

Network Packet Sniffer Tool

Submitted in partial fulfilment of the required for the award of

Bachelors of Computer Science

By

Pradeep D

(REGISTER NO 40290070)



DEPARTMENT OF COMPUTER SCIENCE SCHOOL

OF SCIENCE AND HUMANITIES

SATHYABAMA

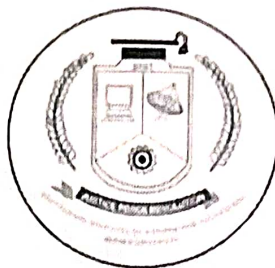
INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

**Accredited with Grade "A" by NAAC | 12B Status by UGC| Approved by
AICTE**

JEPPIAR NAGAR, RAJIV GANDHI SALAI, CHENNAI 600 199

APRIL 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

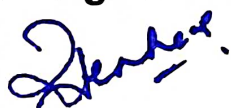
DEPARTMENT OF COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Pradeep D (40290069)** who carried out the project entitled "Network Packet Sniffer Tool" under our supervision from AUGUST 2022 to MAY 2023.


Internal Guide,

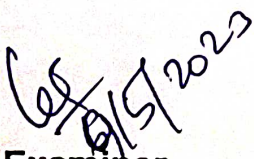
Dr .V. Ulagamuthalvi M.E., Ph.D.

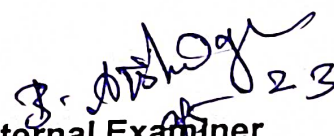

DEAN AND HEAD OF THE DEPARTMENT

Dr. REKHA CHAKRAVARTHI M.E., Ph.D.

Submitted for Viva voce examination held on

09 / 05 / 23


Internal Examiner


External Examiner

DECLARATION

I, Pradeep D (40290069) hereby declare that the Project Report entitled "Network Packet Sniffer Tool" done by us under the guidance of Dr.V.Ulagamuthalvi M.E., Ph.D. at SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY, Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119 is submitted in partial fulfillment of the requirements for the award of Master of Science degree in Computer Science.

DATE: 17.04.2023

PLACE: CHENNAI



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. REKHA CHAKRAVARTHI M.E., Ph.D. Dean of Arts and Science** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my project guides **Dr.V. Ulagamuthalvi M.E., Ph.D.** for their valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff member of the Department of Computer Science who were helpful in many ways for the completion of the project.

Pradeep D

ABSTRACT

The Packet sniffer application software is a powerful tool that captures network data and provides administrators with the necessary information to make informed decisions about network performance and security. The C language sniffer has been rewritten in Java, providing a more efficient and robust application that consumes minimal memory on the hard disk. This new application has been designed to meet the requirements of modern networks, and it is capable of handling large amounts of network traffic without compromising performance.

The Packet sniffer application comprises of five independent modules that work together to provide comprehensive network monitoring and analysis capabilities. The first module is responsible for capturing network data and storing it in a database for later analysis. The second module analyzes the network traffic patterns to identify potential issues, such as high network latency, packet loss, or unusual network activity. The third module provides tools for troubleshooting network problems, allowing administrators to quickly isolate and resolve issues.

One of the key advantages of this application is its small memory footprint. It uses only 1MB of the hard disk space, making it an ideal solution for organizations with limited resources. Additionally, the friendly GUI and simple installation process make it easy for administrators to get started with the application and begin monitoring and analyzing network traffic right away.

TABLE CONTENT

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	V
	LIST OF FIGURES	VIII
1	INTRODUCTION	1
	1.1 OBJECTIVE	1
	1.2 DEFINITION	2
2	LITERATURE SURVEY	3
	2.1 REVIEW ON LITERATURE WORK	3
3	SYSTEM ANALYSIS	5
	3.1 FEASIBILITY STUDY	5
	3.2 TECHNICAL FEASIBILITY	5
	3.3 OPERATIONAL FEASIBILITY	5
	3.4 EXISTING SYSTEM	6
	3.5 PROPOSED SYSTEM	6
4	METHODOLOGY	7
	4.1 REQUIREMRNT ANALYSIS	7
	4.1.1 HARDWARE ENVIRONMENT	7
	4.1.2 SOFTWARE REQUIRMENT	7
	4.2 SYSTEM DESIGN	8
	4.2.1 ARCHITECTURE DIAGRAM	8
	4.2.2 COMPONENT DIAGRAM	9
	4.2.3 DEPLOYMENT DIAGRAM	10
	4.3 MODULES DESCRIPTION	11
	4.3.1 USER INTERFACE MODULE	11

	4.3.2 PACKET SNIFFING MODULE	12
	4.3.3 ANALYZE LAYERS MODULE	12
	4.3.4 FREE MEMORY MODULE	13
	4.3.5 PROTOCOL ANALYSIS MODULE	14
5	SYSTEM TESTING	15
	5.1 TYPES OF TESTING	15
	5.2 UNIT TESTING	15
	5.3 INTEGRATION TESTING	15
	5.4 FUNCTIONAL TESTING	15
	5.5 SYSTEM TEST	16
6	CONCLUSION AND FUTURE WORK	17
	4.1 CONCLUSION	17
	4.2 FUTURE WORK	17
7	REFERENCE	18
8	APPENDIX	19
	8.1.CHECKSTATS CODE	13
	8.2 PACKET DETAILS CODE	18
	8.3 VIEW PACKET CODE	27
	8.4 RAW DETAILS CODE	34
	8.5 SCREEN SHOTS	39

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
4.1	ARCHITECTURE DIAGRAM	8
4.2	COMPONENT DIAGRAM	9
4.3	DEPLOYMENT DIAGRAM	10
8.5.1	HOMEPAGE	44
8.5.2	PACKET DETAILS PAGE	45
8.5.3	PACKET LENGTH PAGE	45

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The sniffing tool is an essential component of a network administrator's toolkit, as it allows them to capture and analyze both IPv4 and IPv6 packets. This is crucial for identifying and diagnosing issues within the network, such as congestion, packet loss, and security breaches. By analyzing these packets, administrators can gain valuable insights into the inner workings of the network and pinpoint the root causes of any problems.

One of the primary uses of the sniffing tool is to analyze IPv4 packets. This involves capturing packets as they move through the network and examining the ports and protocols that are being used. In addition, administrators can also examine the contents of the packets themselves, which can provide valuable information about the source and destination of the packets, as well as their contents. By analyzing this information, administrators can quickly identify any anomalies or issues within the network.

However, as IPv6 becomes more widely used, it is important for network administrators to be able to capture and analyze IPv6 packets as well. While many network tools still focus primarily on IPv4 traffic, the sniffing tool provides a way for administrators to ensure that they are able to diagnose problems across all aspects of the network.

Capturing and analyzing IPv6 packets can be a bit more challenging than IPv4 packets, as there are some differences in the way that they are transmitted. For example, IPv6 packets are larger than IPv4 packets, and they use a different addressing scheme. However, the sniffing tool is able to handle these differences and provides administrators with a way to capture and analyze IPv6 packets in the same way that they would capture and analyze IPv4 packets.

