network\_configuration\_management\_using\_NETCONF\_and\_YANG*.

[21] Ji, H., Zhang, B., Li, G., Gao, X. and Li, Y. (2009-03) *Challenges to the New Network Management Protocol: NETCONF.* IEEE, pp. 832.

[22] Sreerangapuri, (2023) A. *International Journal of INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING*.

[23] Tu, N., Nam, S. and Hong, J.W. (2025) 'Intent-Based Network Configuration Using Large Language Models', *International Journal of Network Management,* 35(1), pp. e2313 Available at:<https://doi.org/10.1002/nem.2313>.

[24] Kim, J., Okhravi, H., Tian, D.(. and Ujcich, B.E. (2024) 'Security Challenges of Intent-Based Networking', (7, 67), 56. Available at:<https://doi.org/10.1145/3639702>.

[25] Afshar, R., Goodrich, M.T., Matias, P., Osegueda, M.C., Berenbrink, P. and Monmege, B. (2022) *Mapping Networks via Parallel kth-Hop Traceroute Queries*.

[26] Dna, C., Multi, C., Meier, A., Ligtenberg, D., Systems, C., Gmbh, S., Metzger, L., Mosimann, P. and Kalberer, J. (2019) *CIsco\_DNA\_Center*.

[27] Shu, X., Pister, A., Tang, J., Chevalier, F. and Bach, B. (2024) 'Does This Have a Particular Meaning? Interactive Pattern Explanation for Network Visualizations', .

[28] Kumar, K. and Dahiya, S. (2017) *Programming Languages: A Survey*.

[29] De La Puente, M. and Perez, H (2023). 'Assessing the Impact of Brilliant.org on Enhancing Mathematics Academic Performance among High School Students in Colombia: A Quasi-Experimental Study', .

[30] Leyk, T., Mcinvale, R. and Chen, L (2017). *Structured Peer Learning Program -An Innovative Approach to Computer Science Education Case Study*.

[31] Widmer, S (1996). *Session 0230 Strategies For Teaching Technical Subjects To Non Technical Engineering Technology Students*.

# 8. Appendices

## 8.1 Appendix A - Script Code

import tkinter as tk

from tkinter import messagebox

from tkinter import simpledialog

import matplotlib.pyplot as plt

from tabulate import tabulate

import numpy as np

import itertools

import math

import nmap

import concurrent.futures

import paramiko

import time

import nmap

import re

from telnetlib import Telnet

import random

import string

class MainMenu:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Network Toolkit")

self.frame = tk.Frame(root)

self.frame.pack(padx=40, pady=40)

title = tk.Label(self.frame, text="Welcome to the Network Toolkit", font=("Arial", 18))

title.pack(pady=(0, 20))

planner\_btn = tk.Button(self.frame, text="Launch Network Planner", font=("Arial", 12), width=25,

command=self.launch\_planner)

planner\_btn.pack(pady=10)

mapper\_btn = tk.Button(self.frame, text="Launch Network Mapper", font=("Arial", 12), width=25,

command=self.launch\_mapper)

mapper\_btn.pack(pady=10)

Config\_btn = tk.Button(self.frame, text="Launch Network Configuration", font=("Arial", 12), width=25,

command=self.config)

Config\_btn.pack(pady=10)

def launch\_planner(self):

self.frame.destroy()

NetworkPlannerApp(self.root)

def launch\_mapper(self):

self.frame.destroy()

Mapper(self.root)

def config(self):

messagebox.showinfo("running configuration", "Running The Configuration file")

Config(self.root)

messagebox.showinfo("Config Complete", "Configuration Complete")

class Config:

def \_\_init\_\_(self, root):

self.root = root

self.devices = self.load\_devices\_from\_file("ip\_assignments.txt")

self.ip\_pattern = re.compile(r"^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$")

self.nm = nmap.PortScanner()

self.run()

def generate\_password(self):

characters = string.ascii\_letters + string.digits + string.punctuation

return "".join(random.choice(characters) for \_ in range(16))

def configure\_firewall(self, connection, firewall\_level, network\_needs, device):

"""Configure firewall based on security level and network needs"""

# Base firewall rules

if (device == "switch" and firewall\_level == "High") or (device == 'router'):

connection.send("ip access-list extended BASE\_FW\n")

time.sleep(.5)

connection.send("permit icmp any any\n") # Allow ping

time.sleep(.5)

connection.send("permit tcp any any eq 22\n") # Allow SSH

time.sleep(.5)

connection.send("deny ip any any log\n") # Deny and log everything else

time.sleep(.5)

connection.send("exit\n")

time.sleep(.5)

# Apply base firewall to external interface

connection.send("interface FastEthernet0/0\n")

time.sleep(.5)

connection.send("ip access-group BASE\_FW in\n")

time.sleep(.5)

connection.send("exit\n")

time.sleep(.5)

if (firewall\_level == 'Moderate' and device == 'router') or firewall\_level.lower() == 'High':

# Create stricter rules for moderate level

connection.send("ip access-list extended MODERATE\_FW\n")

time.sleep(.5)

# Allow specific services based on network needs

if "Web Hosting" in network\_needs:

connection.send("permit tcp any any eq 80\n")

time.sleep(.5)

connection.send("permit tcp any any eq 443\n")

time.sleep(.5)

if "File Sharing" in network\_needs:

connection.send("permit tcp any any eq 445\n")

time.sleep(.5)

connection.send("permit udp any any eq 445\n")

time.sleep(.5)

if "Database" in network\_needs:

connection.send("permit tcp any any eq 3306\n")

time.sleep(.5)

time.sleep(.5)

connection.send("permit icmp any any\n") # Allow ping

time.sleep(.5)

connection.send("permit tcp any any eq 22\n") # Allow SSH

time.sleep(.5)

connection.send("deny ip any any log\n")

time.sleep(.5)

connection.send("exit\n")

time.sleep(.5)

# Apply moderate firewall to all interfaces

connection.send("interface range FastEthernet0/0\n")

time.sleep(.5)

connection.send("ip access-group MODERATE\_FW in\n")

time.sleep(.5)

connection.send("exit\n")

time.sleep(.5)

connection.send("end\n")

time.sleep(0.5)

def configure\_switch(self, device):

ip = device['ip']

firewall\_level = device['firewall\_level']

network\_needs = device['network\_needs']

print(f"\nConfiguring router at {ip} (Firewall: {firewall\_level}, Needs: {network\_needs})")

admin\_pass = self.generate\_password()

guest\_pass = self.generate\_password()

# First try Telnet for basic SSH setup

try:

print("Attempting Telnet connection for initial setup...")

tn = Telnet(host=ip, port=23, timeout=10)

VtyUsername = simpledialog.askstring("Input", "Please enter the Virtual Terminal username: ")

tn.write(VtyUsername.encode('ascii') + b"\n")

VtyPassword = simpledialog.askstring("Input", "Please enter the Virtual Terminal password: ")

tn.write(VtyPassword.encode('ascii') + b"\n")

# Basic configuration to enable SSH

tn.write(b"enable\n")

time.sleep(.5)

tn.write(b"configure terminal\n")

time.sleep(.5)

tn.write(b"ip domain-name CAM\n")

time.sleep(.5)

tn.write(b"crypto key generate rsa\n")

time.sleep(.5)

tn.write(b"1024\n")

time.sleep(.5)

tn.write(b"ip ssh version 2\n")

time.sleep(.5)

tn.write(b"username admin privilege 15 secret " + admin\_pass.encode('ascii') + b"\n")

time.sleep(.5)

tn.write(b"username Guest privilege 0 secret " + guest\_pass.encode('ascii') + b"\n")

time.sleep(.5)

tn.write(b"line vty 0 4\n")

time.sleep(.5)

tn.write(b"transport input ssh\n")

time.sleep(.5)

tn.write(b"login local\n")

time.sleep(.5)

tn.write(b"exit\n")

time.sleep(.5)

tn.write(b"end\n")

time.sleep(.5)

tn.write(b"write mem\n")

time.sleep(.5)

tn.write(b"exit\n")

time.sleep(.5)

print("Basic SSH configuration complete via Telnet")

tn.close()

except Exception as e:

print(f"Telnet configuration failed: {e}")

