SNIFFPRO - A NETWORK MONITORING SYSTEM

A PROJECT REPORT

Submitted by

ABHIJITH VARMA A (2022115003)

RISHI KUMAR U (2022115085)

ROHITH S (2022115304)

A report for the dissertation-II

submitted to the faculty of

INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment

for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING GUINDY

ANNA UNIVERSITY

CHENNAI 600 025

NOV 2024

ANNA UNIVERSITY

CHENNAI - 600 025

BONAFIDE CERTIFICATE

Certified that this project report titled **"SNIFFPRO - A NETWORK MONITORING SYSTEM"** is the bonafide work of **ABHIJITH VARMA A (2022115003), RISHI KUMAR U (2022115085), ROHITH S (2022115304)** who carried out project work under my supervision. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on this or any other candidate.

PLACE:CHENNAI
DATE:

Dr. K. VIDYA
ASSOCIATE PROFESSOR
PROJECT GUIDE
DEPARTMENT OF IST, CEG
ANNA UNIVERSITY
CHENNAI 600025

COUNTERSIGNED

Dr. S. SWAMYNATHAN
HEAD OF THE DEPARTMENT
DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY
COLLEGE OF ENGINEERING GUINDY
ANNA UNIVERSITY
CHENNAI 600025

ABSTRACT

In today's highly interconnected digital landscape, maintaining the performance and reliability of network infrastructure is more critical than ever. Organizations of all sizes rely heavily on their networks for day-to-day operations, making effective network monitoring an essential component of IT management. This project presents the development of a network monitoring application to real-time insights into network traffic and performance.

The primary objective of this project is to develop a tool for monitoring, analyzing, and visualizing network traffic in a user-friendly manner. The application captures and processes network data, offering detailed visualizations to help users understand traffic patterns, identify potential bottlenecks, and make decisions to improve network performance. The functionality of this application is further enhanced by AI-based packet analysis using Meta's Llama 3.1:8b model, which provides insights into network behavior by analyzing packet data in real-time.

ACKNOWLEDGEMENT

It is our privilege to express our deepest sense of gratitude and sincere thanks to **Dr. K. VIDYA,** Associate Professor, Project Guide, Department of Information Science and Technology, College of Engineering, Guindy, Anna University, for her constant supervision, encouragement, and support in our project work. We greatly appreciate the constructive advice and motivation that was given to help us advance our project in the right direction.

We are grateful to **Dr. S. SWAMYNATHAN,** Professor and Head, Department of Information Science and Technology, College of Engineering Guindy, Anna University for providing us with the opportunity and necessary resources to do this project.

We also wish to express our deepest sense of gratitude to the Members of the Project Review Committee: **Dr. M. VIJAYALAKSHMI,** Professor, **Mr. H. RIASUDHEEN,** Teaching Fellow, Department of Information Science and Technology,College of Engineering Guindy, Anna University, for their guidance and useful suggestions that were beneficial in helping us improve our project.

We also thank the faculty members and non teaching staff members of the Department of Information Science and Technology, Anna University, Chennai for their valuable support throughout the course of our project work.

ABHIJITH VARMA A (2022115003)
RISHI KUMAR U (2022115085)
ROHITH S (2022115304)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

NMA	Network Monitoring Application
AI	Artificial Intelligence
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ACL	Access Control List
GUI	Graphical User Interface
LAN	Local Area Network
MAC	Media Access Control
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The fast-paced growth of digital technologies and the increasing reliance on interconnected systems have made network monitoring an important part of IT management. Modern networks deal with a huge amount of data and support protocols like IPv4 and IPv6, which makes managing them a lot more complicated. Without proper tools, issues can stay undetected, leading to serious disruptions and even unexpected downtime in critical services.

Network monitoring doesn't just keep things running smoothly—it helps find and fix problems before they turn into something big. This way of working is super important, especially now, when networks have to support all kinds of applications and technologies that demand uninterrupted performance.

At the same time, new technologies like cloud computing and IoT (Internet of Things) have added more complexity. These environments are dynamic, with devices frequently connecting and disconnecting. To handle this, monitoring tools need to be flexible and capable of working with a variety of data flows. Without this, it becomes hard to ensure high performance in today's world.

1.2 OBJECTIVES

This project focuses on building a network monitoring application (NMA) that is easy to use and simplifies network analysis, even for people

who do not have much technical expertise. A lot of existing tools out there are either too complex or need advanced knowledge to use them effectively, which makes them inaccessible for many users. Our aim is to design an interface that's intuitive, reduces the learning curve, and provides useful insights at a glance.

Another key goal is to include AI-based analysis to make the application smarter. As network traffic becomes more complex, it's harder to manually identify patterns or detect issues. By using AI, the application can automatically highlight unusual activity and offer insights. This makes it easier for users to manage their networks without having to deeply understand all the technicalities.

The project also emphasizes simplicity, with features like real-time monitoring and clear visualizations. These elements are designed to help users quickly address issues and confidently manage their networks.

1.3 CHALLENGES IN NETWORK MONITORING

Network monitoring has its own set of challenges:

1. Handling Large Volumes of Data in Real-Time: As networks grow, the data generated increases massively. Monitoring systems need to process this data quickly and accurately, but delays or inefficiencies can lead to missed alerts or late responses to critical issues.

2. Creating Meaningful Visualizations: Network data in its raw form is hard to interpret. Turning it into easily understandable graphs or charts is crucial to understanding patterns and spotting issues. However, designing these visualizations in a way that's both clear and detailed can be a challenge.

3. **Balancing Monitoring with Network Performance:** Thorough monitoring often puts a strain on network resources. It's important to strike a balance—collecting enough data to provide insights without overloading the network itself.

1.4 SCOPE

The Network Monitoring Application (NMA) is designed to make monitoring more efficient by providing real-time packet capture, analysis, and visualization. This enables administrators to identify trends, diagnose problems, and keep an eye on network activity. Using AI-driven analysis, powered by Meta's Llama 3.1 model, the application also offers advanced insights like anomaly detection and predictive analytics.

By combining real-time monitoring, AI capabilities, and an accessible design, the NMA aims to tackle modern network challenges effectively, making it a powerful yet easy-to-use solution for understanding and managing network traffic.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION TO PACKET SNIFFING

Packet sniffing is a key technique in network management, allowing the capture and analysis of data packets transmitted over a network. It involves intercepting and logging traffic, which is critical for assessing network performance, optimizing data flow, and enhancing protocols. By analyzing packets, administrators can gain valuable insights into network behavior, helping them make informed decisions to maintain performance and address issues.

