NETWORK MONITORING AND MANAGEMENT SYSTEM

PROJECT REPORT

Submitted by

BHARATH KUMAR G

Register No:31022P08001

in partial fulfillment for the award of the degree

of

MASTER OF COMPUTER APPLICATIONS

in

PG DEPARTMENT OF COMPUTER APPLICATIONS

GOVERNMENT THIRUMAGAL MILLS COLLEGE

GUDIYATTAM, VELLORE DIST - 632 602

MARCH/APRIL – 2024

GOVERNMENT THIRUMAGAL MILLS COLLEGE, GUDIYATTAM.

VELLORE DIST – 632 602

PG DEPARTMENT OF COMPUTER APPLICATIONS

PROJECT WORK

MARCH/APRIL - 2024

This is to certify that the project entitled

NETWORK MONITORING AND MANAGEMENT SYSTEM

is the bonafide record of project work done by

BHARATH KUMAR G

Register No: 31022P08001

of MCA (PG Department of Computer Applications) during the year 2023 - 2024.

---------------------------   -----------------------------

Project Guide   Head of the Department

Submitted for the Project Viva-Voce examination held on ……………..

---------------------------   -----------------------------

1. External Examiner   2. External Examiner

DECLARATION

I affirm that the project work titled **NETWORK MONITORING AND MANAGEMENT SYSTEM** being submitted in partial fulfillment for the award of **Master of Computer Applications** is the original work carried out by me. It has not formed the part of any other project work submitted for award of any degree or diploma, either in this or any other University.

BHARATH KUMAR G

31022P08001

I certify that the declaration made above by the candidate is true

Dr. B. REVATHI M.C.A., M.Phil., B.Ed., Ph.D.,

ACKNOWLEDGEMENT

I express my gratitude to our beloved Principal **Dr. G. KRISHNAN M.A., M.Phil., M.ED, Ph.D.,** Government Thirumagal Mills College Gudiyattam, for providing the resource facilities and encouraging gesture for completion of my project.

I wish to express my deep sense of gratitude and heartiest thanks to our professor **Dr. K. ARULANANDAM, MCA., M.Phil., Ph.D.,** Head of the Department of MCA, Government Thirumagal Mills College, Gudiyattam.

I am much privileged to have **Dr. B. REVATHI MCA., M.Phil., B.Ed., Ph.D.,** lecturer in MCA Department, as my guide for this project. I heartily thank her for her overall help and support at all stage.

I also take an opportunity to thank all the TEACHING and NON - TEACHING STAFF for their fullest and valuable assistance provided in this project work.

Last but not the least, I own my special thanks to my beloved all my friends and family member for their help in finishing this project successfully.

Place: Gudiyattam   Submitted by

Date: **BHARATH KUMAR G**

TABLE OF CONTENTS

Abstract

The "**Network Monitoring and Management System**" is a comprehensive solution designed to facilitate the monitoring and management of computer networks. The project is structured into modular components, each addressing specific aspects of network administration. The system encompasses functionalities for network scanning, port detection, device discovery, resource monitoring, real-time monitoring, device management, alerts and notifications, reporting, visualization, and automated network scanning. The modular design promotes scalability and flexibility, allowing users to selectively deploy components based on their network management needs.

The project's directory structure reflects a well-organized approach, with distinct directories for network scanning, port detection, device discovery, resource monitoring, real-time monitoring, device management, alerts and notifications, reporting and visualization, and utilities. The graphical user interface (GUI) component is housed under the "gui" directory, featuring windows for device management, real-time monitoring, resource monitoring, a main application window, a dashboard, settings, and a login interface.

The "auto scan" directory contains modules for automated network scanning, utilizing techniques such as ping and port scanning. The "utils" directory includes utility modules for SNMP (Simple Network Management Protocol) operations, security monitoring, and wireless network monitoring.

The project aims to provide network administrators with a versatile toolset for efficiently monitoring, managing, and securing computer networks. By adopting a modular and organized structure, the system promotes extensibility, allowing for future enhancements and the incorporation of additional features. The utilization of a GUI facilitates user-friendly interactions, making the system accessible to administrators with varying levels of technical expertise.

This network management system serves as a valuable asset for organizations seeking a robust solution for ensuring the optimal performance, security, and stability of their computer networks.

Chapter – 1

Project Information

1.1 Introduction

The "Network Monitoring and Management System" is a comprehensive software solution designed to streamline the supervision and optimization of computer networks. At its core, this system employs a structured file organization strategy, ensuring efficient code management and collaboration among developers. By harnessing the capabilities of Python's Tkinter library, the system facilitates the creation of intuitive graphical user interfaces (GUIs), enabling seamless interaction with network monitoring tools and functionalities. With an emphasis on network health and security, the system integrates advanced functionalities such as port scanning and device discovery, leveraging libraries like Scapy and nmap. Through these capabilities, network administrators can conduct thorough assessments of network infrastructure and promptly address any anomalies or security threats. Additionally, the system incorporates real-time monitoring of CPU, memory, disk, and network usage using Python's psutil library, empowering administrators with actionable insights into system performance. By adopting structured file organization principles, the Network Monitoring and Management System ensures code maintainability, scalability, and fosters efficient collaboration throughout the software development lifecycle.

The Network Monitoring System is a comprehensive software solution designed to monitor and manage network devices, resources, and traffic in real-time. This system aims to provide administrators with tools for efficient network monitoring, device management, and proactive issue detection. Moreover, the system incorporates a task manager module, providing administrators with granular insights into system resource utilization. Through dynamic visualizations and real-time updates, users can monitor CPU, memory, disk, and network usage patterns, enabling proactive resource allocation and optimization. Leveraging matplotlib for data visualization and psutil for system monitoring, the task manager module offers a comprehensive overview of system performance metrics, empowering administrators to make informed decisions to enhance network efficiency and stability. By amalgamating file structures, GUI development, network scanning, and resource monitoring functionalities, the Network Monitoring and Management System emerges as a versatile solution for network administrators seeking to streamline network operations and bolster cybersecurity defences.

1.2 Aim of the system:

The aim of the Network Monitoring and Management System is to provide network administrators with a comprehensive toolkit for overseeing and optimizing network infrastructure. By integrating various modules such as network scanning, device discovery, real-time monitoring, and task management, the system aims to offer a centralized platform for efficiently managing network resources, identifying potential vulnerabilities, and ensuring optimal performance. Ultimately, the goal is to enhance network reliability, security, and efficiency by providing administrators with the necessary tools to monitor, analyze, and respond to network events effectively.

1.3 Objective:

- Develop a user-friendly Graphical User Interface (GUI) for easy navigation and interaction with the Network Monitoring and Management System.
- Implement network scanning functionalities to discover devices, identify network topology, and detect open ports for enhanced security measures.
- Integrate real-time monitoring features to track CPU, memory, disk, and network usage, providing administrators with valuable insights into system performance.
- Incorporate task management capabilities to facilitate the monitoring and control of processes and applications running on network devices.
- Enhance the system's reliability and scalability to accommodate large-scale network infrastructures while ensuring efficient resource utilization.
- Provide alerts and notifications functionality to promptly notify administrators of critical network events or potential security breaches.
- Enable Wi-Fi network management, including listing available networks, discovering devices connected to the network, and retrieving network passwords for authorized access.
- Ensure cross-platform compatibility to support deployment across different operating systems and network environments.
- Offer comprehensive documentation and user guides to assist administrators in effectively utilizing the Network Monitoring and Management System.

Chapter – 2

Requirement and Analysis

2.1 What is Requirement Analysis:-

Requirement analysis for the project involves the systematic process of identifying, documenting, and validating the needs and expectations of stakeholders regarding the network monitoring and management system. It encompasses understanding the functionalities, features, constraints, and objectives of the system to ensure its successful development and deployment. Requirement analysis is crucial for defining the scope of the project and serves as the foundation for designing, implementing, and testing the system. Key aspects of requirement analysis include:

1. **Gathering Requirements:** Engaging with stakeholders, including network administrators, engineers, and end-users, to gather their requirements, preferences, and constraints. This involves conducting interviews, surveys, and workshops to elicit valuable insights into the desired features and functionalities of the system.

2. **Analyzing Requirements:** Analyzing and prioritizing the gathered requirements to identify essential features and functionalities. This involves categorizing requirements into functional and non-functional aspects and identifying dependencies and conflicts among them.

3. **Documenting Requirements:** Documenting the identified requirements in a clear and structured manner using requirement specification documents. These documents outline the system's features, user interactions, data flows, and performance expectations, providing a comprehensive reference for the development team.

4. **Validating Requirements:** Validating the documented requirements with stakeholders to ensure that they accurately represent their needs and expectations. This involves conducting reviews, walkthroughs, and prototyping sessions to gather feedback and make necessary adjustments to the requirements.

5. Managing Requirements: Managing changes to the requirements throughout the project lifecycle to accommodate evolving stakeholder needs and project constraints. This involves establishing a formal change control process to track and evaluate proposed changes and their impact on the system.

2.2 Perspective

The network monitoring and management system presented here offer a comprehensive solution for overseeing and maintaining network infrastructure. It provides users with a multifaceted perspective on network health, performance, and security through a combination of innovative features and intuitive interfaces.

2.2.1 Interface

The interface of the network monitoring and management system serves as the primary point of interaction between users and the system. It encompasses the graphical user interface (GUI) components, command-line interfaces (CLI), and any other means through which users interact with the system. The interface plays a crucial role in providing users with intuitive access to system functionalities, real-time data visualization, and actionable insights into network performance and health.

Graphical User Interface (GUI): The GUI component of the system provides users with a visually appealing and user-friendly interface for accessing various features and functionalities. It includes interactive dashboards, charts, graphs, and tables for displaying network statistics, device statuses, and performance metrics. The GUI enables users to perform tasks such as device discovery, port scanning, CPU and memory monitoring, and network traffic analysis through intuitive point-and-click interactions.

Command-Line Interface (CLI): In addition to the GUI, the system may offer a CLI for advanced users and administrators who prefer command-based interactions. The CLI provides a text-based interface for executing commands, querying system information, and performing administrative tasks efficiently. It allows users to access system functionalities quickly, automate tasks through scripting, and integrate the system with existing network management tools and workflows.

Web Interface: The system may also feature a web-based interface accessible through standard web browsers, enabling users to monitor and manage the network remotely. The web interface provides access to system functionalities from any device with internet connectivity, facilitating seamless remote management and monitoring of network infrastructure.

APIs and Integration: To enhance interoperability and extensibility, the system may offer application programming interfaces (APIs) for integrating with third-party systems, tools, and services. APIs enable developers to build custom integrations, automate workflows, and extend the functionality of the network monitoring and management system according to specific organizational requirements.

2.2.2 Hardware Requirements:-

Server Requirements:

Processor: Quad-core Intel Xeon or equivalent AMD processor.

RAM: Minimum 8 GB DDR4 RAM, recommended 16 GB for optimal performance.

Storage: SSD storage with a minimum of 256 GB for fast data access.

Network Interface: Gigabit Ethernet adapter for high-speed network connectivity.

Operating System: Compatible with Linux distributions (e.g., Ubuntu Server, CentOS) or Windows Server editions.

Network Devices:

Routers: Cisco ISR 4000 series or equivalent for robust routing capabilities.

Switches: Cisco Catalyst 3000 series or equivalent for efficient network switching.

Servers: Dell PowerEdge R-series servers or equivalent for hosting critical services.

Workstations: HP EliteDesk series or equivalent for desktop computing.

Networking Infrastructure:

Cabling: CAT6 or higher-grade Ethernet cables for reliable network connections.

Switches: Managed switches with VLAN support for network segmentation and traffic control.

Routers: Enterprise-grade routers with VPN and firewall capabilities for secure network access.

Monitoring Hardware:

Temperature Sensors: Environmental monitoring sensors (e.g., APC NetBotz) for tracking temperature fluctuations.

Power Usage Monitors: Power monitoring devices (e.g., Eaton Power Xpert) for monitoring power consumption.

UPS (Uninterruptible Power Supply): Ensures uninterrupted power supply to critical network components during outages.

2.2.3 Software Requirements:-

- **Operating System Compatibility:**
  - ➢ **Linux distributions** (e.g., Ubuntu, CentOS) for optimal performance and compatibility with networking tools.
  - ➢ **Windows Server editions** for organizations preferring a Windows-based environment.

- **Windows 10 & 11 editions for** basic operation preforms a windows environment.

- **Programming Language and Frameworks:**
  - **Python 3.x:** Primary programming language for backend development and scripting tasks.
  - **Tkinter:** GUI toolkit for developing the frontend interface, compatible with Python's standard library.
  - **Scapy:** Python library for packet manipulation and network scanning, used for real-time monitoring and packet analysis.
  - **Nmap:** Open-source network scanning tool for network discovery, port scanning, and service enumeration, integrated for network mapping and device discovery functionalities.

- **Networking Tools and Protocols:**
  - **SNMP (Simple Network Management Protocol):** Utilized for device monitoring and management, providing a standardized protocol for collecting and organizing information about managed devices on IP networks.
  - **Scapy and Nmap:** Leveraged for network scanning, packet sniffing, and traffic analysis, enabling real-time monitoring and detection of network anomalies.
- **Visualization Tools:**
  - **Tkinter:** Utilized for building interactive GUI components and visualizing real-time network statistics within the frontend interface.
  - **Matplotlib (Optional):** Python library for creating static, animated, and interactive visualizations, enabling the generation of graphs and charts for network data analysis.
- **Integration with Monitoring Agents:**
  - **Scapy and Nmap:** Integration with monitoring agents running on network devices for data collection and analysis, enabling seamless communication between the monitoring system and monitored devices.

