Dynamic Network Firewall

****

Session: 2022 – 2025

**Submitted by:**

## Irtaza Manzoor 2022-CS-189 Mohsin Ali 2022-CS-193

**Supervised by:**

## Sir Tehseen Ul Hassan Shah

## Department of Computer Sciences

**University of Engineering and Technology, Lahore, Pakistan**

## Contents

1. [Introduction 3](#_TOC_250010)
2. [Project Description 3](#_TOC_250009)
3. [Methodology Used 3](#_TOC_250008)
4. [Key Features and Functionality 4](#_TOC_250007)
5. [Technologies and Tools Used 5](#_TOC_250006)
6. [Flow Chart Diagram 5](#_TOC_250005)
7. [Tool Screenshots 7](#_TOC_250004)
8. [Results and Analysis 8](#_TOC_250003)
9. [Conclusion 8](#_TOC_250002)
10. [Github Link 9](#_TOC_250001)
11. [References 9](#_TOC_250000)

# Introduction

This project aims to develop a comprehensive network packet capture and filtering application using Python. The primary purpose is to provide users with real-time monitoring and control over their network traffic. The significance of this project lies in enhancing network security and helping users understand and manage their network activity. The motivation behind this project is the increasing need for individuals and organizations to safeguard their networks against unauthorized access and potential threats.

# Project Description

The main goal of this project is to create a user-friendly application that captures and filters network packets based on user-defined rules. Objectives include:

* + Developing a real-time packet capture system.
  + Implementing filtering rules based on IP addresses, ports, protocols, and programs.
  + Logging and visualizing network traffic.

The scope of the project includes real-time packet capture, customizable filtering rules, logging, and basic visualization. Advanced intrusion detection and extensive reporting features are outside the scope.

Background context: As internet usage grows, so does the complexity of network security. This project seeks to empower users to take control of their network traffic in a straightforward manner.

# Methodology Used

### Packet Capture:

* + Utilized the pydivert library for capturing network packets in real-time.
  + Created a packet capture loop that continuously captures packets, checks them against the defined rules, and either allows or blocks them based on the rules.

### Rule Definition:

* + Implemented a user interface to allow users to define filtering rules.
  + Rules can be set for ports, IP addresses, protocols, and specific programs. The logic to apply these rules was embedded in the packet capture loop.

### Packet Logging:

* + Maintained a log of all captured packets with detailed information.
  + Logged details include source and destination IPs, ports, protocol, direction, and status. Provided options to filter and view logged packets.

### Real-time Statistics:

* + Displayed real-time packet statistics on the GUI.
  + Tracked and displayed total packets, blocked packets, and allowed packets.

### Live Visualization:

* + Used matplotlib for live chart visualization.
  + Plotted the number of blocked and allowed packets over time, updating the chart at regular intervals to provide real-time insights.

# Key Features and Functionality

1. **Real-time Packet Capture**: Captures network packets in real-time using pydivert.
2. **Customizable Rules**: Users can define rules based on ports, IP addresses, protocols, and programs.
3. **Packet Logging**: Logs captured packets and provides filtering options to view specific logs.
4. **Real-time Statistics**: Displays total, blocked, and allowed packet counts.
5. **Live Chart Visualization**: Plots a live chart showing the number of blocked and allowed packets over time.
6. **CSV Export**: Allows users to save the packet logs to a CSV file for further analysis.

### Unique Aspects:

* + The integration of customizable rules makes this system versatile and user-friendly.
  + Real-time visualization helps users quickly understand network patterns and detect anomalies.