The sniffing tool is an essential tool for network administrators who need to diagnose problems within their networks. By providing the ability to capture and analyze both IPv4 and IPv6 packets, the sniffing tool ensures that administrators are able to identify and resolve issues across all aspects of the network.

1.1 DEFINITION

Packet sniffers, also known as packet analyzers, protocol analyzers, or network analyzers, provide network administrators with a comprehensive view of network traffic. They are powerful tools that can be used to identify sources of network congestion, monitor bandwidth usage, detect network intrusions, and troubleshoot network issues.

Packet sniffers work by capturing packets as they move through the network and then analyzing the contents of these packets to extract valuable information about the network traffic. By examining the packets, administrators can gain insights into the types of traffic that are flowing through the network, the protocols that are being used, and the source and destination of the packets. This information can be used to identify potential bottlenecks, detect security breaches, and troubleshoot issues within the network.

Packet sniffers can be used in a variety of different network environments, from small local area networks to large-scale enterprise networks. They can be used to monitor traffic in real-time or to capture packets for later analysis. This flexibility allows administrators to choose the approach that best suits their needs and to adapt their monitoring strategy as the network evolves over time.

In addition to capturing and analyzing packets, packet sniffers can also provide detailed statistics about network usage. This information can be used to identify trends in network traffic and to make informed decisions about capacity planning and network upgrades. Packet sniffers are powerful tools that provide network administrators with valuable insights into network traffic. By capturing and analyzing packets, administrators can identify issues within the network, troubleshoot problems, and make informed decisions about network upgrades and capacity planning.

CHAPTER 2

LITERATURE SURVEY

2.1 REVIEW ON LITERATURE WORK

Bluetooth traffic has been widely considered an extremely intricate task due to Bluetooth's indiscoverable mode, vendor-dependent adaptive hopping behavior, and the interference in the open 2.4 GHz band. In this paper, we present BlueEar-a practical Bluetooth traffic sniffer. BlueEar features a novel dual-radio architecture where two Bluetooth-compliant radios coordinate with each other on learning the hopping sequence of indiscoverable Bluetooth networks, predicting adaptive hopping behavior, and mitigating the impacts of RF interference. We built a prototype of BlueEar to sniff on Bluetooth classic traffic. [1]

Traffic analysis using the internet is an activity to record data from user activities in using the Internet. This study aims to obtain data about the results of traffic in a graphical form so that it can find out the number of users who access the internet and use bandwidth. In this study, researchers also noted when the peak internet usage time in Telkom Vocational School Pekanbaru. The method used to get the results of the study is the packet sniffing method. [2]

The sniffers are an irreplaceable tool. In the case where WSN covers a large area, single sniffer cannot provide a trace of the complete data traffic, due to the attenuation of the wireless signal. This motivates the implementation of a distributed solution with multiple physical sniffers connected over a network. Overhearing the same data traffic by different physical sniffers is likely in such a scenario, resulting in multiple copies of the same data packet appearing in the merged trace. To overcome this issue, real-time traffic filtering needs to be applied. This paper, motivates and proposes a design of a distributed sniffer that can cover a large deployment. [3]

A real-time wireless sniffing system is able to sense WiFi packets and analyse wireless traffic. This provides an opportunity to obtain insights into the interaction between the humans carrying the mobile devices and the environment. Susceptibility to loss of the wireless data transmission is an important limitation on this idea, and this is complicated by the lack of a standard specification for real deployment of WiFi sniffers. In this paper, we present an experimental analysis of sniffing performance under different wireless environments using off-the-shelf products. Our objective is to identify the possible factors including channel settings and access point configurations that affect sniffing behaviors and performances. [4]

As wireless sensor networks grow larger, more complex and their role more significant, it becomes necessary to have an insight into the network traffic. For this purpose, sniffers play an irreplaceable role. Since a sniffer is a device of limited range, to cover a multi-hop network it is necessary to consider the deployment of multiple sniffers. This motivates the research on the optimal number and position of sniffers in the network. We present a solution based on a minimal dominant set from graph theory. We evaluate the proposed solution and implement it as an extension of the 6TiSCH simulator. Our solution assumes a 50-nodes scenario, deployed in 2×2 km outdoor area, with 10% of packet drops over all channels, when 10 sniffers are used. [5]

CHAPTER 3

SYSTEM ANALYSIS

3.1 FEASIBILITY STUDY

Feasibility study is a process which defines exactly what a project is and what strategic issues need to be considered to assess its feasibility, or likelihood of succeeding. Feasibility studies are useful both when starting a new business, and identifying a new opportunity for an existing business. Ideally, the feasibility study process involves making rational decisions about a number of enduring characteristics of a project, including: Technical feasibility- do we have the technology? If not, can we get it? • Operational feasibility- do we have the resources to build the system? Will the system be acceptable? Will people use it? • Economic feasibility, technical feasibility, schedule feasibility, and operational feasibility- are the benefits greater than the costs?

3.2 TECHNICAL FEASIBILITY

Technical feasibility is concerned with the existing computer system (Hardware, Software etc.) and to what extent it can support the proposed addition. For example, if particular software will work only in a computer with a higher configuration, an additional hardware is required. This involves financial considerations and if the budget is a serious constraint, then the proposal will be considered not feasible.

3.3 OPERATIONAL FEASIBILITY

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantages of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

3.4 EXISTING SYSTEM

The main disadvantages of using a packet sniffer is that it is unable to decrypt SSL traffic without first retrieving the server certificate. This is because SSL traffic is encrypted using a public key infrastructure, which means that the only way to decrypt it is by using the appropriate private key. If the server certificate is not available to the packet sniffer, then the SSL traffic will remain encrypted and unreadable. While packet sniffers can be an effective tool for monitoring and analyzing network traffic, they do have some limitations when it comes to decrypting SSL traffic. Installing a system driver can provide a workaround for this limitation, but it should be done with caution and only after careful consideration of the potential risks and drawbacks.

3.5 PROPOSED SYSTEM

The new sniffing tool is designed to be capable of de-capsulating both IPv4 and IPv6 traffic on a network. To begin the process, the tool will conduct a thorough search of all devices connected to the network and create a list of these devices for the user to select from. Once the user has selected the desired device, the tool will fetch the available packets of that device, whether it is IPv4 or IPv6. The tool will show the available packet lists and once the packets have been captured, the tool will analyze the data and display important information about each packet. This information will be essential for network analysis, and will include details such as the source IP address, source port, destination IP address, destination port, and protocol used for each packet. The proposed sniffing tool will also provide statistical information about the captured packets, such as the total number of packets captured, the number of packets for each protocol, and the number of packets for each source or destination IP address.