# Now connect via SSH for full configuration

try:

print("Attempting SSH connection for full configuration...")

client = paramiko.SSHClient()

client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

client.connect(hostname=ip, port=22, username="admin", password="admin",

look\_for\_keys=False, allow\_agent=False, timeout=10)

connection = client.invoke\_shell()

# Basic router configuration

connection.send("configure terminal\n")

time.sleep(0.5)

connection.send("no enable password\n")

time.sleep(.5)

connection.send("enable secret cisco\n")

time.sleep(.5)

connection.send("banner motd ^WARNING: Unauthorized access prohibited!^\n")

time.sleep(.5)

connection.send("logging buffered 16384\n")

time.sleep(.5)

connection.send("no ip source-route\n")

time.sleep(.5)

connection.send("ip routing\n")

time.sleep(.5)

connection.send("no service tcp-small-servers\n")

time.sleep(.5)

connection.send("no service udp-small-servers\n")

time.sleep(.5)

connection.send("no service finger\n")

time.sleep(.5)

connection.send("no cdp run\n")

time.sleep(.5)

# Configure loopback interface

connection.send("interface Loopback0\n")

time.sleep(.5)

connection.send("ip address 1.1.1.1 255.255.255.255\n")

time.sleep(.5)

connection.send("exit\n")

time.sleep(.5)

# Configure firewall

if firewall\_level == "High":

self.configure\_firewall(connection, firewall\_level, network\_needs, device)

# Save configuration

connection.send("end\n")

connection.send("write mem\n")

time.sleep(1)

# Get output

output = connection.recv(65535).decode('utf-8')

print(f"\nRouter configuration complete for {ip}:\n{output}")

# Save config to file

with open(f"{ip}\_router\_config.txt", "w") as file:

file.write(f"Router IP: {ip}\n")

file.write(f"Firewall Level: {firewall\_level}\n")

file.write(f"Network Needs: {', '.join(network\_needs)}\n")

file.write(f"Admin username: admin\nAdmin password: {admin\_pass}\n")

file.write(f"Guest username: Guest\nGuest password: {guest\_pass}\n")

print(f"Configuration saved to {ip}\_router\_config.txt")

except Exception as e:

print(f"SSH configuration failed: {e}")

finally:

try:

client.close()

except:

pass

def configure\_router(self, device):

ip = device['ip']

firewall\_level = device['firewall\_level']

network\_needs = device['network\_needs']

device\_type = device['device\_type']

print(f"\nConfiguring router at {ip} (Firewall: {firewall\_level}, Needs: {network\_needs})")

admin\_pass = self.generate\_password()

guest\_pass = self.generate\_password()

# First try Telnet for basic SSH setup

try:

print("Attempting Telnet connection for initial setup...")

tn = Telnet(host=ip, port=23, timeout=10)

VtyUsername = simpledialog.askstring("Input", "Please enter the Virtual Terminal username: ")

tn.write(VtyUsername.encode('ascii') + b"\n")

VtyPassword = simpledialog.askstring("Input", "Please enter the Virtual Terminal password: ")

tn.write(VtyPassword.encode('ascii') + b"\n")

# Basic configuration to enable SSH

tn.write(b"enable\n")

time.sleep(.5)

tn.write(b"configure terminal\n")

time.sleep(.5)

tn.write(b"ip domain-name CAM\n")

time.sleep(.5)

tn.write(b"crypto key generate rsa\n")

time.sleep(.5)

tn.write(b"1024\n")

time.sleep(.5)

tn.write(b"ip ssh version 2\n")

time.sleep(.5)

tn.write(b"username admin privilege 15 secret admin\n") #+ admin\_pass.encode('ascii') + b"\n")

time.sleep(.5)

tn.write(b"username Guest privilege 0 secret " + guest\_pass.encode('ascii') + b"\n")

time.sleep(.5)

tn.write(b"line vty 0 4\n")

time.sleep(.5)

tn.write(b"transport input ssh\n")

time.sleep(.5)

tn.write(b"login local\n")

time.sleep(.5)

tn.write(b"exit\n")

time.sleep(.5)

tn.write(b"end\n")

time.sleep(.5)

tn.write(b"write mem\n")

time.sleep(.5)

tn.write(b"exit\n")

time.sleep(.5)

print("Basic SSH configuration complete via Telnet")

tn.close()

except Exception as e:

print(f"Telnet configuration failed: {e}")

# Now connect via SSH for full configuration

try:

print("Attempting SSH connection for full configuration...")

client = paramiko.SSHClient()

client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

client.connect(hostname=ip, port=22, username="admin", password="admin",

look\_for\_keys=False, allow\_agent=False, timeout=10)

connection = client.invoke\_shell()

# Basic router configuration

connection.send("configure terminal\n")

time.sleep(0.5)

connection.send("no enable password\n")

time.sleep(.5)

connection.send("enable secret cisco\n")

time.sleep(.5)

connection.send("banner motd ^WARNING: Unauthorized access prohibited!^\n")

time.sleep(.5)

connection.send("logging buffered 16384\n")

time.sleep(.5)

connection.send("no ip source-route\n")

time.sleep(.5)

connection.send("ip routing\n")

time.sleep(.5)

connection.send("no service tcp-small-servers\n")

time.sleep(.5)

connection.send("no service udp-small-servers\n")

time.sleep(.5)

connection.send("no service finger\n")

time.sleep(.5)

connection.send("no cdp run\n")

time.sleep(.5)

# Configure loopback interface

connection.send("interface Loopback0\n")

time.sleep(.5)

connection.send("ip address 1.1.1.1 255.255.255.255\n")

time.sleep(.5)

connection.send("exit\n")

time.sleep(.5)

# Configure firewall

self.configure\_firewall(connection, firewall\_level, network\_needs, device\_type)

# Save configuration

connection.send("end\n")

connection.send("write mem\n")

time.sleep(1)

# Get output

output = connection.recv(65535).decode('utf-8')

print(f"\nRouter configuration complete for {ip}:\n{output}")

# Save config to file

with open(f"{ip}\_Device\_config.txt", "w") as file:

file.write(f"Router IP: {ip}\n")

file.write(f"Firewall Level: {firewall\_level}\n")

file.write(f"Network Needs: {', '.join(network\_needs)}\n")

file.write(f"Admin username: admin\nAdmin password: {admin\_pass}\n")

file.write(f"Guest username: Guest\nGuest password: {guest\_pass}\n")

print(f"Configuration saved to {ip}\_router\_config.txt")

except Exception as e:

print(f"SSH configuration failed: {e}")

finally:

try:

client.close()

except:

pass

def validate\_ip(self, ip):

"""Validate IP address format"""

return bool(self.ip\_pattern.match(ip))

def run(self):

"""Main method to configure all devices"""

if not self.devices:

print("No devices to configure!")

return

for device in self.devices:

if not self.validate\_ip(device['ip']):

print(f"\nInvalid IP skipped: {device['ip']}")

continue

if device['device\_type'] == 'router':

self.configure\_router(device)

elif device['device\_type'] == 'switch':

self.configure\_switch(device)

else:

print(f"\nUnknown device type skipped: {device['device\_type']}")

def load\_devices\_from\_file(self, filename):