Packet sniffing tools capture data such as source and destination addresses, protocols, and payloads. These tools are becoming increasingly indispensable as modern networks grow more complex due to the proliferation of cloud services, IoT devices, and diverse protocols. This makes packet sniffing a fundamental practice in ensuring reliable and efficient network operations.

2.1.1 Significance of Packet Sniffing in Network Analysis

Packet sniffing plays a critical role in both real-time monitoring and retrospective analysis. Real-time monitoring enables the immediate detection of network issues, minimizing disruptions to end users. By logging historical data, administrators can also diagnose recurring problems and plan for long-term improvements.

Packet sniffing helps identify performance bottlenecks and potential

causes of delay, such as misconfigurations or bandwidth limitations. These capabilities make it a cornerstone of effective network management, offering diagnostics that enhance system reliability and efficiency, as noted by Shaw and Parveen[1].

2.1.2 Methods for Efficient Packet Sniffing

Various methods for intercepting and analyzing packets have been explored, each tailored to different network environments. For example, Chomsiri[2] discusses alternatives to ARP spoofing, which enhances security and accuracy in packet capture, addressing vulnerabilities typically associated with traditional methods.

Another area of focus is minimizing packet loss during capture. Techniques like using dedicated hardware or optimizing software reduce system overhead, ensuring captured data accurately reflects network activity. These improvements make packet sniffing more effective for traffic analysis and network diagnostics, as highlighted by Chomsiri et al.[2].

2.1.3 Continuous Sniffing for Performance Assessment

Continuous packet sniffing provides valuable data for assessing network reliability over time. By tracking metrics like packet loss, latency, and throughput, administrators can identify connectivity issues, diagnose delays, and optimize routing. This is particularly useful in large networks, where intermittent monitoring may miss transient issues.

Aggregating data from continuous sniffing also helps detect long-term trends, recurring problems, or seasonal variations in network load. This comprehensive view enables better capacity planning, ensuring the network

can handle peak demands while maintaining quality of service, as shown by Liu and Xu[3].

2.2 CHALLENGES IN PACKET ANALYSIS

Despite its advantages, packet sniffing poses challenges in interpreting and managing the vast amount of data generated. Tools like Wireshark capture extensive raw data, which can overwhelm users without technical expertise. Identifying relevant patterns and anomalies in this data often requires significant manual effort.

Current tools often lack advanced analytics to help users extract meaningful insights. As Sikos[4] suggests, integrating machine learning into packet analysis could enable automated detection of unusual traffic patterns. This would improve usability and make analysis faster and more effective.

2.3 ENHANCING USER EXPERIENCE THROUGH VISUALIZATION

Clear and intuitive visualizations are crucial for making packet data actionable. Effective tools transform raw data into formats that highlight trends and anomalies, making them accessible even to non-technical users. Interactive dashboards and graphical representations allow users to explore traffic patterns dynamically, improving their ability to monitor and manage networks.

Such visualizations not only enhance user satisfaction but also broaden the accessibility of packet sniffing tools, enabling more informed decision-making, as noted by Asrodia and Patel[3].

2.4 COMPLEXITY IN EXISTING TOOLS: WIRESHARK

Wireshark, a widely used packet sniffing tool, demonstrates both the potential and challenges of such software. While its structured tabular display provides a detailed view of captured traffic, its sheer volume of information can overwhelm users, particularly those unfamiliar with network analysis, as pointed out by Kumar et al.[5].

2.4.1 Packet Details Complexity

Wireshark's Packet Details pane provides a hierarchical breakdown of protocol layers, offering in-depth insights for experienced users. However, for beginners, the detailed and expansive information can be difficult to interpret. This highlights the need for simplifying data presentation without sacrificing detail, as observed by Malek and Amran[6].

2.4.2 Potential for AI Integration in Packet Analysis

A limitation of tools like Wireshark is the lack of advanced analytics, such as machine learning. Integrating AI could help identify patterns and flag unusual activity, making analysis faster and more predictive. This would reduce the reliance on manual inspection, enabling administrators to proactively address network issues, as discussed by Sikos et al.[4].

2.5 CONCLUSION

The literature underscores the evolving nature of packet sniffing and its role in network monitoring. Combining real-time capture, efficient methods, and user-friendly visualizations is essential for navigating modern network complexities. Future tools can further enhance usability by integrating AI and other advanced features, ensuring they cater to both technical and non-technical users effectively.

CHAPTER 3

SYSTEM DESIGN

3.1 APPLICATION FLOW

This section explains the steps involved in processing, storing, analyzing, and visualizing network packets. Each step plays a role in how the App(SniffPro) works, providing real-time insights.

3.1.1 Packet Capture

The first thing the App(SniffPro) performs is capturing packets from the network interface. This is done using Rust-based libraries like **pnet** and **libpcap**, which help with low-level access to network traffic. Capturing at this level allows the system to keep an eye on network data in real-time. Each packet has metadata like timestamp, source IP, destination IP, protocol type, and payload data, all of which are really important for later analysis.

3.1.1.1 Filtering

Once the packets are captured, they go through a filtering process, which is handled by the FilterModule. This part applies rules to keep only the packets that match certain criteria, like protocol type or IP address range**(IPv4, IPv6, Ethernet)**. It is to reduce the amount of data the system has to deal with, so it can focus on the more relevant packets. This helps improve performance by distinguishing between normal traffic and any unusual traffic.

3.1.2 Storage and Retrieval

The data and analysis results are stored in an **SQLite** database. SQLite is a lightweight, embedded database system that lets us store and retrieve network packets and results efficiently. The database helps maintain data persistence, so users can access historical data for long-term analysis. This setup also supports time-based analysis, which is important when examining past trends or incidents.

3.1.3 Visualization

The visualization module, built using **Recharts.js**, turns packet data into a more understandable format. Key metrics like traffic volume and protocol usage are shown using charts and graphs, so users can quickly grasp the network packets. The visualizations make it easy to interpret network data, providing immediate insights into traffic patterns.