CHAPTER 4

METHODOLOGY

4.1 REQUIREMENT ANALYSIS

Requirement analysis determines the requirements of a new system. This project analyses on product and resource requirement, which is required for this successful system. The product requirement includes input and output requirements it gives the wants in term of input to produce the required output. The resource requirements give in brief about the software and hardware that are needed to achieve the required functionality.

4.1.1 Hardware Environment

- Processor: Minimum 1 GHz; Recommended 2GHz or more.
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi).
- Hard Drive: Minimum 8 GB; Recommended 24 GB or more.
- Memory (RAM): Minimum 1 GB; Recommended 4 GB or above.

4.1.2 Software Requirements

- Java, JDK 8.1 or above.
- Operating System: Windows: 7 or Newer

4.2 SYSTEM DESIGN

4.2.1 ARCHITECTURE DIAGRAM

The purpose of an architecture diagram is to provide a high-level view of the system being designed or implemented. It shows the components of the system and their relationships with one another, as well as how they interact with external systems and stakeholders. The diagram can help to communicate the overall design of the system to stakeholders and team members, allowing them to better understand how the system functions and how its different components work together.

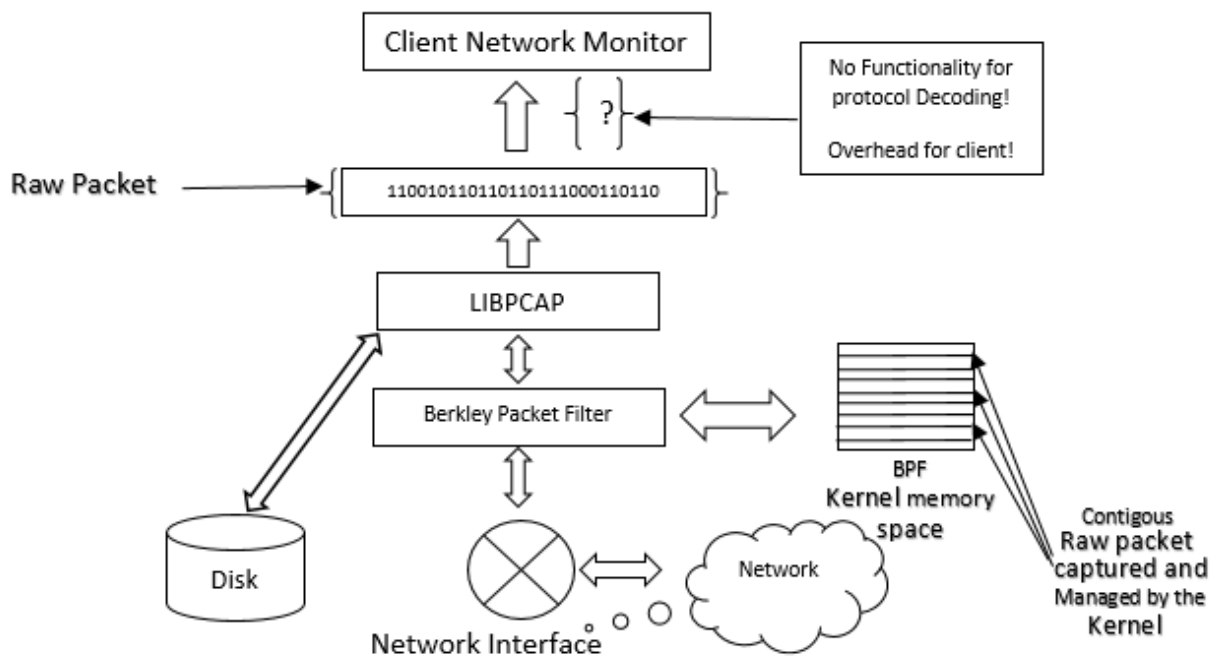


Figure 4.1 Architecture Diagram

4.2.2 COMPONENT DIAGRAM

The purpose of a component diagram is to show the relationship between different components in a system. For the purpose of UML 2.0, the term "component" refers to a module of classes that represent independent systems or subsystems with the ability to interface with the rest of the system.

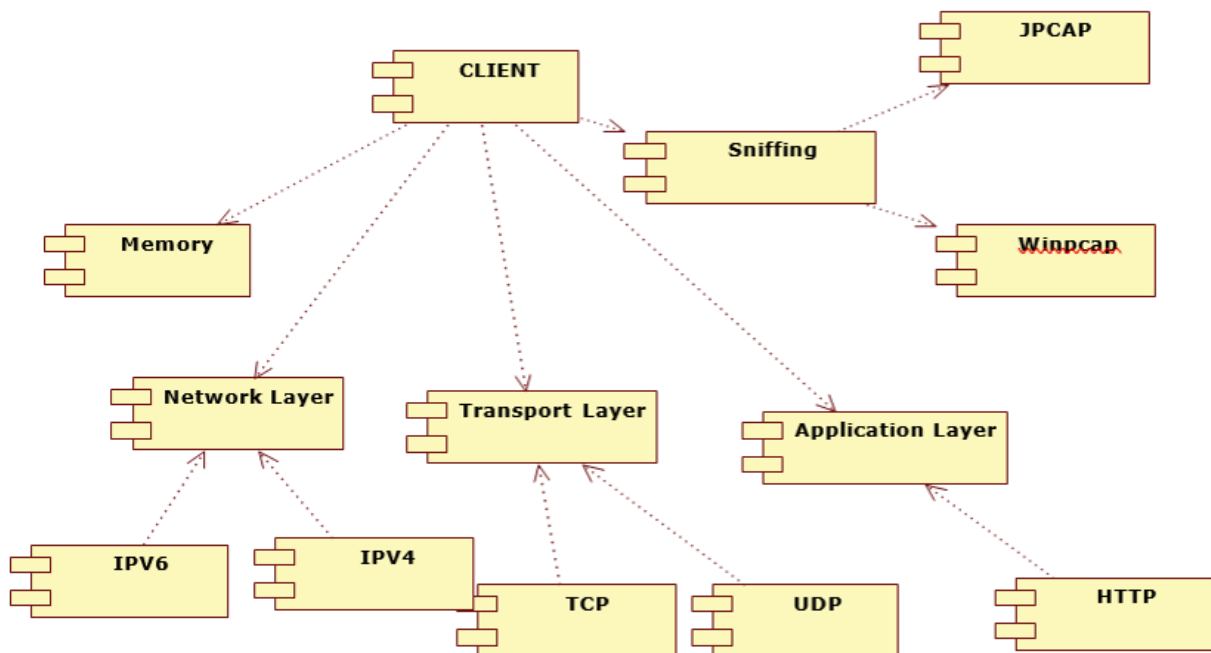


Figure 4.2 Component Diagram

4.2.3 DEPLOYMENT DIAGRAM

Deployment diagrams are effective for visualizing, specifying, and documenting the following types of systems:

- Embedded systems that use hardware that is controlled by external stimuli; for example, a display that is controlled by temperature change
- Client/server systems that typically distinguish between the user interface and the persistent data of a system
- Distributed systems that have multiple servers and can host multiple versions of software artifacts, some of which might even migrate from node to node