"""Load devices from file with format: name|ip|type"""

devices = []

try:

with open(filename, 'r') as file:

lines = [line.strip() for line in file.readlines() if line.strip()]

# Last line contains firewall level and network needs

if lines:

last\_line = lines[-1]

if '|' in last\_line:

firewall\_level, needs\_str = last\_line.split('|')

firewall\_level = firewall\_level.strip()

network\_needs = [need.strip() for need in needs\_str.strip("[]").split(',')]

else:

firewall\_level = 'Low'

network\_needs = []

# Process device entries

for line in lines[:-1]:

if '|' not in line:

continue

name, ip, device\_type = map(str.strip, line.split('|'))

devices.append({

'name': name,

'ip': ip,

'device\_type': device\_type.lower(),

'firewall\_level': firewall\_level,

'network\_needs': network\_needs

})

return devices

except FileNotFoundError:

print(f"Error: File {filename} not found!")

return []

except Exception as e:

print(f"Error reading device file: {e}")

return []

class NetworkPlannerApp:

def \_\_init\_\_(self, root):

self.root = root

root.title("Network Planning Tool")

self.question\_index = 0

self.answers = {}

self.frames = []

self.questions = [

{"key": "Workstations", "text": "How many workstations (desktops) will be present on this network?",

"type": "entry", "help": "Enter the total number of workstations (computers) that need to be connected."},

{"key": "Scalability", "text": "How much scalability (future growth) do you need?", "type": "options",

"options": ["Low/None (6:1 workstations to switch)", "Moderate (4:1)", "High (2:1)"],

"help": "Low: 6 workstations per switch\nModerate: 4 workstations per switch\nHigh: 2 workstations per switch"},

{"key": "Redundancy", "text": "How much redundancy (fault tolerance) do you require?", "type": "options",

"options": ["Low", "Moderate", "High"],

"help": "Low: Basic connections\nModerate: Redundant paths between switches\nHigh: Full mesh with redundant servers"},

{"key": "Security", "text": "What level of security do you need?", "type": "options",

"options": ["Low", "Moderate", "High"],

"help": "Basic: Perimeter firewall only\nStandard: Internal segmentation with ACLs\nHigh: Per-device controls and zones"},

{"key": "ServerNeeds", "text": "What server services will you need?", "type": "checklist",

"options": ["File Sharing", "Web Hosting", "Database"],

"help": "Select all server services that will be required on your network"}

]

self.build\_ui()

def build\_ui(self):

self.main\_frame = tk.Frame(self.root)

self.main\_frame.pack(padx=20, pady=20)

self.label = tk.Label(self.main\_frame, text="", font=("Arial", 14), wraplength=500)

self.label.pack(pady=10)

self.input\_var = tk.StringVar()

self.input\_widget = None

self.checkboxes = []

self.button\_frame = tk.Frame(self.main\_frame)

self.button\_frame.pack(pady=10)

self.back\_button = tk.Button(self.button\_frame, text="Back", command=self.prev\_question)

self.back\_button.grid(row=0, column=0, padx=5)

self.next\_button = tk.Button(self.button\_frame, text="Next", command=self.next\_question)

self.next\_button.grid(row=0, column=1, padx=5)

self.help\_button = tk.Button(self.button\_frame, text="Help", command=self.show\_help)

self.help\_button.grid(row=0, column=2, padx=5)

self.display\_question()

def show\_help(self):

q = self.questions[self.question\_index]

messagebox.showinfo("Help", q.get("help", "No help available for this question."))

def display\_question(self):

self.clear\_input()

if self.question\_index >= len(self.questions):

self.show\_summary()

return

q = self.questions[self.question\_index]

self.label.config(text=q["text"])

if q["type"] == "entry":

self.input\_widget = tk.Entry(self.main\_frame, textvariable=self.input\_var, font=("Arial", 12))

self.input\_widget.pack()

elif q["type"] == "options":

self.input\_widget = tk.OptionMenu(self.main\_frame, self.input\_var, \*q["options"])

self.input\_var.set(q["options"][0])

self.input\_widget.pack()

elif q["type"] == "checklist":

self.input\_widget = tk.Frame(self.main\_frame)

self.checkboxes = []

for option in q["options"]:

var = tk.IntVar()

cb = tk.Checkbutton(self.input\_widget, text=option, variable=var)

cb.pack(anchor="w")

self.checkboxes.append((option, var))

self.input\_widget.pack()

prev\_answer = self.answers.get(q["key"])

if prev\_answer:

if q["type"] == "checklist":

for option, var in self.checkboxes:

if option in prev\_answer:

var.set(1)

else:

self.input\_var.set(prev\_answer)

self.back\_button.config(state="normal" if self.question\_index > 0 else "disabled")

def clear\_input(self):

if self.input\_widget:

self.input\_widget.destroy()

self.input\_widget = None

self.input\_var.set("")

self.checkboxes = []

def next\_question(self):

q = self.questions[self.question\_index]

q\_key = q["key"]

if q["type"] == "entry":

answer = self.input\_var.get().strip()

if q\_key == "Workstations":

if not answer.isdigit() or int(answer) <= 0:

messagebox.showerror("Invalid input", "Please enter a valid positive number of workstations.")

return

answer = int(answer)

elif q["type"] == "checklist":

answer = [option for option, var in self.checkboxes if var.get() == 1]

if not answer and q\_key == "ServerNeeds":

answer = ["None"]

else:

answer = self.input\_var.get()

self.answers[q\_key] = answer

self.question\_index += 1

self.display\_question()

def prev\_question(self):

self.question\_index -= 1

self.display\_question()

def show\_summary(self):

self.clear\_input()

self.label.config(text="Network Planning Summary:")

summary\_text = ""

for key, val in self.answers.items():

if isinstance(val, list):

val = ", ".join(val)

summary\_text += f"{key}: {val}\n"

summary\_label = tk.Label(self.main\_frame, text=summary\_text, font=("Arial", 12), justify="left")

summary\_label.pack(pady=10)

self.back\_button.config(state="normal")

self.next\_button.config(text="Generate Network", command=self.generate\_network)

self.help\_button.config(state="disabled")

def generate\_network(self):

# Get all answers

workstations = self.answers["Workstations"]

scalability = self.answers["Scalability"]

redundancy = self.answers["Redundancy"]

security = self.answers["Security"]

server\_needs = self.answers["ServerNeeds"]

if "Low/None" in scalability:

ws\_per\_switch = 6

elif "Moderate" in scalability:

ws\_per\_switch = 4

else: # High

ws\_per\_switch = 2

# Calculate number of switches needed

num\_switches = math.ceil(workstations / ws\_per\_switch)

num\_routers = math.ceil(num\_switches / 3)

#determine architectur

if (("Web Hosting" not in server\_needs and "Database" not in server\_needs and workstations <= 108 and "High" in redundancy and "Low/None" in scalability) or

( "Web Hosting" not in server\_needs and "Database" not in server\_needs and workstations <= 72 and "High" in redundancy and "Moderate" in scalability) or

( "Web Hosting" not in server\_needs and "Database" not in server\_needs and workstations <= 36 and "High" in redundancy and "High" in scalability)):

branch\_angles\_deg=[-30, 0, 30, 60]

router\_radius=50

switch\_distance=50

workstation\_distance=10

angles = np.linspace(0, 2 \* np.pi, 6, endpoint=False)

router\_positions = [(np.cos(a) \* router\_radius, np.sin(a) \* router\_radius) for a in angles[:num\_routers]]

plt.figure(figsize=(12, 12))

connections = []

switch\_positions = []

