Thesis Project - D0032D

Network Guard
– A Python-Powered Network Monitoring Solution

Mohamad Al Homsi

Computer Networking - Higher Education Diploma - 2024

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

[This page intentionally left blank]

Abstract

In today's interconnected world, the reliability and performance of network infrastructure are critical for business operations and service delivery. Network Guard, a Python-powered network monitoring solution, is designed to address these challenges by providing comprehensive, real-time insights into network performance. This thesis details the development, implementation, and evaluation of Network Guard, focusing on its ability to perform network discovery, connection monitoring, and bandwidth analysis. The tool leverages Python's flexibility and Scapy's powerful packet manipulation capabilities to offer a scalable, user-friendly interface that meets the diverse needs of modern network environments. Our evaluations demonstrate Network Guard's effectiveness in real-time diagnostics and its potential to enhance network stability and performance, making it a valuable asset for network engineers and administrators.

Table of Contents

Aim

The main goals of the study are to:

- ***Develop Network Guard:*** Develop a solution for network monitoring called Network Guard, which can detect and analyze network problems in real time.

- ***Provide a comprehensive view:*** Implement network discovery, connection monitoring and bandwidth monitoring features in Network Manager to provide users with a comprehensive view of their network infrastructure.

- ***Be scalable and flexible:*** Network Guard is designed to be scalable, flexible and adaptable to different network environments. It enables future growth and change.

- ***Enable real-time detection:*** By using real-time diagnostics, Network Guard can detect network problems as they occur and react and resolve them quickly.

- ***Create a user-friendly interface:*** Design an intuitive user interface for Network Guard that enables network engineers and administrators to easily view network data, interpret information and take action.

- ***Evaluation of performance and usability:*** Tests and evaluations will be conducted to assess the performance, usability, and effectiveness of a network's global environment.

By achieving these goals, this publication aims to contribute to the development of network monitoring technology and provide organizations with a powerful tool to ensure network stability and performance.

Background and Motivation

1.1 Motivation

1.1.1 Network Monitoring

In today's connected world, where businesses are highly dependent on digital infrastructure, the importance of network monitoring cannot be overstated. Downtime, and poor performance can lead to serious consequences. By monitoring the network, organizations can reduce the risk of downtime, security threats and ensure a smooth user experience for customers and employees.

Network monitoring is an important part of modern network management, ensuring the stability and performance of computer networks. This includes monitoring and analyzing networks, devices and service signals to detect and respond to malicious activity and performance issues. Network monitoring allows organizations to quickly identify and resolve problems, optimize network performance and improve it.

1.1.2 Challenges in Network Monitoring

Although network monitoring is important, it presents many challenges for organizations. As network environments become more complex, including the installation of more devices, deployment of new applications, and the advancing data types, it becomes increasingly difficult to gain a complete understanding of network performance. Additionally, monitoring tools often lack the scalability, flexibility, and automation needed to keep up the pace with dynamic network environments.

1.1.3 Research Gap

Many network monitoring tools and techniques exist, but new solutions are needed to address the limitations of traditional methods. This thesis attempts to bridge this gap by considering the development of "Network Guard", a network monitoring solution built in Python and designed to detect network issues in real-time and easily report them. The network monitoring tool developed during the course of this thesis is designed to leverage the flexibility and extensibility of Python and Scapy to provide a complete and customizable solution for network monitoring in a variety of environments.

1.1.4 Motivation for Network Guard

The motivation to develop Network Guard is derived from the need of a scalable and user-friendly network monitoring solution that addresses the shortcomings of existing tools. Network Guard is designed to harness the power of Python and Scapy to provide network engineers and administrators with a complete toolbox for monitoring and managing their networks. The solution supports real-time network problem detection and intelligent display of network data, helping users make quick decisions and actions to ensure network stability and security.

1.1.5 Benefits of Network Guard

Network Guard has a number of advantages over traditional network monitoring tools.

1. Flexibility*:* Python and Scapy are reliable and flexible in a sense that they adapt to different network environments. Additionally, the monitoring functionality can be adapted based on your needs.

2. Scalability: Network Guard is designed to dynamically scale to accommodate network growth and increasing data volumes while ensuring continuous monitoring and reliability.

3. Easy-to-use interface: A comprehensive graphical user interface makes it possible for engineers and network managers to easily view network data, interpret observations and perform tasks.

4. Real-time detection: Network Guard uses real-time diagnostics to quickly detect network issues so that they can be resolved quickly with minimal downtime.

5. Comprehensive monitoring: Network Guard provides monitoring functions including network discovery, connection monitoring, filtering and bandwidth monitoring, providing users with a comprehensive overview of the network structure.

1.2 Theoretical Background

1.2.1 Python

Python is a high-level, interpreted programming language known for its simplicity and readability. Python's clear syntax and dynamic nature makes it ideal for rapid application development and scripting. With features like dynamic typing, dynamic binding, and robust built-in data structures, Python enhances productivity and reduces maintenance costs. It supports modularity through modules and packages, encouraging code reuse and making it easier to manage large projects.[*1*]

1.2.2 Scapy

Scapy is a powerful Python library used for network packet manipulation. It enables users to create, send, receive, and analyze network packets, making it a versatile tool for tasks such as network probing, scanning, and even attacks. Scapy excels in two primary functions: sending packets and receiving responses. Users can interactively craft custom packets, monitor network traffic, and perform various network-related tasks within a single Python environment. Its flexibility and ease of use make Scapy invaluable for network professionals and developers interested in network communications. [*2*]

1.2.3 Tkinter

Tkinter is a Python library used to create graphical user interfaces (GUIs). It offers a set of tools that allow developers to build windows, buttons, menus, and other GUI components for their applications. Based on the Tk GUI toolkit, Tkinter is cross-platform, making it suitable for developing GUI

applications that run on various operating systems. With Tkinter, developers can create interactive and visually appealing desktop applications using straightforward code.[3]

1.2.4 ARP (Address Resolution Protocol)

The Address Resolution Protocol (ARP) is a crucial protocol in TCP/IP networks that binds 32-bit IP addresses to 48-bit MAC addresses, enabling data packets to be transmitted directly on a local network. ARP functions efficiently by resolving IP addresses to MAC addresses and sending ARP requests to cache the responses, ensuring smooth communication. It plays a significant role in network performance, particularly in LAN environments, and is an essential part of applications like Network Guard, specifically for the performed monitoring in the application.[4]

1.2.5 Broadcast

In networking, broadcasting is a method of communication where a single packet is sent from one device to all devices on a local network segment. This technique is useful when the sender needs to distribute information to all network participants without knowing their specific addresses.[5]

How does broadcasting work?

Broadcast Address: In Ethernet networks, a specific address (FF:FF:FF:FF:FF) is designated as the broadcast address. Packets sent to this address are received by all devices on the local network.

Broadcast Frame: When a device sends a broadcast frame, all devices on the network segment receive it. Each device examines the frame to decide whether to process it or not.

Usage in ARP: ARP uses broadcasts to send requests to all devices on the network, asking for the MAC address associated with a specific IP address. All devices receive the request, but only the one with the matching IP address responds.[5]

1.2.6 MAC (Media Access Control)

A MAC address is a unique 48-bit identifier assigned to a network interface, essential for data transmission on an Ethernet network. Operating at the data link layer, MAC addresses are used to send specific packets to network devices using unicast, multicast, and broadcast methods. They are crucial in security protocols like ARP, which maps IP addresses to MAC addresses to facilitate data transfer.[6]

1.2.7 IP (The Internet Protocol)

The Internet Protocol (IP) is the backbone of modern networks, enabling data transfer across interconnected networks. IP uses unique numerical identifiers called IP addresses to identify devices and route data packets. Operating on a connectionless packet exchange model, IP provides flexible and scalable communication. Understanding IP is fundamental for grasping network architecture, essential for tools like Network Guard that monitor and manage networks using the IP protocol. IPv4 addresses are 32-bit numbers divided into four octets, separated by periods (e.g. 192.168.1.1). [4]

1.2.8 ICMP (Internet Control Message Protocol) and ping

The Internet Control Message Protocol (ICMP) operates at the network layer, handling error reporting and network path diagnostics. Encapsulated within an IP header, ICMP messages use a specific protocol field followed by the ICMP payload.

The 'ping' utility, a key network diagnostic tool, sends an echo packet to a target device to check its reachability and measure round-trip time. By exchanging ICMP echo request and reply packets, ICMP and Ping help network administrators evaluate connectivity and latency. This is useful for verifying device availability and assessing network infrastructure. ICMP testing is valuable for diagnosing issues, detecting packet loss, and evaluating overall network performance.[7]

1.2.9 LAN (Local Area Network)

A Local Area Network (LAN) is a group of devices connected within a single physical location, such as a building, office, or home.[8]

- Key characteristics of a LAN:

Limited Geographic Area: LANs typically cover a small geographic area, like a single building, office, home, or a group of buildings in close proximity.[9]

High Data Transfer Rates: LANs are known for their high data transfer speeds, usually ranging from 10 Mbps to 10 Gbps, which makes them suitable for applications requiring fast and reliable communication.[9]

Private Ownership: LANs are usually owned, managed, and maintained by a single organization or individual, which provides control over the network infrastructure and security.[9]

Wired and Wireless Connections: LANs can use both wired connections (such as Ethernet cables) and wireless connections (such as Wi-Fi) to link devices.[9]

- Components of a LAN:

Network Interface Cards (NICs): Each device on a LAN has a NIC, which provides the hardware interface between the computer and the network.[9]

Switches and Hubs: These devices are used to connect multiple devices on the LAN, facilitating communication between them. Switches are more advanced than hubs, as they can direct data to specific devices rather than broadcasting it to all connected devices.[9]

Routers: While routers are typically associated with connecting different networks, they can also be used in a LAN to manage traffic and provide connectivity to external networks, such as the Internet.[9]

Cables and Wireless Access Points: Physical cables (like Ethernet) or wireless access points (Wi-Fi routers) connect the devices to the network, enabling communication and data transfer.[9]

Methodology

2.1 Choice of Python and Scapy

2.1.1 Python

Python was chosen as the core programming language for Network Guard for several reasons:

- ***Simplicity and readability:*** Python's syntax is clean and easy to understand, which speeds up development and reduces the likelihood of errors. This is especially beneficial for collaborative projects and future maintenance.
- ***Extensive Libraries:*** Python offers a large variety of libraries and frameworks that simplify various tasks. For network monitoring, libraries like Scapy, Tkinter and threading are invaluable.
- ***Community Support:*** Python has a large and active community. This means extensive documentation, many tutorials and forums where developers can seek help and share knowledge.
- ***Cross-Platform Compatibility:*** Python runs on many operating systems, including Windows, macOS, and various distributions of Linux, making Network Guard highly portable.

2.1.2 Scapy

Scapy was chosen for network packet manipulation and analysis because of its powerful features:

- ***Comprehensive packet handling:*** Scapy can handle a wide range of network protocols, making it versatile for various types of network analysis and monitoring.
- ***Flexibility:*** Scapy allows creating, sending, sniffing and dissecting network packets, which is important for both network discovery and bandwidth monitoring.
- ***Integration with Python:*** Since Scapy is a Python library, it integrates seamlessly with Python applications, enabling the development of complex network monitoring functions with ease.

2.2 Design and Architecture

Network Guard's architecture has been carefully designed to achieve the goals of comprehensive network monitoring, real-time detection, scalability, flexibility, and ease of use. Network Guard is divided into two main components:

1. A network discovery and connection monitoring application.
2. A bandwidth monitoring application.

Each component is designed to perform specific functions while complementing each other to provide a comprehensive view of the network infrastructure.

2.2.1 Network discovery and connection monitoring application

This application focuses on identifying and tracking devices connected to the network. It is built using Tkinter for the GUI and Scapy for network interactions. The app maintains three main views: connected devices, new devices, and offline devices.

The main components are:

1. Graphical User Interface (GUI):
- View connected devices: Displays the IP and MAC addresses of currently connected devices.

- ***Show New Devices:*** Displays devices that have recently joined the network.

- Show Disconnected Devices: Lists devices that were previously connected but are no longer detected.

2. Network monitoring logic:
- Device Discovery: Uses ARP requests to discover devices on the network. The ***"get_devices"*** method sends ARP requests and processes the responses to generate a list of active devices.

- Real-time monitoring: A separate thread runs the ***"Monitor_devices"*** function, which constantly scans the network, updates device lists, and detects changes.

The setup part starts by initializing the GUI and then monitoring is begun. It generates a thread to run the monitoring loop.