2.2.4 Existing System:

The existing system is a basic network monitoring solution designed to provide administrators with limited insights into their network infrastructure. It consists of a set of loosely integrated scripts and tools for performing basic network monitoring tasks, such as device discovery, port scanning, and resource monitoring. However, the system lacks a unified and user-friendly interface, making it challenging for administrators to efficiently manage and troubleshoot network issues.

1. Script-based Monitoring Tools:

- ➢ The existing system relies on standalone scripts for performing various monitoring tasks, such as device discovery, port scanning, and resource monitoring.
- ➢ These scripts are typically written in Python and executed manually by administrators.

2. Limited Graphical Interface:

- ➢ The system lacks a comprehensive graphical user interface (GUI) for centralized monitoring and management.
- ➢ Administrators interact with the system primarily through command-line interfaces (CLIs) or individual script executions.

3. Basic Network Monitoring Functionality:

- ➢ **Device Discovery:** Utilizes simple network scanning techniques to discover devices connected to the network.
- ➢ **Port Scanning:** Performs basic port scanning to identify open ports on network devices.
- ➢ **Resource Monitoring:** Monitors CPU, memory, and disk usage on network devices using basic system commands.

Drawbacks of the Existing System:

1. Lack of Centralized Management:

- The absence of a centralized management interface makes it challenging for administrators to oversee and control network monitoring tasks efficiently.
- Manual execution of scripts results in time-consuming and error-prone monitoring processes.

2. Limited Scalability:

- The existing system lacks scalability and struggles to accommodate growing network infrastructures and monitoring requirements.
- Adding new monitoring functionalities or integrating with additional network devices requires significant manual effort and code modifications.

3. Poor User Experience:

- The reliance on command-line interfaces and individual scripts hinders user experience, particularly for administrators unfamiliar with command-line operations.
- Lack of intuitive visualization tools makes it difficult to interpret monitoring data and identify network issues effectively.

4. Security Concerns:

- The absence of robust security mechanisms, such as authentication and encryption, poses security risks, particularly when accessing sensitive network data.
- Limited access controls make the system vulnerable to unauthorized access and potential security breaches.

5. Inefficient Resource Utilization:

- ➢ Resource monitoring capabilities are basic and lack granularity, limiting administrators' ability to identify and address performance bottlenecks effectively.
- ➢ Lack of real-time monitoring features impedes timely detection and resolution of network issues, leading to potential downtime and service disruptions.

2.2.5 Proposed System:

The proposed system is a comprehensive network monitoring solution aimed at providing administrators with robust tools for efficiently managing and monitoring network infrastructure. Leveraging a modular architecture and a user-friendly graphical interface, the system offers a wide range of functionalities, including device discovery, real-time monitoring, resource management, and reporting.

1. Modular Architecture:

- ➢ The system will be organized into modular components, each responsible for specific monitoring tasks such as device discovery, port scanning, resource monitoring, and real-time monitoring.
- ➢ Modularity enables flexibility, scalability, and easier maintenance of the system.

2. Graphical User Interface (GUI):

- ➢ The system will feature a user-friendly GUI built using Tkinter, providing administrators with intuitive visualizations and controls for managing network monitoring tasks.
- ➢ The GUI will offer dashboards, reports, and interactive tools for monitoring network performance and detecting anomalies.

3. Enhanced Monitoring Functionalities:

- ➢ Device Discovery: Utilizes advanced scanning techniques (e.g., SNMP, Nmap) for comprehensive device discovery and network mapping.

- Port Scanning: Implements robust port scanning mechanisms to identify open ports and services on network devices accurately.
- Resource Monitoring: Monitors CPU, memory, disk usage, and network traffic in real-time, providing granular insights into resource utilization.

4. Security Measures:

- Authentication and Access Control: Implements secure user authentication mechanisms to control access to the system's functionalities.
- Encryption: Utilizes encryption protocols (e.g., TLS/SSL) to secure data transmission between the frontend and backend components.
- Firewall Integration: Integrates with firewall configurations to protect the monitoring system from unauthorized access attempts.

5. Real-Time Monitoring and Alerts:

- Real-Time Monitoring: Provides real-time monitoring of network traffic, latency, and packet loss using tools like Scapy, enabling prompt detection and resolution of network issues.
- Alerts and Notifications: Generates alerts and notifications for critical network events, enabling administrators to respond proactively to potential issues.

2.3 System Function:

The network monitoring and management system is equipped with a range of functions designed to address various aspects of network administration, performance monitoring, and security management. These functions enable users to effectively monitor network activity, analyse performance metrics, troubleshoot issues, and implement security measures. Key system functions include:

Device Discovery: The system facilitates the discovery of devices within the network, providing users with detailed information about connected devices, including IP and MAC addresses, vendor details, and status indicators.

Port Scanning: Users can perform port scanning to identify open ports on target devices, enabling them to assess the accessibility and security posture of network endpoints.

Resource Monitoring: The system continuously monitors resource utilization metrics such as CPU, memory, and disk usage, allowing users to track system performance trends, identify bottlenecks, and optimize resource allocation.

Network Traffic Analysis: Through real-time network traffic analysis, users can gain insights into bandwidth utilization, traffic patterns, and application-level protocols, facilitating network optimization and capacity planning.

Alerting and Notifications: The system includes alerting mechanisms to notify users of critical events, threshold breaches, or anomalous network behavior, enabling proactive intervention and remediation.

Graphical Visualization: Utilizing graphical visualization techniques such as charts, graphs, and heatmaps, the system presents network data in a visually intuitive manner, enhancing situational awareness and decision-making.

Remote Management: Remote management capabilities allow users to administer the system and perform monitoring tasks from anywhere with internet access, ensuring flexibility and accessibility for distributed network environments.

Customization and Extensibility: The system supports customization and extensibility through APIs and modular architecture, enabling users to integrate third-party tools, develop custom plugins, and extend functionality to suit specific requirements.

2.4 User Characteristics

The network monitoring and management system caters to a diverse range of users with varying levels of technical expertise and responsibilities within the organization. User characteristics include:

1. System Administrators: Experienced IT professionals responsible for configuring, maintaining, and securing network infrastructure. They require comprehensive visibility into network performance, security posture, and resource utilization to ensure optimal operation and mitigate risks effectively.

2. Network Engineers: Skilled professionals specializing in network design, implementation, and optimization. They leverage the system to analyze network traffic, troubleshoot connectivity issues, and fine-tune network configurations to meet performance objectives and business requirements.

3. Security Analysts: Security-focused personnel tasked with monitoring and mitigating cybersecurity threats across the network. They utilize the system to detect anomalous behaviour, investigate security incidents, and enforce access controls to safeguard sensitive data and critical assets.

4. IT Managers: Decision-makers responsible for strategic planning, budgeting, and resource allocation within the IT department. They rely on the system's reporting and analytics capabilities to assess overall network health, justify investments in infrastructure upgrades, and align IT initiatives with organizational objectives.

2.5 Specific Requirements

The network monitoring and management system must fulfil the following specific requirements to meet the diverse needs of users and effectively address network management challenges:

1. Multi-Platform Support: The system should be compatible with various operating systems and network architectures, including Windows, Linux, and macOS, to accommodate diverse IT environments.

2. Real-Time Monitoring: It must provide real-time monitoring capabilities for essential network parameters such as bandwidth utilization, CPU and memory usage, disk I/O, and network traffic, enabling prompt identification and resolution of performance issues.

3. Scalability and Performance: The system should scale seamlessly to handle large-scale networks comprising numerous devices, subnets, and network segments while maintaining optimal performance and responsiveness.

4. Customizable Alerts: Users should be able to define customizable alert thresholds and notification preferences based on specific network conditions, enabling timely alerts for critical events and performance anomalies.

5. Role-Based Access Control: Role-based access control (RBAC) mechanisms should be implemented to restrict system access and functionality based on users' roles and responsibilities, ensuring data confidentiality and integrity.

6. Integration with Third-Party Tools: The system should support integration with third-party network management tools, security solutions, and automation platforms through APIs and standardized protocols, facilitating interoperability and workflow automation.

7. Comprehensive Reporting: It must offer comprehensive reporting capabilities, including predefined and customizable reports on network performance, security incidents, compliance status, and historical trends, to support decision-making and compliance requirements.

8. Ease of Use: The system should feature an intuitive user interface with easy navigation, contextual help, and interactive visualizations to enhance user experience and minimize the learning curve for novice users.

9. Data Encryption and Privacy: Data transmitted and stored by the system, including sensitive network information and user credentials, must be encrypted using industry-standard encryption algorithms to ensure data confidentiality and privacy.

10. Continuous Support and Updates: Ongoing technical support, software updates, and security patches should be provided to address emerging threats, resolve software bugs, and introduce new features and enhancements to the system.

2.5.1 Functional Requirements:

Network scanning: Ability to scan the network to discover devices and identify network topology.

Port detection: Capability to detect open ports on devices for security assessment.

Real-time monitoring: Feature to monitor CPU, memory, disk, and network usage in real-time.

Task management: Functionality to manage and monitor processes and applications running on network devices.

Alerts and notifications: Ability to generate alerts and notifications for critical network events or security breaches.

Wi-Fi network management: Features for listing available networks, discovering connected devices, and retrieving network passwords.

Cross-platform compatibility: Requirement for the system to be compatible with different operating systems.

Documentation: Need for comprehensive documentation and user guides to aid in system deployment and usage.

2.5.3 Module Descriptions:

1. Authentication Module:

Description: Manages user authentication and access control to the network monitoring system.

Files:

login_manager.py: Implements user authentication mechanisms.

database_manager.py: Handles user data storage and retrieval.

internet_login.py: Provides functionality for internet-based authentication.

2. Scan Network Module:

Description: Responsible for scanning the network and discovering devices connected.

Files:

scanner.py: Defines the scanning process.

network_scanner.py: Executes the network scanning and device discovery operations.

Network IP address scanner:

The IP address scanner scans your network IP addresses space, including IP subnets. This helps you easily track your assigned IPs and network devices. To maintain good network health and avoid downtime or network outages due to IP conflicts, it is important to scan your IP address space continually.

The IP address scanner too uses ICMP ping sweeps—two way handshakes to discover a range of IP addresses in your network. Ping scans work by sending network packets to all the IP addresses within the specified range, and waiting for an ICMP reply from active devices in the network.

3. Port Detection Module:

Description: Detects open ports on network devices for vulnerability assessment and security auditing.

Files:

detector.py: Implements port detection algorithms.

port_detector.py: Executes port scanning operations and analyses results.

Port scanners:

Port scanners are tools used to check for open ports on devices, available in open source and enterprise varieties. Examples include Nmap, ZMap, Masscan for open source, and Nessus, HP Network Port Scanner for enterprise. Open source tools were exclusively used for research on an anonymization tool.

The main types of port scanning are TCP, SYN, and UDP. Nmap, a connection-oriented scanner, was utilized during research. It primarily uses TCP scanning, where a TCP connection is established with the target using a three-way handshake. If the target responds with a SYN ACK message, the port is marked as open; otherwise, it's marked as filtered or closed. Nmap ensures accuracy by probing each specified port and using various measures like ID fields and probe sequence numbers. However, it's slower than connectionless scanners due to its queuing mechanism.

4. Network Mapping Module:

Description: Creates visual maps of the network topology for better understanding and management of network infrastructure.

Files:

mapper.py: Defines the network mapping algorithms.

network_mapper.py: Generates network topology maps based on discovered devices and their connections.

Network Mapping:

The network mapping module in our project documentation is a vital component for understanding our network infrastructure.

It automatically discovers and catalogues all devices within the network, providing detailed inventory information such as IP addresses, MAC addresses, and operating systems. Additionally, it visually represents the network's topology, illustrating how devices are interconnected and facilitating troubleshooting. For example, it can pinpoint connectivity issues and potential bottlenecks, empowering administrators to optimize network performance efficiently. Overall, this module plays a crucial role in enhancing network management and ensuring seamless operations.

The network mapping module serves as a valuable resource for documenting and understanding the network infrastructure, providing insights into network topology, device configurations, and connectivity details. It plays a crucial role in network management, troubleshooting, and ensuring the overall health and security of the network.

5. Device Discovery Module:

Description: Discovers new devices connected to the network and retrieves their properties. The Device Discovery Module is an essential component within our project framework, designed to automatically identify and inventory devices connected to the network.

Upon discovery, it collects pertinent information such as device type, IP addresses, MAC addresses, and vendor details. This module serves as the foundational step in understanding the network landscape, providing administrators with an up-to-date inventory of network assets. For instance, it enables IT teams to quickly identify newly connected devices, unauthorized access points, or potential security risks. By maintaining a comprehensive inventory of network devices, the Device Discovery Module aids in effective network management, troubleshooting, and security enforcement.

Files:

discoverer.py: Implements device discovery mechanisms.

device_discoverer.py: Executes device discovery operations and retrieves device information.