# IP counters

network\_counter = 1 # For 3rd octet

interface\_counter = 1 # For 4th octet

switch\_count = 1

firewall\_count = 1

workstation\_count = 1

# Process each router sequentially

for router\_num in range(num\_routers):

rx, ry = router\_positions[router\_num]

router\_name = f"Router{router\_num+1}"

router\_switches = [] # Track switch names and positions for intra-switch connections

# 1. Draw the router

plt.scatter(rx, ry, color='blue')

plt.text(rx, ry, router\_name, fontsize=10, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightgrey"))

# 2. Connect switches to this router

for j, offset in enumerate(branch\_angles\_deg):

angle = np.arctan2(ry, rx) + np.deg2rad(offset) # This should result in a scalar

x = rx + np.cos(angle) \* switch\_distance

y = ry + np.sin(angle) \* switch\_distance

if j < 3 and switch\_count <= num\_switches: # Switch branches

switch\_name = f"Switch{switch\_count}"

if security == 'High':

# With firewall between router and switch

mx, my = (rx + x)\*0.7, (ry + y)\*0.7

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red')

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Router to Firewall

plt.plot([rx, mx], [ry, my], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((rx+mx)/2, (ry+my)/2, f"{ip1}",

fontsize=7, ha='center', color='purple')

interface\_counter += 1

# Firewall to Switch

plt.plot([mx, x], [my, y], 'black')

interface\_counter += 1

firewall\_count += 1

else:

# Direct router-switch connection

plt.plot([rx, x], [ry, y], 'black')

ip1 = f"192.168.{network\_counter}.1"

ip2 = f"192.168.{network\_counter}.2"

plt.text((rx+x)/2, (ry+y)/2, f"{ip1}/{ip2}",

fontsize=7, ha='center', color='purple')

connections.append({

'name' :switch\_name,

'ip' : ip2,

'type' : 'switch'

})

network\_counter += 1

# Draw switch

plt.scatter(x, y, color='green')

plt.text(x, y, switch\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightyellow"))

switch\_positions.append((x, y, switch\_name))

switch\_count += 1

router\_switches.append((x, y, switch\_name))

switch\_positions.append((x, y, switch\_name))

ws\_list = [] # Store workstation names and positions for interconnections

# 3. Connect workstations to this switch

for k in range(6):

if workstation\_count > workstations:

break

angle = k \* (2 \* np.pi / 6)

wx = x + np.cos(angle) \* workstation\_distance

wy = y + np.sin(angle) \* workstation\_distance

ws\_name = f"WS{workstation\_count}"

# Draw workstation and connection

plt.scatter(wx, wy, color='orange')

plt.plot([x, wx], [y, wy], 'brown')

plt.text(wx, wy, ws\_name, fontsize=7, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="mistyrose"))

# Assign IPs

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((x+wx)/2, (y+wy)/2, f"{ip1}/{ip2}",

fontsize=6, ha='center', color='brown')

connections.append({

'name': ws\_name,

'ip': ip2,

'type': 'workstation'

})

ws\_list.append((wx, wy, ws\_name))

network\_counter +=1

interface\_counter = 1

workstation\_count += 1

# Connect each workstation to every other on this switch

for i in range(len(ws\_list)):

for j in range(i + 1, len(ws\_list)):

x1, y1, ws1 = ws\_list[i]

x2, y2, ws2 = ws\_list[j]

plt.plot([x1, x2], [y1, y2], linestyle='dotted', color='gray', linewidth=0.8)

elif j == 3: # Firewall stub

fw\_name = f"FW{firewall\_count}"

plt.scatter(x, y, color='red')

plt.plot([rx, x], [ry, y], 'darkred', linewidth=2)

plt.text(x, y, fw\_name, fontsize=9, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

ip1 = f"192.168.{network\_counter}.1"

plt.text((rx+x)\*0.4, (ry+y)\*0.4, f"{ip1}",

fontsize=7, ha='center', color='darkred')

connections.append({

'name': router\_name,

'ip': ip1,

'type': 'router'

})

network\_counter += 1

interface\_counter = 1

firewall\_count += 1

for i, (x1, y1, sw1) in enumerate(router\_switches):

for j in range(i + 1, len(router\_switches)):

x2, y2, sw2 = router\_switches[j]

# Draw connection

plt.plot([x1, x2], [y1, y2], 'darkgreen', linestyle='--')

network\_counter += 1

interface\_counter = 1

# Connect routers to each other

for i, j in itertools.combinations(range(num\_routers), 2):

r1, r2 = router\_positions[i], router\_positions[j]

router1, router2 = f"Router{i+1}", f"Router{j+1}"

if security in ['Moderate', 'High']:

# With firewall between routers

mx, my = (r1[0]+r2[0])/2, (r1[1]+r2[1])/2

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red')

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Router1 to Firewall

plt.plot([r1[0], mx], [r1[1], my], 'gray')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

plt.text((r1[0]+mx)/2, (r1[1]+my)/2, f"{ip1}",

fontsize=7, ha='center', color='darkblue')

interface\_counter += 1

# Firewall to Router2

plt.plot([mx, r2[0]], [my, r2[1]], 'gray')

ip2 = f"192.168.{network\_counter}.1"

plt.text((mx+r2[0])/2, (my+r2[1])/2, f"{ip2}",

fontsize=7, ha='center', color='darkblue')

connections.append({

'name': router2,

'ip': ip2,

'type': 'router'

})

network\_counter += 1

interface\_counter = 1

firewall\_count += 1

else:

# Direct router-router connection

plt.plot([r1[0], r2[0]], [r1[1], r2[1]], 'gray')

ip1 = f"192.168.{network\_counter}.1"

ip2 = f"192.168.{network\_counter}.2"

plt.text((r1[0]+r2[0])/2, (r1[1]+r2[1])/2, f"{ip1}/{ip2}",

fontsize=7, ha='center', color='darkblue')

connections.append({

'name': router2,

'ip': ip2,

'type': 'router'

})

network\_counter += 1

interface\_counter = 1

plt.title(f"Hex Network\n"

f"Security: {security.capitalize()}, "

f"Scalability: {redundancy.capitalize()}\n"

f"{num\_routers} Routers, {num\_switches} Switches, {workstations} Workstations")

plt.axis('equal')

plt.axis('off')

plt.tight\_layout()

plt.show()

elif "Web Hosting" not in server\_needs and "Database" not in server\_needs:

# Place routers in a straight horizontal line

branch\_angles\_deg=[-30, 0, 30, 60]

router\_spacing=15

switch\_distance=6

workstation\_distance=3

router\_positions = [(i \* router\_spacing, 0) for i in range(num\_routers)]

router\_names = [f"Router{i+1}" for i in range(num\_routers)]

plt.figure(figsize=(12, 8))

connections = []

all\_switch\_groups = []

switch\_positions = []

# IP counters

network\_counter = 1 # For 3rd octet

interface\_counter = 1 # For 4th octet

switch\_count = 1

firewall\_count = 1

workstation\_count = 1

# Process each router's switches and workstations first

for router\_num in range(num\_routers):

rx, ry = router\_positions[router\_num]

router\_name = router\_names[router\_num]

# Draw the router

plt.scatter(rx, ry, color='blue', s=100)

plt.text(rx, ry, router\_name, fontsize=10, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightgrey"))

# Connect switches to this router

switch\_group1 = []

switch\_group2 = []

base\_angle = np.pi/2 # Pointing upwards

for j, offset in enumerate(branch\_angles\_deg):

angle = base\_angle + np.deg2rad(offset)

x = rx + np.cos(angle) \* switch\_distance

y = ry + np.sin(angle) \* switch\_distance

if j < 3 and switch\_count <= num\_switches: # Switch branches

switch\_name = f"Switch{switch\_count}"

if security == 'High':

# With firewall between router and switch

mx, my = (rx + x)\*0.7, (ry + y)\*0.7

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red', s=80)