3.1.4 AI Analysis

AI analysis is another important component. The AI module uses **Meta's LLaMA 3.1:8b parameter model** to identify patterns and potential bottlenecks in network traffic. The model helps detect anomalies and potential security threats based on the patterns it sees in the data. LLaMA 3.1 adds a layer of intelligence to the system, making real-time monitoring more responsive and insightful.

3.2 ARCHITECTURE

The SniffPro architecture is divided into several components, each

responsible for different tasks like packet processing, storage, and AI analysis. Figure 3.1 shows the architecture diagram for a clear understanding.

3.2.1 Packet Capture Engine

The Packet Capture Engine is the core module that handles network packets. Even though packets come from an external network interface (part of the overall system but not included in the project itself), this engine is responsible for processing them. It has several parts:

- **Packet Capture Module:** This part receives raw packet data from the external network interface and prepares it for processing inside SniffPro.
- **Filtering Module:** This module applies filters to the incoming packets, ensuring that only the relevant data is kept for analysis. Filters can be set by the user based on things like protocol, IP address, or packet type.
- **Storage Module:** This module organizes and saves the relevant packet metadata, allowing users to access historical data for further analysis and real-time monitoring.

3.2.2 Storage

The SQLite Database acts as the storage for all packet data, analysis results, and configuration settings. It uses efficient indexing and optimized read/write operations to enable quick retrieval of historical data, supporting detailed analysis. The database also handles concurrent access from different

processes, which helps maintain data integrity even when the system is under heavy load.

3.2.3 AI Analysis

LLaMA, which powers the AI component of SniffPro, specializes in pattern recognition and traffic analysis. By using historical data, it detects deviations from typical traffic patterns and highlights potential bottlenecks or inefficiencies. It also provides predictive insights, letting users take steps to resolve issues before they worsen. In short, LLaMA adds smart analysis capabilities to SniffPro, enabling it to handle more complex network problems.

3.2.4 Process Flow

- **Packet Capture Process:** The Network Interface captures raw packets, attaches some basic metadata, and sends them off to the Rust Packet Capture Engine for further processing. The system keeps track of how well the packets are being captured, allowing for adjustments if necessary.
- **Processing Pipeline:** The Rust Engine receives the packets and applies active filter rules through the Filter Module. The filtered packets are then passed to Flow Aggregation, which organizes them into logical flows. After that, the processed data is either stored or sent off for further analysis.
- **Storage Operations:** The processed data and flow records are saved in the SQLite Database. The database is well-organized for easy access and ensures efficient retrieval over time. Periodic clean-up operations are performed to keep the storage running smoothly.

- **Analysis Workflow:** The LLaMA model conducts deeper analysis, detecting any anomalies and providing predictive insights based on the data. These results are then stored and ready to be displayed on the frontend.
- **Frontend Integration:** The Tauri Bridge takes the analysis results, formats them for the React UI, and updates the dashboard. Data is visualized in the form of bar graphs, pie charts, and other helpful formats.

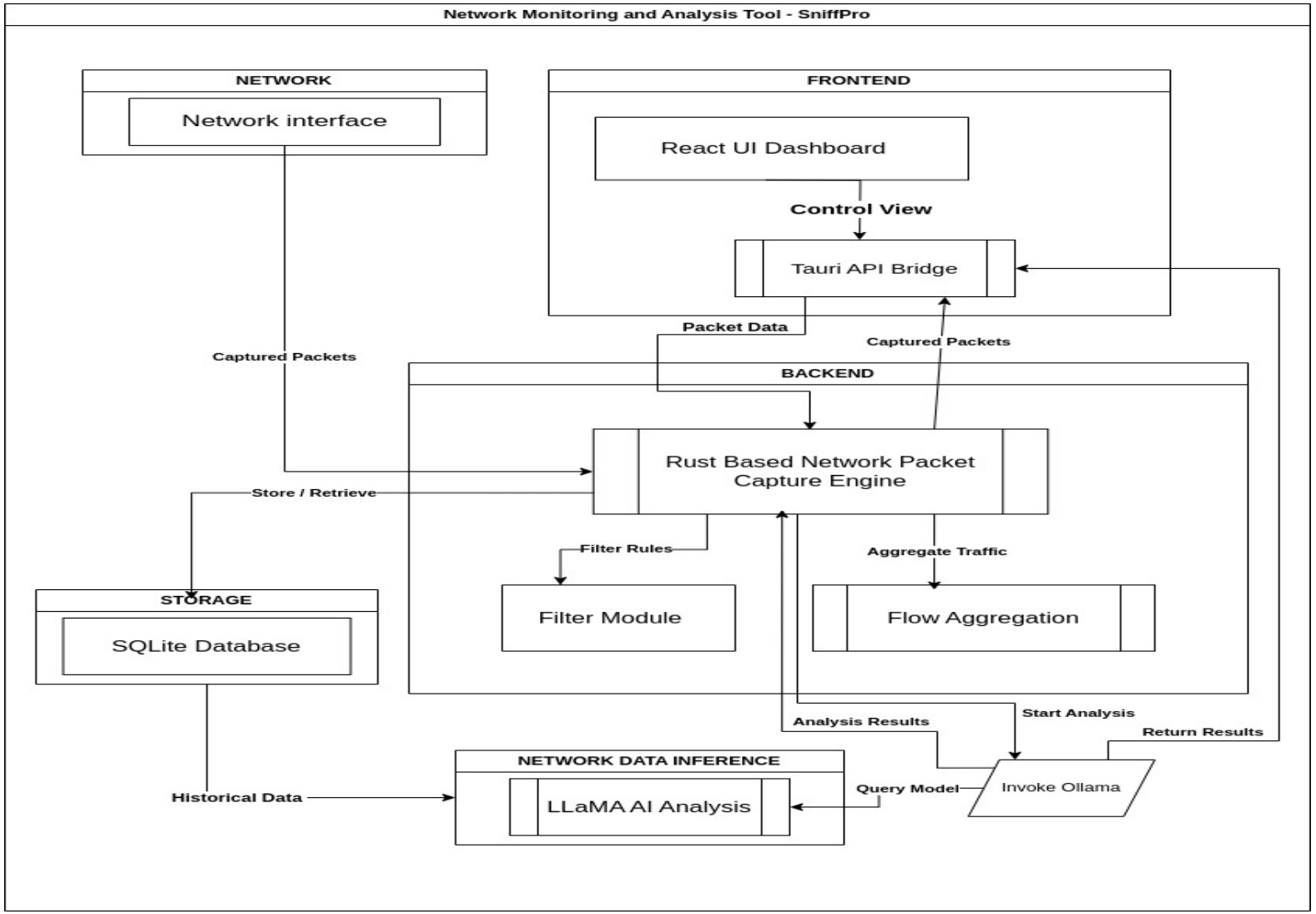


Figure 3.1: SniffPro Architecture Diagram

CHAPTER 4

IMPLEMENTATION

4.1 TECHNOLOGY STACK

4.1.1 Frontend

- **React**: React is a JavaScript library that us helps build dynamic and interactive user interfaces. Using react makes it easy to create reusable UI elements, which speeds up development and improves the overall user experience.
- **Tauri**: Tauri is a tool that serves as the bridge between the frontend and backend, enabling the creation of a lightweight desktop application.