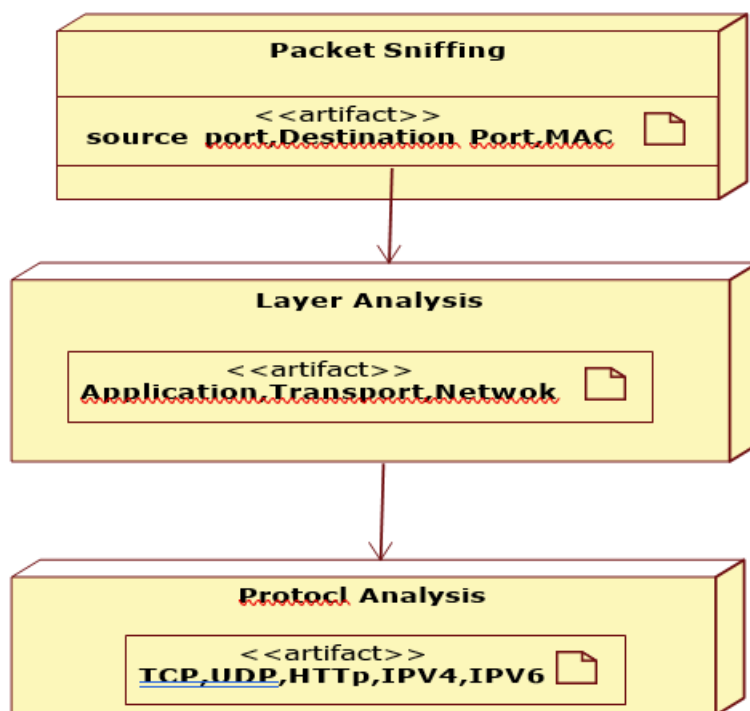


Figure 4.3 Deployment Diagram

4.3 Module Description

- **User Interface Module.**
- **Packet Sniffing Module.**
- **Analyze layers Module.**
- **Free Memory Module.**
- **Protocol Analysis Module.**

4.3.1 User Interface module:

The user interface (UI) module is an essential component of any software application, and the Packet sniffer application is no exception. The UI module of the Packet sniffer application is designed with end-users in mind, providing an easy-to-use interface that allows users to access the application's features with ease. The UI module is designed to be visually attractive, providing an aesthetically pleasing look that is easy on the eyes.

The Swing framework in core Java to create the UI. Swing is a powerful tool that enables the creation of modern, visually appealing interfaces that are both functional and user-friendly. By using Swing, the developers have created a UI that is consistent and unified throughout the application, providing a seamless user experience.

The Packet sniffer UI is designed to be intuitive and easy to navigate. The UI provides users with a clear and concise view of the application's features, allowing them to quickly access the information they need. The UI module also includes features such as tooltips, which provide users with additional information about specific features and functions.

The UI module of the Packet sniffer application is designed with the end-users in mind, providing an attractive, easy-to-use interface that is consistent and unified throughout the application. By using Swing in core Java, the developers have ensured that the application is cross-platform compatible, allowing users to access the application on a wide range of devices.

4.3.2 Packet Sniffing Module:

The packet capturing module is a crucial component of the Packet sniffer application, responsible for capturing packets that traverse a machine's network interface. This module intercepts all the packets that are sent and received by the Network Interface Card (NIC) of the machine on which the sniffer is installed. This is possible when the NIC is set to the promiscuous mode, which allows it to receive all the packets sent to the network, including packets that are not intended for the machine.

Once the packet capturing module has captured the packets, it stores them in a buffer for further analysis. The size of the buffer can be configured by the user, allowing for the control of the number of packets that can be captured and stored at any given time. This is important because capturing too many packets can consume excessive amounts of memory, which can lead to performance issues.

The packet capturing module is a critical component of the Packet sniffer application, as it allows network administrators to analyze network traffic for optimal performance and security. By analyzing network traffic, administrators can identify potential network congestion, packet loss, or security breaches, and take corrective action. Additionally, the packet capturing module can be used to monitor bandwidth usage, detect network intrusions, and troubleshoot network issues.

4.3.3 Analyze layers Module:

This Analyze Layer module is responsible for analyzing the layers in the system, specifically the Transport Layer, Application Layer, and Network Layer. This module utilizes various techniques to extract information from the packets, such as packet parsing and protocol analysis. Packet parsing involves breaking down the packets into their constituent parts, while protocol analysis involves analyzing the headers of the packets to determine the protocol used. Additionally, this module provides a graphical representation of the usage of different layers during packet capturing time. The module is capable of displaying the graph in two different manners, line graph and pie graph, allowing for a more comprehensive understanding of the packet flow.

The Analyze Layer module is an essential component of the Packet Sniffer application as it enables network administrators to gain insight into network traffic by analyzing different layers. By breaking down packets into their constituent parts, administrators can identify potential bottlenecks, security breaches, and other issues within the network. Moreover, the ability to display graphical representations of layer usage enhances the overall usability of the application, providing users with an intuitive and user-friendly experience. Overall, the Analyze Layer module plays a crucial role in ensuring optimal network performance and security.

4.3.4 Free Memory Module:

The Free Memory module in the Packet Sniffer application is designed to monitor the usage of computer memory during the packet capturing process. This module provides useful insights into the resource utilization of the system, making it a critical component of the application. The module offers both numerical and graphical representations of the amount of free memory available at the time of packet capturing. The numerical representation provides the exact amount of free memory available in bytes, kilobytes or megabytes, depending on the size of the memory.

The graphical representation of the memory usage is designed to be easily understandable, even for non-technical users. It displays the memory usage in a visual format that is easy to interpret. The Free Memory module is particularly useful for network administrators who need to ensure that the system has enough resources to run the Packet Sniffer application and other essential applications simultaneously.

The Free Memory module works by monitoring the usage of memory during the packet capturing process. It captures the amount of free memory available at regular intervals and displays the results in the numerical and graphical formats. The graphical representation of memory usage can be displayed in a variety of different formats, including line graphs and pie charts. This enables users to easily monitor memory usage over time and detect any spikes or drops in memory usage that may be indicative of a problem.

4.3.5 Protocol Analysis Module:

The Protocol Analysis module is responsible for analyzing the different protocols used by the system, such as TCP, UDP, and HTTP. This module is critical in providing a detailed understanding of the network traffic and helps in troubleshooting protocol-related issues. The module displays information about the source and destination ports of each protocol, along with the packet length. The source and destination ports for TCP and UDP protocols are particularly useful in identifying the source of network congestion or detecting any network intrusions.

Moreover, the packet length information helps in analyzing the data transfer rate of each protocol. The module provides a comprehensive view of the protocols used in the network and enables administrators to make informed decisions about network performance and security. Additionally, it can display the information in a user-friendly format, making it easy for non-technical users to understand the network traffic. Overall, the Protocol Analysis module plays a crucial role in ensuring optimal network performance and maintaining network security.