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Router to Firewall

plt.plot([rx, mx], [ry, my], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((rx+mx)/2, (ry+my)/2, f"{ip1}",

fontsize=7, ha='center', color='purple')

interface\_counter +=1

# Firewall to Switch

plt.plot([mx, x], [my, y], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((mx+x)/2, (my+y)/2, f"{ip2}",

fontsize=7, ha='center', color='purple')

interface\_counter =1

firewall\_count += 1

else:

# Direct router-switch connection (low/moderate)

plt.plot([rx, x], [ry, y], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((rx+x)/2, (ry+y)/2, f"{ip1}/{ip2}",

fontsize=7, ha='center', color='purple')

connections.append({

'name': switch\_name,

'ip': ip2,

'type': 'switch'

})

network\_counter += 1

interface\_counter = 1

# Draw switch

plt.scatter(x, y, color='green', s=80)

plt.text(x, y, switch\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightyellow"))

switch\_positions.append((x, y, switch\_name))

switch\_group1.append((x, y, switch\_name))

switch\_group2.append((x, y, switch\_name))

switch\_count += 1

# Connect workstations to switches

for sx, sy, switch\_name in switch\_group2:

for k in range(ws\_per\_switch):

if workstation\_count > workstations:

break

angle = k \* (2 \* np.pi / ws\_per\_switch)

wx = sx + np.cos(angle) \* workstation\_distance

wy = sy + np.sin(angle) \* workstation\_distance

ws\_name = f"WS{workstation\_count}"

# Draw workstation and connection

plt.scatter(wx, wy, color='orange', s=60)

plt.plot([sx, wx], [sy, wy], 'brown')

plt.text(wx, wy, ws\_name, fontsize=7, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="mistyrose"))

# Assign IPs

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((sx+wx)/2, (sy+wy)/2, f"{ip1}/{ip2}",

fontsize=6, ha='center', color='brown')

connections.append({

'name': ws\_name,

'ip': ip2,

'type': 'workstation'

})

network\_counter +=1

workstation\_count += 1

elif j == 3: # Firewall stub (always present)

fw\_name = f"FW{firewall\_count}"

plt.scatter(x, y, color='red', s=80)

plt.plot([rx, x], [ry, y], 'darkred', linewidth=2)

plt.text(x, y, fw\_name, fontsize=9, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

ip1 = f"192.168.{network\_counter}.1"

plt.text((rx+x)\*0.4, (ry+y)\*0.4, f"{ip1}",

fontsize=7, ha='center', color='darkred')

connections.append({

'name': router\_name,

'ip': ip1,

'type': 'router'

})

network\_counter += 1

interface\_counter = 1

firewall\_count += 1

switch\_group2 = []

# Add switch-to-switch connections for moderate redundancy

if redundancy == 'Moderate' and len(switch\_group1) > 1:

for (x1, y1, sw1), (x2, y2, sw2) in itertools.combinations(switch\_group1, 2):

plt.plot([x1, x2], [y1, y2], color='lightgreen', linestyle='dashed')

all\_switch\_groups.append(switch\_group1)

for i in range(num\_routers - 1):

router1 = router\_names[i]

router2 = router\_names[i + 1]

r1 = router\_positions[i]

r2 = router\_positions[i + 1]

if security in ['Moderate', 'High']:

# Add firewall between routers

mx, my = (r1[0]+r2[0])/2, (r1[1]+r2[1])/2

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red', s=80)

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Router1 to Firewall

plt.plot([r1[0], mx], [r1[1], my], 'gray', linewidth=2)

ip1 = f"192.168.{network\_counter}.1"

plt.text((r1[0]+mx)/2, (r1[1]+my)/2, f"{ip1}",

fontsize=7, ha='center', color='darkblue')

# Firewall to Router2

plt.plot([mx, r2[0]], [my, r2[1]], 'gray', linewidth=2)

ip2 = f"192.168.{network\_counter}.2"

plt.text((mx+r2[0])/2, (my+r2[1])/2, f"{ip2}",

fontsize=7, ha='center', color='darkblue')

connections.append({

'name': router2,

'ip': ip2,

'type': 'router'

})

firewall\_count += 1

network\_counter += 1

else:

# Direct router-to-router connection

plt.plot([r1[0], r2[0]], [r1[1], r2[1]], 'gray', linewidth=2)

ip1 = f"192.168.{network\_counter}.1"

ip2 = f"192.168.{network\_counter}.2"

plt.text((r1[0]+r2[0])/2, (r1[1]+r2[1])/2, f"{ip1}/{ip2}",

fontsize=7, ha='center', color='darkblue')

connections.append({

'name': router2,

'ip': ip2,

'type': 'router'

})

network\_counter += 1

plt.title(f"Linear Network\n"

f"Security: {security.capitalize()}, "

f"Redundancy: {redundancy.capitalize()}, "

f"Scalability: {scalability.capitalize()}\n"

f"{num\_routers} Routers, {num\_switches} Switches, {workstations} Workstations")

plt.axis('equal')

plt.axis('off')

plt.tight\_layout()

plt.show()

else:

router\_spacing=15

switch\_distance=6

workstation\_distance=3

branch\_angles\_deg=[-30, 0, 30, 60]

# Place routers in a straight horizontal line

router\_positions = [(i \* router\_spacing, 0) for i in range(num\_routers)]

router\_names = [f"Router{i+1}" for i in range(num\_routers)]

plt.figure(figsize=(12, 8))

connections = []

all\_switch\_groups = []

switch\_positions = []

# IP counters

network\_counter = 1 # For 3rd octet

interface\_counter = 1 # For 4th octet

switch\_count = 1

firewall\_count = 1

workstation\_count = 1

# Process each router's switches and workstations first

for router\_num in range(num\_routers):

rx, ry = router\_positions[router\_num]

router\_name = router\_names[router\_num]

# Draw the router

plt.scatter(rx, ry, color='blue', s=100)

plt.text(rx, ry, router\_name, fontsize=10, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightgrey"))

# Connect switches to this router

switch\_group1 = []

switch\_group2 = []

base\_angle = np.pi/2 # Pointing upwards

for j, offset in enumerate(branch\_angles\_deg):

angle = base\_angle + np.deg2rad(offset) # This should result in a scalar

x = rx + np.cos(angle) \* switch\_distance

y = ry + np.sin(angle) \* switch\_distance

if j < 3 and switch\_count <= num\_switches: # Switch branches

switch\_name = f"Switch{switch\_count}"

if security == 'High':

# With firewall between router and switch

mx, my = (rx + x)\*0.7, (ry + y)\*0.7

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red', s=80)

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Router to Firewall

plt.plot([rx, mx], [ry, my], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((rx+mx)/2, (ry+my)/2, f"{ip1}",

fontsize=7, ha='center', color='purple')

# Firewall to Switch

plt.plot([mx, x], [my, y], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((mx+x)/2, (my+y)/2, f"{ip1}",

fontsize=7, ha='center', color='purple')

connections.append({

'name': switch\_name,

'ip': ip1,

'type': 'switch'

})

interface\_counter =1

firewall\_count += 1

else:

# Direct router-switch connection (low/moderate)

plt.plot([rx, x], [ry, y], 'black')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((rx+x)/2, (ry+y)/2, f"{ip1}/{ip2}",

fontsize=7, ha='center', color='purple')

connections.append({

'name': switch\_name,

'ip': ip2,

'type': 'switch'