1. Initial scan: Compiles a basic list of connected devices.

2. Continuous monitoring: Frequently scans the network, comparing the current list of devices to the baseline to identify new and disconnected devices.

3. UI Updates: Updates the respective views in the GUI to reflect the current state of the network.

2.2.2 Bandwidth monitoring application

This application, being the second major part of Network Guard, measures and reports the bandwidth usage of devices on the network. It is implemented using Scapy for packet sniffing and the Python threading module for concurrent execution.

Key components:

1. Package size tracking: The ***"packet_sizes"*** dictionary tracks the cumulative size of packets associated with each device's IP address.

2. Calculate bandwidth: The ***"Calculate_bandwidth"*** function updates the ***"packet_sizes"*** dictionary based on packets captured by Scapy. The ***"print_bandwidth_usage"***, which is a function related to the ***"Calculate_bandwidth"*** function, runs in a separate thread, periodically calculating and printing each device's bandwidth usage.

The part starts by launching a thread to handle printing bandwidth usage. It then uses Scapy's Sniff function to capture network traffic and update bandwidth statistics.

1. Packet capture: The ***"Calculate_bandwidth"*** function is called for each packet, updating the total packet size for the devices involved.

2. Periodic reporting: The ***"print_bandwidth_usage"*** function runs in a loop, calculating bandwidth usage every second and resetting the counters for the next interval.

2.2.3 Integration and scalability

Both applications can work independently or together to provide a full range of network monitoring capabilities:

- Co-operation: By running both applications simultaneously, users can monitor device connections and bandwidth usage in real-time, and get a comprehensive understanding of network activity.

- Data Sharing: The architecture allows for potential integration as bandwidth data can be visualized within the GUI, enhancing the user experience.

2.2.3.1 Network Guard is designed to be scalable and adaptable

- Modular design: Each component (network discovery, connection monitoring, bandwidth monitoring) is modular, allowing for easy improvements and additions.

- Adaptability: The system can be configured for different network environments and scales very well regardless if it is a small home network or a larger enterprise network.

- Future Growth: The architecture supports future extensions, such as adding more sophisticated network analysis tools, integration with other network management systems, or improving the user interface.

2.3 System Architecture and Implementation

2.3.1 Components of Network Guard

Network Guard is composed of several key components that work together to provide comprehensive network monitoring and analysis. These components include the Network Discovery module, the Connection Monitoring module, and the Bandwidth Monitoring module. Each component is designed to perform specific tasks, contributing to the overall functionality and efficiency of the system.

The component overview:

Network Discovery Module: This module is responsible for identifying all devices connected to the network, capturing their IP and MAC addresses. It provides the basis for connection monitoring by maintaining an up-to-date list of active devices.

Connection Monitoring Module: This module tracks changes in the network, specifically identifying new devices connecting and existing devices disconnecting. It helps maintain real-time awareness of network status.

Bandwidth Monitoring Module: This module measures and reports the bandwidth usage of each device on the network. By tracking network packet size and frequency, it provides insights into network load and usage patterns.

2.3.2 Network Discovery

Network discovery is a crucial component of Network Guard, responsible for identifying all devices connected to the network. This process involves sending ARP requests to the network and capturing the responses to map out the devices.

The implementation Details:

ARP Requests: The ***"get_devices"*** method utilizes ARP to probe the local network. It constructs an ARP request packet using the Scapy library, targeting the entire subnet (192.168.100.0/24 in this case).

Packet Construction: The ARP request is encapsulated in an Ethernet frame with a broadcast destination address (ff:ff:ff:ff:ff:ff), ensuring all devices on the network receive the request.

Sending and Receiving: The constructed packet is sent out, and the responses are captured using Scapy's ***"srp"*** function. The responses contain the IP and MAC addresses of the devices that replied.

Device List: The method iterates through the responses, extracting the IP and MAC addresses and storing them in a list.

- Code Snippet:

```
def get_devices(self, interface='enp0s3'):
    arp = ARP(pdst='192.168.100.0/24')
    ether = Ether(dst='ff:ff:ff:ff:ff:ff')
    packet = ether/arp
    result = srp(packet, timeout=2, iface=interface, verbose=0)[0]

    devices = []
    for sent, received in result:
        devices.append((received.psrc, received.hwsrc))

    return devices
```

This process ensures that Network Guard maintains an updated list of all devices currently connected to the network, providing a comprehensive view of the network's topology.

2.3.3 Connection Monitoring

Connection monitoring is designed to track the state of devices on the network, identifying new connections and detecting when devices disconnect. This real-time monitoring capability is essential for maintaining network security and stability.
The implementation Details:

Initial Device List: At startup, the ***"monitor_devices"*** method retrieves the initial list of connected devices using the ***"get_devices"*** method.

Continuous Monitoring: The method enters a loop, periodically calling ***"get_devices"*** to get the current state of the network.

Detecting Changes: It compares the current list of devices with the previously known list to identify new and disconnected devices:

- ***New Devices:*** Devices that appear in the current list but were not in the previous list are classified as new devices.
- ***Disconnected Devices:*** Devices that were in the previous list but do not appear in the current list are classified as disconnected devices.

UI Updates: The method updates the respective trees (***"new_tree"***, ***"disconnected_tree"***, and ***"connected_tree"***) in the GUI to reflect these changes.

- Code Snippet:

```
def monitor_devices(self, interface='enp0s3'):
    self.connected_devices = self.get_devices(interface)
```

```
while True:
    current_devices = self.get_devices(interface)

    disconnected_devices = [d for d in self.connected_devices if d not in current_devices]
    new_devices = [d for d in current_devices if d not in self.connected_devices]

    if disconnected_devices:
        for item in self.disconnected_tree.get_children():
            self.disconnected_tree.delete(item)

        for device in disconnected_devices:
            self.disconnected_tree.insert('', 'end', values=device)

        self.connected_devices = [d for d in self.connected_devices if d not in disconnected_devices]

    if new_devices:
        for item in self.new_tree.get_children():
            self.new_tree.delete(item)

        for device in new_devices:
            self.new_tree.insert('', 'end', values=device)

        self.connected_devices.extend(new_devices)

    for item in self.connected_tree.get_children():
        self.connected_tree.delete(item)

    for device in self.connected_devices:
        self.connected_tree.insert('', 'end', values=device)

    sleep(1)
```

This functionality ensures that network administrators can quickly identify any changes in the network's device landscape, allowing for timely interventions.

2.3.4 Bandwidth Monitoring

Bandwidth monitoring is a key feature of Network Guard, providing insights into the data usage patterns of the devices on the network. This helps in identifying potential bandwidth control and understanding overall network load.

The implementation Details:

Packet Sniffing: The ***"sniff_packets"*** method uses Scapy's ***"sniff"*** function to capture all packets on the network. It runs in a separate thread to continuously monitor network traffic.

Packet Size Calculation: The ***"calculate_bandwidth"*** method processes each captured packet to determine its size. It updates a dictionary (***"packet_sizes"***) that tracks the total data usage for each device.

Source and Destination: The method checks both the source and destination IP addresses of the packet to account for all traffic associated with the monitored devices.

Thread Safety: A lock (***"packet_sizes_lock"***) ensures thread-safe updates to the ***"packet_sizes"*** dictionary.

Bandwidth Display: The ***"print_bandwidth_usage"*** method periodically calculates the bandwidth usage (Kilobytes per second) for each device and updates the ***"bandwidth_tree"*** in the GUI.

- Code Snippet:

- Packet Sniffing:

```
def sniff_packets(self):
    sniff(iface='enp0s8', prn=self.calculate_bandwidth)
```

- Bandwidth Calculation:

```
def calculate_bandwidth(self, packet):
    if IP in packet:
        src_ip = packet[IP].src
        dst_ip = packet[IP].dst

        with self.packet_sizes_lock:
            if any(src_ip == d[0] for d in self.connected_devices):
                self.packet_sizes[src_ip] = self.packet_sizes.get(src_ip, 0) + len(packet)

            if any(dst_ip == d[0] for d in self.connected_devices):
                self.packet_sizes[dst_ip] = self.packet_sizes.get(dst_ip, 0) + len(packet)
```

- Displaying Bandwidth Usage:

```
def print_bandwidth_usage(self):
    self.bandwidth_tree_ready.wait()

    while True:
        for item in self.bandwidth_tree.get_children():
            self.bandwidth_tree.delete(item)

        with self.packet_sizes_lock:
            for ip, size in self.packet_sizes.items():
                bandwidth = (size / 1024) / 1.0
```

```
                self.bandwidth_tree.insert('', 'end', values=(ip, "{:.2f}".format(bandwidth)))
                self.packet_sizes[ip] = 0

        sleep(1)
```

- User Interface:
A separate window (***"bandwidth_window"***) is created to display the bandwidth usage of each device. The ***"bandwidth_tree"*** is populated with the IP addresses and corresponding bandwidth usage, providing a clear and intuitive view of network performance.

The system's architecture and implementation ensure that it is scalable, flexible, and capable of adapting to various network environments, meeting the goals of the project

2.4 Establishing the environment

To effectively test and validate the functionality of Network Guard, a controlled laboratory environment was established. This environment was designed to simulate a typical network setup and facilitate comprehensive testing of network discovery, connection monitoring, and bandwidth monitoring features.

2.4.1 Lab Setup

2.4.1.1 Hardware Configuration

Computers: The lab was equipped with eight personal computers (PCs). Each PC was connected to a central switch, providing a local network environment.

Switch: A network switch was used to connect all eight PCs, ensuring efficient and reliable communication between the devices.

Router: The switch was connected to a router, which managed the network traffic and provided internet connectivity to all the PCs. The router was configured with appropriate network settings to facilitate seamless communication.

2.4.1.2 Network Configuration

IP Address Assignment: The router was configured to assign dynamic IP addresses to each PC through a DHCP server to ensure consistency during testing. This setup allowed for easy identification and monitoring of each device on the network.

Internet Access: Ensuring that all PCs had access to the internet was crucial for simulating real-world network usage. The router was configured with PAT to enable internet access for all connected devices, allowing for testing scenarios that included both local and external network traffic.

2.4.2 Software, Hardware and Tools

To establish and test the functionality of Network Guard, a diverse array of software, hardware, and tools were employed. The selection was aimed at simulating a realistic network environment and ensuring comprehensive monitoring capabilities. Below is a detailed overview of the software, hardware, and tools used in this project.

2.4.2.1 Hardware

Cisco Catalyst 2960 Switch: This switch was crucial for managing the local network connections among the eight PCs. It facilitated efficient data traffic routing and ensured robust connectivity.

Cisco 1840 Router: This router was employed to manage network traffic and provide internet connectivity to all the PCs. It was configured to support various network scenarios, including DHCP for IP address allocation and PAT for internet access.

Personal Computers: The lab setup included eight Windows 10 PCs. Each PC was connected to the network switch, simulating a typical small office or lab environment. These PCs were used to generate network traffic, test connectivity, and monitor network performance.

2.4.2.2 Software

VirtualBox: Oracle VM VirtualBox was used to create virtual environments necessary for running multiple instances of network operating systems. This tool was essential for simulating diverse network setups and testing Network Guard under various conditions.

Ubuntu 22.04 LTS: This Linux distribution served as the primary operating system for running the Network Guard application. Ubuntu's stability and support for a wide range of software packages made it an ideal choice for development and testing.

Wireshark: This network protocol analyzer was used to capture and inspect the data packets traversing the network. Wireshark was invaluable for diagnosing network issues and verifying the accuracy of Network Guard's monitoring capabilities.

Python and Pythons Libraries: Python was the core programming language used to develop Network Guard. Various libraries, such as Scapy for network packet manipulation and Tkinter for the graphical user interface, were utilized to enhance functionality and user experience.

Ping Utility: This simple yet powerful tool was used to test the connectivity between devices on the network. By sending ICMP echo requests, the ping utility helped in identifying latency issues and ensuring that devices were reachable.

PuTTY: This SSH and telnet client was employed for configuring and managing the Cisco switch and router. PuTTY provided a reliable interface for accessing the command-line interfaces (CLI) of the networking hardware, facilitating configuration and troubleshooting.

2.4.3 Testing Process

Initial Configuration: The Network Guard application was configured with the appropriate network interface (enp0s8) and (enp0s3) to monitor the local network. The application's parameters were set to ensure accurate data collection and real-time monitoring.

Baseline Data Collection: An initial scan of the network was conducted to establish a baseline of connected devices. This involved capturing the IP and MAC addresses of all devices connected to the switch with help of ARP protocol.

Simulated Usage Scenarios: To simulate real-life network usage, participants were asked to perform various activities on the PCs. These activities included browsing the internet, downloading files, and watching streaming videos. The goal was to generate diverse network traffic patterns that Network Guard could monitor and analyze.