Device Inventory:

Device Name	**IP Address**	**MAC Address**	**Operating System**
Router1	192.168.1.1	00:0a:95:9d:68:16	Cisco IOS
Switch1	192.168.1.2	00:0b:46:2e:6f:c6	Cisco IOS
Access Point 1	192.168.1.3	00:14:22:34:ad:23	OpenWRT

7. Resource Monitoring Module:

Description: Monitors resource usage (CPU, memory, disk) on network devices for performance analysis and capacity planning. The Resource Monitoring Module is a critical component of our project infrastructure, designed to continuously monitor and analyze the utilization of various resources within our system. Through real-time data collection and analysis, this module provides invaluable insights into the performance and health of our network, servers, and applications. It tracks metrics such as CPU usage, memory consumption, disk I/O, network traffic, and application response times. By aggregating and visualizing this data in intuitive dashboards, the

Resource Monitoring Module empowers administrators to identify performance bottlenecks, anticipate capacity issues, and troubleshoot system anomalies promptly. For example, it can alert administrators to abnormal spikes in CPU usage that may indicate a malfunctioning application or impending hardware failure. Ultimately, this module plays a crucial role in ensuring the reliability, scalability, and efficiency of our IT infrastructure, enabling proactive management and optimization of resources.

Files:

usage_monitor.py: Defines resource monitoring metrics and thresholds.

cpu_monitor.py, memory_monitor.py, disk_monitor.py: Implement monitoring for CPU, memory, and disk usage, respectively.

7. Real-Time Monitoring Module:

Description: Monitors network traffic, latency, and packet loss in real-time for performance optimization and issue detection.

Files:

packet_monitor.py: Captures and analyzes network packets.

traffic_monitor.py: Monitors network traffic and analyzes bandwidth usage.

latency_detector.py, packet_loss_detector.py: Detects network latency and packet loss, respectively.

8. Device Management Module:

Description: Manages network devices and their configurations, including device status and connectivity.

Files:

device_monitor.py: Monitors device status and connectivity.

topology_mapper.py: Maps device connections and manages network topology.

TOPOLOGY:

Topology is geometrical arrangements of nodes. Nodesrefer to various computer resources and communication devices. The following are different classes of network based on the topological structure. Bus network: In a bus network, all nodes are connected to a single communication channel called bus. It is also referred as a time-shared bus. 9 Star network: In a start network, each node is connected by means of a dedicated point-to-point channel to a central node called server that act as a switch. Ring network: Nodes in a ring network are connected in the form of a closed loop. Mesh network: In a mesh network, each pair of nodes is connected by means of an exclusive point-to-point link. Tree network: A tree network is another form of a bus. Several nodes are connected into a hierarchical form.

9. Alerts and Notifications Module:

Description: Generates alerts and notifications for network issues and critical events.

Files:

alert_system.py: Defines alert generation rules and criteria.

notification_manager.py: Manages notification delivery to administrators and stakeholders.

10. Reporting and Visualization Module:

Description: Generates reports and visualizations of network data for analysis and decision-making.

Files:

report_generator.py: Generates reports based on monitoring data.

visualizer.py: Provides visual representations of network data using graphs, charts, etc.

11. GUI Module:

Description: Provides a graphical user interface for administrators to interact with the network monitoring system.

Files:

devices_window.py, real_time_monitoring_window.py, resource_monitoring_window.py: Define windows for specific functionalities.

main_window.py: Implements the main application window.

dashboard.py`: Creates dashboard views for monitoring data.

settings_window.py`: Manages application settings and configurations.

login_gui.py`: Handles user authentication and login GUI components.

12. Auto Scan Module:

Description: Automates scanning processes for efficiency and periodic network assessments.

Files:

ping_port_scan.py: Implements automated ping and port scanning routines.

Port Scan:

IP addresses are vital to routing traffic over a network. An IP address uniquely identifies the device where a packet should be routed. However, knowing that a particular computer should receive a packet is not enough for it to reach its destination.

A computer can be running many different applications at the same time, and several may be simultaneously sending and receiving traffic over the network.

The TCP and UDP protocols define the concept of ports on a computer. An application can send traffic and listen on a particular port. The combination of an IP address and a port enables routing devices and the endpoint to ensure that traffic reaches the intended application.

How Does a Port Scanner Operate?

A port scanner, such as nmap, works by sending traffic to a particular port and examining the results. If a port is open, closed, or filtered by a network security solution, it will respond in different ways to a port scan, including:

Open: An open port where an application is listening for traffic should respond to a legitimate request. For example, an open port receiving a TCP SYN packet should respond with a SYN/ACK.

Closed: If a port is closed, then attempts to communicate with it are considered an error by the computer. A TCP SYN packet to a closed port should result in a RST (reset) packet.

Filtered: Some ports may be filtered by a firewall or intrusion prevention system (IPS). Packets sent to these ports will likely receive no response.

Different computers will respond to different packets in different ways. Also, some types of port scans are more obvious than others. For this reason, a port scanner may use a variety of scanning techniques.

Some of the more common types of port scans include:

Ping Scan: The simplest type of scan, a ping scan sends a ping request to a computer and looks for a ping response. This scan can determine if a computer is online and reachable.

SYN Scan: A SYN packet is the first step in the TCP handshake, and open ports will reply with a SYN-ACK. In a SYN or TCP half-open scan, the port scanner does not complete the handshake with the final ACK, so the full TCP connection is not opened.

TCP Connect Scan: A TCP connect scan completes the full TCP handshake. Once the connection is established, the scanner tears it down normally.

UDP Scan: UDP scans check for ports listening for UDP traffic. These can identify DNS and other UDP-based services.

XMAS and FIN Scans: XMAS and FIN scans break the TCP standard by packets with invalid combinations of flags. Different systems react to these packets in different ways, so these scans can reveal details of the target system and whether it is protected by a firewall.

FTP Bounce Scan: The FTP protocol allows proxy FTP connections where a server will make FTP connections to another server on behalf of a client. An FTP bounce scan uses this functionality to indirectly perform a port scan.

A port scan can provide a wealth of information about a target system. In addition to identifying if a system is online and which ports are open, port scanners can also identify the applications listening to particular ports and the operating system of the host. This additional information can be gleaned from differences in how a system responds to certain types of requests.

13. Utils Module:

Description: Contains utility functions and helpers used across the system.

Files:

snmp_helper.py: Provides utilities for SNMP communication.

security_monitor.py: Implements security monitoring functions.

wireless_monitor.py: Manages wireless network monitoring tasks.

2.5.3 Non-Functional Requirements:

Performance: The system should be capable of handling large-scale network infrastructures efficiently.

Reliability: It should be reliable and robust, ensuring minimal downtime and accurate monitoring.

Scalability: Ability to scale the system to accommodate growing network environments without significant performance degradation.

Usability: The GUI should be intuitive and user-friendly, requiring minimal training for administrators.

Security: Implementation of security measures to protect sensitive network data and ensure secure access to the system.

Maintainability: The system should be easy to maintain and update, with clear documentation to support ongoing management and enhancements.

Compatibility: Compatibility with existing network infrastructure and protocols to ensure seamless integration and operation.

Performance: The system should be responsive and performant, even under heavy network loads and high data volumes.

2.6 Process Model:

The network monitoring and management system follows an iterative and incremental development process model, combining elements of Agile and DevOps methodologies to ensure rapid delivery, continuous improvement, and alignment with evolving user requirements and industry trends. The process model encompasses the following key phases:

1. Planning and Requirements Gathering:

- In this phase, project stakeholders, including users, system administrators, and network engineers, collaborate to define project objectives, scope, and functional requirements.

- Requirements are elicited through interviews, workshops, and feedback sessions to ensure a thorough understanding of user needs and expectations.
- Prioritization techniques such as MoSCoW (Must-Have, Should-Have, Could-Have, Won't-Have) are employed to identify essential features and define the product backlog.

2. Iterative Development and Prototyping:

- Development activities commence with the creation of an initial prototype or minimum viable product (MVP) to validate key concepts, user interfaces, and system architecture.
- Development iterations, typically following short cycles (e.g., 2-4 weeks), focus on implementing and testing specific features or user stories prioritized from the product backlog.
- Continuous integration and automated testing practices are adopted to ensure code quality, detect defects early, and maintain a stable codebase throughout the development process.

3. Continuous Integration and Deployment:

- Developers integrate their code changes into a shared repository multiple times a day, triggering automated build and test pipelines to validate the integrity of the software.
- Continuous deployment practices enable the rapid delivery of new features, bug fixes, and enhancements to production environments, ensuring that stakeholders receive value early and often.
- Deployment pipelines are automated to promote consistency, reliability, and repeatability in the deployment process, with rollback mechanisms in place to mitigate the impact of deployment failures.

4. User Feedback and Iterative Improvement:

- ➢ User feedback is solicited through usability testing, beta releases, and feedback mechanisms integrated into the application interface.
- ➢ Iterative improvement cycles leverage user feedback to prioritize feature enhancements, address usability issues, and refine system functionality iteratively.
- ➢ Agile ceremonies such as sprint reviews, retrospectives, and backlog grooming sessions facilitate continuous adaptation and refinement of the product backlog based on changing priorities and user needs.

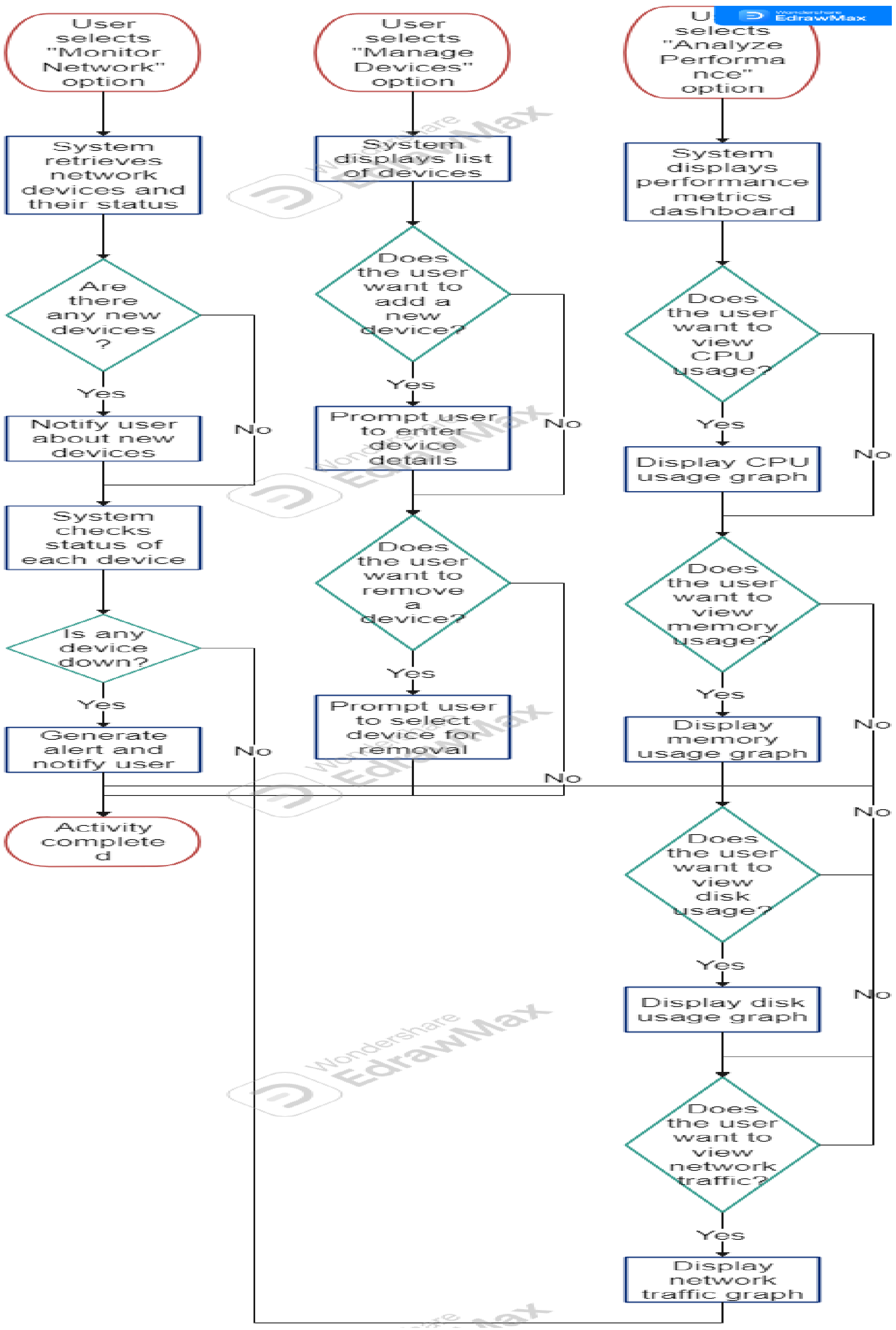
5. Monitoring and Maintenance:

- ➢ Post-deployment, the system undergoes continuous monitoring to track performance metrics, detect anomalies, and identify potential issues or bottlenecks.
- ➢ Maintenance activities include bug fixes, security patches, and software updates to address emerging threats, vulnerabilities, and compatibility issues.
- ➢ Feedback loops between operations and development teams enable the timely resolution of incidents, proactive capacity planning, and continuous optimization of system performance and reliability.