})

network\_counter += 1

interface\_counter = 1

# Draw switch

plt.scatter(x, y, color='green', s=80)

plt.text(x, y, switch\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightyellow"))

switch\_positions.append((x, y, switch\_name))

switch\_group1.append((x, y, switch\_name))

switch\_group2.append((x, y, switch\_name))

ws\_list = [] # Store workstation names and positions for interconnections

switch\_count += 1

# Connect workstations to switches

for sx, sy, switch\_name in switch\_group2:

for k in range(ws\_per\_switch):

if workstation\_count > workstations:

break

angle = k \* (2 \* np.pi / ws\_per\_switch)

wx = sx + np.cos(angle) \* workstation\_distance

wy = sy + np.sin(angle) \* workstation\_distance

ws\_name = f"WS{workstation\_count}"

# Draw workstation and connection

plt.scatter(wx, wy, color='orange', s=60)

plt.plot([sx, wx], [sy, wy], 'brown')

plt.text(wx, wy, ws\_name, fontsize=7, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="mistyrose"))

# Assign IPs

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((sx+wx)/2, (sy+wy)/2, f"{ip1}/{ip2}",

fontsize=6, ha='center', color='brown')

connections.append({

'name': ws\_name,

'ip': ip2,

'type': 'workstation'

})

network\_counter +=1

ws\_list.append((wx, wy, ws\_name))

workstation\_count += 1

# Connect each workstation to every other on this switch

if redundancy == "High":

for i in range(len(ws\_list)):

for j in range(i + 1, len(ws\_list)):

x1, y1, ws1 = ws\_list[i]

x2, y2, ws2 = ws\_list[j]

plt.plot([x1, x2], [y1, y2], linestyle='dotted', color='gray', linewidth=0.8)

interface\_counter =1

elif j == 3: # Firewall stub (always present)

fw\_name = f"FW{firewall\_count}"

plt.scatter(x, y, color='red', s=80)

plt.plot([rx, x], [ry, y], 'darkred', linewidth=2)

plt.text(x, y, fw\_name, fontsize=9, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

ip1 = f"192.168.{network\_counter}.1"

plt.text((rx+x)\*0.4, (ry+y)\*0.4, f"{ip1}",

fontsize=7, ha='center', color='darkred')

connections.append({

'name': router\_name,

'ip': ip1,

'type': 'router'

})

network\_counter += 1

interface\_counter = 1

firewall\_count += 1

switch\_group2 = []

# Add switch-to-switch connections for moderate redundancy

if redundancy in ['Moderate', 'High'] and len(switch\_group1) > 1:

for (x1, y1, sw1), (x2, y2, sw2) in itertools.combinations(switch\_group1, 2):

plt.plot([x1, x2], [y1, y2], color='lightgreen', linestyle='dashed')

all\_switch\_groups.append(switch\_group1)

server\_count = math.ceil(num\_routers/2)

server\_positions = []

server\_router\_name = None

server\_y\_offset = -8 # Below the routers

server\_x\_spacing = 4

# Position servers in a horizontal row

for i in range(server\_count):

sx = i \* server\_x\_spacing + 2 # offset to make it visible

sy = server\_y\_offset

s\_name = f"Server{i+1}"

server\_positions.append((sx, sy, s\_name))

plt.scatter(sx, sy, color='teal', s=80)

plt.text(sx, sy, s\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightcyan"))

if server\_count > 1:

# Place the server router centrally

sr\_x = (server\_positions[0][0] + server\_positions[-1][0]) / 2

sr\_y = server\_y\_offset + 3

server\_router\_name = "ServerRouter"

plt.scatter(sr\_x, sr\_y, color='blue', s=100)

plt.text(sr\_x, sr\_y, server\_router\_name, fontsize=10, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="lightgrey"))

# Connect each server to the server router

for sx, sy, s\_name in server\_positions:

if security == 'High':

# Add firewall between server and server router

mx, my = (sx + sr\_x)\*0.6, (sy + sr\_y)\*0.6

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red', s=80)

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Server to Firewall

plt.plot([sx, mx], [sy, my], 'cyan')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

plt.text((sx+mx)/2, (sy+my)/2, f"{ip1}",

fontsize=6, ha='center', color='teal')

connections.append({

'name': s\_name,

'ip': ip1,

'type': 'server'

})

interface\_counter += 1

# Firewall to Server Router

plt.plot([mx, sr\_x], [my, sr\_y], 'cyan')

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

plt.text((mx+sr\_x)/2, (my+sr\_y)/2, f"{ip2}",

fontsize=6, ha='center', color='teal')

interface\_counter = 1

firewall\_count += 1

else:

# Direct server-router connection (low/moderate)

plt.plot([sx, sr\_x], [sy, sr\_y], 'cyan')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((sx+sr\_x)/2, (sy+sr\_y)/2, f"{ip1}/{ip2}",

fontsize=6, ha='center', color='teal')

connections.append({

'name': s\_name,

'ip': ip1,

'type': 'server'

})

network\_counter += 1

interface\_counter = 1

# Connect servers to routers and track which routers belong to which server

server\_router\_mapping = {}

total\_routers = len(router\_names)

routers\_per\_server = math.ceil(total\_routers // server\_count)

for i, (sx, sy, s\_name) in enumerate(server\_positions):

start\_index = i \* routers\_per\_server

end\_index = start\_index + 2 # each server gets 2 routers

connected\_indices = list(range(start\_index, min(end\_index, total\_routers)))

server\_router\_mapping[s\_name] = [router\_names[idx] for idx in connected\_indices]

for idx in connected\_indices:

router = router\_names[idx]

rx, ry = router\_positions[idx]

if security == 'High':

# Add firewall between server and router

mx, my = (sx + rx)\*0.6, (sy + ry)\*0.6

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red', s=80)

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Server to Firewall

plt.plot([sx, mx], [sy, my], 'skyblue', linestyle='dotted')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

plt.text((sx+mx)/2, (sy+my)/2, f"{ip1}",

fontsize=6, ha='center', color='skyblue')

connections.append({

'name': s\_name,

'ip': ip1,

'type': 'server'

})

interface\_counter += 1

# Firewall to Router

plt.plot([mx, rx], [my, ry], 'skyblue', linestyle='dotted')

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

plt.text((mx+rx)/2, (my+ry)/2, f"{ip2}",

fontsize=6, ha='center', color='skyblue')

interface\_counter = 1

firewall\_count += 1

else:

# Direct server-router connection (low/moderate)

plt.plot([sx, rx], [sy, ry], 'skyblue', linestyle='dotted')

ip1 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

ip2 = f"192.168.{network\_counter}.{interface\_counter}"

interface\_counter += 1

plt.text((sx+rx)/2, (sy+ry)/2, f"{ip1}/{ip2}",

fontsize=6, ha='center', color='skyblue')

connections.append({

'name': s\_name,

'ip': ip1,

'type': 'server'

})

network\_counter += 1

interface\_counter = 1

# Connect routers that share the same server only when redundancy is high

if redundancy == 'High':

# Create a reverse mapping from router to server

router\_server\_mapping = {}

for server, routers in server\_router\_mapping.items():

for router in routers:

router\_server\_mapping[router] = server

# Find all pairs of routers that share the same server

for i in range(num\_routers):

for j in range(i+1, num\_routers):

router1 = router\_names[i]

router2 = router\_names[j]

# Check if both routers are connected to the same server

if (router1 in router\_server\_mapping and

router2 in router\_server\_mapping and

router\_server\_mapping[router1] == router\_server\_mapping[router2]):

r1 = router\_positions[i]

r2 = router\_positions[j]

if security in ['Moderate', 'High']:

# With firewall between routers

mx, my = (r1[0]+r2[0])/2, (r1[1]+r2[1])/2

fw\_name = f"FW{firewall\_count}"

plt.scatter(mx, my, color='red', s=80)

plt.text(mx, my, fw\_name, fontsize=8, ha='center', va='center',

bbox=dict(boxstyle="round", facecolor="salmon"))