Monitoring and Analysis: Throughout the testing period, Network Guard continuously monitored the network. The application's ability to detect new devices, identify disconnected devices, and measure bandwidth usage was rigorously tested. Any anomalies or issues were documented for further analysis.

Feedback Collection: Participants were asked to provide feedback on the network performance and any noticeable issues. This feedback was used to assess the effectiveness of Network Guard in a simulated real-world environment.

2.4.4 Observation and Outcome

The initial testing phase was successful, demonstrating that Network Guard could effectively monitor the network environment and provide accurate, real-time data on network activities. The system's ability to detect new and disconnected devices, along with its bandwidth monitoring capabilities, was verified. The simulated real-life scenario provided valuable insights into the system's performance under dynamic network conditions, highlighting its robustness and reliability.
By establishing this controlled lab environment and conducting comprehensive testing, the groundwork was laid for further development and refinement of Network Guard. The results of these tests confirmed that Network Guard is capable of providing a comprehensive view of the network infrastructure, making it a valuable tool for network monitoring.

Evaluation and Results

This section presents the findings from comprehensive evaluations of Network Guard, including performance and usability testing conducted within a controlled laboratory environment. Illustrations and figures are referenced throughout to support the analysis visually.
The results from the performance and usability evaluations indicate that Network Guard is a reliable and efficient tool for network monitoring, capable of supporting network management tasks with accuracy and user-friendly operation

4.1 Performance Evaluation

The performance evaluation of Network Guard was conducted in a controlled lab environment to assess its effectiveness in real-time network monitoring. The primary focus was on the application's ability to detect network devices, monitor bandwidth usage, and maintain accuracy under varying network conditions.

The laboratory setup has been configured to mimic a small office network with eight personal computers connected through a central switch to a router (see ***Figure 4:*** Network Topology). This setup simulates typical network traffic patterns through various activities such as web browsing, data transmission, and video streaming. The dynamic assignment of IP addresses and unrestricted internet access have been critical in replicating a realistic network environment.

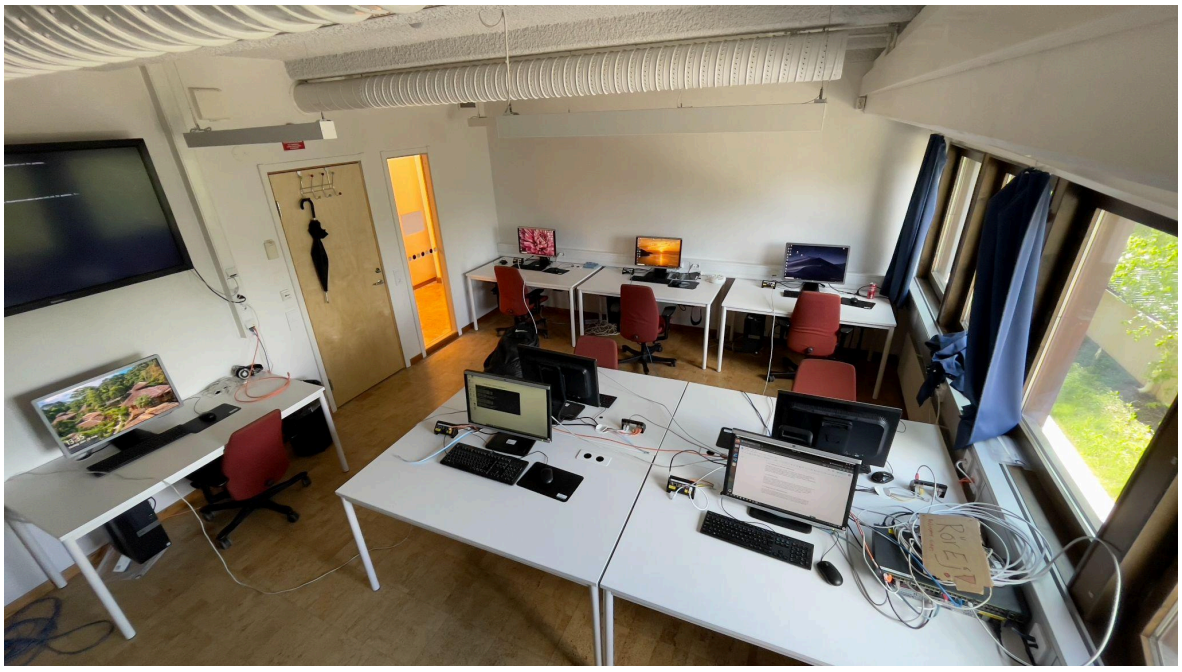

Figure 1: shows what the lab environment looks like

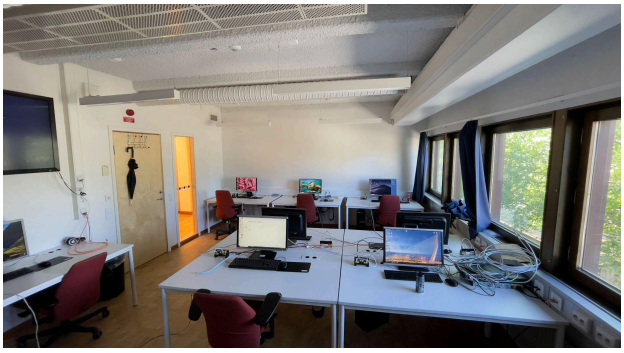

Figure 2: shows a different angle of the environment

Figure 3: shows a different angle of the environment

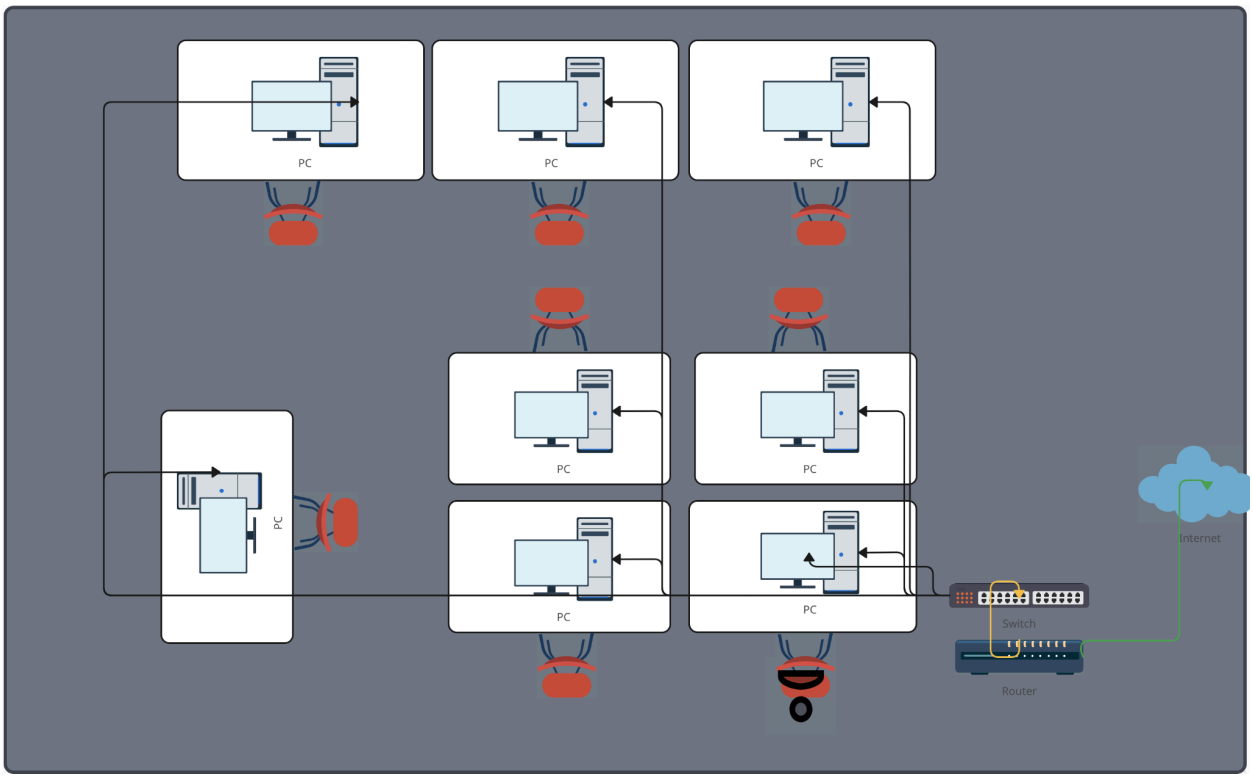

Figure 4: shows the logical topology of the environment

To evaluate Network Guard's performance, various network activities were simulated, including web browsing, sending data, and watching video streaming. These activities generated diverse traffic patterns for Network Guard to analyze. (see ***Figure 5-10***)

Figure 5: Generated traffic from running a game.

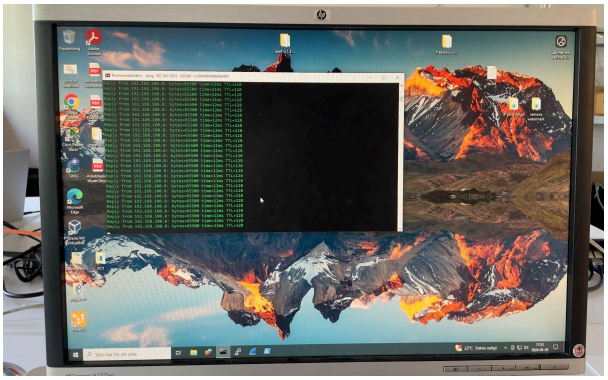

Figure 6 Generated traffic from running Ping

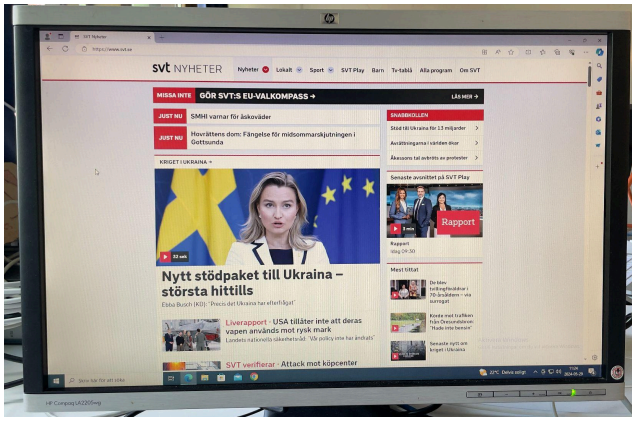

Figure 7: Generated traffic from reading news

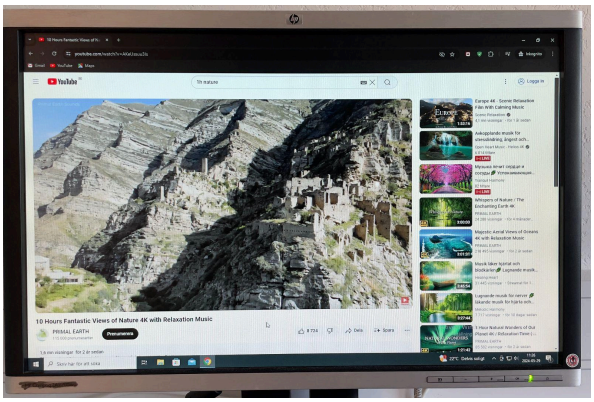

Figure 8: Generated traffic from running youtube

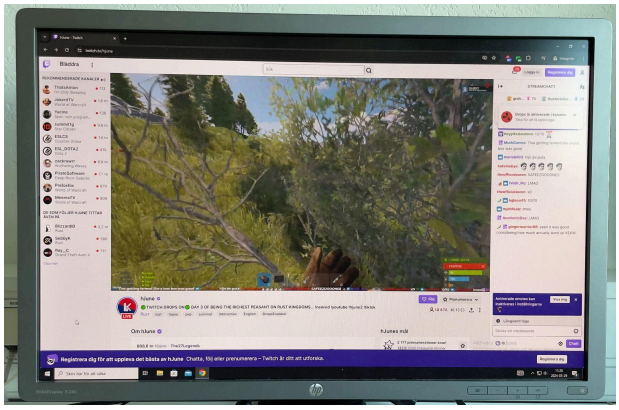

Figure 9: Generated traffic from watching Twitch

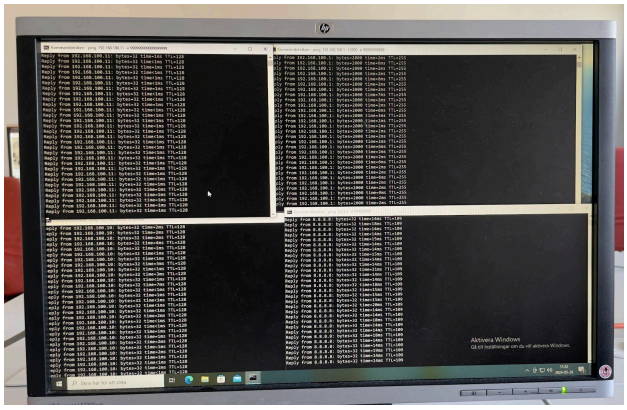

Figure 10: Generated traffic from running several ping

Network Guard has been evaluated based on its ability to detect new and disconnected devices, its accuracy in bandwidth measurement, and its overall responsiveness under different traffic conditions. Tools like Wireshark have been employed to validate the traffic data, capturing 17,119 packets within 3.5 seconds to indicate robust traffic flow management (see ***Figure 11:*** Wireshark Traffic Capture).