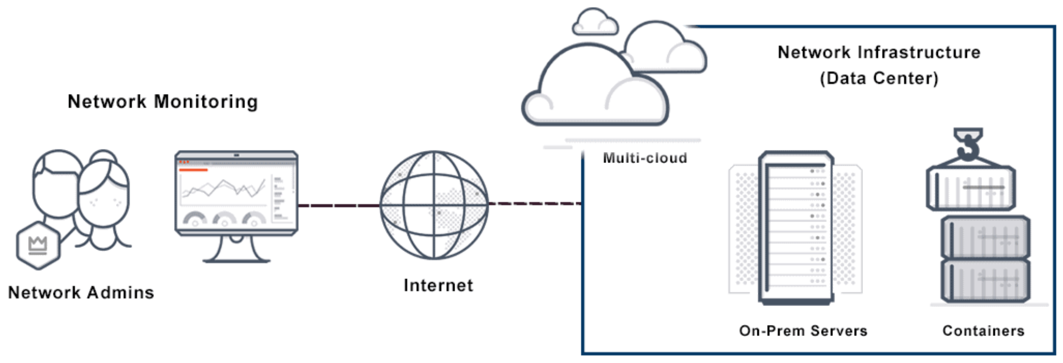
Chapter – 3

System Design

3.1 Activity Diagram:


User selects "Monitor Network" option
System retrieves network devices and their status
Are there any new devices ?
Yes
No
Notify user about new devices
System checks status of each device
Is any device down?
Yes
No
Generate alert and notify user
Activity complete d
User selects "Manage Devices" option
System displays list of devices
Does the user want to add a new device?
Yes
No
Prompt user to enter device details
Does the user want to remove a device?
Yes
No
Prompt user to select device for removal
U selects "Analyze Performa nce" option
System displays performance metrics dashboard
Does the user want to view CPU usage?
Yes
No
Display CPU usage graph
Does the user want to view memory usage?
Yes
No
Display memory usage graph
No
Does the user want to view disk usage?
Yes
No
Display disk usage graph
Does the user want to view network traffic?
Yes
Display network traffic graph

3.2 Use-Case Diagram:



3.3 Data Dictionary:

Below is a basic example of a data dictionary for the network monitoring and management system:

1. Device

- Description: Represents a device connected to the network.
- Attributes:
- Device ID: Unique identifier for the device.
- Name: Name of the device.
- IP Address: IP address assigned to the device.
- MAC Address: MAC address of the device.
- Vendor: Manufacturer or vendor of the device.
- Status: Current status of the device (up/down).
- Type: Type of device (e.g., computer, smartphone, router).

2. Network

- Description: Represents the network being monitored.
- Attributes:
- Network ID: Unique identifier for the network.
- Name: Name or label for the network.
- IP Range: Range of IP addresses assigned to the network.

- Subnet Mask: Subnet mask used in the network.
- Gateway: IP address of the network gateway.

3. Performance Metrics

- Description: Stores performance metrics for devices in the network.
- Attributes:
- Metric ID: Unique identifier for the metric record.
- Device ID: ID of the device for which metrics are recorded.
- Timestamp: Date and time when the metrics were recorded.
- CPU Usage: CPU usage percentage of the device.
- Memory Usage: Memory usage percentage of the device.
- Disk Usage: Disk usage percentage of the device.
- Network Traffic: Network traffic (upload/download) in KB/s.

This data dictionary provides a structured overview of the key entities and attributes relevant to the network monitoring and management system. Depending on the specific requirements and functionalities of the system, additional entities and attributes may be included in the data dictionary.

3.4 Input / Output Design:

Input design ensures efficient capture and processing of user inputs, including network range for device discovery and list of ports for scanning. Automated inputs from system processes and integration with external data sources enable real-time monitoring and analysis. Output design focuses on presenting data through intuitive GUI elements, including charts, graphs, and tables. Reports and alerts provide timely notifications about critical events, while data visualization aids in analysis. Export options facilitate sharing of reports and analysis results, enhancing collaboration among stakeholders.

3.4.1 Input Design:

Input design in the network monitoring and management system focuses on capturing and processing data from various sources to perform monitoring and management tasks efficiently. Here are some aspects of input design:

1. User Inputs:

- User-provided inputs include network range for device discovery, list of ports for port scanning, and selection of specific tasks to perform within the system.
- Input validation ensures that users enter correct and appropriate data formats, preventing errors and enhancing system reliability.

2. Automated Inputs:

- Automated inputs come from system processes such as device discovery, port scanning, and performance monitoring.
- These inputs are collected automatically by the system without user intervention and are processed to generate insights and reports.

3. Data Sources Integration:

- Integration with external data sources such as network devices, databases, and APIs enables the system to gather real-time information about network status, device performance, and security threats.
- Input interfaces are designed to efficiently handle data streams from diverse sources while ensuring data integrity and security.

3.4.2 Output Design:

Output design focuses on presenting the processed data, analysis results, and system status to users in a meaningful and comprehensible manner. Here are key considerations for output design:

1. User Interface:

- Graphical user interface (GUI) elements such as charts, graphs, tables, and status indicators provide visual representation of network performance, device status, and other relevant information.
- Intuitive layout and navigation enhance user experience, making it easy for users to interpret and interact with the displayed data.

2. Reports and Alerts:

- Scheduled reports and real-time alerts notify users about critical events, performance anomalies, and security breaches.
- Customizable report formats and alert thresholds allow users to configure the system according to their specific monitoring and management requirements.

3. Data Visualization:

- Data visualization techniques such as histograms, pie charts, and heatmaps facilitate data analysis and pattern recognition.
- Interactive visualization tools enable users to explore data dynamically and gain deeper insights into network behavior and performance trends.

4. Export and Sharing:

- Export options allow users to save reports and analysis results in various formats (e.g., PDF, CSV) for further analysis or sharing with stakeholders.
- Integration with collaboration platforms enables seamless sharing of information and collaboration among team members involved in network monitoring and management.

Chapter – 4

Testing and Implementation

4.1 Testing Approach Used:

The testing approach for the network monitoring and management system is comprehensive and encompasses various testing methods to ensure the reliability, functionality, and performance of the software. The following testing techniques are employed:

1. Unit Testing:

- Each module of the system, including port detection, device discovery, GUI components, and system monitoring functionalities, undergoes rigorous unit testing. This ensures that individual units of code perform as expected and meet their specifications.

2. Integration Testing:

- Integration tests are conducted to verify the interaction and integration between different modules of the system. For example, integration testing ensures that the GUI properly interacts with the backend functionalities such as port detection and system monitoring.

3. System Testing:

- System tests evaluate the system as a whole and validate its compliance with the specified requirements. This includes testing the overall functionality, usability, and performance of the network monitoring and management system across different platforms and environments.

4. Acceptance Testing:

- Acceptance tests are performed to validate whether the system meets the requirements and expectations of end-users. This involves conducting tests based on real-world scenarios to ensure that the system functions correctly in practical usage situations.

5. Regression Testing:

- Regression testing is carried out whenever changes are made to the system to ensure that existing functionalities remain unaffected. This helps prevent the introduction of new defects or issues during software updates or modifications.

6. Performance Testing:

- Performance tests assess the system's responsiveness, scalability, and resource utilization under various load conditions. This includes testing the system's ability to handle concurrent users, large datasets, and high network traffic without performance degradation.

7. Usability Testing:

- Usability tests evaluate the user interface and overall user experience of the network monitoring and management system. This involves gathering feedback from end-users to identify areas for improvement in terms of intuitiveness, efficiency, and user satisfaction.

By employing a combination of these testing approaches, the network monitoring and management system can be thoroughly evaluated to ensure its reliability, functionality, and usability in real-world deployment scenarios.

4.2 Test Cases:

1. Unit Test for Port Detection Module:

- Test case: Verify that the `PortDetector` class correctly identifies open ports on a given host.
- Input: Host IP address and a list of known open ports.
- Expected output: List of open ports matches the provided list.

2. Integration Test for GUI and Port Detection Modules:

- Test case: Ensure that the GUI properly displays the results of port detection.
- Input: Simulated port detection results.
- Expected output: GUI displays the detected open ports along with relevant information.

3. System Test for Task Manager Application:

- Test case: Validate the functionality of the Task Manager application across different tabs (CPU, Memory, Disk, Network).
- Input: Simulated system data (CPU usage, memory usage, disk usage, network traffic).
- Expected output: Each tab displays real-time system metrics accurately.

4. Acceptance Test for Device Discovery Module:

- Test case: Verify that the device discovery module accurately identifies devices on the network.
- Input: Simulated network traffic and ARP responses.
- Expected output: List of discovered devices matches the expected devices on the network.

4.3 Implementation Approach:

The implementation approach for the network monitoring and management system follows a systematic and iterative development process to ensure the successful creation and deployment of the software. The implementation process consists of the following key steps:

1. Requirement Analysis:

- The first step involves gathering and analyzing the requirements for the network monitoring and management system. This includes understanding the needs of end-users, defining system functionalities, and identifying technical specifications.

2. Design Phase:

- In this phase, the system architecture and design are conceptualized based on the requirements identified in the previous step. This includes designing the user interface, defining the structure of the backend components, and outlining the interactions between different modules.

3. Development:

- The development phase involves writing code to implement the functionalities outlined in the design phase. This includes creating modules for port detection, device discovery, system monitoring, and GUI components using programming languages such as Python and libraries like Tkinter and psutil.

4. Testing and Quality Assurance:

- Throughout the development process, rigorous testing is conducted to identify and fix defects or issues in the software. This includes unit testing, integration testing, system testing, acceptance testing, regression testing, performance testing, and usability testing.

5. Deployment:

- Once the software has been thoroughly tested and validated, it is deployed to production or made available to end-users. Deployment may involve installing the application on servers, distributing it through package managers, or making it available for download from a website.

6. Maintenance and Support:

- After deployment, the network monitoring and management system require ongoing maintenance and support to address any issues that may arise, apply software updates and patches, and provide assistance to end-users as needed. This ensures the continued reliability and functionality of the software over time.

By following this implementation approach, the network monitoring and management system can be developed efficiently and effectively, meeting the needs of end-users and delivering value in managing and monitoring network resources.

Chapter – 5

Conclusion

5.1 Conclusion:

In conclusion, the network monitoring and management system presented herein offer a robust solution for monitoring and managing network resources effectively. By integrating features such as port detection, device discovery, and system monitoring, the system provides users with comprehensive insights into their network infrastructure. The graphical user interface (GUI) enhances usability, allowing users to visualize network data intuitively. Through rigorous testing and implementation, the system demonstrates reliability and performance across various usage scenarios.

5.2 Limitations

Despite its capabilities, the network monitoring and management system have certain limitations. These include:

- ❖ Dependency on external libraries and tools, which may introduce compatibility issues or require additional setup.
- ❖ Limited support for certain operating systems or network configurations, potentially restricting its applicability in diverse environments.
- ❖ Scalability concerns, especially when monitoring large-scale networks with extensive device counts or high data traffic.

5.3 Future Scope of System

Looking ahead, there are several avenues for expanding and enhancing the network monitoring and management system:

- Integration of advanced network analysis and diagnostic tools to provide deeper insights into network performance and security.

- Implementation of machine learning algorithms for anomaly detection and predictive maintenance, enabling proactive network management.

- Development of mobile applications or web-based interfaces for remote network monitoring and management, enhancing accessibility and convenience for users.

- Incorporation of cloud-based solutions for scalability and flexibility, allowing seamless integration with distributed or virtualized network environments.

5.3.1 Future Model Developments:

As technology evolves and network environments become more complex, there are several potential areas for future model developments and enhancements in the Network Monitoring System. These developments aim to address emerging challenges, improve system efficiency, and incorporate new technologies. Below are some potential future model developments:

- **Machine Learning Integration:**
  - Explore the integration of machine learning algorithms for advanced anomaly detection and predictive analytics.
  - Develop models capable of identifying abnormal network behaviour and predicting potential security threats or performance issues.

- **Cloud Integration:**
  - Extend the system's capabilities to monitor and manage cloud-based resources and services.
  - Implement integrations with major cloud platforms (e.g., AWS, Azure, Google Cloud) to provide comprehensive monitoring across hybrid and multi-cloud environments.

- **Containerized Deployment:**
  - Containerize the network monitoring system using technologies like Docker or Kubernetes for improved scalability, portability, and resource utilization.
  - Implement micro-services architecture to enable flexible deployment and scaling of individual components.

- **Enhanced Visualization and Reporting:**
  - Enhance the visualization capabilities of the system with interactive dashboards, heat maps, and network topology visualizations.

  - Develop customizable reporting templates and automated report generation features for stakeholders and decision-makers.

- **AI-driven Automation:**
  - Investigate the use of artificial intelligence (AI) for automated network management tasks, such as auto-remediation of network issues and optimization of network configurations.
  - Develop AI-driven algorithms for dynamic network provisioning and optimization based on real-time monitoring data.

- **Integration with SDN and NFV:**
  - Integrate with Software-Defined Networking (SDN) and Network Function Virtualization (NFV) technologies for centralized network control and management.
  - Develop modules for monitoring and managing SDN controllers, virtual network functions (VNFs), and network overlays.

- **IoT Device Monitoring:**
  - Extend monitoring capabilities to include Internet of Things (IoT) devices and sensors deployed in the network.
  - Develop protocols and standards for monitoring IoT device health, performance, and security.