# Router1 to Firewall

plt.plot([r1[0], mx], [r1[1], my], 'gray', linewidth=2)

ip1 = f"192.168.{network\_counter}.1"

plt.text((r1[0]+mx)/2, (r1[1]+my)/2, f"{ip1}",

fontsize=7, ha='center', color='darkblue')

# Firewall to Router2

plt.plot([mx, r2[0]], [my, r2[1]], 'gray', linewidth=2)

ip2 = f"192.168.{network\_counter}.2"

plt.text((mx+r2[0])/2, (my+r2[1])/2, f"{ip2}",

fontsize=7, ha='center', color='darkblue')

connections.append({

'name': router2,

'ip': ip2,

'type': 'router'

})

firewall\_count += 1

network\_counter += 1

else:

# Direct router-router connection (low security)

plt.plot([r1[0], r2[0]], [r1[1], r2[1]], 'gray', linewidth=2)

ip1 = f"192.168.{network\_counter}.1"

ip2 = f"192.168.{network\_counter}.2"

plt.text((r1[0]+r2[0])/2, (r1[1]+r2[1])/2, f"{ip1}/{ip2}",

fontsize=7, ha='center', color='darkblue')

connections.append({

'name': router2,

'ip': ip2,

'type': 'router'

})

network\_counter += 1

plt.title(f"Linear Network\n"

f"Security: {security.capitalize()}, "

f"Redundancy: {redundancy.capitalize()}, "

f"Scalability: {scalability.capitalize()}\n"

f"{num\_routers} Routers, {num\_switches} Switches, {workstations} Workstations")

plt.axis('equal')

plt.axis('off')

plt.tight\_layout()

plt.show()

# Return to main menu

with open("ip\_assignments.txt", "w") as f:

for connection in connections:

f.write(f"{connection['name']} | {connection['ip']} | {connection['type']}\n")

f.write(f"{security} | {server\_needs}")

self.main\_frame.destroy()

MainMenu(self.root)

def display\_network\_info(self, workstations, switches, routers, servers, firewalls, server\_needs):

info = [

["Workstations", workstations],

["Switches", switches],

["Routers", routers],

["Servers", servers],

["Firewalls", firewalls],

["Server Services", ", ".join(server\_needs) if server\_needs else "None"]

]

message = "Network Configuration Summary:\n\n"

message += tabulate(info, headers=["Component", "Quantity"], tablefmt="grid")

messagebox.showinfo("Network Summary", message)

class Mapper:

def \_\_init\_\_(self, root):

self.root = root

root.title("Network Scanner")

self.hosts = []

self.build\_ui()

def build\_ui(self):

self.main\_frame = tk.Frame(self.root)

self.main\_frame.pack(padx=20, pady=20)

self.label = tk.Label(self.main\_frame, text="Enter an IP address or subnet:", font=("Arial", 14))

self.label.pack(pady=10)

self.input\_var = tk.StringVar()

self.entry = tk.Entry(self.main\_frame, textvariable=self.input\_var, width=30)

self.entry.pack(pady=5)

self.button\_frame = tk.Frame(self.main\_frame)

self.button\_frame.pack(pady=10)

self.add\_button = tk.Button(self.button\_frame, text="Add Host", command=self.add\_host)

self.add\_button.grid(row=0, column=0, padx=5)

self.discover\_button = tk.Button(self.button\_frame, text="Discover Subnet", command=self.discover\_subnet)

self.discover\_button.grid(row=0, column=1, padx=5)

self.finish\_button = tk.Button(self.button\_frame, text="Finish", command=self.finish\_input)

self.finish\_button.grid(row=0, column=2, padx=5)

self.hosts\_listbox = tk.Listbox(self.main\_frame, width=50)

self.hosts\_listbox.pack(pady=10)

def add\_host(self):

ip = self.input\_var.get().strip()

if ip:

self.hosts.append(ip)

self.hosts\_listbox.insert(tk.END, ip)

self.input\_var.set("")

else:

messagebox.showwarning("Input Error", "Please enter a valid IP address.")

def discover\_subnet(self):

subnet = self.input\_var.get().strip()

if not subnet:

messagebox.showwarning("Input Error", "Please enter a subnet (e.g., 192.168.0.0/24)")

return

scanner = nmap.PortScanner()

try:

scanner.scan(hosts=subnet, arguments="-sn")

discovered = [host for host in scanner.all\_hosts() if scanner[host].state() == "up"]

if not discovered:

messagebox.showinfo("Discovery Result", "No live hosts found.")

return

for host in discovered:

if host not in self.hosts:

self.hosts.append(host)

self.hosts\_listbox.insert(tk.END, host)

messagebox.showinfo("Discovery Complete", f"Discovered {len(discovered)} live host(s).")

self.input\_var.set("")

except Exception as e:

messagebox.showerror("Scan Error", f"Failed to scan subnet.\nError: {e}")

def finish\_input(self):

if not self.hosts:

messagebox.showinfo("No Hosts", "You haven't entered or discovered any IP addresses yet.")

return

messagebox.showinfo("Scanning", f"Scanning the following hosts:\n{', '.join(self.hosts)}")

Mapper.run(self.hosts)

@staticmethod

def scan\_host(host, file\_handle):

try:

scanner = nmap.PortScanner()

scanner.scan(host, arguments="")

result = f"\nHost: {host}\n"

result += f"State: {scanner[host].state()}\n"

for proto in scanner[host].all\_protocols():

result += f"Protocol: {proto}\n"

ports = scanner[host][proto].keys()

for port in ports:

state = scanner[host][proto][port]['state']

result += f"Port: {port} State: {state}\n"

print(result)

file\_handle.write(result)

except Exception as e:

error\_msg = f"Error scanning {host}: {e}\n"

print(error\_msg)

file\_handle.write(error\_msg)

@staticmethod

def run(target\_ips):

scanner = nmap.PortScanner()

live\_hosts = []

for ip in target\_ips:

try:

scanner.scan(hosts=ip, arguments="-sn")

if ip in scanner.all\_hosts() and scanner[ip].state() == "up":

live\_hosts.append(ip)

except Exception as e:

print(f"Error scanning {ip}: {e}")

if not live\_hosts:

print("No live hosts found.")

return

with open("scanresults.txt", "w") as f:

with concurrent.futures.ThreadPoolExecutor(max\_workers=10) as executor:

futures = [executor.submit(Mapper.scan\_host, host, f) for host in live\_hosts]

concurrent.futures.wait(futures)

print("\nScan results saved to scanresults.txt")

if \_\_name\_\_ == "\_\_main\_\_":

## 8.2 Appendix B - Low level Manual Set up Guide

Manual Network Hardware Set up Guide

By Cameron Finlay

Student ID: 2101510

# **Abstract**

This guide is aimed at non-technical users to assist with manual set up of network hardware. This guide will cover how to connect devices to a power supply, connect devices together using ethernet cables and setting up the devices so that they can communicate with each other and be Secure Shelled (SSH) into. This initial set up will allow for further configuration to be done by the automated network set up tool which will assist in planning the network. This guide also assists with interpreting the plan and correctly connecting devices in the way intended.

Contents

Abstract. 2

1. Hardware. 4

1.1 Appropriate Hardware. 4

1.2 Cabling. 5

1.2.1 Power Supply. 5

1.2.2 Ethernet Cables. 6

1.2.3 Console Cable. 6

2. Software. 7

2.1 Manual configuration. 7

References. 8

# **1. Hardware**

## **1.1 Appropriate Hardware**

There are various considerations when picking the correct hardware for a network. The First consideration is expected size. This will determine how many devices are present on the network.