4.1.2 Backend

- **Rust**: Rust was chosen for its outstanding performance and memory safety. It is ideal to build high performance systems, especially those requiring real time network packet processing.
- Packet Capture Libraries: **pnet & libpcap** are used for packet capturing, providing the necessary low level access to network interfaces. These allow the application to handle high-throughput data without significant overhead.

4.1.3 Database

- **SQLite**: SQLite is chosen for its lightweight, serverless design, making it ideal for fast and efficient storage of structured data. It handles packet data and analysis results effectively, ensuring easy management of historical data.
- **Data Formats**: The application uses JSON for data serialization, ensuring smooth communication between components. Its human readable format simplifies data exchange, and ensures compatibility with different systems.

4.1.4 Visualisation

Interactive data visualizations of packet data is being implemented using **Recharts.js**. Recharts.js allows us to create visually engaging, responsive, and dynamic charts that provide insights into packet data patterns over time. By visualizing packet data, we transform raw network information into meaningful, easily interpretable patterns, allowing users to:

- Track Traffic Volume Over Time: Show how packet flow varies, highlighting peak times and potential overloads.
- Analyze Protocol Usage: Identify the prevalence of protocols like TCP, UDP, HTTP, etc., which helps in recognizing common usage patterns or pinpointing suspicious protocol activity.
- Spot Anomalies and Patterns: Understand sudden spikes or drops in traffic, helping to quickly diagnose issues.

4.1.5 AI Analysis

The AI analysis module uses **Meta's LLaMA 3.1:8b** to analyze network traffic, identify patterns, and detect potential bottlenecks. This advanced model enhances real-time monitoring with intelligent and accurate insights, providing more use of the network data to the user.

4.2 PHASE 1 : PACKET SNIFFING

First phase would be to establish the packet sniffing functionality. The application captures essential network data by utilising pnet and libpcap libraries.

4.2.1 Setting Dependencies

- **Rust** : v1.73.0
- **pnet** : v0.31.0
- **tokio** : v1.52.0
- **rusqlite** : v0.29.0
- **chrono** : v0.4.30

4.2.2 Packet Structure

To capture packet metadata, a PacketData struct (Figure 4.1) is defined. This struct holds key fields like *timestamp, source, destination, protocol*, and various payload representations. By deriving Serialize and

```rust
#[derive(Serialize, Deserialize)]
struct PacketData {
    timestamp: String,
    packet_type: String,
    source: String,
    destination: String,
    protocol: Option<String>,
    payload_base64: String,
    payload_hex: String,
    payload_raw: Vec<u8>,
    payload_string: String,
}
```

Figure 4.1: PacketData Structure

Deserialize traits, the structure can be easily converted to and from JSON, which supports storing in SQLite or displaying in the frontend.

4.2.3 Packet Sniffing Logic

A custom packet sniffing logic is written in Rust to facilitate the basic function of our NMA. Using the **pnet::datalink** library, we access network interfaces of the device in which the application runs and establish a packet capture loop. The *start_sniffer* function is responsible for starting the sniffing process, allowing users to select a network interface on which to listen.

4.3 PHASE 2 : EXTENDED PACKET SNIFFING & STORAGE

In this phase, the packet sniffing module has been significantly enhanced to improve the capture and storage of various network protocols and packet types. The system now supports detailed handling of Ethernet, IPv4, and IPv6 packets, allowing for more precise analysis of network traffic across multiple layers. Additionally, the integration of a SQLite database for persistent storage enables reliable recording and retrieval of historical data, facilitating continuous traffic analysis and comprehensive forensic inspections.

4.3.1 Extensive Packet Sniffing

The existing sniffing logic is expanded to handle these various packet types and protocols, which are taken care of by helper functions *handle_ethernet_packets, handle_ipv4_packets & handle_ipv6_packets*. This approach allows for protocol specific extraction and processing, ensuring that each packet type's unique attributes are correctly identified and stored. These functions follow a modular design to allow for future expansion with additional protocols, such as TCP, UDP, or ICMP, making it adaptable for more complex network analysis.

4.3.1.1 Ethernet Packets

The *handle_ethernet_packets* function inspects the Ethernet header, extracting details such as source and destination MAC addresses and packet type. Since Ethernet serves as the link-layer encapsulation for most packets, this step is essential to determine the type of network layer protocol (IPv4 or IPv6) within each packet.

4.3.1.2 IPv4 Packets

The *handle_ipv4_packets* function focuses on the details of IPv4 packets, extracting information like the source and destination IP addresses, time-to-live, and protocol type (e.g., TCP or UDP). This functionality enables the tracking of individual packet flows, improving SniffPro's ability to perform indepth packet inspection and analysis.

4.3.1.3 IPv6 Packets

Similarly, *handle_ipv6_packets*, similar to the function written for IPV4, parses IPv6 packets, which differ in structure and fields from IPv4. This function extracts source and destination IP addresses, payload length, and next header type. IPv6 traffic often includes extension headers, which are handled here to support modern network infrastructures where IPv6 is prevalent.

4.3.2 Database Integration

Rusqlite makes it straightforward to interact with an SQLite database. A connection is established with the database. By creating a local database file (e.g., packets.db), we can consistently store packet data directly on the system. This approach allows us to capture and maintain a detailed historical record of network traffic locally, without any external dependency on networked or cloud-based databases.

4.3.2.1 Schema Design

The schema includes fields to store a detailed view of each packet, which can be used for in-depth analysis or retrospective examination.