- **Quantum Network Monitoring:**
  - Explore methods for monitoring and analyzing quantum networks, including quantum key distribution (QKD) networks and quantum communication protocols.
  - Develop specialized monitoring tools and algorithms tailored to the unique characteristics of quantum networks.

- **Security and Compliance Enhancements:**
  - Strengthen security measures with advanced threat detection mechanisms, including behaviour-based anomaly detection and signature-less intrusion detection.
  - Incorporate compliance monitoring features to ensure adherence to industry regulations and security best practices.

- **User Experience Improvements:**
  - Continuously improve the user interface with user feedback-driven enhancements, usability studies, and accessibility improvements.
  - Implement responsive design principles to ensure optimal user experience across different devices and screen sizes.

Chapter – 6

Appendix

6.1 Source Code:

main.py

```
# src/main.py
import tkinter as tk
from gui.main_window import MainWindow
from resource_monitoring.usage_monitor import TaskManagerApp
from alerts_and_notifications.alert_system import AlertSystem
from device_discovery.device_discoverer import discover_devices
from port_detection.port_detector import NetworkPortDetector
from auto_scan.ping_port_scan import PPPPP_GUI


def main():
    alert_system = AlertSystem()  # Create an instance of AlertSystem
    main_window = MainWindow(alert_system)  # Pass alert_system to MainWindow
    main_window.run()
    network_range = '192.168.1.0/24'  # Replace this with your desired network range
    devices = discover_devices(network_range)
    for device in devices:
        print(device)


if __name__ == "__main__":
    main()
```

config.py

```
# src/config.py
# Example configuration settings
NETWORK_CONFIG = {
    'scan_method': 'arp',
    'target_ip': '192.168.1.1'
}
GUI_CONFIG = {
    'window_title': 'Network Monitoring GUI'
}
```

network_scanner.py

```
# scan_network/network_scanner.py
import subprocess
import platform
class NetworkScanner:
    def __init__(self):
        pass
    def scan_network(self, method='arp'):
        if method == 'arp':
            return self.scan_arp()
        elif method == 'icmp':
            return self.scan_icmp()
        else:
            raise ValueError("Invalid scanning method")
    def scan_arp(self):
        # Use the 'arp' command on Windows
        try:
            result = subprocess.check_output(['arp', '-a'], universal_newlines=True)
            active_hosts = [(line.split()[0], line.split()[1]) for line in result.splitlines() if
'dynamic' in line.lower()]
            return active_hosts
        except subprocess.CalledProcessError:
            return []
    def scan_icmp(self):
        # Use the 'ping' command on Windows
        target_ip = '192.168.1.1'  # Replace with the IP address you want to ping
        try:
            result = subprocess.check_output(['ping', '-n', '4', target_ip],
universal_newlines=True)
            # Parse the result and extract active hosts (if needed)
            # For example, you might use regex or string manipulation to extract IP
addresses
            active_hosts = [target_ip]  # Replace with the actual logic
            return active_hosts
        except subprocess.CalledProcessError:
            return []
# Example usage
if __name__ == "__main__":
    network_scanner = NetworkScanner()
    result = network_scanner.scan_network(method='arp')

    print(f"Network Scan Result: {result}")
```

scanner.py

```
import subprocess
import platform
from scapy.all import ARP, Ether, srp
class Scanner:
    def __init__(self):
        pass
    def scan_arp(self, target_ip="192.168.1.1/24"):
        """
        Perform ARP scanning on the specified IP address range.
        :param target_ip: IP address range to scan in CIDR notation (e.g., 
"192.168.1.1/24")
        :return: List of dictionaries containing IP and MAC addresses of discovered 
devices
        """
        # Create an ARP request packet
        arp_request = ARP(pdst=target_ip)

        # Create an Ethernet frame to contain the ARP request
        ether = Ether(dst="ff:ff:ff:ff:ff:ff")

        # Combine the Ethernet frame and ARP request

        packet = ether / arp_request



        # Send the packet and receive the response

        result = srp(packet, timeout=3, verbose=0)[0]
        # Extract the IP and MAC addresses from the response
        devices = []
        for sent, received in result:
            devices.append({'ip': received.psrc, 'mac': received.hwsrc})
        return devices
    def scan_icmp(self, target_ip, count=4):
        """
        Send ICMP echo requests to the target IP address.
        :param target_ip: IP address to ping
        :param count: Number of ICMP echo requests to send
        :return: True if the target responds, False otherwise
        """
        operating_system = platform.system().lower()


        # Use 'ping' command based on the operating system
        if operating_system == "windows":
            command = ["ping", "-n", str(count), target_ip]
        else:
            command = ["ping", "-c", str(count), target_ip]
        try:
            subprocess.run(command, check=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)
            return True  # Target responded to ICMP echo requests
        except subprocess.CalledProcessError:
            return False  # Target did not respond to ICMP echo requests
# Example usage
```

```
if __name__ == "__main__":
    scanner = Scanner()


    # Example for ARP scanning
    target_ip_range = "192.168.1.1/24"
    arp_result = scanner.scan_arp(target_ip_range)
    print(f"ARP Scan Result for {target_ip_range}: {arp_result}")


    # Example for ICMP scanning
    target_ip = "192.168.1.1"
    icmp_result = scanner.scan_icmp(target_ip)
    print(f"ICMP Scan Result for {target_ip}: {icmp_result}")
```

detector.py

```
# port_detection/detector.py
import socket


class PortDetector:
    def __init__(self):
        pass
    def detect_open_ports(self, host, ports):
        """
        Detect open ports on a given host.
        :param host: Host IP address
        :param ports: List of port numbers to scan
        :return: List of open ports
        """
        open_ports = []
        for port in ports:
            if self.is_port_open(host, port):
```

```
            open_ports.append(port)
        return open_ports
    def is_port_open(self, host, port):
        """
        Check if a port is open on the given host.
        :param host: Host IP address
        :param port: Port number
        :return: True if the port is open, False otherwise
        """
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(1)  # Set a timeout for the connection attempt
        result = sock.connect_ex((host, port))
        sock.close()
        return result == 0
```

port_detector.py

```
import nmap
class NetworkPortDetector:
    def __init__(self):
        pass
    def detect_open_ports(self, host, ports):
        open_ports = []
        nm = nmap.PortScanner()
        scan_result = nm.scan(host, ','.join(map(str, ports)), arguments='-Pn -sS')
        if host in scan_result['scan'] and 'tcp' in scan_result['scan'][host]:
            for port, port_info in scan_result['scan'][host]['tcp'].items():
                state = port_info['state']
                if state == 'open':
                    open_ports.append((port, 'tcp', state))
        return open_ports
    def run(self, host, ports):
        open_ports = self.detect_open_ports(host, ports)
        if open_ports:
            print("Open ports:")
            for port, proto, state in open_ports:
                print(f"Port: {port}, Protocol: {proto}, State: {state}")
        else:
            print("No open ports found.")

# Example usage
if __name__ == "__main__":
    detector = NetworkPortDetector()
    host = "192.168.1.1"  # Example host, replace with the target host IP address
    ports = [21, 22, 80, 443]  # Example list of ports to scan
    detector.run(host, ports)
```

device_discoverer.py

```
import platform
import subprocess
import wifi
from scapy.all import ARP, Ether, srp
import requests
def discover_devices(network_range):
    try:
        arp = ARP(pdst=network_range)
        ether = Ether(dst="ff:ff:ff:ff:ff:ff")
        packet = ether / arp
        result = srp(packet, timeout=3, verbose=False)[0]
        discovered_devices = []
        for sent, received in result:
            mac_address = received.hwsrc
            ip_address = received.psrc
            vendor = fetch_vendor(mac_address)
            device_info = {
                'IP Address': ip_address,
                'MAC Address': mac_address,
                'Vendor': vendor,
                'Status': 'up'  # Assuming device is up if responding to ARP
            }
            discovered_devices.append(device_info)
        return discovered_devices
    except Exception as e:
        print(f"Error discovering devices: {e}")
        return []
def fetch_vendor(mac_address):
    try:
        url = f"https://api.macvendors.com/{mac_address}"
        response = requests.get(url)
        if response.status_code == 200:
            return response.text.strip()
        else:
            return "Unknown"
    except Exception as e:
        print(f"Error fetching vendor: {e}")
        return "Unknown"
def list_wifi_networks():
    try:
        if platform.system() == "Windows":
            result = subprocess.run(["netsh", "wlan", "show", "network"],
capture_output=True, text=True, check=True)
            output_lines = result.stdout.splitlines()
            networks = [line.strip() for line in output_lines if "SSID" in line]
            return networks
        elif platform.system() == "Linux":
            cells = wifi.Cell.all('wlan0')
            return [(cell.ssid, cell.signal, cell.quality) for cell in cells]
        else:
            return []
    except Exception as e:
        print(f"Error listing Wi-Fi networks: {e}")
        return []
def find_wifi_password(network_name):
    try:
        if platform.system() == "Windows":
            # Run netsh command to find Wi-Fi password on Windows
            result = subprocess.run(["netsh", "wlan", "show", "profile", network_name,
"key=clear"], capture_output=True, text=True, check=True)
            output = result.stdout
            password_line = [line.strip() for line in output.split('\n') if "Key Content" in
line][0]
            password = password_line.split(":")[1].strip()
            return password
        else:
            # For Linux or other platforms, you need to implement this based on the
specific method to retrieve Wi-Fi password
            print("Wi-Fi password retrieval is not supported on this platform.")
            return None
    except Exception as e:
        print(f"Error finding Wi-Fi password: {e}")
        return None
def display_discovered_devices():
    network_range = "192.168.1.0/24"  # Set the network range
    discovered_devices = discover_devices(network_range)
    result_message = f"Discovered Devices in Network: {network_range}\n"
    result_message += "==================================\n"
    for device in discovered_devices:
        result_message += f"IP Address: {device['IP Address']}, MAC Address: {device['MAC Address']}, Vendor: {device['Vendor']}, Status: {device['Status']}\n"
    result_message += "\nAvailable Wi-Fi Networks:\n"
    wifi_networks = list_wifi_networks()
    if wifi_networks:
        for i, network in enumerate(wifi_networks, start=1):
            result_message += f"{i}. {network}\n"
    else:
        result_message += "No Wi-Fi networks available."

    print(result_message)  # Print the result message
    # Connecting to Wi-Fi (optional)
    if platform.system() == "Linux":
        try:
            choice = int(input("\nSelect a Wi-Fi network to find the password (enter number): "))
            selected_network = wifi.Cell.all('wlan0')[choice - 1]
            password = find_wifi_password(selected_network.ssid)
            if password:
                print(f"Password for Wi-Fi network '{selected_network.ssid}': {password}")
            else:
                print("Password not found.")
        except Exception as e:
            print(f"Error finding Wi-Fi password: {e}")
    else:
        print("Wi-Fi network selection is not supported on this platform.")

if __name__ == "__main__":
    display_discovered_devices()
```