Next is cabling and the amount of data that will be the expected to be transferred across the connection there are three different types of relevant of ethernet cables. The first is a standard ethernet cable which is for regular connections transferring up to 10 megabits per second. The second is fast ethernet which is capable of up to 100 megabits per second and the third and final is gigabit ethernet which supports speeds up 1000 megabits per second. The downside of each is that the price increases with speed so forward planning is important to save money.

For example,

If you have a connection between the internet which supports up to 80 megabits per second then you would need a fast ethernet cable, but if the connection between that router with connection to the internet and a server is only expected to get 8 megabits per second at peak times then both can be regular ethernet cables as long as there is no other traffic coming through the router than that of the server

The third consideration is scalability this will determine how many additional cable slots any given router or switch will need above the currently used amount.

Routers are needed to connect multiple networks together. This means that depending on the number of networks present there may need to be multiple routers.

For example,

Two different servers with IP address 192.168.29.1/24 and 192.168.30/24 would need to be connected by a router whereas servers with 192.168.29.1/24 and 192.168.29.2/24 can be connected directly together. This is not required knowledge for the guide.

## **1.2 Cabling**

### **1.2.1 Power Supply**

There are two different types of cables that will need to be connected to all devices present on the network the first is the power cable. The majority of power cables use the cable shown below in figure 1. However, if your device comes with its power supply cable use the one that is included with it. There should be a symbol to denote where the power cable should be inserted.

Figure 1 Most common cable

### **1.2.2 Ethernet Cables**

Ethernet cables are an essential way of connecting devices together within a computer network. This is because they allow for data to travel in-between devices and depending on the ethernet cable the data transfer can be extremely fast. As mentioned earlier there are three main types of ethernet cable the normal (10 megabytes per second) the fast (100 megabytes per second) and the gigabit (1000 megabytes per second). However, all these cables look and act very similarly to each other. There is no real difference between them other than the speed and cost of them.

Figure 1 Ethernet cable Ports (p-themes, 2025)

Shown above is ethernet cable ports this is where you must plug in ethernet cables (shown below in figure 2). This is the first step to creating a functional connection between devices. When plugging in make sure to take care which port you are putting the cable into so that in the next step the device can be configured properly.

Figure 2 ethernet cable (Ellis, 2025)

### **1.2.3 Console Cable**

There will be a console port on a device that does not come configured with the ability to accept Secure Shell (SSH) or Telnet connections. This port is used to do the software portion of the configuration and as such is required and must be done before proceeding to the next stage of configuration. The Console cable will be used to connect a PC or laptop directly to the router, switch or server. The types of cables are USB-serial or USB-C etc. Once connected use a terminal emulator such as PuTTy for windows or screen for Mac and Linux.

# **2. Software**

### **2.1 Manual configuration**

First you will need to be able to access the command line interface of the device you whish to configure. Next for devices that have an IOS operating system such as cisco routers and switches use the following commands replacing the <> with the appropriate information. The below commands will allow any automated script you have to connect to the device via telnet and do the remaining configuration

1. Conf t

2. User <username> privilege 15 secret <password>

3. Line vty 0 4

4. Transport input telnet

5. Login local

6. Exit

7. Enable secret <password>

The Second step for connecting devices is to set the IP address for the ethernet cable and then make sure that the connection is up and running. To input the IP address and subnet for the required connection use the below commands.

1. Int fa<fast ethernet number>

2. Ip add <Ip address> <subnet>

3. No shut

The final step is to enable and configure Open shortest Path First (OSPF) this will allow the devices to move the information sent to them to the correct place as efficiently as possible. The below commands should be done only be used once per device except command 3 that should be repeated for each interface as necessary.

1. router ospf 1

2. router-id <unique router ID ie 1.1.1.1>

3. network <interface network> <wildcard mask (opposite of netmask)> area 0

4. network 1.1.1.1 0.0.0.0 area 0

5. ip route 0.0.0.0 0.0.0.0 [next-hop-ip or interface]

# **References**

Quora. (2019). *Are most router power cords standardized?* [online] Available at: https://www.quora.com/Are-most-router-power-cords-standardized [Accessed 25 Mar. 2025].

p-themes (n.d.). *What Is An Ethernet Port?* [online] Infinity Cable Products. Available at:<https://infinity-cable-products.com/blogs/hardware/what-is-an-ethernet-port>. [Accessed 30 Mar. 2025]

Ellis, J. (2022). What Is An Ethernet Cable And What Does It Do? *What Is An Ethernet Cable And What Does It Do?* [online] 1 Jan. Available at:<https://www.comms-express.com/blog/what-does-an-ethernet-cable-do/>. [Accessed 30 Mar. 2025]‌

‌

## 8.3 Appendix C - Survey Question 4 and 5 Results

|  |  |
| --- | --- |
| Response 1 | I would learn how many devices I would want and what services they need |
| Response 2 | I would look at the environment that the network will be deployed in and figure out how I can get the network configured like in the image generated from the program |
| Response 3 | Start by collating what sort of hardware you will need and how much of each device is required for your network, cable the network appropriately and ensure that it is done in a logical fashion and then configure the devices (IP addresses etc) |
| Response 4 | Find out what I would need for the services my company would need. How many workstations, how secure everything would be etc |
| Response 5 | When setting up a new network there are three things to consider. Network size, data transferred and scalability. Once these factors are decided then equipment can be bought and the network can be assembled. |
| Response 6 | Consider appropriate hardware and the cabling required |
| Response 7 | Appropriate hardware and cables |
| Response 8 | determine how many devices are present, calibrate amount of data expected to be stransfered, determine how many additional able slots any router or switch will need |
| Response 9 | Decide the number of people, service and work stations I require. |
| Response 10 | Before deploying, I would pre-configure devices with correct IP settings, update firmware, apply security settings, label them clearly, and test functionality in a lab environment to ensure they’re ready for live deployment. |
| Response 11 | Required services, number of users, number of workstations along with any necessary security requirements. |
| Response 12 | Hardware, cabling, power supply |
| Response 13 | You would need to get all the generated devices, power cabling for the devices, all the required ethernet connections. you would also need software for the devices. you would also need a console cable to program into. |
| Response 14 | I need the appropriate hardware, the correct power supply and cables, an Ethernet and cable and console cable and finally the correct software and knowledge on manual configuration |

Table 4 Survey Question 4 answers

|  |  |
| --- | --- |
| Response 1 | a Power cable and a ehternet cable |
| Response 2 | I would need a power cable to provide power to the networking equipment and ethernet cables to allow the networking devices to communicate with each other in and out of the network, then I would use the command line to manually configure the devices by assigning IP addresses for example and then conduct some tests to see if everything has been configured properly |
| Response 3 | Ethernet cables, power supply cables, console cables. You need to give devices their addresses,. |
| Response 4 | It would vary on the technology needed for the network, and what cabling they needed to worm sufficient |
| Response 5 | Multiple ethernet and power cables would be needed. |
| Response 6 | Console cable, ethernet, power cables and some manual configuration |
| Response 7 | Console cables, ethernet cables and using a power supply. Alongside a bit of manual config |
| Response 8 | connect power supply cable to all devices, attach Ethernet cable and connect console cable |
| Response 9 | I would considered the hardware required and consult the guide for manual configuration. |
| Response 10 | For the initial stages of configuration, I use console cables (usually RJ45 to USB or USB-C) to connect to the network device. I access the CLI via a terminal program, set IP addresses, hostnames, and basic routing or VLANs as needed. |
| Response 11 | Power and ethernet cables, along with console cables. |
| Response 12 | Ethernet cables, console cables, manual configuration |
| Response 13 | power cable, ethernet , console are the cables required. |
| Response 14 | You turn it on and plug in the correct power supply. Connect the correct cables, Ethernet and console cables. |

Table 5 Survey Question 5 answers