- **timestamp**: Records the exact time the packet was captured, crucial for time-based analysis.
- **packet_type**: Indicates the type of packet (Ethernet, IPv4, IPv6).
- **source & destination**: Capture the source and destination addresses, which could be IP or MAC addresses depending on the packet type.
- **protocol**: Specifies the protocol used (e.g., TCP, UDP).
- **payload_base64, payload_hex, payload_raw, and payload_string**: Store different representations of the packet's payload data:

4.4 PHASE 3 : TAURI & REACT CONFIGURATION

We now focus on setting up a GUI. Tauri and React are integrated to create a frontend for the application, enabling users to control the sniffer, choose network interfaces, and view captured data.

4.4.1 Tauri Configuration

Initialising a Tauri project results in the connection of Rust and React components automatically, ensuring seamless interaction betweeen the frontend and backend.

Next step is to write functions in the backend Rust script that would serve as the interface between the frontend and backend, essentially maintaining an API Bridge. Additional helper functions are added, with the prefix header *#[tauri::command]* so as to instruct the compiler that this is a Tauri function.

Few examples, like the start_packet_sniffer and stop_packet_sniffer functions are used for this very purpose. Some more include list_names, which lists down the available interfaces in the system, and get_table_data function which is for retrieving data from existing tables.

4.4.2 Table display

Data is retrieved from the SQLite database and presented in a tablular format. This component enables users to view captured data in a structured manner, making it easier to analyze traffic.

4.5 PHASE 4 : VISUALISATION

In this phase, SniffPro's data is brought to life with visualizations using **Recharts.js**. These visualizations enable users to monitor traffic, identify bottlenecks, and analyze protocol specific behavior, all within an user friendly GUI.

Given the volume of packet data collected, three primary graph models are being implemented for efficient and comprehensive data analysis.

4.5.1 Source & Destination Frequency

A bar chart is used to display the frequency of source and destination

IP addresses. This information helps identify which hosts generate the most traffic or potentially cause network issues.

4.5.2 Network Traffic Flow

To showcase the rate of flow of packet traffic, a time series (line) graph is used. It allows users to observe traffic trends and detect unusual spikes that could indicate potential threats or issues.

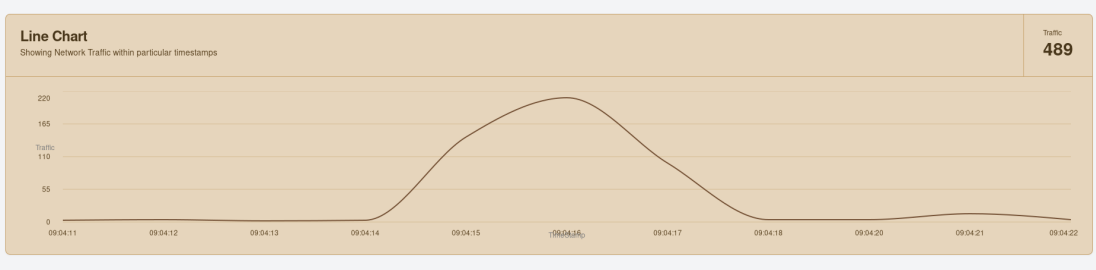


Figure 4.2: Sample Time-series Line Graph

4.5.3 Protocol Distribution

A pie chart is used to showcase the distribution of protocols (IPv4, IPv6) in the captured data. It offers a clear view of which protocols dominate the network, enabling quick assessment of whether certain protocols are over or underutilized.

4.6 PHASE 5 : AI ANALYSIS

To provide advanced network analysis capabilities, this phase integrates **Meta's LLaMA 3.1:8b** model through the Ollama API, locally.

Using the LLaMA model, the application can now perform traffic pattern analysis, or even packet by packet analysis, depending on the user's input, providing users with advanced insights.

Post this, SniffPro will be able to derive unusual network traffic patterns, like potential bottlenecks, overflowing etc., enhancing the application's ability to detect and respond to network anomalies. This integration gives SniffPro the transition from a traditional sniffer to an intelligent network monitoring tool, capable of analysing the network traffic.

CHAPTER 5

RESULTS

After the successful phases of implementation, we have developed a full fledged NMA, that is compatible across the major operating systems, Windows, Linux and macOS. It has almost no learning curve given the ease of usage and simple User Interface. This means that users who might not even have knowledge of network fundamentals can use this and engage in network traffic analysis.

5.1 LANDING PAGE

This is the first page (Figure 5.1) of the application, where the user can choose from the dropdown list of available network interfaces on the user's system, and then click on the *Start Sniffer* button to start the Sniffing and packet capture process. Once the user wishes to stop this, they can click on the *Stop Sniffer* button.

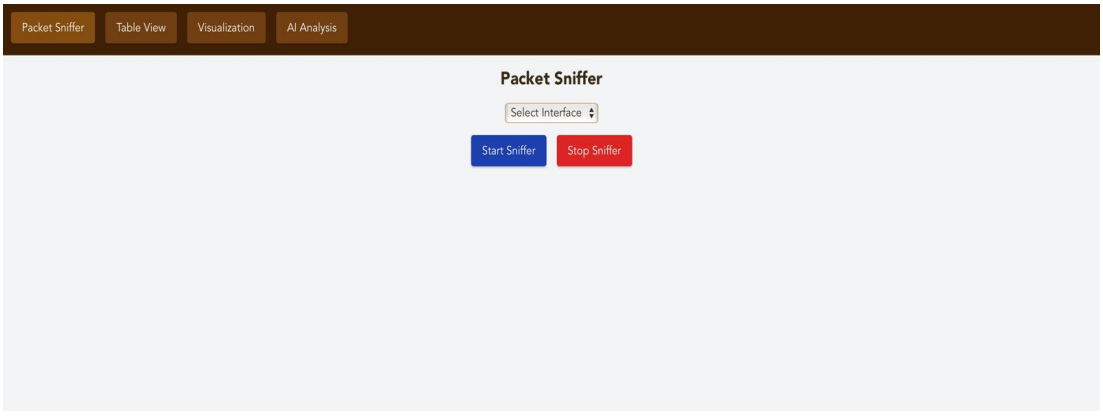


Figure 5.1: Packet Sniffer View

5.2 PACKET TABLE DISPLAY

This page of the application enables users to select from the database created in the user's system, which is intended to store the captured packet data. Each table generated is a different packet sniffing session, so the user can pick accordingly and view the captured packets.