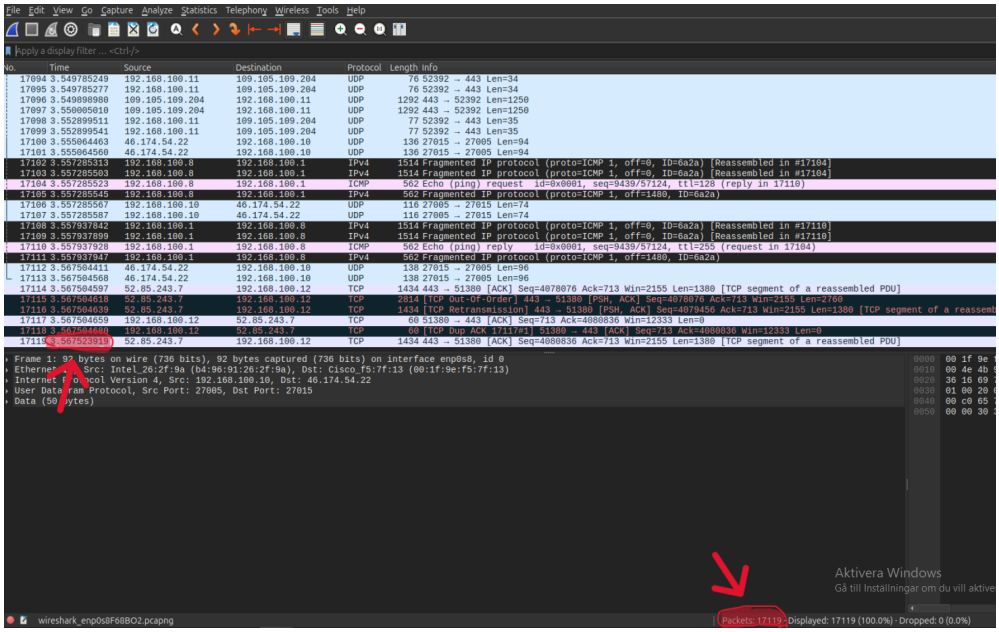

Figure 11: shows that Wireshark has received 17119 packet in 3.5 seconds

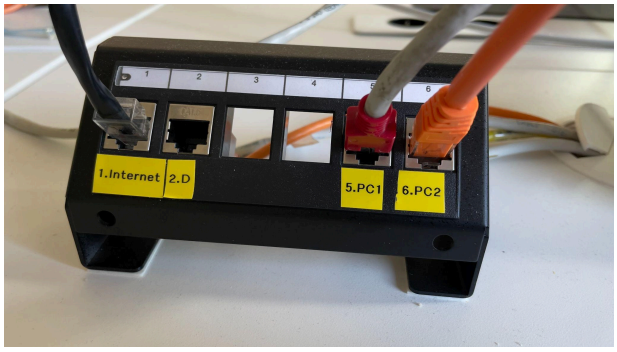

Figure 12: shows the first NIC "PC1" and "PC2"

The Management PC had two NIC interfaces, one for sniffing packets and the other to send ARP and analyze it.

Throughout the testing period, Network Guard continuously monitored the network, detecting new devices, identifying disconnected devices, and measuring bandwidth usage. Notably, the system maintains its performance without degradation, even under varying network conditions, illustrating its suitability for real-world applications. Key highlights include:

Device Detection: Continuous and flawless detection of new and disconnected devices is noted, with updates occurring in real-time (see ***Figures 14 and 16:*** Device Disconnection Alerts).

Bandwidth Monitoring: Network Guard successfully tracks and reports bandwidth usage accurately for each connected device (see ***Figure 13:*** Bandwidth Monitoring Interface).

Traffic Analysis Compatibility: Integration with Wireshark demonstrates the application's ability to accurately capture and analyze network traffic.

Network Guard demonstrated strong performance in monitoring network activities. It accurately detected and reported new devices, promptly identified disconnected devices, and provided real-time bandwidth usage statistics. The application effectively managed the dynamic network environment, maintaining a comprehensive and up-to-date view of the network infrastructure. No significant performance degradation was observed during the testing period, indicating that Network Guard can handle typical network loads without compromising accuracy or speed.

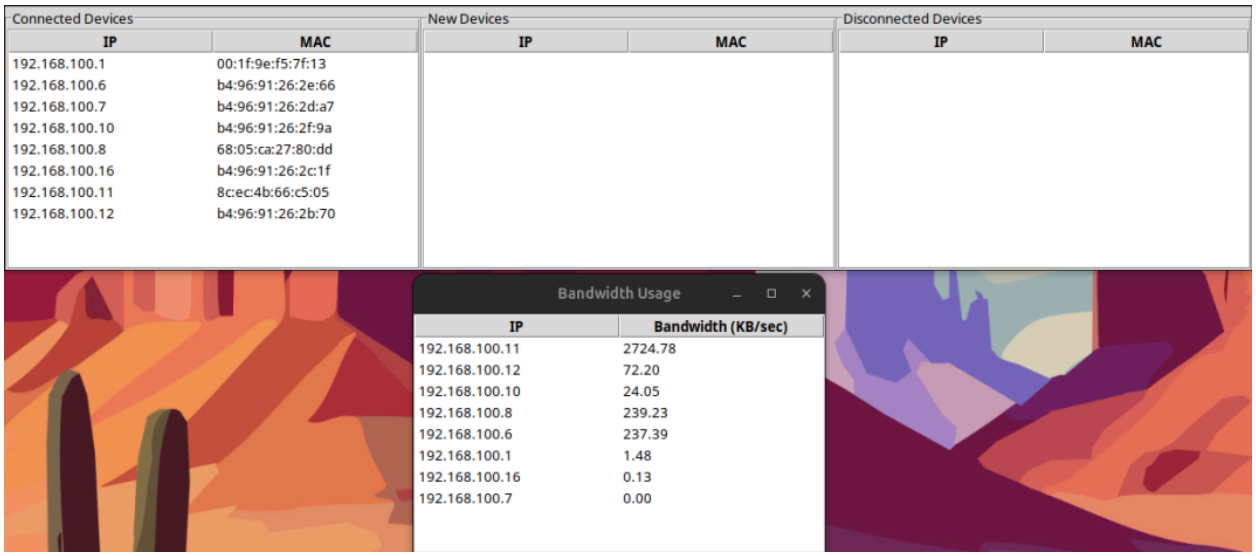

Figure 13: the app shows a visual representation of its interface, listing the connected devices and monitoring the bandwidth usage for each user in KB/sec.

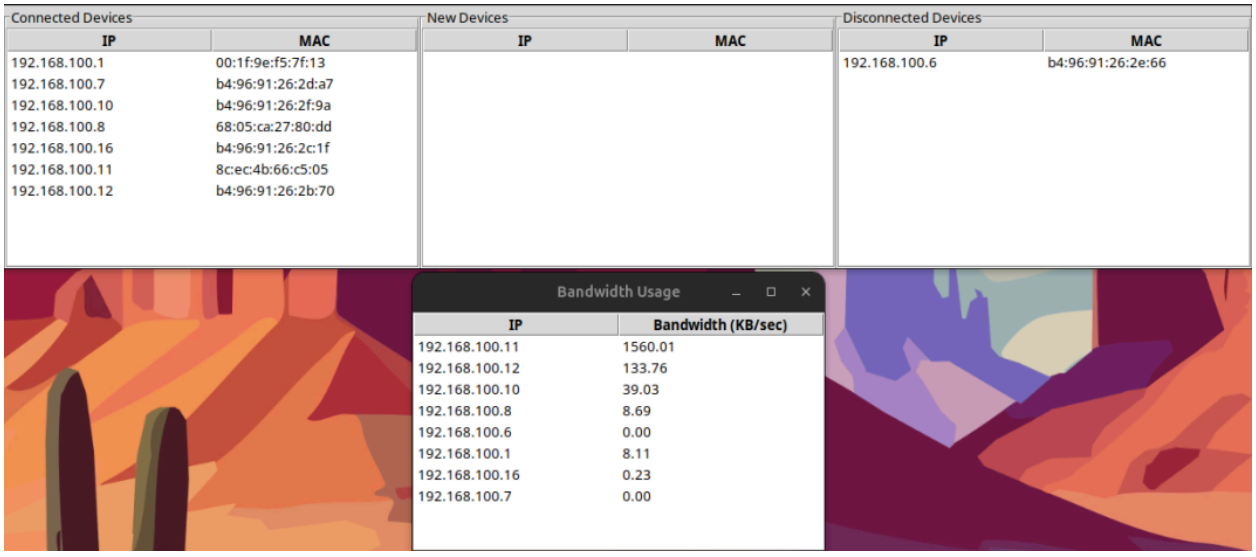

Figure 14: indicate that user with the IP address 192.168.100.6 has disconnected

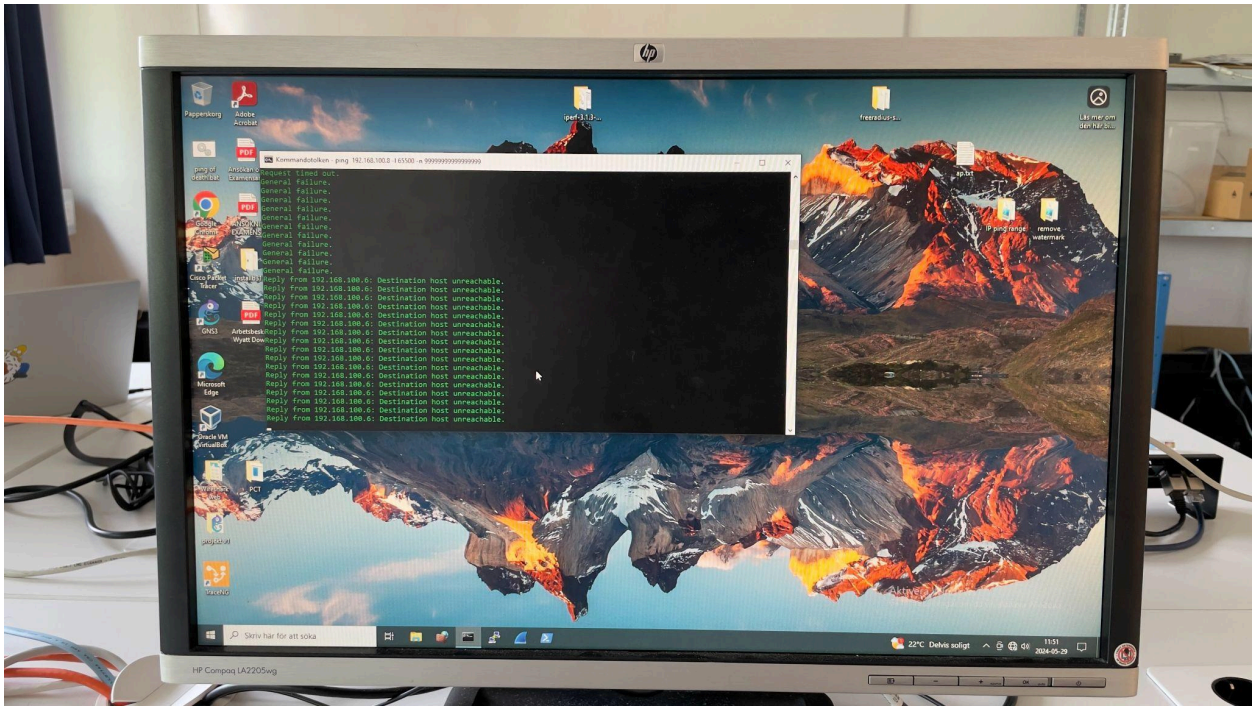

Figure 15: the user with the IP address 192.168.100.6 is unable to establish further connectivity through pings