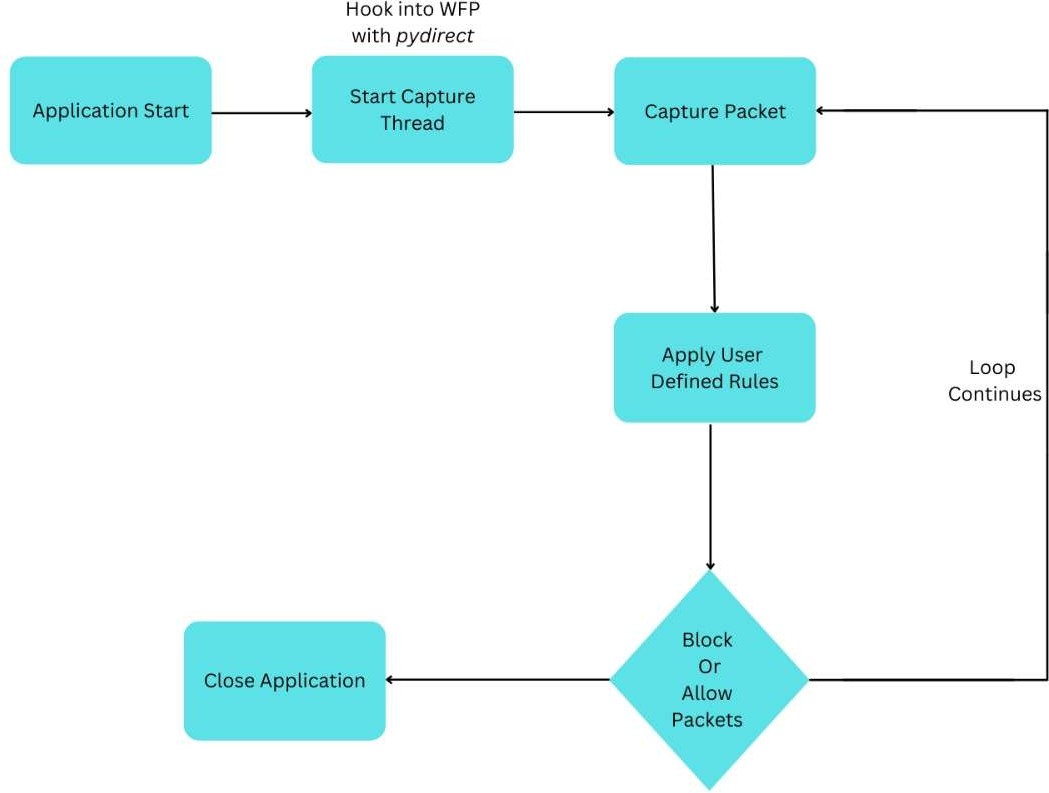
### User Interaction Examples:

* + Users define filtering rules via the GUI.
  + Users start and stop packet capture with simple controls.
  + Users view real-time statistics and live charts directly within the application.

# Technologies and Tools Used

1. **Python**: The main programming language, chosen for its simplicity and extensive libraries.
2. **pydivert:** Used for capturing and filtering network packets due to its efficiency and ease of use.
3. **psutil:** Used to match packets to specific programs, providing deeper insight into the traffic.
4. **tkinter:** Employed for building the graphical user interface, enabling user interactions.
5. **matplotlib:** Selected for plotting live charts to visualize packet traffic effectively.
6. **csv:** Utilized for exporting packet logs to a CSV file, facilitating further analysis and reporting.
7. **Threading:**Used to ensure the packet capture process runs concurrently with the GUI. By using threading, the application can handle real-time packet capture and user interactions without freezing or slowing down. A dedicated thread manages the packet capture, allowing the main thread to handle the user interface and other tasks efficiently.

# Flow Chart Diagram

****

### Flowchart: Packet Capture and Filtering

### Start Application:

* + Initialize the application and GUI.

### Start Capture Thread:

* + Begin packet capture by starting a dedicated thread that uses pydivert to interface with WFP.

### Capture Packets:

* + **Intercept Packets**: WFP intercepts packets at the network stack.
  + **Pass to pydivert**: Intercepted packets are passed to pydivert for processing.

### Apply Filtering Rules:

* + **User-defined Rules**: Packets are checked against user-defined rules:
    - **IP Address**: Matches destination IP.
    - **Port**: Matches destination port.
    - **Protocol**: Matches protocol (TCP/UDP).
    - **Program**: Matches the program creating the traffic (using psutil).

### Log Packet Details:

* + Record details (source IP, destination IP, ports, protocol, direction, status) into logs.

### Block or Allow Packet:

* + **Block Packet**: If packet matches blocking criteria, it is dropped.
  + **Allow Packet**: If packet is allowed by rules, it is sent back to the network stack.