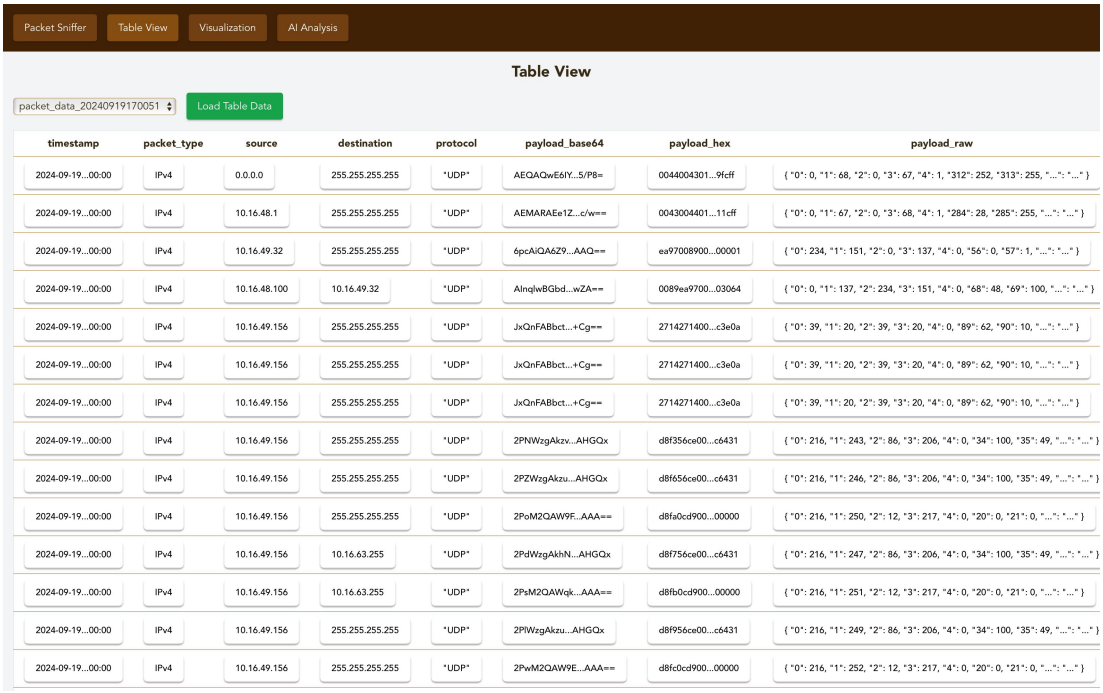


Figure 5.2: Packet Data Display

5.3 VISUALISATION

Post a sniffing session, the user can navigate to the Visualisation page where the user can access specific tables from the SQLite database and then proceed to visualise the data captured from the network interface.