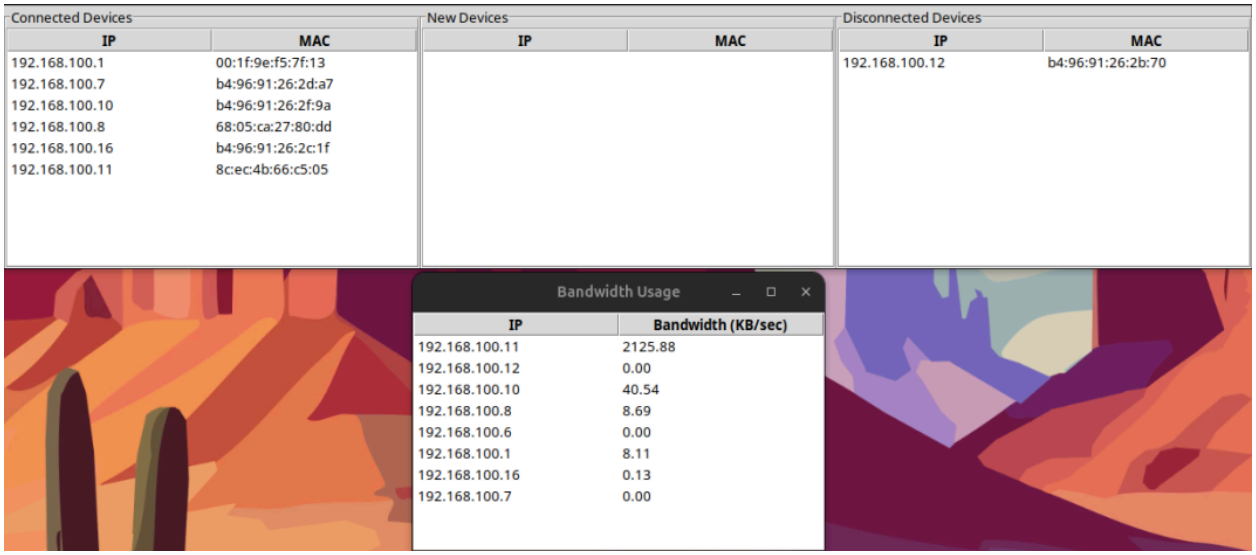

Figure 16: indicate that user with the IP address 192.168.100.12 has disconnected

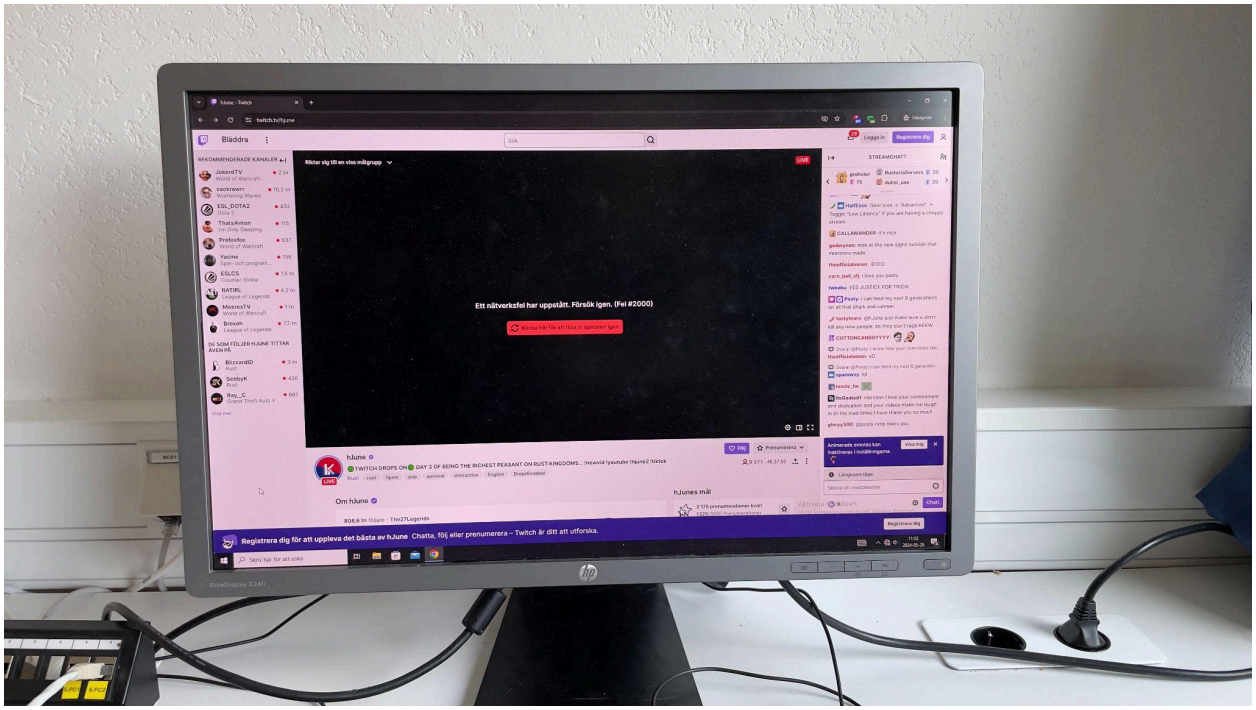

Figure 17: the user with the IP address 192.168.100.12 is unable to access the streaming service

Connected Devices

IP	MAC
192.168.100.1	00:1f:9e:f5:7f:13
192.168.100.7	b4:96:91:26:2d:a7
192.168.100.10	b4:96:91:26:2f:9a
192.168.100.8	68:05:ca:27:80:dd
192.168.100.16	b4:96:91:26:2c:1f
192.168.100.11	8c:ec:4b:66:c5:05
192.168.100.12	b4:96:91:26:2b:70
192.168.100.6	b4:96:91:26:2e:66

New Devices

IP	MAC
192.168.100.6	b4:96:91:26:2e:66

Disconnected Devices

IP	MAC
192.168.100.12	b4:96:91:26:2b:70

Bandwidth Usage

IP	Bandwidth (KB/sec)
192.168.100.11	1893.69
192.168.100.12	961.32
192.168.100.10	53.54
192.168.100.8	36.56
192.168.100.6	51.38
192.168.100.1	8.11
192.168.100.16	0.17
192.168.100.7	0.00

Figure 18: indicate that user with the IP address 192.168.100.6 has established a new connection

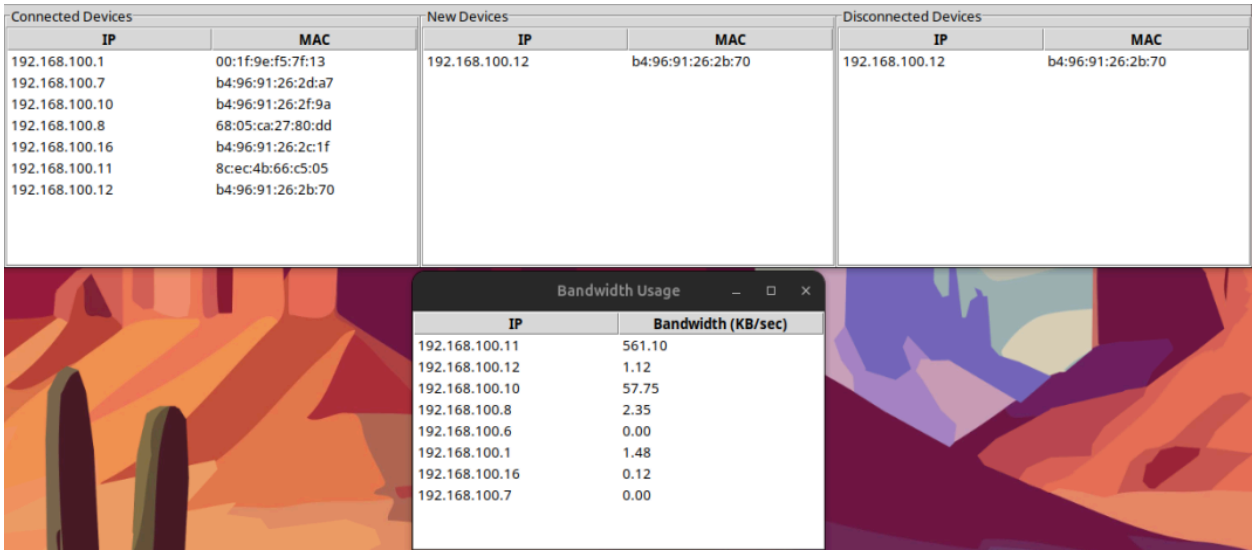

Figure 19: indicate that user with the IP address 192.168.100.12 has established a new connection

4.2 Usability Testing

Usability testing was conducted to evaluate the user interface and overall user experience of Network Guard. The goal was to ensure that the application is intuitive, easy to navigate, and provides meaningful insights to network engineers.

One participant involved in the usability testing was a network engineer student. The participant received a brief tutorial on Network Guard's features and functionalities before commencing the test.

The participant has been tasked with initiating the application, connecting and disconnecting devices, and observing the changes in bandwidth usage displayed by the system (see ***Figures 20-23:*** Usability Testing Screenshots).

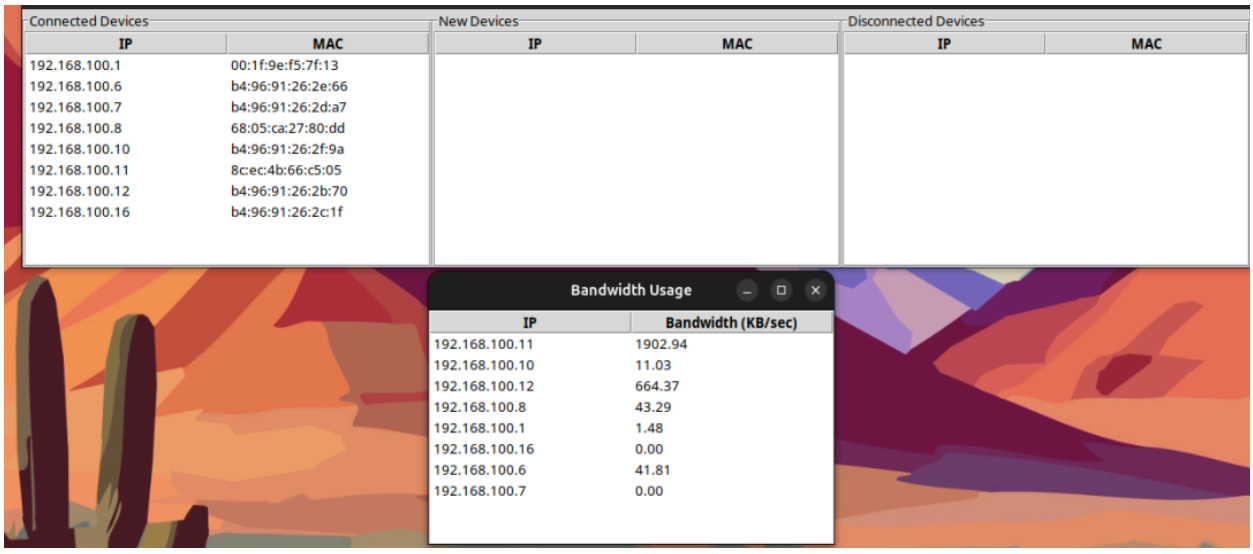

Figure 20: the participant has initiated the application

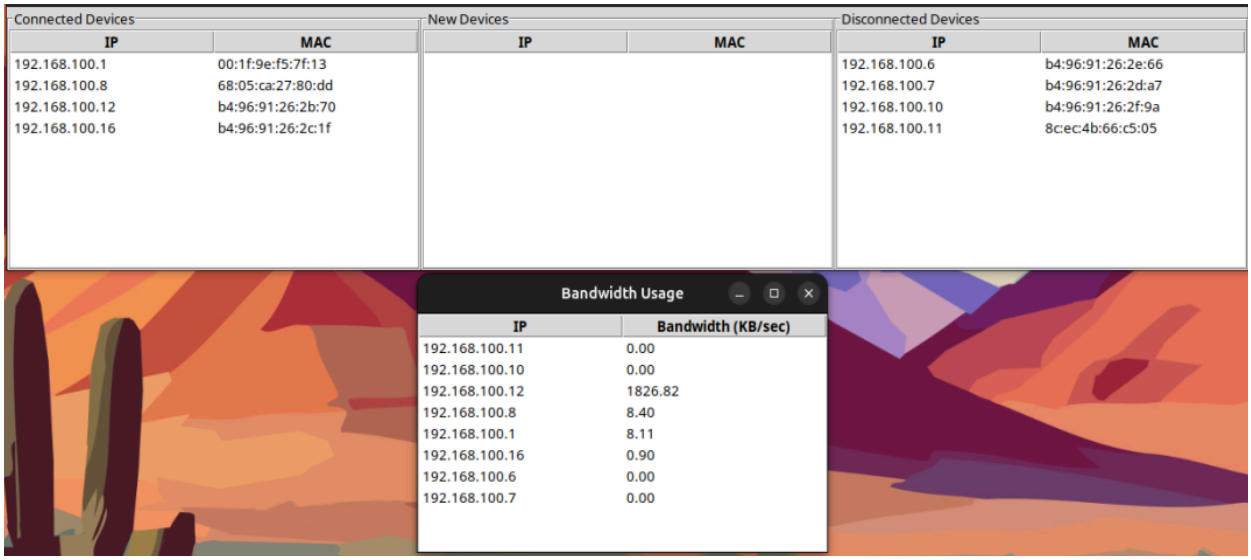

Figure 21: the participant has disconnected four devices and observed that the bandwidth for the disconnected devices is 0