CHAPTER 5

SYSTEM TESTING

5.1 TYPES OF TESTS

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

5.2 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

5.3 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or 45 fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

5.4 FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input: identified classes of valid input must be accepted.

Invalid Input: identified classes of invalid input must be rejected.

Functions: identified functions must be exercised.

Output: identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

5.5 SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and 46 predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

The Packet Sniffer application is designed to be installed on a collision domain that uses switches instead of the broadcast domain (HUB). This is because the use of HUBS in network settings is becoming less common due to its broadcasting nature. One of the key advantages of this application compared to similar works is that it displays the layers involved in packet sniffing and the protocols used. The application boasts a rich and user-friendly graphical user interface (GUI) that was developed using the Java Swing technology, making it easy to use. Another significant advantage of using Java is its platform independence, which means that the Packet Sniffer can run on any platform that supports Java. The installation file for the Packet Sniffer is only 587 KB, making it highly economical in terms of memory usage. The application is based on object-oriented design principles, which makes it easy to adapt to any further changes that may be necessary.

6.2 FUTURE WORK

As technology advances, networks are becoming more complex and larger in size. Future work in the development of packet sniffers could focus on incorporating Artificial Intelligence (AI) integration. By using AI, packet sniffers could learn and adapt to network behavior, allowing for real-time analysis and improved network security. Additionally, cloud integration could be implemented to enable remote access to the packet sniffer's functionality and data, which would increase accessibility and convenience.

Furthermore, developing protocol support for the packet sniffer would enhance its ability to detect and analyze a wider range of network protocols, making it a more versatile tool for network administrators. Lastly, visualization tools could be developed to provide a more user-friendly experience, allowing for easier interpretation of data and quick identification of network anomalies.

7. REFERENCES

1. Wahhab Albazrqaoe, Jun Huang and Guoliang Xing, "Network Traffic Monitoring" 2018.[1]
2. Apri Siswanto, Abdul Syukur, Evizal Abdul Kadir and Suratin, "Network Traffic Monitoring" 2019.[2]
3. Jovan Crnogorac, Jelena Kovač, Enis Kočan and Mališa Vučinić, "Distributed IEEE 802.15.4 Sniffer" 2019.[3]
4. Yan Li, Johan Barthelemy, Shuai Sun, Pascal Perez and Bill, "Wi-Fi Sniffing Performance Evaluation" 2020.[4]
5. Jovan Crnogorac, Jelena Kovač, Enis Kočan and Mališa Vučinić, "Sniffing Multi-channel Wireless Sensor Networks" 2020.[5]
6. R. Spangler, "Packet sniffer detection with Antisniff."
7. J. C. McEachen, T. H. Siang, and G. Kirykos, "Design and Implementation of a Modular Wireless Sensor Network Sniffer," 2006.
8. P. Asrodia and H. Patel, "Network traffic analysis using Packet Sniffer," International Journal of Engineering Research and Applications, vol. 2, no. 3, pp. 854–85

8. APPENDIX

8.1 CheckStats

```
package com.first;

import org.pcap4j.core.*;
import org.pcap4j.core.PcapNetworkInterface.*;
import java.util.*;
import javax.swing.*;
import org.pcap4j.packet.*;

public class CheckStats extends javax.swing.JFrame {

    PcapHandle ha = null;
    List<Packet> p1 = new ArrayList();
    int i;

    public CheckStats(List<Packet> p,int index,PcapHandle handle) {
        initComponents();
        p1 = p;
        i = index;
        ha = handle;
        try{
            PcapStat stat = handle.getStats();
            DefaultListModel model = (DefaultListModel)jList1.getModel();
            model.addElement("Packets Received: " + stat.getNumPacketsReceived());
            model.addElement("Packets Dropped: " + stat.getNumPacketsDropped());
            model.addElement("Packets Dropped by the Interface : " +
stat.getNumPacketsDroppedByIf());
```

```
        }catch(PcapNativeException | NotOpenException e){  
    }  
}
```

```
@SuppressWarnings("unchecked")
```

```
private void initComponents() {
```

```
    jLabel2 = new javax.swing.JLabel();  
    jButton1 = new javax.swing.JButton();  
    jLabel1 = new javax.swing.JLabel();  
    jScrollPane2 = new javax.swing.JScrollPane();  
    jList1 = new javax.swing.JList();  
    jLabel3 = new javax.swing.JLabel();
```

```
    jLabel2.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/first/1  
TYAzzTJ60x-qg5N81ElU9A.png"))); // NOI18N
```

```
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);  
    addWindowListener(new java.awt.event.WindowAdapter() {  
        public void windowOpened(java.awt.event.WindowEvent evt) {  
            formWindowOpened(evt);  
        }  
    });  
    getContentPane().setLayout(null);
```

```
    jButton1.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N  
    jButton1.setText("BACK");  
    jButton1.addActionListener(new java.awt.event.ActionListener() {  
        public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```

        jButton1ActionPerformed(evt);
    }
});
getContentPane().add(jButton1);
jButton1.setBounds(180, 260, 105, 38);

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel1.setForeground(new java.awt.Color(255, 255, 255));
jLabel1.setText("STATISTICS");
getContentPane().add(jLabel1);
jLabel1.setBounds(190, 40, 95, 38);

jList1.setModel(new DefaultListModel());
jScrollPane2.setViewportView(jList1);

getContentPane().add(jScrollPane2);
jScrollPane2.setBounds(130, 90, 200, 130);

jLabel3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/first/1
TYAzzTJ60x-qg5N81EIU9A.png"))); // NOI18N
jLabel3.setText("jLabel3");
getContentPane().add(jLabel3);
jLabel3.setBounds(0, 0, 430, 350);

pack();
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
    new PacketDetails(p1,i,ha).setVisible(true);
}

```

```

        this.setVisible(false);
    } //GEN-LAST:event_jButton1ActionPerformed

    private void formWindowOpened(java.awt.event.WindowEvent evt) { //GEN-FIRST:event_formWindowOpened
        setExtendedState(JFrame.MAXIMIZED_BOTH);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
                javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

            java.util.logging.Logger.getLogger(CheckStats.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (InstantiationException ex) {

            java.util.logging.Logger.getLogger(CheckStats.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {

            java.util.logging.Logger.getLogger(CheckStats.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

            java.util.logging.Logger.getLogger(CheckStats.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

```

```

    }

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            }
    });
}

```

```

private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JList jList1;
private javax.swing.JScrollPane jScrollPane2;
}

```

8.2 Packet Details

```

package com.first;

import org.pcap4j.core.*;
import org.pcap4j.core.PcapNetworkInterface.*;
import java.util.*;
import javax.swing.*;
import org.pcap4j.packet.*;

public class PacketDetails extends javax.swing.JFrame {

    PcapHandle ha = null;
    List<Packet> p1 = new ArrayList();
    int i;

    public PacketDetails(List<Packet> p,int index,PcapHandle handle) {
        p1 = p;
    }
}