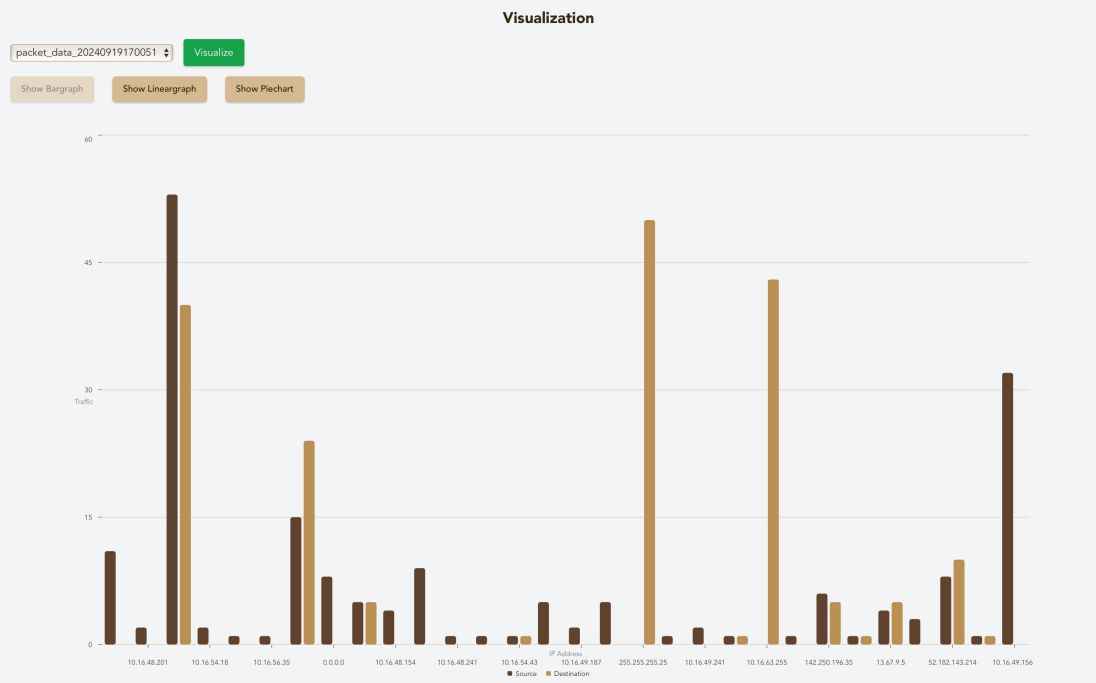


Figure 5.3: Source & Destination Frequency

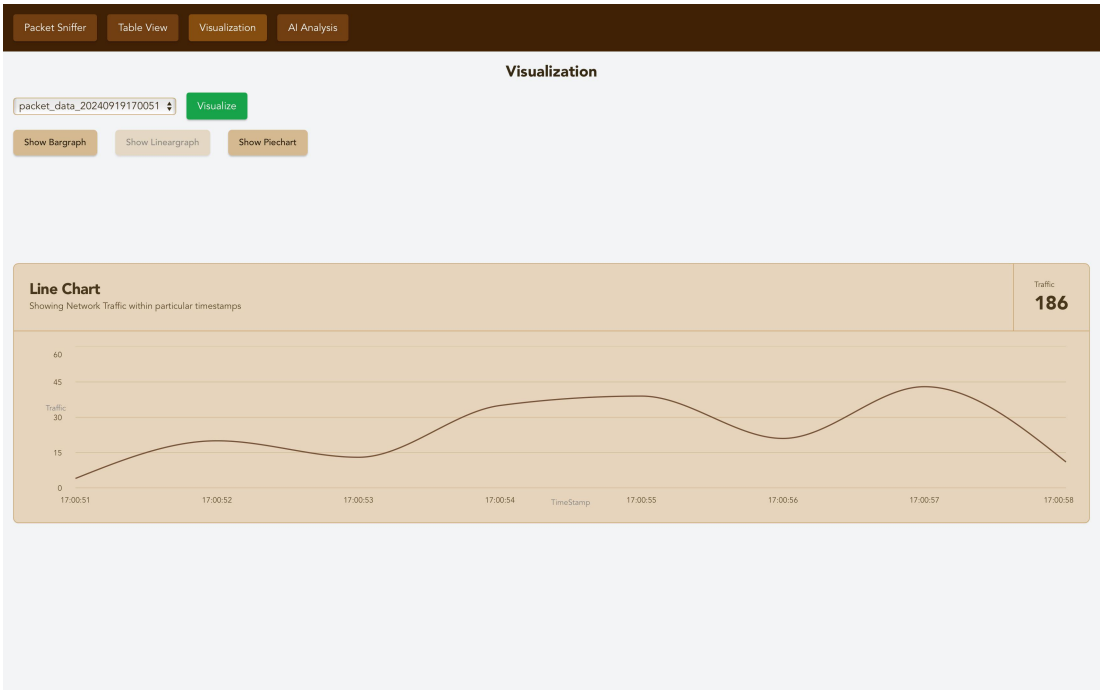


Figure 5.4: Traffic Flow

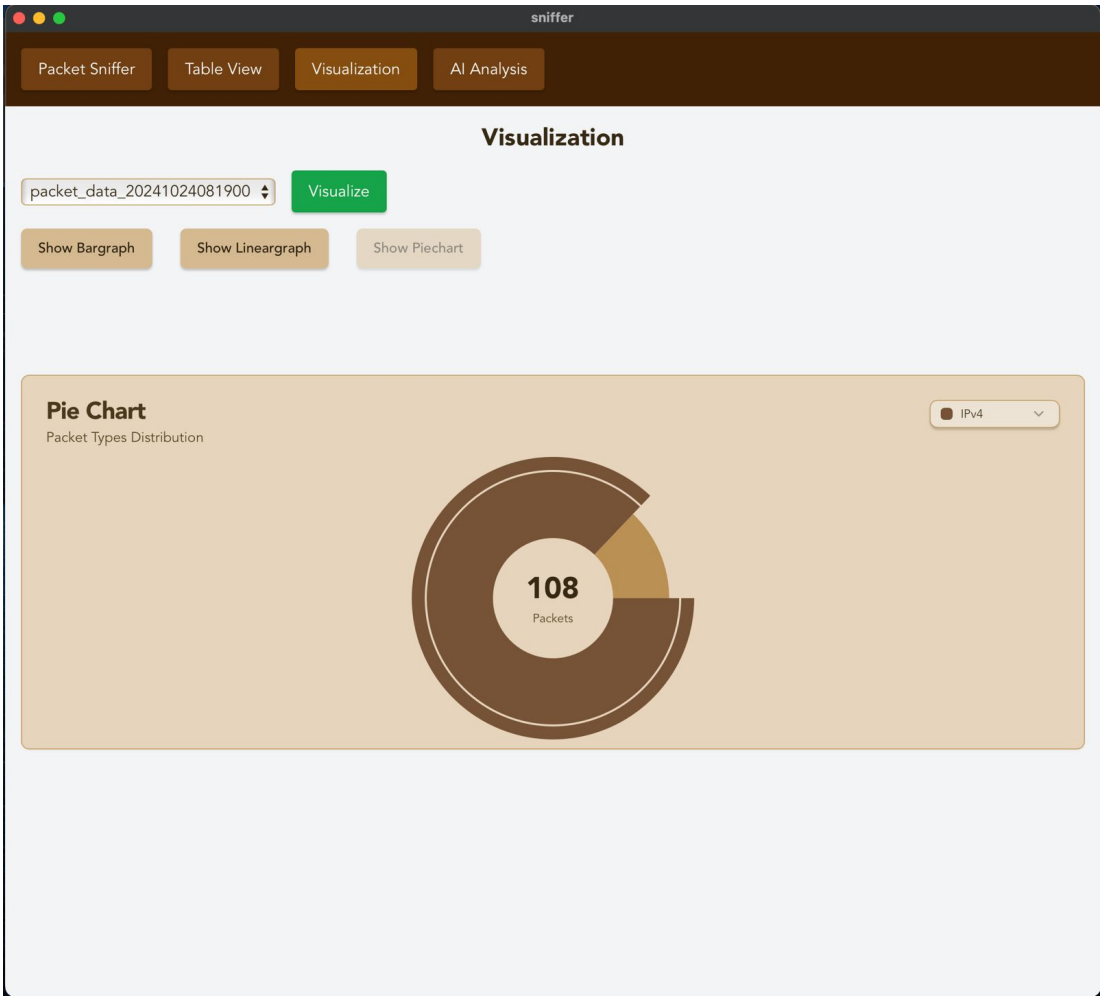


Figure 5.5: Protocol Distribution

5.4 AI ANALYSIS

Utilisation of Llama3.1 for AI analysis has proved to be a highly useful addition to this application's functionality. In the AI analysis page, the user can choose the table which they wish to derive analysis on, and select the type of packet along with it's source and destination. It is also possible to input the entire table to the model, and retrieve insights from it.