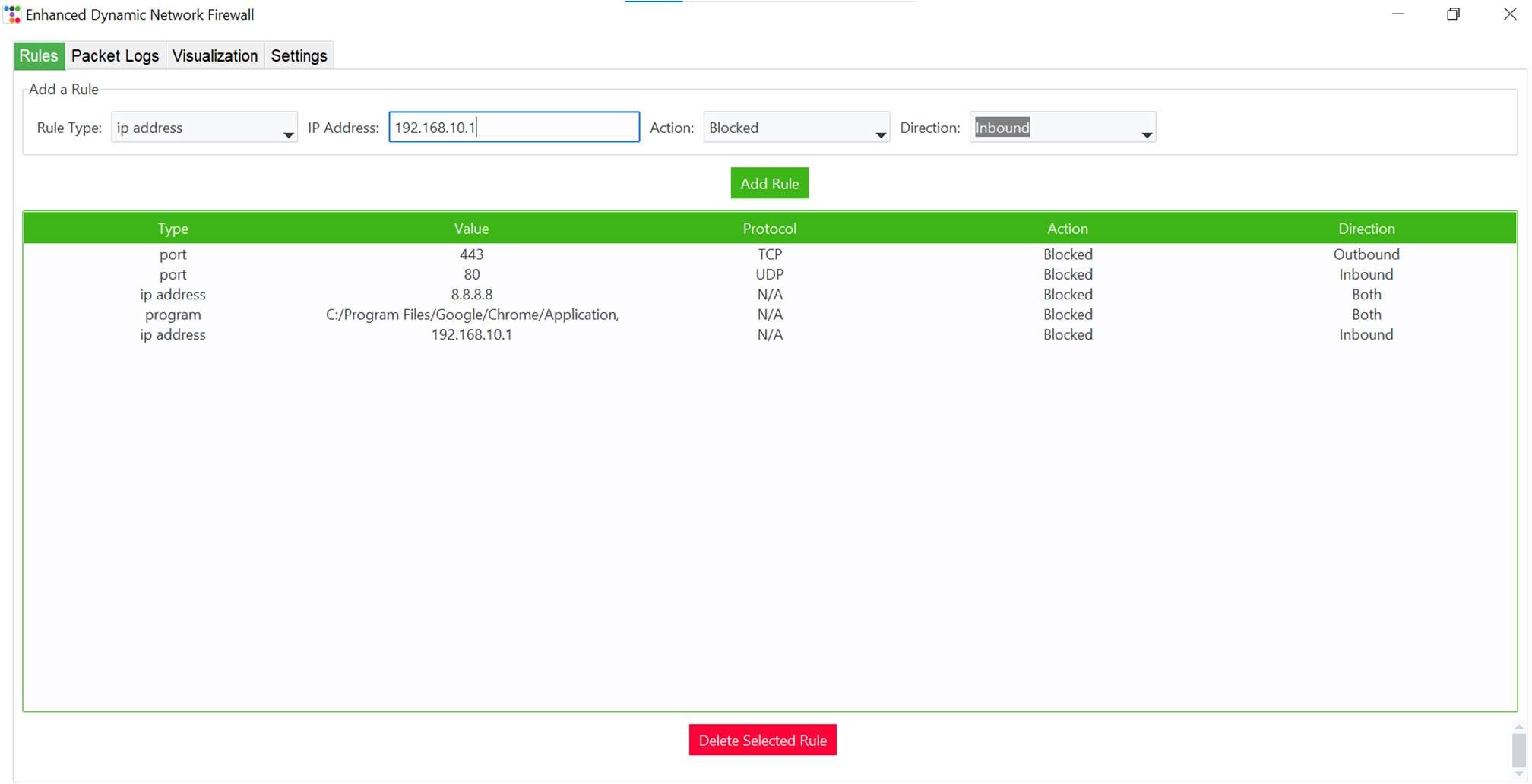
### Update Real-time Statistics:

* + Update and display packet counts (total, blocked, allowed) in real-time on the GUI.