```

```

        i = index;
        ha = handle;
        initComponents();
        jLabel4.setText(String.valueOf(p1.get(i).getHeader()));
    }

```

```

@SuppressWarnings("unchecked")

```

```

private void initComponents() {

```

```

    jOptionPane1 = new javax.swing.JOptionPane();

```

```

    jButton1 = new javax.swing.JButton();

```

```

    jButton2 = new javax.swing.JButton();

```

```

    jButton3 = new javax.swing.JButton();

```

```

    jButton4 = new javax.swing.JButton();

```

```

    jButton5 = new javax.swing.JButton();

```

```

    jButton6 = new javax.swing.JButton();

```

```

    jButton7 = new javax.swing.JButton();

```

```

    jLabel1 = new javax.swing.JLabel();

```

```

    jLabel2 = new javax.swing.JLabel();

```

```

    jLabel4 = new javax.swing.JLabel();

```

```

    jLabel3 = new javax.swing.JLabel();

```

```

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

```

```

addWindowListener(new java.awt.event.WindowAdapter() {

```

```

    public void windowOpened(java.awt.event.WindowEvent evt) {

```

```

        formWindowOpened(evt);

```

```

    }

```

```

});

```

```

getContentPane().setLayout(null);

```

```
jButton1.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton1.setText("Get Packet Header");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});
getContentPane().add(jButton1);
jButton1.setBounds(80, 100, 150, 90);
```

```
jButton2.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton2.setText("Get Packet Raw Data");
jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
    }
});
getContentPane().add(jButton2);
jButton2.setBounds(240, 100, 170, 90);
```

```
jButton3.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton3.setText("Get Packet Payload");
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton3ActionPerformed(evt);
    }
});
getContentPane().add(jButton3);
```

```

jButton3.setBounds(420, 100, 150, 90);

jButton4.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton4.setText("Get Packet Length");
jButton4.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton4ActionPerformed(evt);
    }
});
getContentPane().add(jButton4);
jButton4.setBounds(240, 200, 170, 90);

jButton5.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton5.setText("Check Statistics");
jButton5.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton5ActionPerformed(evt);
    }
});
getContentPane().add(jButton5);
jButton5.setBounds(80, 200, 150, 90);

jButton6.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton6.setText("Dump Packets");
jButton6.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton6ActionPerformed(evt);
    }
});

```

```

getContentPane().add(jButton6);
jButton6.setBounds(420, 200, 150, 90);

jButton7.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton7.setText("EXIT");
jButton7.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton7ActionPerformed(evt);
    }
});
getContentPane().add(jButton7);
jButton7.setBounds(280, 310, 70, 30);

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel1.setForeground(new java.awt.Color(255, 255, 255));
jLabel1.setText("Selected Packet: ");
getContentPane().add(jLabel1);
jLabel1.setBounds(80, 40, 120, 15);

jLabel2.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel2.setForeground(new java.awt.Color(255, 255, 255));
getContentPane().add(jLabel2);
jLabel2.setBounds(220, 40, 160, 0);

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel4.setForeground(new java.awt.Color(255, 255, 255));
getContentPane().add(jLabel4);
jLabel4.setBounds(230, 40, 290, 20);

```

```
jLabel3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/first/1TYAzzTJ60x-qg5N81EIU9A.png"))); // NOI18N
```

```
jLabel3.setText("jLabel3");
```

```
getContentPane().add(jLabel3);
```

```
jLabel3.setBounds(0, 0, 620, 370);
```

```
pack();
```

```
}
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    new Header(p1,i,ha).setVisible(true);
```

```
    this.setVisible(false);
```

```
}
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    new RawData(p1,i,ha).setVisible(true);
```

```
    this.setVisible(false);
```

```
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    new Payload(p1,i,ha).setVisible(true);
```

```
    this.setVisible(false);
```

```
//GEN-LAST:event_jButton3ActionPerformed
```

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    new Length(p1,i,ha).setVisible(true);
```

```
    this.setVisible(false);
```

```
}
```

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
```

```

        new CheckStats(p1,i,ha).setVisible(true);
        this.setVisible(false);
    }

    private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {

        try{
            PcapDumper dumper = ha.dumpOpen("E:\\vidur.pcap");
            for(int i = 0;i<p1.size();i++){
                dumper.dump(p1.get(1),ha.getTimestamp());
            }
            dumper.close();
            ha.close();

            JOptionPane.showMessageDialog(null,"Successfully Dumped in
E:\\","Success!", JOptionPane.INFORMATION_MESSAGE);
        }catch(PcapNativeException | NotOpenException e){}
    }

    private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {

        System.exit(0);
    }

    private void formWindowOpened(java.awt.event.WindowEvent evt) {

        setExtendedState(JFrame.MAXIMIZED_BOTH);
    }

```

```

public static void main(String args[]) {
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(PacketDetails.class.getName()).log(java.util.logging.L
evel.SEVERE, null, ex);

        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(PacketDetails.class.getName()).log(java.util.logging.L
evel.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(PacketDetails.class.getName()).log(java.util.logging.L
evel.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(PacketDetails.class.getName()).log(java.util.logging.L
evel.SEVERE, null, ex);

        }

        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
            }
        });
    }
}

```

```

private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
private javax.swing.JButton jButton4;
private javax.swing.JButton jButton5;
private javax.swing.JButton jButton6;
private javax.swing.JButton jButton7;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JOptionPane jOptionPane1;
}

```

8.3 View Packet Module

```

package com.first;
import org.pcap4j.core.*;
import org.pcap4j.core.PcapNetworkInterface.*;
import java.util.*;
import javax.swing.*;
import org.pcap4j.packet.*;

public class ViewPackets extends javax.swing.JFrame {

    PcapNetworkInterface device;
    List<Packet> p = new ArrayList();
    int index;

```

```

PcapHandle handle = null;
public ViewPackets(List<PcapNetworkInterface> d, int i) {
    initComponents();
    jLabel7.setText(d.get(i).getDescription());
    index = i;
    device = d.get(i);
}

@SuppressWarnings("unchecked")
private void initComponents() {

    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jButton1 = new javax.swing.JButton();
    jScrollPane1 = new javax.swing.JScrollPane();
    jList1 = new javax.swing.JList();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jButton3 = new javax.swing.JButton();
    jLabel6 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    jLabel8 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowOpened(java.awt.event.WindowEvent evt) {
            formWindowOpened(evt);
        }
    });
    getContentPane().setLayout(null);

    jLabel1.setFont(new java.awt.Font("Segoe UI Symbol", 1, 14)); // NOI18N