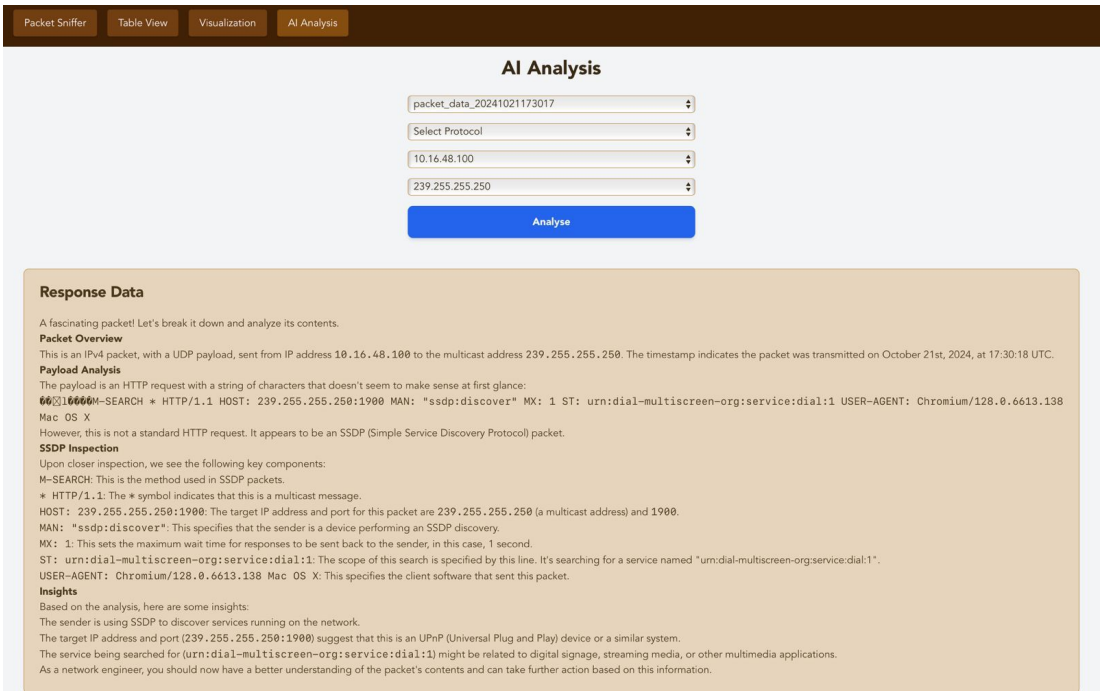
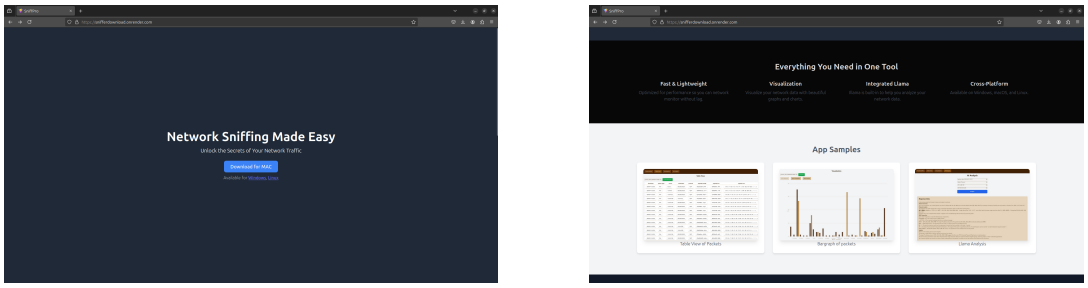


Figure 5.6: Analysis

5.5 WEBSITE

Additionally, we have also deployed a website for SniffPro that entails details about the NMA and a download link, from where users can download the NMA, for any platform, be it Windows, Linux or macOS.



((a)) ((b))

Figure 5.7: SniffPro Website

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

The SniffPro project has successfully demonstrated the feasibility and effectiveness of a NMA that is able to capture packet data from the user specified network interface to provide insights into network traffic. By integrating Rust for backend processing and React for the frontend user interface, we created a robust NMA capable of capturing, analyzing, and visualizing network packets in real-time, while also being very easy to use. The NMA effectively captures a wide range of packet data, including source and destination IPs, protocol types, and payload sizes. The use of SQLite for data storage ensures that captured data can be efficiently queried and analyzed. Additionally, usage of AI for analysing the packet and flow data has proved to be beneficial, that can make the data amount to valuable information.

6.1.1 Insights

- **User Engagement:** Interactive visualizations enhance user engagement by allowing dynamic exploration of network data.
- **Real-time Monitoring:** The ability to start and stop packet sniffing sessions in real-time helps users focus on specific events or time periods, making it easier to diagnose network issues or track unusual activity.

- **Protocol Analysis:** Analyzing protocol distribution provides valuable insights into network behavior, highlighting the dominance of certain protocols.
- **AI Integration:** AI analysis adds intelligence to SniffPro. By analyzing historical data, it can offer predictive insights and recommendations that help network administrators in making informed decisions.
- **Scalability:** SniffPro's architecture is built to scale, enabling future improvements such as support for additional protocols and integration with other monitoring tools.

6.2 FUTURE WORK

Real time Alerting Mechanism: Implement a notification system to alert users in real time about suspicious activities or anomalies during packet sniffing. Alerts could include unusual traffic spikes, unexpected protocol usage, or connections from blacklisted IP addresses.

Enhanced Protocol Support: Expand support for additional protocols beyond TCP, UDP, and ICMP, including specialized protocols like HTTP/2, MQTT, or custom application-layer protocols. This would give users a more comprehensive view of their network traffic.

User Authentication and Role Management: Introduce authentication to secure access and access control to assign different levels of access depending on the user's role (e.g., admin, analyst).

Advanced Analytics Features: Develop advanced analytics features using machine learning to identify patterns in historical data and predict future traffic behavior. This could include anomaly detection algorithms that learn normal traffic patterns and flag deviations.

REFERENCES

[1] Ranju Shaw and Suraiya Parveen. Literature review on packet sniffing: Essential for cybersecurity & network security. In *2024 5th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pages 715–719, 2024.

[2] Thawatchai Chomsiri. Sniffing packets on lan without arp spoofing. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, pages 472–477, 2008.

[3] Kai Liu and Menghan Xu. Path connectivity assessment based on continuous sniffing. In *2022 4th International Academic Exchange Conference on Science and Technology Innovation (IAECST)*, pages 991–994, 2022.

[4] Leslie F. Sikos. Packet analysis for network forensics: A comprehensive survey. *Forensic Science International: Digital Investigation*, 32:200892, 2020.

[5] Shivam Kumar, Suryansh Jigyasu, and Vikas Singh. Network traffic monitor and analysis usingpacket sniffer. *International Journal of Research in Engineering and Science (IJRES)*, 9(7):77–82, 2021.

[6] Muhammad Syaffiq Abdul Malek and Ahmad Roshidi Amran. A study of packet sniffing as an imperative security solution in cybersecurity. *Journal of Engineering Technology*, 9(1):96–101, 2021.

[7] Pallavi Asrodia and Hemlata Patel. Analysis of various packet sniffing tools for network monitoring and analysis. *International Journal of Electrical, Electronics and Computer Engineering*, 1(1):55–58, 2012.