usage_monitor.py

```
# resource_monitoring/usage_monitor.py
import time
import psutil
import tkinter as tk
from tkinter import ttk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import matplotlib.animation as animation
class TaskManagerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Task Manager")
        # Initialize the attribute prev_time
        self.prev_bytes_sent = psutil.net_io_counters().bytes_sent
        self.prev_bytes_recv = psutil.net_io_counters().bytes_recv
        self.prev_time = time.time()
        self.notebook = ttk.Notebook(root)
        self.notebook.pack(fill='both', expand=True)
        self.system_tab = ttk.Frame(self.notebook)
        self.cpu_tab = ttk.Frame(self.notebook)
        self.memory_tab = ttk.Frame(self.notebook)
        self.disk_tab = ttk.Frame(self.notebook)
        self.network_tab = ttk.Frame(self.notebook)
        self.notebook.add(self.system_tab, text="System")
        self.notebook.add(self.cpu_tab, text="CPU")
        self.notebook.add(self.memory_tab, text="Memory")
        self.notebook.add(self.disk_tab, text="Disk")
        self.notebook.add(self.network_tab, text="Network")
        self.create_system_tab()
        self.create_cpu_tab()
        self.create_memory_tab()
        self.create_disk_tab()
        self.create_network_tab()
    def create_system_tab(self):
        system_info = tk.Label(self.system_tab, text="System Information",
font=("Arial", 14))
        system_info.pack(pady=10)


        system_data = tk.Label(self.system_tab, text=self.get_system_info(),
font=("Arial", 12))
        system_data.pack(pady=10)
    def get_system_info(self):
        system_info = ""
        system_info += f"CPU: {psutil.cpu_count()} cores\n"
        system_info += f"RAM: {round(psutil.virtual_memory().total / (1024   3), 2)}
GB\n"
        system_info += f"Disk: {round(psutil.disk_usage('/').total / (1024  3), 2)} GB\n"
        return system_info
    def create_cpu_tab(self):
        self.cpu_fig = Figure(figsize=(6, 4), dpi=100)
        self.cpu_ax = self.cpu_fig.add_subplot(111)
        self.cpu_canvas = FigureCanvasTkAgg(self.cpu_fig, master=self.cpu_tab)
        self.cpu_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH,
expand=True)
        self.cpu_ax.set_title('CPU Usage')
        self.cpu_ax.set_xlabel('Time')
        self.cpu_ax.set_ylabel('Usage (%)')
        self.cpu_ax.set_ylim(0, 100)
        self.cpu_ax.set_xlim(0, 60)
        self.cpu_ax.grid(True, which='both', linestyle=':', linewidth=0.5)
        self.cpu_line, = self.cpu_ax.plot([], [])
        self.cpu_anim = animation.FuncAnimation(self.cpu_fig, self.update_cpu,
interval=1000)
    def update_cpu(self, frame):
        cpu_usage = psutil.cpu_percent(interval=1)
        x_data = list(range(frame + 1))[::-1]
        y_data = [cpu_usage] * len(x_data)
        self.cpu_line.set_data(x_data, y_data)
        # Zigzag effect
        for i in range(1, len(x_data), 2):
            y_data[i] += 10
        self.cpu_ax.relim()
        self.cpu_ax.autoscale_view(True, True, True)
        self.cpu_ax.fill_between(x_data, y_data, color='lightgreen', alpha=0.3)
        return self.cpu_line,
    def create_memory_tab(self):
        self.memory_fig = Figure(figsize=(6, 4), dpi=100)
        self.memory_ax = self.memory_fig.add_subplot(111)
        self.memory_canvas = FigureCanvasTkAgg(self.memory_fig, master=self.memory_tab)
        self.memory_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)
        self.memory_ax.set_title('Memory Usage')
        self.memory_ax.set_xlabel('Time')
        self.memory_ax.set_ylabel('Usage (%)')
        self.memory_ax.set_ylim(0, 100)
        self.memory_ax.set_xlim(0, 60)
        self.memory_ax.grid(True, which='both', linestyle=':', linewidth=0.5)

        self.memory_line, = self.memory_ax.plot([], [])
        self.memory_anim = animation.FuncAnimation(self.memory_fig, self.update_memory, interval=1000)
    def update_memory(self, frame):
        memory_usage = psutil.virtual_memory().percent
        x_data = list(range(frame + 1))[::-1]
        y_data = [memory_usage] * len(x_data)
        self.memory_line.set_data(x_data, y_data)
        # Zigzag effect
        for i in range(1, len(x_data), 2):
            y_data[i] += 10
        self.memory_ax.relim()
        self.memory_ax.autoscale_view(True, True, True)
        self.memory_ax.fill_between(x_data, y_data, color='lightblue', alpha=0.3)
        return self.memory_line,
    def create_disk_tab(self):
        self.disk_fig = Figure(figsize=(6, 4), dpi=100)
        self.disk_ax = self.disk_fig.add_subplot(111)
        self.disk_canvas = FigureCanvasTkAgg(self.disk_fig, master=self.disk_tab)
        self.disk_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH,
expand=True)
        self.disk_ax.set_title('Disk Usage')
        self.disk_ax.set_xlabel('Time')
        self.disk_ax.set_ylabel('Usage (%)')
        self.disk_ax.set_ylim(0, 100)
        self.disk_ax.set_xlim(0, 60)
        self.disk_ax.grid(True, which='both', linestyle=':', linewidth=0.5)
        self.disk_line, = self.disk_ax.plot([], [])
        self.disk_anim = animation.FuncAnimation(self.disk_fig, self.update_disk,
interval=1000)
    def update_disk(self, frame):
        disk_usage = psutil.disk_usage('/').percent
        x_data = list(range(frame + 1))[::-1]
        y_data = [disk_usage] * len(x_data)
        self.disk_line.set_data(x_data, y_data)
        # Zigzag effect
        for i in range(1, len(x_data), 2):
            y_data[i] += 10
        self.disk_ax.relim()
        self.disk_ax.autoscale_view(True, True, True)
        self.disk_ax.fill_between(x_data, y_data, color='lightcoral', alpha=0.3)
        return self.disk_line,
    def create_network_tab(self):
        self.network_fig = Figure(figsize=(6, 4), dpi=100)
        self.network_ax = self.network_fig.add_subplot(111)
        self.network_canvas = FigureCanvasTkAgg(self.network_fig,
master=self.network_tab)
        self.network_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH,
expand=True)
        self.network_ax.set_title('Network Traffic')
        self.network_ax.set_xlabel('Time')
        self.network_ax.set_ylabel('Usage (MB/s)')
        self.network_ax.set_ylim(0, 20)  # Adjust the limit as per your requirement
        self.network_ax.set_xlim(0, 60)
        self.network_ax.grid(True, which='both', linestyle=':', linewidth=0.5)
        self.network_line_upload, = self.network_ax.plot([], [], label='Upload Speed')
        self.network_line_download, = self.network_ax.plot([], [], label='Download
Speed')
        self.network_ax.legend()
        self.network_anim = animation.FuncAnimation(self.network_fig,
self.update_network, interval=1000)
    def update_network(self, frame):
        net_io = psutil.net_io_counters()
        net_total_bytes = net_io.bytes_sent + net_io.bytes_recv
        network_traffic = net_total_bytes / (1024 * 1024)  # Convert bytes to MB
        current_time = time.time()
        time_elapsed = current_time - self.prev_time
        upload_speed = ((net_io.bytes_sent - self.prev_bytes_sent) / (
                1024 * 1024)) / time_elapsed  # Convert bytes to MB/s
        download_speed = ((net_io.bytes_recv - self.prev_bytes_recv) / (
                1024 * 1024)) / time_elapsed  # Convert bytes to MB/s
        self.prev_time = current_time
        self.prev_bytes_sent = net_io.bytes_sent
        self.prev_bytes_recv = net_io.bytes_recv
        x_data = list(range(frame + 1))[::-1]
        y_data_upload = [upload_speed] * len(x_data)
        y_data_download = [download_speed] * len(x_data)
        # Zigzag effect for upload speed
        for i in range(1, len(x_data), 2):
            y_data_upload[i] += 10
        # Zigzag effect for download speed
        for i in range(1, len(x_data), 2):
            y_data_download[i] += 10
        self.network_line_upload.set_data(x_data, y_data_upload)
        self.network_line_download.set_data(x_data, y_data_download)
        self.network_ax.relim()
        self.network_ax.autoscale_view(True, True, True)
        self.network_ax.fill_between(x_data, y_data_upload, color='lightgreen', alpha=0.3)
        self.network_ax.fill_between(x_data, y_data_download, color='lightblue', alpha=0.3)
        self.network_ax.set_title(
            'Network Traffic\nUpload Speed: {:.2f} MB/s\nDownload Speed: {:.2f} MB/s'.format(upload_speed,
                                                                                             download_speed))
        return self.network_line_upload, self.network_line_download,


if __name__ == "__main__":
    root = tk.Tk()
    app = TaskManagerApp(root)
    app.prev_bytes_sent = psutil.net_io_counters().bytes_sent
    app.prev_bytes_recv = psutil.net_io_counters().bytes_recv
    root.mainloop()
```

main_window.py

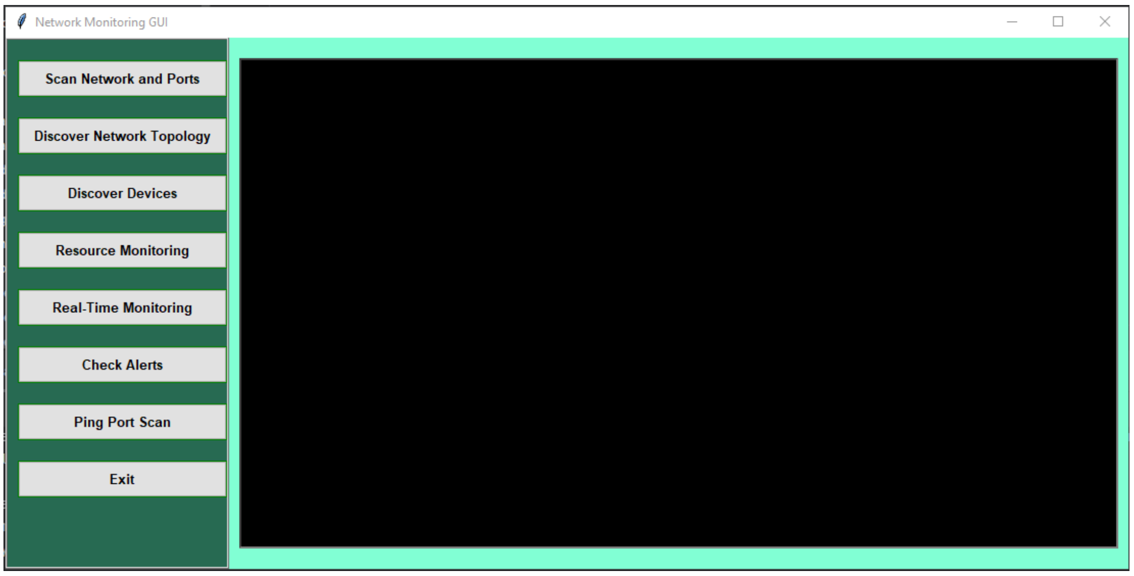
```
#src/gui/main_window.py
import tkinter as tk
import asyncio
import subprocess
import os
from tkinter import ttk
from device_discovery.device_discoverer import discover_devices, list_wifi_networks
from scan_network.network_scanner import NetworkScanner
from scan_network.scanner import Scanner
from port_detection.port_detector import NetworkPortDetector
from network_mapping.mapper import Mapper
from resource_monitoring.usage_monitor import TaskManagerApp
from alerts_and_notifications.alert_system import AlertSystem
from gui.real_time_monitoring_window import RealTimeMonitoringWindow
from resource_monitoring.usage_monitor import TaskManagerApp as ResourceTaskManagerApp
# Import the PPPPP_GUI class from src/auto_scan/ping_port_scan
from auto_scan.ping_port_scan import PPPPP_GUI


class MainWindow:
    def __init__(self, alert_system):  # Accept alert_system as a parameter
        self.alert_system = alert_system  # Store alert_system as an attribute
        self.root = tk.Tk()
        self.root.title("Network Monitoring GUI")
        self.root.geometry("1100x520")  # Set the initial size of the window
        self.root.resizable(False, False)  # Disable resizing both horizontally and vertically
        self.main_frame = ttk.Frame(self.root, style="Main.TFrame", relief="ridge")
        self.main_frame.pack(fill="both", expand=True)
        self.style = ttk.Style()
        self.style.configure("Main.TFrame", background="#81FFD4")
```

```
# Initialize other components
self.result_textbox = None
self.resource_monitor = None
# Initialize the Scanner instance
self.scanner = Scanner()
self.options_frame = ttk.Frame(self.main_frame, width=200, padding=(10, 10, 0, 10), style="Options.TFrame", borderwidth=2, relief="ridge")
self.options_frame.pack(side="left", fill="y")
self.style.configure("Options.TFrame", background="#276A52", foreground="white")
# Increase the width of the Result frame
self.result_frame = ttk.Frame(self.main_frame, style="Result.TFrame", width=600)
self.result_frame.pack(side="right", fill="both", expand=True)
self.style.configure("Result.TFrame", background="#81FFD4")
# Set a fixed width for all buttons
button_width = 25
#buttons
self.scan_button = ttk.Button(self.options_frame, text="Scan Network and Ports", command=self.scan_and_detect_ports, style="Blue.TButton", width=button_width) self.scan_button.pack(pady=10, ipadx=10, ipady=5)


self.discover_button = ttk.Button(self.options_frame, text="Discover Network Topology", command=self.discover_topology, style="Blue.TButton", width=button_width)
self.discover_button.pack(pady=10, ipadx=10, ipady=5)
self.discover_device_button = ttk.Button(self.options_frame, text="Discover Devices", command=self.display_discovered_devices, style="Blue.TButton", width=button_width)
self.discover_device_button.pack(pady=10, ipadx=10, ipady=5)
self.resource_monitor = None
# Resource Monitoring button
self.resource_monitor_button = ttk.Button(self.options_frame, text="Resource Monitoring", command=self.open_resource_monitoring, style="Blue.TButton", width=button_width)
```