```

```

jLabel1.setForeground(new java.awt.Color(255, 255, 255));
jLabel1.setText("Interface Chosen:");
getContentPane().add(jLabel1);
jLabel1.setBounds(70, 20, 130, 20);
getContentPane().add(jLabel2);
jLabel2.setBounds(133, 11, 274, 0);

jButton1.setFont(new java.awt.Font("Lucida Sans", 1, 18)); // NOI18N
jButton1.setText("Start capturing!");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});
getContentPane().add(jButton1);
jButton1.setBounds(530, 20, 230, 30);

jList1.setModel(new DefaultListModel());
jList1.addListSelectionListener(new javax.swing.event.ListSelectionListener() {
    public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
        jList1ValueChanged(evt);
    }
});
jScrollPane1.setViewportView(jList1);

getContentPane().add(jScrollPane1);
jScrollPane1.setBounds(70, 70, 1030, 420);

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jLabel3.setForeground(new java.awt.Color(255, 255, 255));
jLabel3.setText("Selected Packet: ");
getContentPane().add(jLabel3);

```

```

jLabel3.setBounds(90, 550, 130, 17);
getContentPane().add(jLabel4);
jLabel4.setBounds(463, 554, 312, 0);

jButton3.setFont(new java.awt.Font("Lucida Sans", 1, 18)); // NOI18N
jButton3.setText("Click for detailed view");
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton3ActionPerformed(evt);
    }
});
getContentPane().add(jButton3);
jButton3.setBounds(820, 540, 280, 30);

jLabel6.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N
jLabel6.setForeground(new java.awt.Color(255, 255, 255));
getContentPane().add(jLabel6);
jLabel6.setBounds(220, 550, 420, 14);

jLabel7.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N
jLabel7.setForeground(new java.awt.Color(255, 255, 255));
getContentPane().add(jLabel7);
jLabel7.setBounds(220, 20, 210, 14);

jLabel8.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/first/1
TYAzzTJ60x-qg5N81EIU9A.png"))); // NOI18N
jLabel8.setText("jLabel8");
getContentPane().add(jLabel8);
jLabel8.setBounds(0, 0, 1150, 620);

pack();
}

```

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {GEN-
FIRST:event_jButton1ActionPerformed

    int snaplen = 65536;
    int timeout = 150;
    try{
        handle = device.openLive(snaplen,PromiscuousMode.PROMISCUOUS,timeout);
    }catch(PcapNativeException e){}

    DefaultListModel model = (DefaultListModel)jList1.getModel();

    PacketListener listener = new PacketListener(){
        @Override
        public void gotPacket(Packet packet){
            model.addElement(packet);
            p.add(packet);
        }
    };

    try{
        int max = 100;
        handle.loop(max,listener);
    }catch(InterruptedException | PcapNativeException | NotOpenException
e){e.printStackTrace();}
}

private void jList1ValueChanged(javax.swing.event.ListSelectionEvent evt) {GEN-
FIRST:event_jList1ValueChanged

    index = jList1.getSelectedIndex();
    jLabel6.setText(String.valueOf(index) + " :" + p.get(index).getHeader());
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {GEN-
FIRST:event_jButton3ActionPerformed

```

```

        new PacketDetails(p,index,handle).setVisible(true);
        this.setVisible(false);
    }

    private void formWindowOpened(java.awt.event.WindowEvent evt) {GEN-
FIRST:event_formWindowOpened
        setExtendedState(JFrame.MAXIMIZED_BOTH);
    }//GEN-LAST:event_formWindowOpened

    public static void main(String args[]) {

        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

            java.util.logging.Logger.getLogger(ViewPackets.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);

        } catch (InstantiationException ex) {

            java.util.logging.Logger.getLogger(ViewPackets.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {

            java.util.logging.Logger.getLogger(ViewPackets.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

            java.util.logging.Logger.getLogger(ViewPackets.class.getName()).log(java.util.logging.Le
vel.SEVERE, null, ex);

        }
    }

```

```

java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {

    }
});
}

```

```

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton3;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JList jList1;
private javax.swing.JScrollPane jScrollPane1;
}

```

8.4 Packet Details

```

package com.first;
import java.util.*;
import org.pcap4j.core.PcapNetworkInterface;
import org.pcap4j.util.NifSelector;
import java.io.IOException;
import org.pcap4j.core.*;
import javax.swing.*;
import java.sql.*;

```

```

public class NetworkInterfaces extends javax.swing.JFrame {

    int index;
    List<PcapNetworkInterface> device = null;
    public NetworkInterfaces() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jScrollPane2 = new javax.swing.JScrollPane();
        jList1 = new javax.swing.JList();
        jLabel1 = new javax.swing.JLabel();
        jButton2 = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jButton3 = new javax.swing.JButton();
        jLabel7 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowOpened(java.awt.event.WindowEvent evt) {
                formWindowOpened(evt);
            }
        }

```

```

});
getContentPane().setLayout(null);

jButton1.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton1.setText("Check available interfaces");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});
getContentPane().add(jButton1);
jButton1.setBounds(250, 370, 195, 40);

jList1.setModel(new DefaultListModel());
jList1.addListSelectionListener(new javax.swing.event.ListSelectionListener() {
    public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
        jList1ValueChanged(evt);
    }
});
jScrollPane2.setViewportView(jList1);

getContentPane().add(jScrollPane2);
jScrollPane2.setBounds(26, 70, 680, 294);

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel1.setForeground(new java.awt.Color(255, 255, 255));
jLabel1.setText("Selected Interface:");
getContentPane().add(jLabel1);
jLabel1.setBounds(213, 430, 120, 15);

jButton2.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jButton2.setText("View Packets");

```

```

jButton2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton2ActionPerformed(evt);
    }
});
getContentPane().add(jButton2);
jButton2.setBounds(300, 470, 140, 30);
getContentPane().add(jLabel2);
jLabel2.setBounds(139, 404, 570, 0);

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
jLabel3.setForeground(new java.awt.Color(255, 255, 255));
jLabel3.setText("NETWORK PACKET SNIFFER");
getContentPane().add(jLabel3);
jLabel3.setBounds(240, 20, 290, 22);

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel4.setForeground(new java.awt.Color(255, 255, 255));
getContentPane().add(jLabel4);
jLabel4.setBounds(360, 430, 150, 0);

jLabel6.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
jLabel6.setForeground(new java.awt.Color(255, 255, 255));
getContentPane().add(jLabel6);
jLabel6.setBounds(350, 430, 260, 20);

jButton3.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N
jButton3.setText("EXIT");
jButton3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton3ActionPerformed(evt);
    }
}

```

```

});
getContentPane().add(jButton3);
jButton3.setBounds(310, 520, 110, 40);

jLabel7.setIcon(new javax.swing.ImageIcon(getClass().getResource("/com/first/1
TYAzzTJ60x-qg5N81ElU9A.png"))); // NOI18N
jLabel7.setText("jLabel7");
getContentPane().add(jLabel7);
jLabel7.setBounds(0, 10, 730, 560);

pack();
} // </editor-fold> // GEN-END: initComponents

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST:event_jButton1ActionPerformed
    // TODO add your handling code here:
    try{
        device = Pcaps.findAllDevs();
    } catch (PcapNativeException e){
        e.printStackTrace();
    }