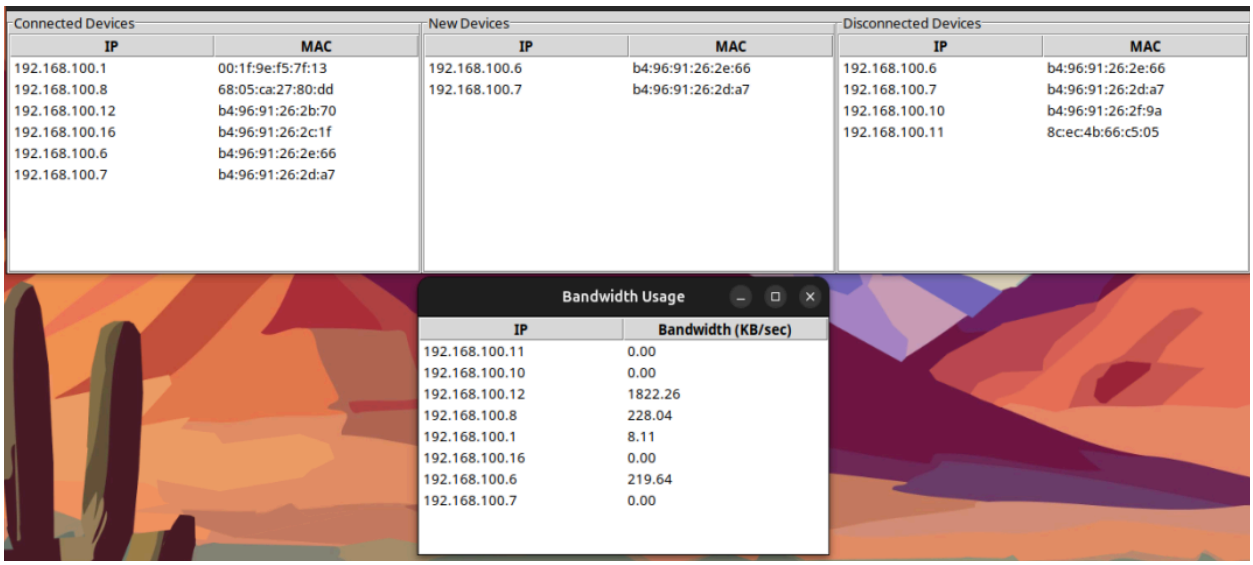

Figure 22: the participant has connected two devices and observed that the bandwidth for the connected devices start to use the bandwidth

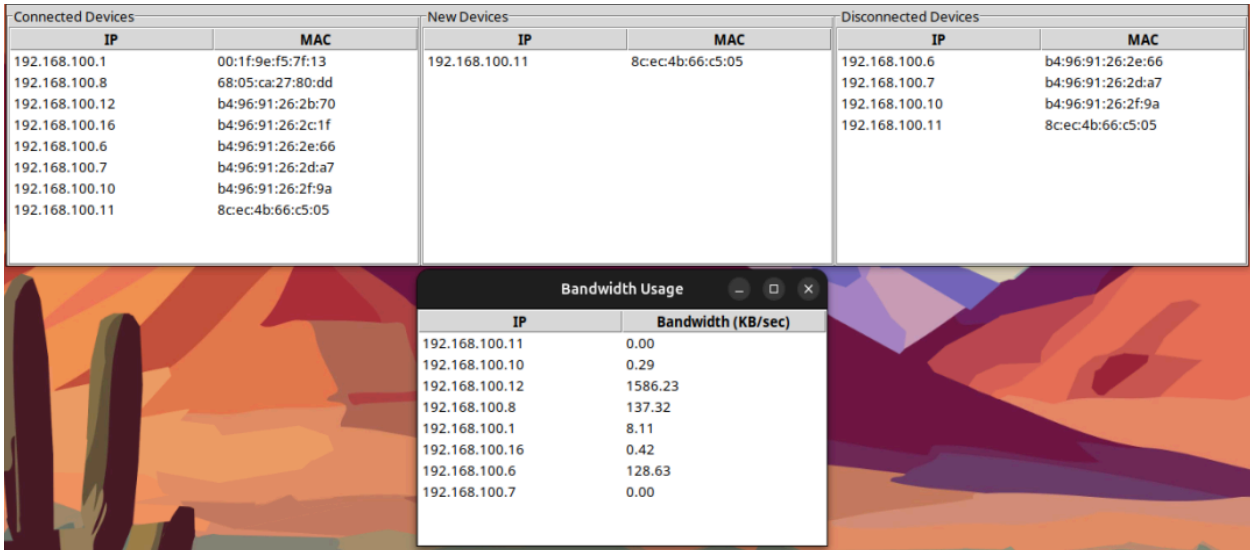

Figure 23: the participant has connected one device.

Discussion

5.1 Comparison with Existing Tools

5.1.1 Network Discovery Protocols

PRTG Network Monitor utilizes ICMP to discover devices on the network, as shown in ***figure 24***. This method involves sending ICMP echo requests (pings) to all IP addresses within the specified range and waiting for replies to determine which devices are active. In contrast, Network Guard employs ARP for device discovery. ARP is used to map IP addresses to MAC addresses, which can be more reliable in certain network environments, especially those that might block ICMP traffic for security reasons.

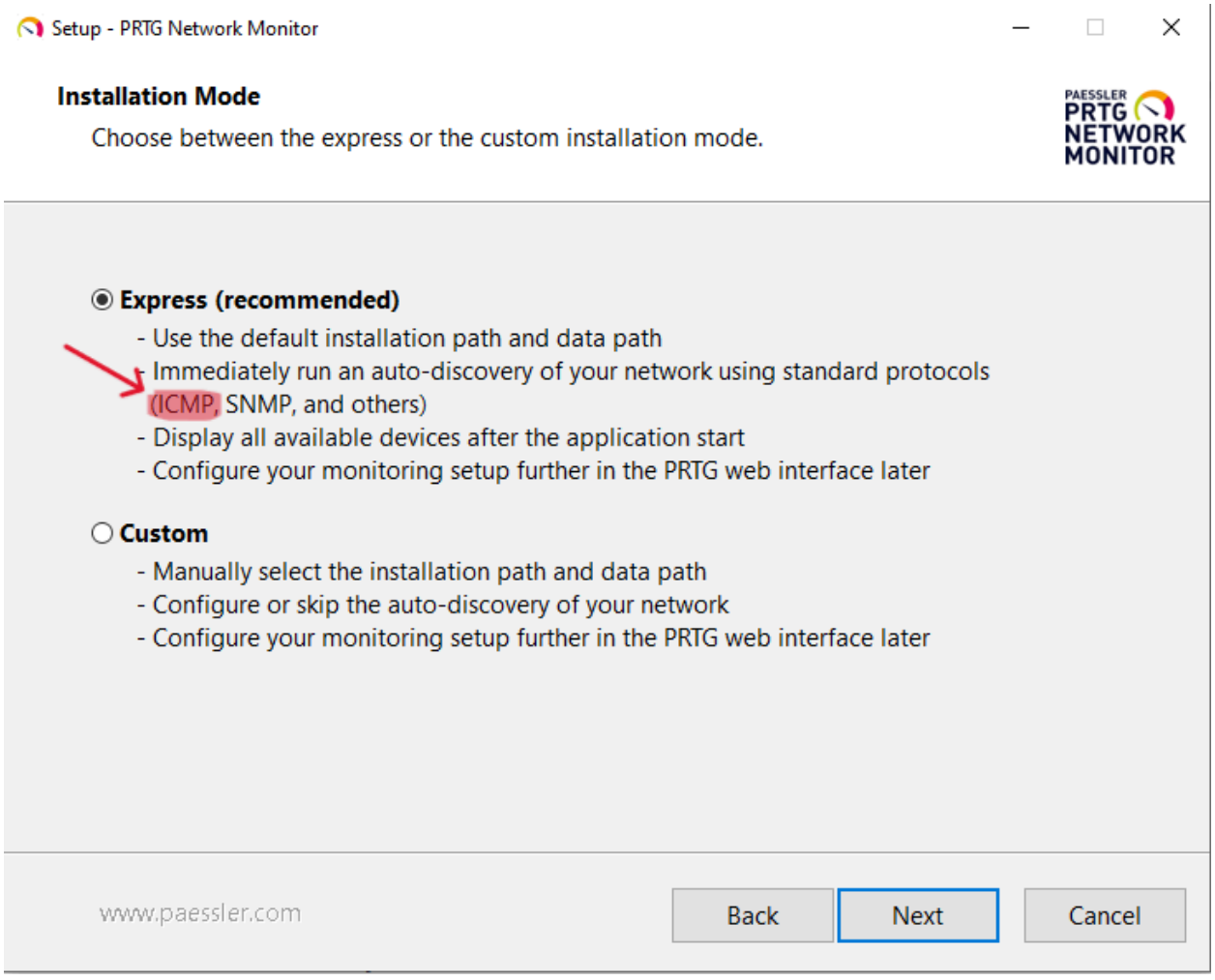

Figure 24: Installation window for PRTG.

5.1.2 Testing Device Connectivity

To evaluate the effectiveness of PRTG's discovery process, we performed a series of tests. Initially, PRTG successfully detected all connected devices in the network, including a device with the IP address 192.168.100.6, as shown in ***figure 25***. This confirmed that PRTG's ICMP-based discovery could accurately identify active devices.

Next, we disconnected the device with IP 192.168.100.6 and observed PRTG's response. PRTG has a built-in update interval of 30 seconds to refresh the device list. After the first 30-second interval, PRTG marked the device as "Warning," indicating a potential connectivity issue (***Figure 26***). Following another 30-second interval, PRTG changed the status of 192.168.100.6 to "Disconnected" (***Figure 27***), demonstrating its capability to detect and update the status of network devices in real time.

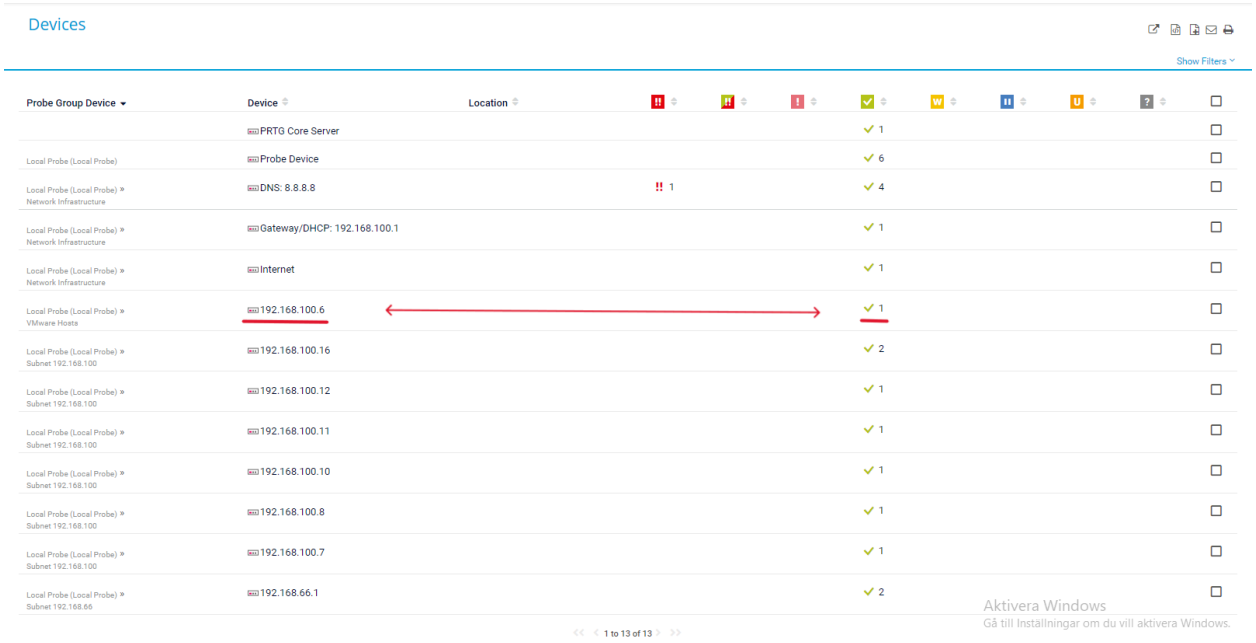

Figure 25: PRTG statistics showing that the host 192.168.100.6 is connected.