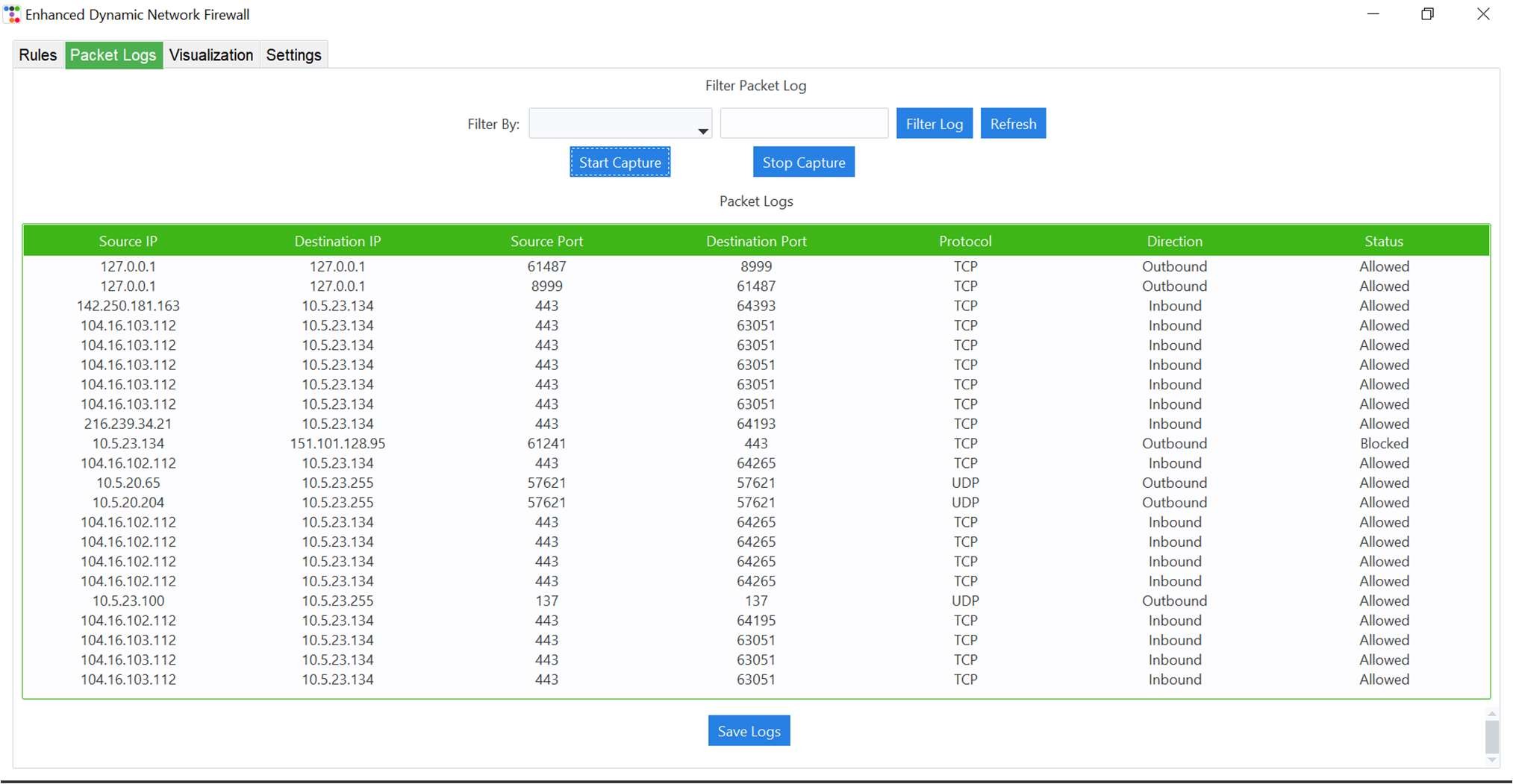
### Visualize Packet Traffic:

* + Use matplotlib to plot live charts of blocked and allowed packets over time.