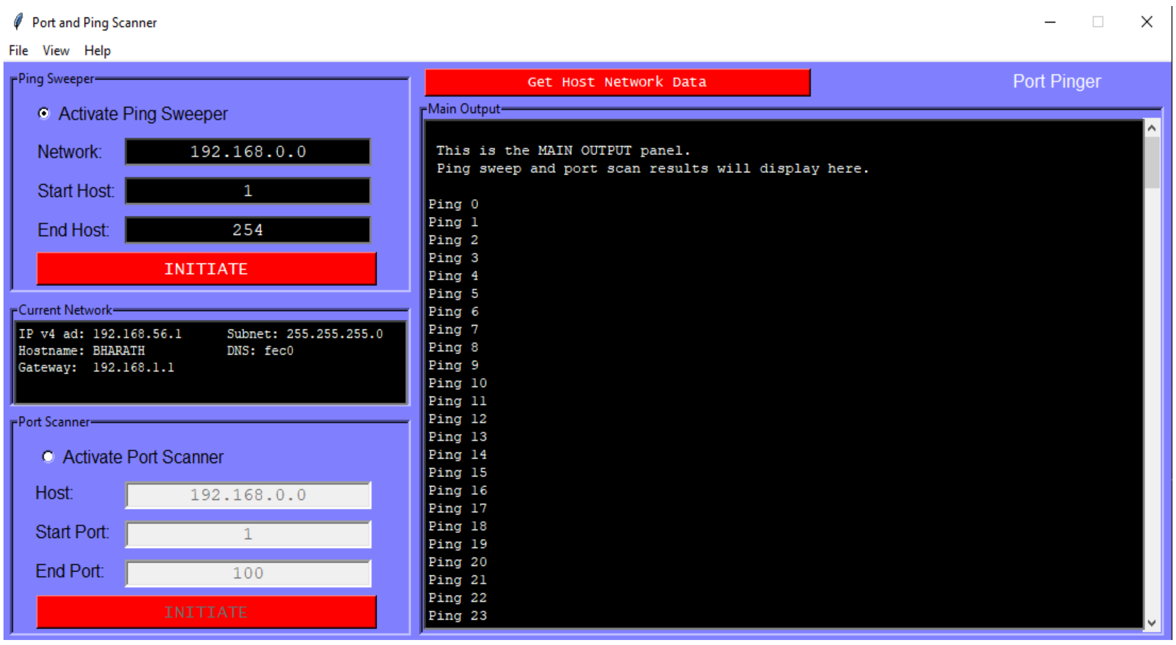
```
        self.resource_monitor_button.pack(pady=10, ipadx=10, ipady=5)


        self.real_time_monitoring_button = ttk.Button(self.options_frame, text="Real-Time Monitoring", command=self.open_real_time_monitoring, style="Blue.TButton", width=button_width)

        self.real_time_monitoring_button.pack(pady=10, ipadx=10, ipady=5)

        self.real_time_monitor = None

        # Add a new button for checking alerts

        self.check_alerts_button = ttk.Button(self.options_frame, text="Check Alerts", command=self.check_alerts, style="Blue.TButton", width=button_width)

        self.check_alerts_button.pack(pady=10, ipadx=10, ipady=5)


        self.ping_port_button = ttk.Button(self.options_frame, text="Ping Port Scan", command=self.ping_port_scan, style="Blue.TButton", width=button_width)

        self.ping_port_button.pack(pady=10, ipadx=10, ipady=5)


        self.exit_button = ttk.Button(self.options_frame, text="Exit", command=self.root.destroy, style="Blue.TButton", width=button_width)

        self.exit_button.pack(pady=10, ipadx=10, ipady=5)

        # Center the Entry widget

        self.result_textbox_frame = ttk.Frame(self.result_frame, style="Result.TFrame")

        self.result_textbox_frame.pack(pady=(20, 20), padx=(10, 10), fill="both", expand=True)

        self.result_textbox = tk.Text(self.result_textbox_frame, font=("Helvetica", 12), wrap=tk.WORD, foreground="white", borderwidth=3, background="black", relief="sunken", width=30,  height=10)

        self.result_textbox.pack(expand=True, fill="both")   # Set expand and fill properties

        # Prevent the Entry frame from automatically adjusting to the size of its children

        self.result_textbox_frame.pack_propagate(False)

        # Configure a style for blue buttons with black text

        self.style.configure("Blue.TButton", foreground="black", background="green", font=("Helvetica", 10, "bold"))

    def display_discovered_devices(self):

        network_range = "192.168.1.0/24"  # Set the network range
        discovered_devices = discover_devices(network_range)
        result_message = "Discovered Devices:\n"
        if discovered_devices:
            for device in discovered_devices:
                result_message += f"\nIP Address: {device['IP Address']},\nMAC Address: {device['MAC Address']},\nVendor: {device['Vendor']},\nStatus: {device['Status']}\n"
        else:
            result_message += "No devices discovered."
        # Append information about Wi-Fi networks
        wifi_networks = list_wifi_networks()
        result_message += "\n\nAvailable Wi-Fi Networks:\n"
        if wifi_networks:
            for i, network in enumerate(wifi_networks, start=1):
                result_message += f"{i}. {network}\n"
        else:
            result_message += "No Wi-Fi networks available."

        self.display_result(result_message)

    def discover_topology(self):
        network_mapper = Mapper()
        topology_result = network_mapper.discover_topology(method='wmi')
        self.display_result(f"Network Topology Result: {topology_result}")
        #self.display_result.pack(pady=10, ipadx=10, ipady=5)
    def initialize_result_textbox(self, result_textbox):
        self.result_textbox = result_textbox
    def scan_and_detect_ports(self):
        # Function to create the input frame
        def create_input_frame():
            input_frame = ttk.Frame(self.result_frame, style='Input.TFrame')  # Add style parameter
            # Create a custom style for the input frame
            self.style = ttk.Style()
            self.style.configure('Input.TFrame', background='#81FFD4') # Set
    background color
            # Set style for labels
            self.style.configure('InputLabel.TLabel',background='#81FFD4',
    foreground='black')  # Set label color
            # Set style for button
            self.style.configure('ScanButton.TButton', background='blue', relief='raised')
    # Set button color and relief
            # Create labels and entry widgets for IP address and ports
            ip_label = ttk.Label(input_frame, text="Enter IP Address:",
    style='InputLabel.TLabel')  # Apply label style
            ip_label.pack(side="left", padx=5, pady=5)
            ip_entry = ttk.Entry(input_frame)
            ip_entry.pack(side="left", padx=5, pady=5)
            # Label to instruct users to enter ports manually
            ports_label = ttk.Label(input_frame, text="Enter Ports (comma-separated):",
                            style='InputLabel.TLabel')  # Apply label style
            ports_label.pack(side="left", padx=5, pady=5)
            ports_entry = ttk.Entry(input_frame)
            ports_entry.pack(side="left", padx=5, pady=5)

            # Function to handle scan button click
            def start_scan():
                ip = ip_entry.get()
                ports_input = ports_entry.get()

                # If ports are not provided, use a list of common ports to scan
                if not ports_input:
                    ports = [21, 22, 23, 25, 53, 80, 110, 143, 443, 465, 587, 993, 995]
                else:
                    ports = [int(port.strip()) for port in ports_input.split(",")]
                result_message = ""
                try:
                    # Perform network scanning using the Scanner instance
                    scanner = Scanner()
                    scanner_result = scanner.scan_arp()  # Use the ARP scanning method from scanner.py
                    scanner_result_message = "Successfully scanned with Scanner\n"
                    # Prepare the table header and data for ARP scan
                    arp_table_header = ["IP Address", "MAC Address"]
                    arp_table_data = [(device['ip'], device['mac']) for device in scanner_result]
                    # Display the ARP scan results as a table
                    arp_table_string = "IP Address\tMAC Address\n"
                    for row in arp_table_data:
                        arp_table_string += f"{row[0]}\t{row[1]}\n"
                    # Combine the scanner result message and the ARP table
                    result_message += f"Network Scan Result (Scanner):\n{scanner_result_message}\n\n"
                    result_message += arp_table_string
                except Exception as e:
                    print(f"Error during network scan with Scanner: {e}")
                    result_message += "Error during network scan with Scanner."
                try:
                    # Perform port detection using NetworkPortDetector
                    port_detector = NetworkPortDetector()
                    open_ports = port_detector.detect_open_ports(host=ip, ports=ports)
                    if open_ports:
                        # If no ports were entered manually, show all open ports
                        if not ports_input:
                            result_message += f"\nOpen Ports for Common Ports: {open_ports}"
                        else:
                            result_message += f"\nOpen Ports: {open_ports}"
                    else:
                        result_message += "\nNo open ports found."
                except Exception as e:
                    print(f"Error during port detection: {e}")
                    result_message += "\nError during port detection."


                # Display the result in the result text box
                self.display_result(result_message)
            # Create a button to start the scan
            scan_button = ttk.Button(input_frame, text="Start Scan",
    command=start_scan,
                                    style='ScanButton.TButton')  # Apply button style
            scan_button.pack(side="left", padx=5, pady=5)
            return input_frame
        # Hide the input frame if it exists
        if hasattr(self, "input_frame") and self.input_frame.winfo_ismapped():
            self.input_frame.pack_forget()
        else:
            # Create and show the input frame
            self.input_frame = create_input_frame()
            self.input_frame.pack(fill="both", expand=True)
    def open_resource_monitoring(self):
        if self.resource_monitor is None:
            # Create a new top-level window for resource monitoring
            self.resource_monitor_window = tk.Toplevel(self.root)
            self.resource_monitor_window.title("Resource Monitoring")
            # Create a new TaskManagerApp instance for resource monitoring
            self.resource_monitor =
    ResourceTaskManagerApp(self.resource_monitor_window)
        else:
            # Show or raise the existing resource monitoring window
            self.resource_monitor_window.lift()
            # Show the result_textbox if it exists
            if self.result_textbox is not None:
            self.result_textbox.pack(in_=self.result_textbox_frame,expand=True,
fill="both")
    def clear_result_frame(self):
        # Recreate the result_textbox
        self.result_textbox = tk.Text(self.result_textbox_frame, font=("Helvetica", 12),
wrap=tk.WORD, foreground="white", borderwidth=3, background="black",
relief="sunken", width=30,   height=10)
        self.result_textbox.pack(expand=True, fill="both")  # Set expand and fill
properties
    def open_real_time_monitoring(self):
        if self.real_time_monitor is None:
            self.real_time_monitor = RealTimeMonitoringWindow(self.result_textbox)
        else:
            self.real_time_monitor.update_real_time_info()
    def display_result(self, message):
        self.result_textbox.delete(1.0, tk.END)  # Clear previous content
        self.result_textbox.insert(tk.END, message)
        if self.result_textbox:
            self.result_textbox.delete(1.0, tk.END)  # Clear previous content
            self.result_textbox.insert(tk.END, message)
        else:
            print("Result text box not initialized. Cannot display result.")
    def ping_port_scan(self):
        # Use PPPPP_GUI for ping port scan and display the results in a new window
        ping_port_result = PPPPP_GUI(self.root)


        # Create a new Toplevel window for displaying the ping port scan result
        ping_port_result_window = ttk.Toplevel(window)
        ping_port_result_window.title("Ping Port Scan Result")
        # Set the geometry of the new window
        ping_port_result_window.geometry("1100x520")
        # Create a frame in the Toplevel window to display the result
        new_frame = ttk.frame(ping_port_result_window.master, font=("Helvetica", 12), wrap=tk.WORD, foreground="white", borderwidth=3, background="black", relief="sunken", width=30, height=10)
        ping_port_gui.pack(expand=True, fill="both")
        # new_frame.insert(tkk.END, ping_port_result)


    def manage_devices(self):
        # Call the device management function and display the result
        device_management_result = manage_devices()
        self.display_result(device_management_result)
    def check_alerts(self):
        alerts = self.alert_system.get_alerts()  # Access alert_system attribute
        if alerts:
            alert_messages = [str(alert) for alert in alerts]
            result_message = "\n".join(alert_messages)
        else:
            result_message = "No alerts to display."
        self.display_result(result_message)
    def run(self):
        self.root.mainloop()


if __name__ == "__main__":
    alert_system = AlertSystem()  # Create an instance of AlertSystem
    main_window = MainWindow(alert_system)  # Pass alert_system to MainWindow
    main_window.run()
```

6.2 System Snapshots:

MAIN DESIGN:



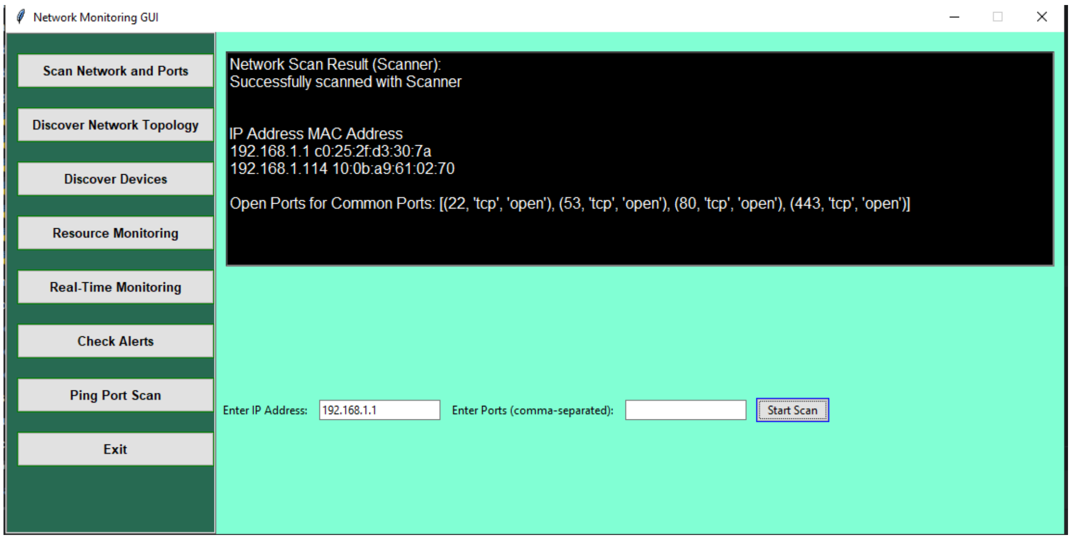
PING PORT DESIGN:



```
IP v4 ad: 192.168.56.1        Subnet: 255.255.255.0
Hostname: BHARATH             DNS: fec0
Gateway:  192.168.1.1
```

```
 This is the MAIN OUTPUT panel.
 Ping sweep and port scan results will display here.

Ping 0
Ping 1
Ping 2
Ping 3
Ping 4
Ping 5
Ping 6
Ping 7
Ping 8
Ping 9
Ping 10
Ping 11
Ping 12
Ping 13
Ping 14
Ping 15
Ping 16
Ping 17
Ping 18
Ping 19
Ping 20
Ping 21
Ping 22
Ping 23
```