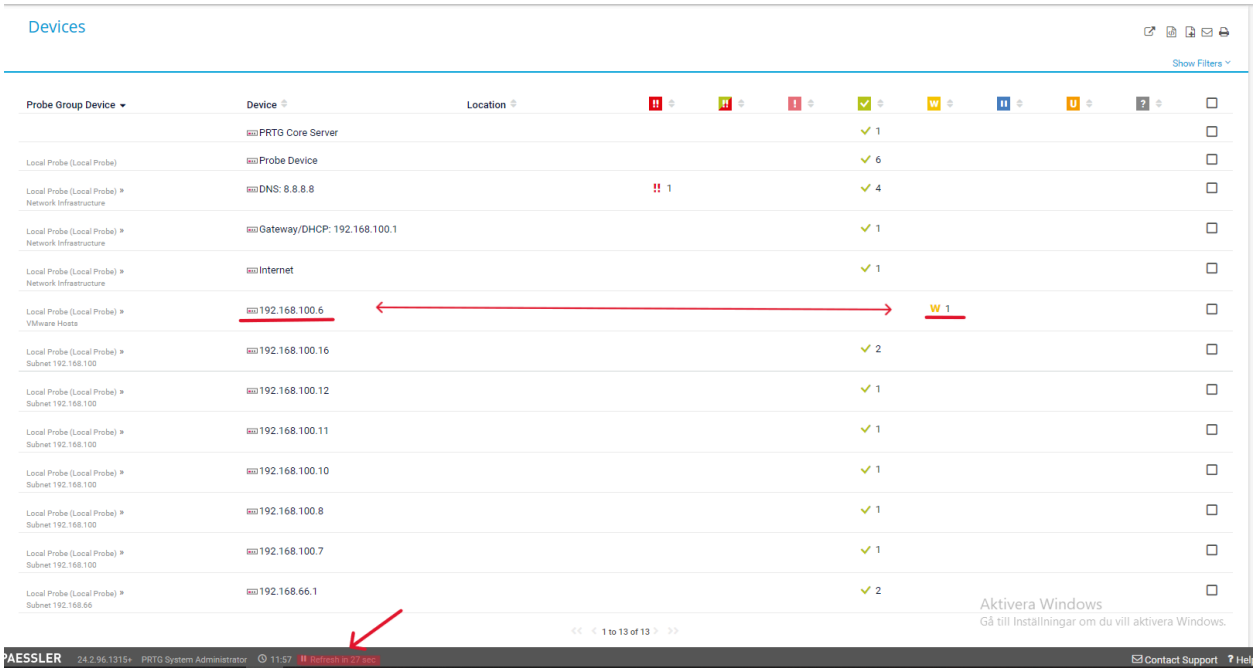

Figure 26: PRTG statistics showing that the state of the host 192.168.100.6 is at “Warning”.

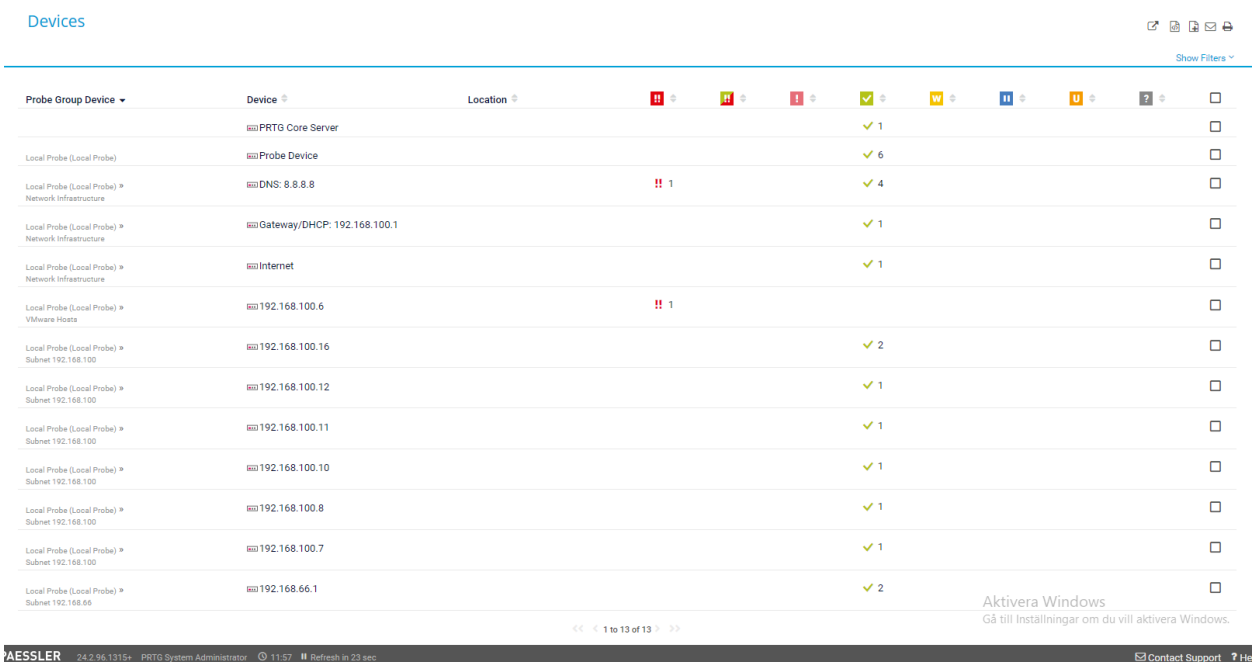

Figure 27: PRTG statistics showing that the state of the host 192.168.100.6 is at “Error”.

5.1.3 Bandwidth Monitoring

When it comes to bandwidth monitoring, PRTG offers a general overview by displaying the highest and lowest bandwidth usage, which can be useful for identifying extreme cases but lacks detailed granularity. On the other hand, Network Guard provides a more detailed analysis by showing the actual bandwidth usage of each connected device. This feature enables network administrators to have a comprehensive view of the network's bandwidth distribution, identifying which devices are consuming the most resources in real-time.

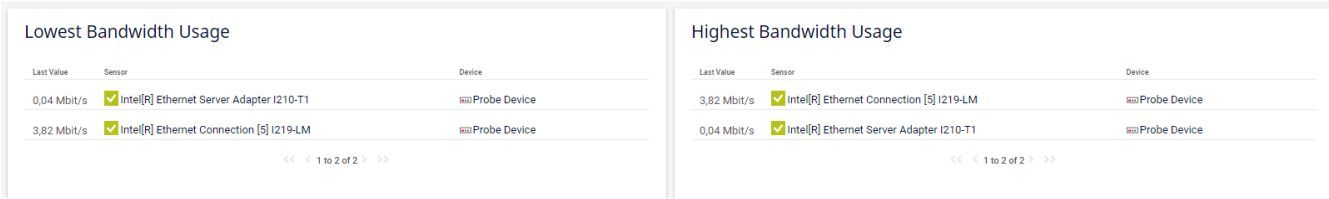

Figure 28: Bandwidth usage displayed on PRTG.

5.1.4 Usability and User Interface

PRTG Interface Complexity: PRTG's extensive features can make the user interface complex, potentially overwhelming for new users. Navigating through various settings and dashboards can require a steep learning curve.

Network Guard's GUI Simplicity: Network Guard aims for simplicity and ease of use, but this can limit the depth of customization and advanced features compared to more established tools like PRTG.

5.1.5 Scalability and Performance

PRTG Scalability: PRTG is designed to handle large-scale networks, but the performance can degrade as the number of monitored devices increases. Ensuring timely updates and accurate monitoring in high-traffic networks can be demanding.

Network Guard's Real-Time Monitoring: Network Guard’s real-time monitoring capability demands significant processing power and efficient handling of network packets, especially in environments with high device turnover and traffic volume.

5.2 Challenges and Opportunities

5.2.1 Challenges

Requirement for Dual Network Interface Cards (NICs): One significant challenge encountered during the development of Network Guard was the necessity for the system to be equipped with two Network Interface Cards (NICs). This requirement stems from the need for one NIC to perform network sniffing, capturing packets to monitor the network traffic, while the other NIC is used to actively discover devices on the network using ARP. This dual-NIC setup ensures that the network monitoring and discovery processes do not interfere with each other, maintaining the integrity and performance of both tasks. However, the need for two NICs can be a limitation, particularly for users with hardware constraints or in environments where additional NICs are not readily available.

Limited Testing on Real Networks: Another challenge was the inability to apply and thoroughly test the application on a real, functioning network. Testing in a live network environment is crucial for understanding how the application performs under actual conditions, including varying traffic loads, network configurations, and potential security threats. However, access to a real network was restricted, limiting the scope of testing primarily to simulated environments. This limitation means that while preliminary tests indicate promising results, the application's performance, scalability, and reliability in real-world scenarios remain to be fully validated.

Legal and Ethical Considerations: Conducting network monitoring and device discovery involves capturing and analyzing network traffic, which can raise significant legal and ethical issues. One of the primary challenges faced was obtaining the necessary permissions to test the application on any network. Without explicit authorization, network monitoring can be deemed illegal and intrusive, violating privacy and security regulations. This legal constraint not only limited the testing opportunities but also necessitated stringent adherence to ethical guidelines and regulatory compliance, ensuring that any deployment of Network Guard respects privacy and legal standards.

5.2.2 Opportunities

Scalability and Extensibility: Despite the challenges, Network Guard is developed with a scalable and flexible architecture, designed to adapt to the evolving needs of network monitoring. The application’s codebase is modular, allowing for easy addition of new functionalities or adjustments to existing ones. This design ensures that as network environments grow and new monitoring requirements emerge, Network Guard can be quickly updated and enhanced without extensive rework. For example, integrating new network protocols, adding advanced analytics features, or improving user interface components can be seamlessly incorporated into the existing framework.

Future Development and Enhancements: The scalable nature of Network Guard presents numerous opportunities for future development. Potential enhancements include incorporating support for additional network discovery protocols such as SNMP or NetFlow, which can provide more comprehensive network insights. Additionally, advanced analytics and reporting features could be developed to offer deeper visibility into network performance and potential issues. These enhancements would not only improve the

functionality and reliability of Network Guard but also position it as a versatile tool capable of meeting diverse network monitoring needs.

Broadening Testing Environments: Moving forward, securing opportunities to test Network Guard in a variety of real-world network environments will be crucial. Collaborating with organizations that can provide access to their networks under controlled and authorized conditions will allow for more extensive validation of the application's capabilities. These partnerships can offer valuable feedback, highlighting areas for improvement and verifying the application's effectiveness in different network scenarios. Real-world testing will also help in fine-tuning the application's performance and addressing any unforeseen challenges that may arise in live environments.

Compliance and Ethical Use: Establishing clear guidelines and obtaining necessary permissions for network monitoring will ensure that Network Guard is used ethically and legally. Developing robust documentation and consent processes can facilitate smoother deployment in organizational networks. Furthermore, educating users about the legal implications and ethical considerations of network monitoring can foster responsible use of the application. By prioritizing compliance and ethical practices, Network Guard can build trust with users and stakeholders, promoting its adoption in various sectors.

By addressing these challenges head-on and leveraging the identified opportunities, Network Guard can evolve into a more comprehensive and robust network monitoring solution. Continuous development, real-world testing, and adherence to legal and ethical standards will enhance its capabilities and ensure it meets the diverse needs of modern network environments.

Conclusion

6.1 Summary of Results

The development and evaluation of Network Guard resulted in several significant findings:

Network Guard successfully implemented network discovery features that enable the identification of devices and connections within a network. Using Scapy, the tool was able to detect active devices, map their connections, and monitor their status. This capability is crucial for maintaining an up-to-date view of the network infrastructure, which is essential for effective network management and troubleshooting.