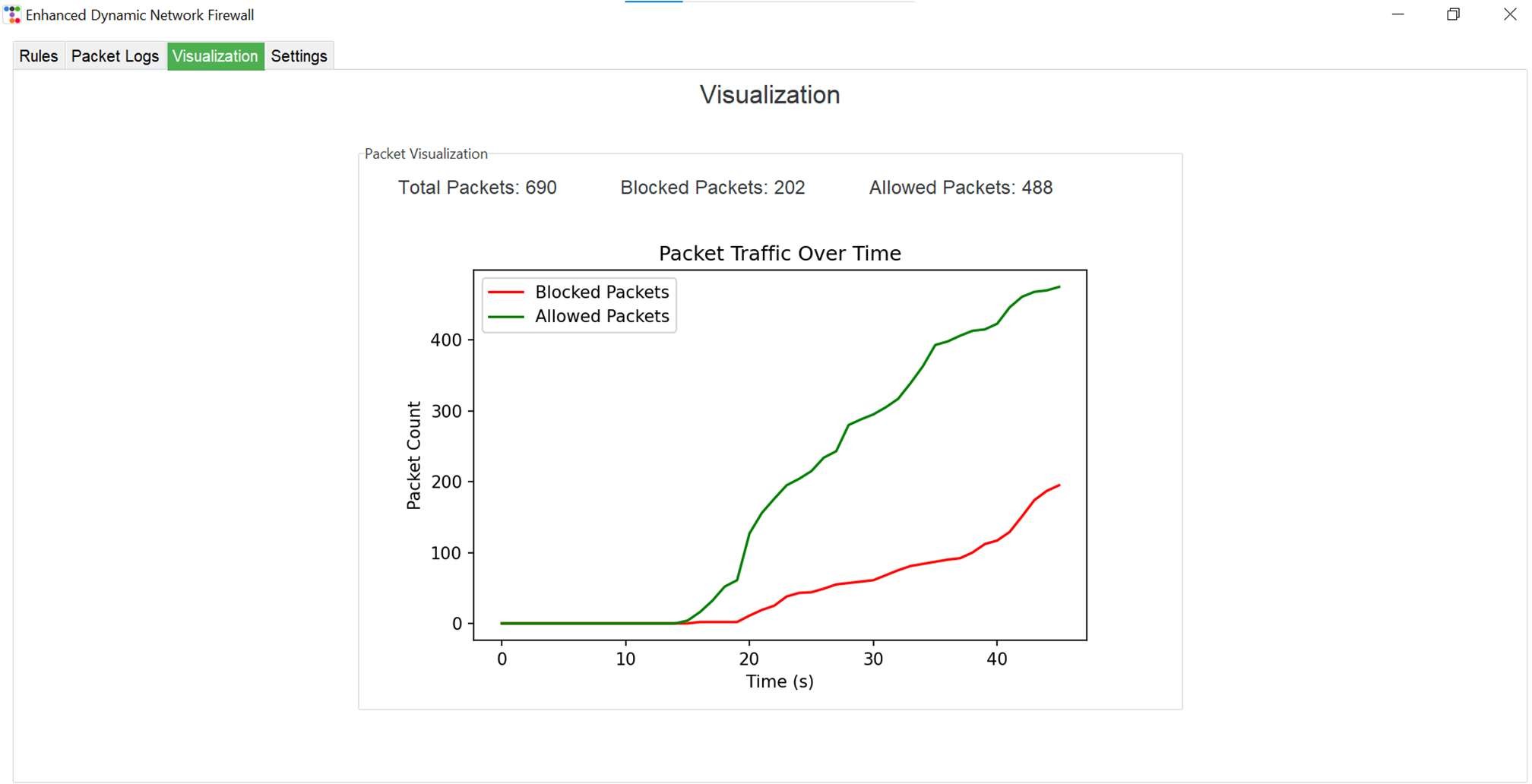
# Tool Screenshots

****

*Figure 1: Rule Definition Table*

**

*Figure 2: Packet Logging and Filtering*



# Results and Analysis

*Figure 3: Real Time Statistics*

1. **Results:** The application successfully captures, filters, and logs network traffic in real- time. The customizable rules and visualizations provide clear insights into network activity.
2. **Alignment with Objectives:** The outcomes align well with the project objectives, providing a functional and user-friendly tool for network monitoring.
3. **Tables:** Displaying logged packets with detailed information.
4. **Charts:** Live charts illustrating the number of blocked and allowed packets over time.
5. **Insights:** Users can quickly identify unusual traffic patterns, aiding in the detection of potential security threats. The CSV export feature enables detailed post-capture analysis.

# Conclusion

This project demonstrates the successful development of a network packet capture and filtering tool. Key takeaways include the importance of customizable filtering rules and real-time visualization in network monitoring.

### Success and Challenges:

* + **Success**: Achieved real-time capture, filtering, logging, and visualization of network

traffic.

* + **Challenges**: Ensuring accurate matching of packets to specific programs was complex but essential.

### Potential Improvements:

1. **Deep Packet Inspection (DPI)**: Implement DPI to analyze the content of packets beyond headers, identifying patterns in payload data.
2. **Machine Learning Integration**: Use machine learning algorithms to detect and predict unusual patterns, potentially identifying threats more accurately.
3. **Resource Management**: Improve resource management to ensure minimal impact on system performance during heavy network traffic.
4. **Support for Other Operating Systems**: Extend the application’s compatibility to support other operating systems like macOS and Linux.

# Github Link

Project Repository Link: https://github.com/Irtazamanzoor009/Dynamic-Network-Firewall

# References

1. Python Software Foundation. (n.d.). *Python documentation*. Retrieved from https://docs.python.org/
2. Microsoft Docs. (n.d.). *pydivert (Windows Filtering Platform) documentation*. Retrieved from https://docs.microsoft.com/en-us/windows/win32/fwp/windows-filtering-platform- start-page
3. pydivert documentation: *https://pypi.org/project/pydivert/*
4. tkinter documentation: *https://docs.python.org/3/library/tkinter.html*
5. psutil documentation: *https://pypi.org/project/psutil/*
6. matplotlib documentation: *https://matplotlib.org/stable/contents.html*
7. Real Python. (n.d.). *Introduction to Python*. Retrieved from https://realpython.com/
8. Geeks for Geeks. (n.d.). *Network Programming in Python*. Retrieved from https://[www.geeksforgeeks.org/network-programming-in-python/](http://www.geeksforgeeks.org/network-programming-in-python/)