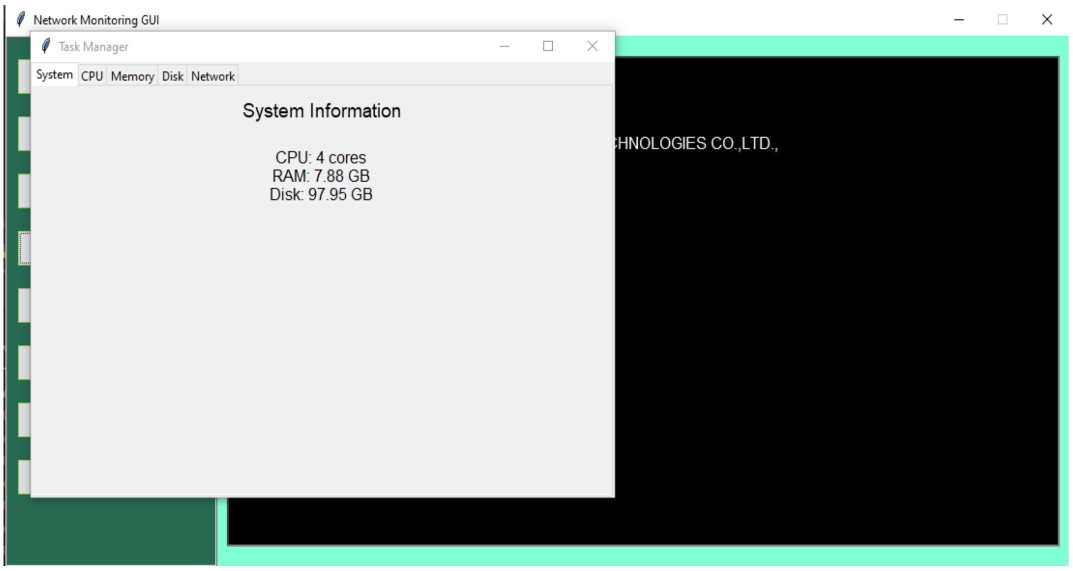
Device Discovery Inputs:



```
Network Scan Result (Scanner):
Successfully scanned with Scanner


IP Address MAC Address
192.168.1.1 c0:25:2f:d3:30:7a
192.168.1.114 10:0b:a9:61:02:70

Open Ports for Common Ports: [(22, 'tcp', 'open'), (53, 'tcp', 'open'), (80, 'tcp', 'open'), (443, 'tcp', 'open')]
```

Enter IP Address: 192.168.1.1 Enter Ports (comma-separated):

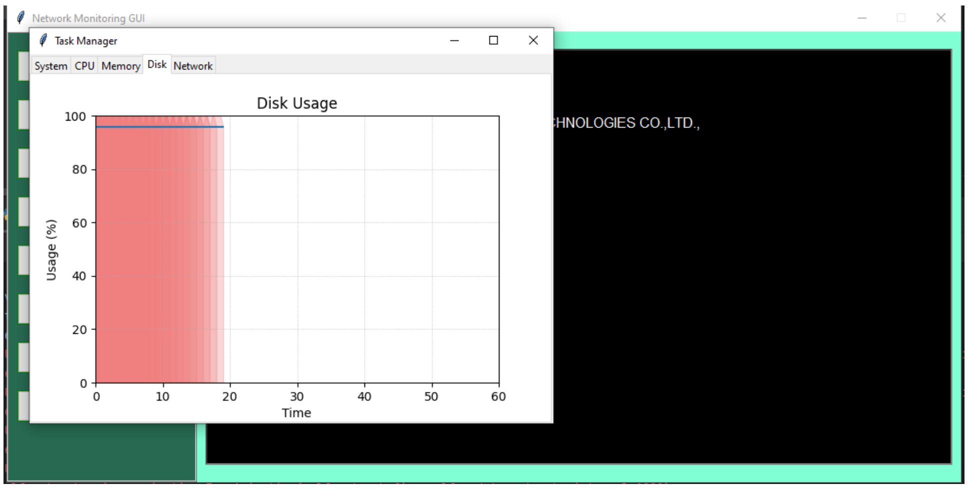
Resource Monitoring Inputs:

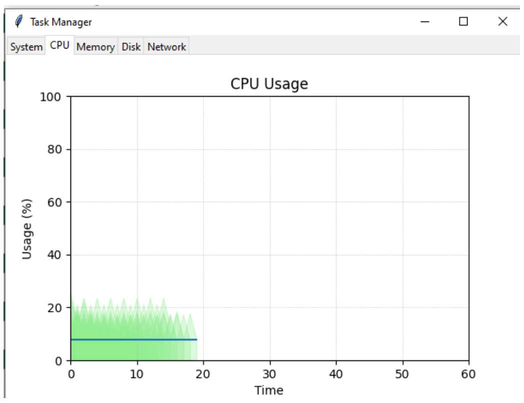


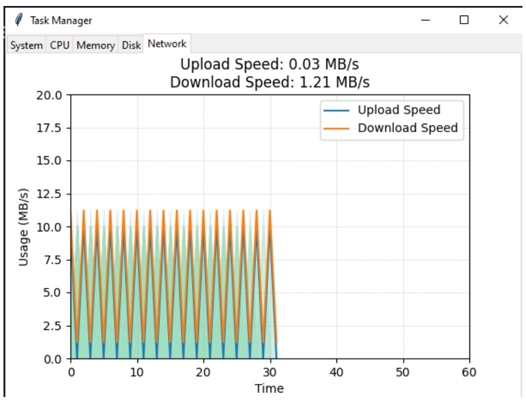
System Information

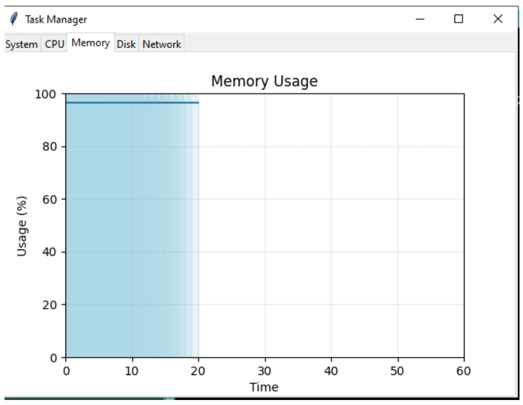
CPU: 4 cores
RAM: 7.88 GB
Disk: 97.95 GB

HNOLOGIES CO.,LTD.,

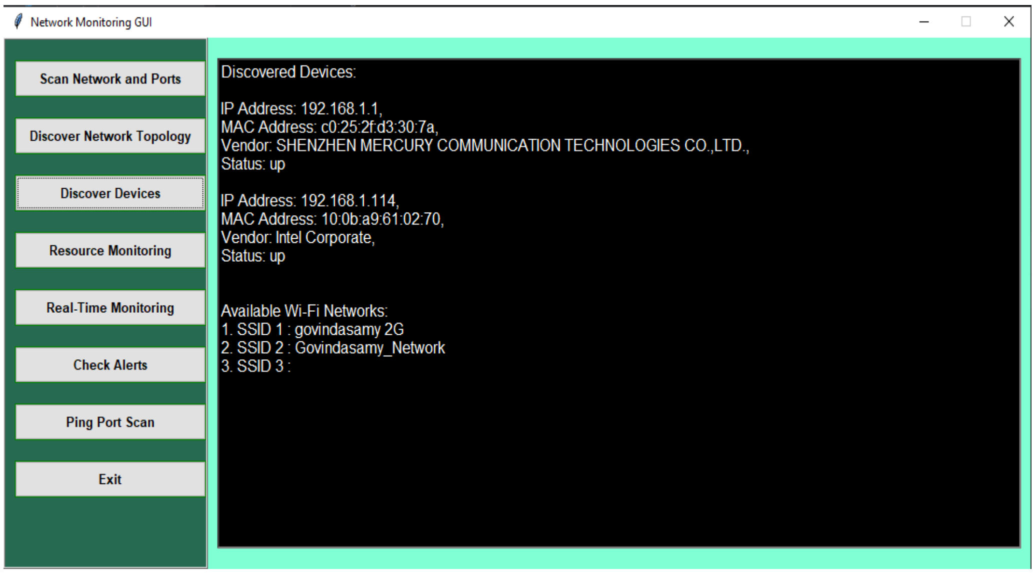

Network Monitoring GUI
Task Manager
System
CPU
Memory
Disk
Network
Disk Usage
Usage (%)
Time
HNOLOGIES CO.,LTD.,


Task Manager
System
CPU
Memory
Disk
Network
CPU Usage
Usage (%)
Time


Task Manager
System
CPU
Memory
Disk
Network
Upload Speed: 0.03 MB/s
Download Speed: 1.21 MB/s
Upload Speed
Download Speed
Usage (MB/s)
Time

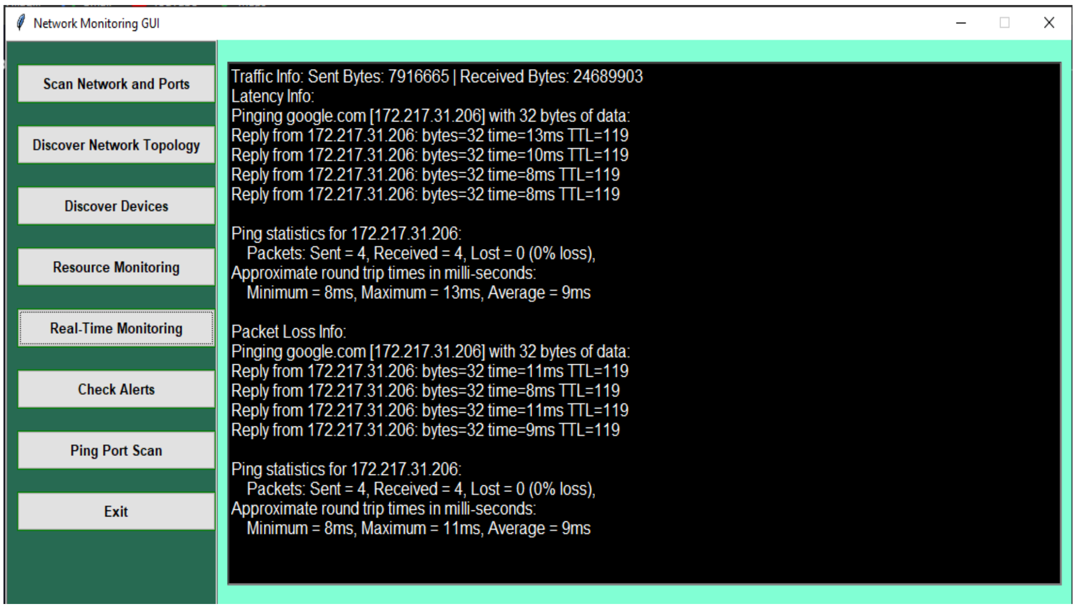

Task Manager
System
CPU
Memory
Disk
Network
Memory Usage
Usage (%)
Time

Real-Time Monitoring:



```
Discovered Devices:

IP Address: 192.168.1.1,
MAC Address: c0:25:2f:d3:30:7a,
Vendor: SHENZHEN MERCURY COMMUNICATION TECHNOLOGIES CO.,LTD.,
Status: up

IP Address: 192.168.1.114,
MAC Address: 10:0b:a9:61:02:70,
Vendor: Intel Corporate,
Status: up


Available Wi-Fi Networks:
1. SSID 1 : govindasamy 2G
2. SSID 2 : Govindasamy_Network
3. SSID 3 :
```

Device Discovery:



```
Traffic Info: Sent Bytes: 7916665 | Received Bytes: 24689903
Latency Info:
Pinging google.com [172.217.31.206] with 32 bytes of data:
Reply from 172.217.31.206: bytes=32 time=13ms TTL=119
Reply from 172.217.31.206: bytes=32 time=10ms TTL=119
Reply from 172.217.31.206: bytes=32 time=8ms TTL=119
Reply from 172.217.31.206: bytes=32 time=8ms TTL=119

Ping statistics for 172.217.31.206:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 8ms, Maximum = 13ms, Average = 9ms

Packet Loss Info:
Pinging google.com [172.217.31.206] with 32 bytes of data:
Reply from 172.217.31.206: bytes=32 time=11ms TTL=119
Reply from 172.217.31.206: bytes=32 time=8ms TTL=119
Reply from 172.217.31.206: bytes=32 time=11ms TTL=119
Reply from 172.217.31.206: bytes=32 time=9ms TTL=119

Ping statistics for 172.217.31.206:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 8ms, Maximum = 11ms, Average = 9ms
```

Chapter – 7

Bibliography

7.1 Bibliography

- Python Software Foundation. (2022). Python 3.9.9 Documentation. Retrieved from https://docs.python.org/3.9/index.html
- Tkinter Documentation. (2022). Tkinter GUI Toolkit. Retrieved from https://docs.python.org/3/library/tkinter.html
- OpenAI. (2022). GPT-3.5. Retrieved from https://openai.com/gpt-3
- GitHub Repository: Retrieved from https://github.com/
- Real Python Tutorials. Retrieved from https://realpython.com/
- Nmap - Free Security Scanner for Network Exploration & Security Audits. [Online]. Available: https://nmap.org/