One of the core strengths of Network Guard is its real-time monitoring capabilities. The tool was designed to capture and analyze network traffic in real time, providing immediate insights into bandwidth usage and potential network issues. This real-time analysis helps network administrators quickly identify and address performance bottlenecks and other anomalies, thereby reducing downtime and improving overall network performance.

A key objective of this project was to develop a user-friendly interface that allows easy access to network data and diagnostic tools. Network Guard's interface, built using Tkinter, provides a clear and intuitive layout for viewing network statistics, configuring monitoring parameters, and interpreting diagnostic results. Usability testing confirmed that the interface meets the needs of network administrators, offering a straightforward means of interacting with the monitoring system.

Network Guard was designed with scalability in mind, ensuring that it can adapt to growing and evolving network environments. The modular architecture of the application allows for easy integration of new functionalities and protocols, making it a flexible solution that can be tailored to specific network requirements. This adaptability ensures that Network Guard remains relevant and effective even as network infrastructures expand and new technologies are introduced.

Through extensive testing in controlled environments, Network Guard demonstrated reliable performance across various network scenarios. The tool's ability to handle high volumes of traffic and maintain accurate monitoring results under different conditions underscores its robustness and dependability as a network monitoring solution.

When compared to existing network monitoring solutions, Network Guard stands out due to its open-source nature, ease of customization, and the powerful combination of Python and Scapy. While traditional tools often come with limitations in terms of flexibility and cost, Network Guard offers a cost-effective alternative that does not compromise on functionality or performance.

In conclusion, the development of Network Guard has proven to be a significant contribution to the field of network monitoring. Its ability to provide real-time diagnostics, coupled with a scalable and user-friendly design, positions it as a valuable tool for ensuring network stability and performance. The results of this thesis highlight the potential for further enhancements and the broader applicability of Network Guard in diverse network environments.

Reference

[1] *python.org, (n.d.), "What is Python? Executive Summary", Retrieved May 7, 2024, from: https://www.python.org/doc/essays/blurb/*

[2] *Philippe Biondi and the Scapy community, (n.d.), "About Scapy", Retrieved May 8, 2024, from: https://scapy.readthedocs.io/en/latest/introduction.html#about-scapy*

[3] *David Amos, (n.d.), "Python GUI Programming With Tkinter", Retrieved May 8, 2024, from: https://realpython.com/python-gui-tkinter/*

[4] *W. Richard Stevens, (2011), "TCP/IP Illustrated, Volume 1: The Protocols", Addison Wesley, 2 ISBN 0-321-33631-3, Retrieved May 14, 2024*

[5] *RJ Atkinson, SN Bhatti and U. St Andrews. (November 2012). "Address Resolution Protocol (ARP) for the Identifier-Locator Network Protocol for IPv4 (ILNPv4)", Retrieved May 17, 2024. From: https://www.rfc-editor.org/rfc/rfc6747*

[6] *William Stallings, (2007). "Data and Computer Communications". ISBN 8131715361. Retrieved May 19, 2024*

[7] *David Cao, (July 8, 2023). "Understanding Ping Command and ICMP with Examples". Retrieved May 19, 2024, from : https://www.howtouselinux.com/post/ping-icmp*

[8] *Cisco.com, (n.d.), "What is LAN?". Retrieved May 20, 2024, from : https://www.cisco.com/c/en/us/products/switches/what-is-a-lan-local-area-network.html*

[9] *lisedunetwork.com, (2014) Updated:December 28, 2023, "Components of Local Area Network (LAN)". Retrieved May 29, 2024, from : https://www.lisedunetwork.com/components-of-local-area-network-lan/*

Source Code

```
import tkinter as tk
from tkinter import ttk
from threading import Thread, Event, Lock
from scapy.all import ARP, Ether, srp, sniff, IP
from time import sleep


class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.packet_sizes = {}
        self.packet_sizes_lock = Lock()
        self.connected_devices = []
        self.bandwidth_tree_ready = Event()
        self.create_widgets()
        self.start_monitoring()

    def create_widgets(self):
        self.connected_frame = tk.LabelFrame(self, text="Connected Devices")
        self.connected_frame.pack(side="left")
        self.connected_tree = ttk.Treeview(self.connected_frame, columns=("IP", "MAC"), show="headings")
        self.connected_tree.heading("IP", text="IP")
        self.connected_tree.heading("MAC", text="MAC")
        self.connected_tree.pack()
        self.new_frame = tk.LabelFrame(self, text="New Devices")
        self.new_frame.pack(side="left")
        self.new_tree = ttk.Treeview(self.new_frame, columns=("IP", "MAC"), show="headings")
        self.new_tree.heading("IP", text="IP")
        self.new_tree.heading("MAC", text="MAC")
        self.new_tree.pack()
        self.disconnected_frame = tk.LabelFrame(self, text="Disconnected Devices")
        self.disconnected_frame.pack(side="left")
        self.disconnected_tree = ttk.Treeview(self.disconnected_frame, columns=("IP", "MAC"), show="headings")
        self.disconnected_tree.heading("IP", text="IP")
        self.disconnected_tree.heading("MAC", text="MAC")
        self.disconnected_tree.pack()
```

```
def start_monitoring(self):
    self.monitor_thread = Thread(target=self.monitor_devices)
    self.monitor_thread.start()
    self.sniff_thread = Thread(target=self.sniff_packets)
    self.sniff_thread.start()
    self.create_bandwidth_window()

def get_devices(self, interface='enp0s3'):
    arp = ARP(pdst='192.168.100.0/24')
    ether = Ether(dst='ff:ff:ff:ff:ff:ff')
    packet = ether/arp
    result = srp(packet, timeout=2, iface=interface, verbose=0)[0]
    devices = []
    for sent, received in result:
        devices.append((received.psrc, received.hwsrc))
    return devices


def monitor_devices(self, interface='enp0s3'):
    self.connected_devices = self.get_devices(interface)
    while True:
        current_devices = self.get_devices(interface)
        disconnected_devices = [d for d in self.connected_devices if d not in current_devices]
        new_devices = [d for d in current_devices if d not in self.connected_devices]

        if disconnected_devices:
            for item in self.disconnected_tree.get_children():
                self.disconnected_tree.delete(item)

            for device in disconnected_devices:
                self.disconnected_tree.insert('', 'end', values=device)

            self.connected_devices = [d for d in self.connected_devices if d not in disconnected_devices]

        if new_devices:
            for item in self.new_tree.get_children():
                self.new_tree.delete(item)

            for device in new_devices:
                self.new_tree.insert('', 'end', values=device)

            self.connected_devices.extend(new_devices)
```

```
        for item in self.connected_tree.get_children():
            self.connected_tree.delete(item)

        for device in self.connected_devices:
            self.connected_tree.insert('', 'end', values=device)

        sleep(1)

def calculate_bandwidth(self, packet):
    if IP in packet:
        src_ip = packet[IP].src
        dst_ip = packet[IP].dst

        with self.packet_sizes_lock:
            if any(src_ip == d[0] for d in self.connected_devices):
                self.packet_sizes[src_ip] = self.packet_sizes.get(src_ip, 0) + len(packet)

            if any(dst_ip == d[0] for d in self.connected_devices):
                self.packet_sizes[dst_ip] = self.packet_sizes.get(dst_ip, 0) + len(packet)


def sniff_packets(self):
    sniff(iface='enp0s8', prn=self.calculate_bandwidth)

def create_bandwidth_window(self):
    self.bandwidth_window = tk.Toplevel(self.master)
    self.bandwidth_window.title("Bandwidth Usage")
    self.bandwidth_tree = ttk.Treeview(self.bandwidth_window, columns=("IP", "Bandwidth"), show="headings")
    self.bandwidth_tree.heading("IP", text="IP")
    self.bandwidth_tree.heading("Bandwidth", text="Bandwidth (KB/sec)")
    self.bandwidth_tree.pack()
    self.bandwidth_tree_ready.set()
    self.bandwidth_thread = Thread(target=self.print_bandwidth_usage)
    self.bandwidth_thread.start()

def print_bandwidth_usage(self):
    self.bandwidth_tree_ready.wait()

    while True:
        for item in self.bandwidth_tree.get_children():
            self.bandwidth_tree.delete(item)
```

```
            with self.packet_sizes_lock:
                for ip, size in self.packet_sizes.items():
                    bandwidth = (size / 1024) / 1.0
                    self.bandwidth_tree.insert('', 'end', values=(ip, "{:.2f}".format(bandwidth)))
                    self.packet_sizes[ip] = 0

            sleep(1)

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

Device Configurations

9.1 Router Configuration

```
Building configuration...

Current configuration : 1364 bytes
!
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname R1
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$8SVk$SG8XyX2BrmVe7eUEaWLCm.
!
no aaa new-model
memory-size iomem 15
ip cef
!
!
no ip dhcp use vrf connected
ip dhcp excluded-address 192.168.100.1
ip dhcp excluded-address 192.168.100.99
!
ip dhcp pool F1
  network 192.168.100.0 255.255.255.0
  default-router 192.168.100.1
  dns-server 8.8.8.8
!
!
ip auth-proxy max-nodata-conns 3
ip admission max-nodata-conns 3
!
!
!
!
```

```
!
!
!
!
!
interface FastEthernet0/0
 ip address 192.168.0.113 255.255.255.0
 ip nat outside
 ip virtual-reassembly
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 192.168.100.1 255.255.255.0
 ip nat inside
 ip virtual-reassembly
 duplex auto
 speed auto
!
interface Serial0/0/0
 no ip address
 shutdown
 clock rate 125000
!
interface Serial0/0/1
 no ip address
 shutdown
 clock rate 125000
!
ip forward-protocol nd
ip route 0.0.0.0 0.0.0.0 192.168.0.1
!
!
ip http server
no ip http secure-server
ip nat pool INTERNET 192.168.0.113 192.168.0.113 netmask 255.255.255.0
ip nat inside source list EX pool INTERNET overload
!
ip access-list extended EX
 permit ip any any
!
!
!
```

```
!
!
control-plane
!
!
!
line con 0
line aux 0
line vty 0 4
 login
!
scheduler allocate 20000 1000
end
```

9.2 Switch Configuration

```
Building configuration...

Current configuration : 2085 bytes
!
version 12.2
no service pad
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname Switch
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$H.U8$xvelDeA/MfQ82t4ngucnW0
!
!
!
no aaa new-model
system mtu routing 1500
!
!
!
!
```

```
!
!
!
!
!
spanning-tree mode pvst
spanning-tree extend system-id
no spanning-tree vlan 1
!
vlan internal allocation policy ascending
!
!
!
interface FastEthernet0/1
!
interface FastEthernet0/2
!
interface FastEthernet0/3
!
interface FastEthernet0/4
!
interface FastEthernet0/5
!
interface FastEthernet0/6
!
interface FastEthernet0/7
!
interface FastEthernet0/8
!
interface FastEthernet0/9
!
interface FastEthernet0/10
!
interface FastEthernet0/11
!
interface FastEthernet0/12
!
interface FastEthernet0/13
!
interface FastEthernet0/14
!
interface FastEthernet0/15
!
```

```
interface FastEthernet0/16
!
interface FastEthernet0/17
!
interface FastEthernet0/18
!
interface FastEthernet0/19
!
interface FastEthernet0/20
!
interface FastEthernet0/21
!
interface FastEthernet0/22
!
interface FastEthernet0/23
!
interface FastEthernet0/24
!
interface FastEthernet0/25
!
interface FastEthernet0/26
!
interface FastEthernet0/27
!
interface FastEthernet0/28
!
interface FastEthernet0/29
!
interface FastEthernet0/30
!
interface FastEthernet0/31
!
interface FastEthernet0/32
!
interface FastEthernet0/33
!
interface FastEthernet0/34
!
interface FastEthernet0/35
!
interface FastEthernet0/36
!
interface FastEthernet0/37
```

```
!
interface FastEthernet0/38
!
interface FastEthernet0/39
!
interface FastEthernet0/40
!
interface FastEthernet0/41
!
interface FastEthernet0/42
!
interface FastEthernet0/43
!
interface FastEthernet0/44
!
interface FastEthernet0/45
!
interface FastEthernet0/46
!
interface FastEthernet0/47
!
interface FastEthernet0/48
!
interface GigabitEthernet0/1
!
interface GigabitEthernet0/2
!
interface Vlan1
 no ip address
!
ip http server
ip http secure-server
!
line con 0
line vty 5 15
!
!
monitor session 1 source interface Fa0/1 , Fa0/3 - 48
monitor session 1 destination interface Fa0/2
end
```