    DefaultListModel model = (DefaultListModel)jList1.getModel();
    for( int i = 0; i < device.size(); i++){
        model.addElement(i+1 + ". Name: " + device.get(i).getName() + " | Description: "
+ device.get(i).getDescription() + "\n");
    }
} // GEN-LAST:event_jButton1ActionPerformed

private void jList1ValueChanged(javax.swing.event.ListSelectionEvent evt) { // GEN-
FIRST:event_jList1ValueChanged
    index = jList1.getSelectedIndex();
    jLabel6.setText(device.get(index).getDescription());
} // GEN-LAST:event_jList1ValueChanged

```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton2ActionPerformed
```

```
    new ViewPackets(device,index).setVisible(true);
```

```
    this.setVisible(false);
```

```
//GEN-LAST:event_jButton2ActionPerformed
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton3ActionPerformed
```

```
    System.exit(0);
```

```
//GEN-LAST:event_jButton3ActionPerformed
```

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {//GEN-FIRST:event_formWindowOpened
```

```
    setExtendedState(JFrame.MAXIMIZED_BOTH);
```

```
//GEN-LAST:event_formWindowOpened
```

```
public static void main(String args[]) {
```

```
    for (javax.swing.UIManager.LookAndFeelInfo info :  
    javax.swing.UIManager.getInstalledLookAndFeels()) {
```

```
        if ("Nimbus".equals(info.getName())) {
```

```
            javax.swing.UIManager.setLookAndFeel(info.getClassName());
```

```
            break;
```

```
        }
```

```
    }
```

```
    } catch (ClassNotFoundException ex) {
```

```
        java.util.logging.Logger.getLogger(NetworkInterfaces.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
    } catch (InstantiationException ex) {
```

```
        java.util.logging.Logger.getLogger(NetworkInterfaces.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
    } catch (IllegalAccessException ex) {
```

```
java.util.logging.Logger.getLogger(NetworkInterfaces.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
```

```
java.util.logging.Logger.getLogger(NetworkInterfaces.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
```

```
    }
```

```
    /* Create and display the form */
```

```
    java.awt.EventQueue.invokeLater(new Runnable() {
```

```
        public void run() {
```

```
            new NetworkInterfaces().setVisible(true);
```

```
        }
```

```
    });
```

```
}
```

```
private javax.swing.JButton jButton1;
```

```
private javax.swing.JButton jButton2;
```

```
private javax.swing.JButton jButton3;
```

```
private javax.swing.JLabel jLabel1;
```

```
private javax.swing.JLabel jLabel2;
```

```
private javax.swing.JLabel jLabel3;
```

```
private javax.swing.JLabel jLabel4;
```

```
private javax.swing.JLabel jLabel6;
```

```
private javax.swing.JLabel jLabel7;
```

```
private javax.swing.JList jList1;
```

```
private javax.swing.JScrollPane jScrollPane2;
```

```
}
```

8.5 Screenshots

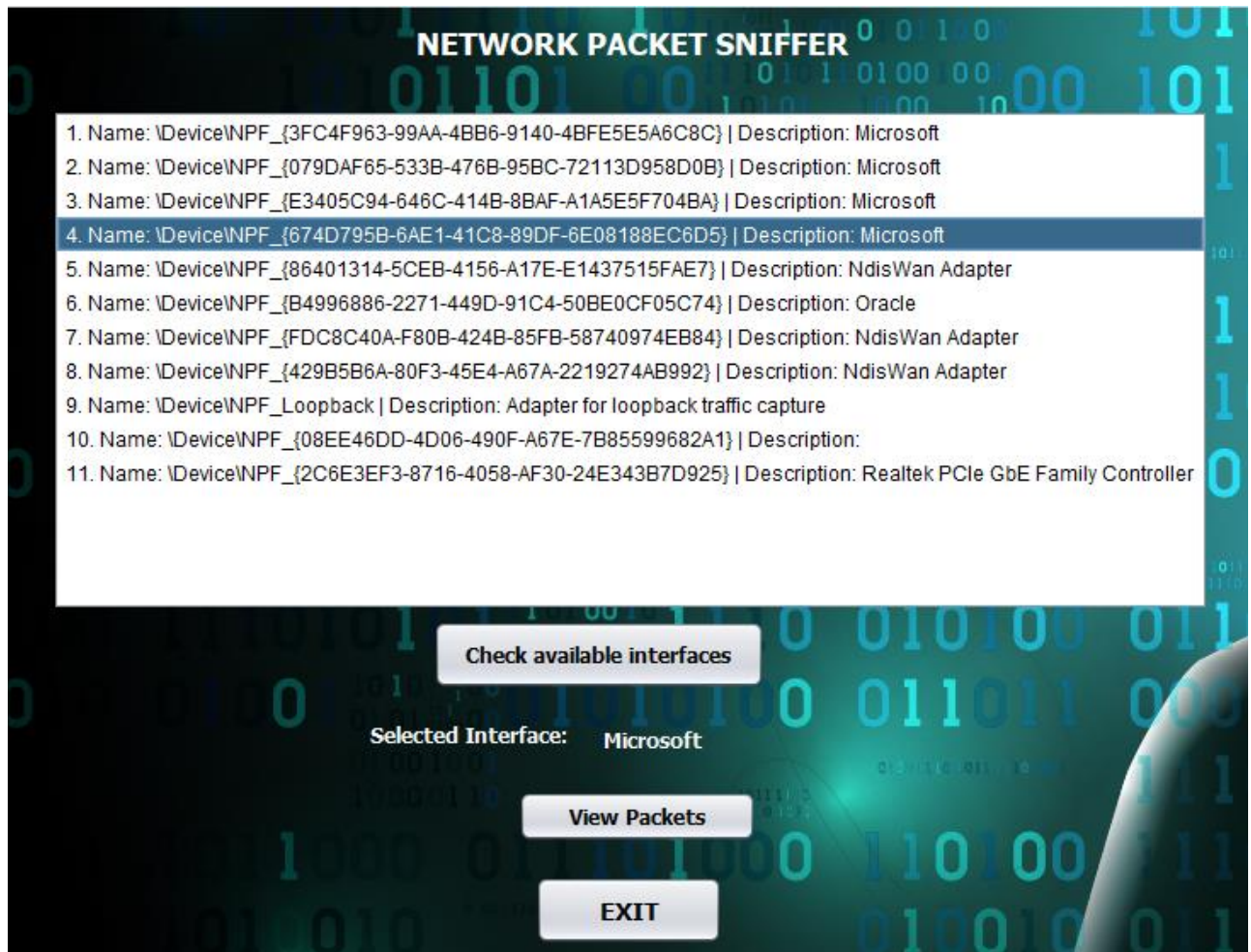


Figure 8.5.1 Home page

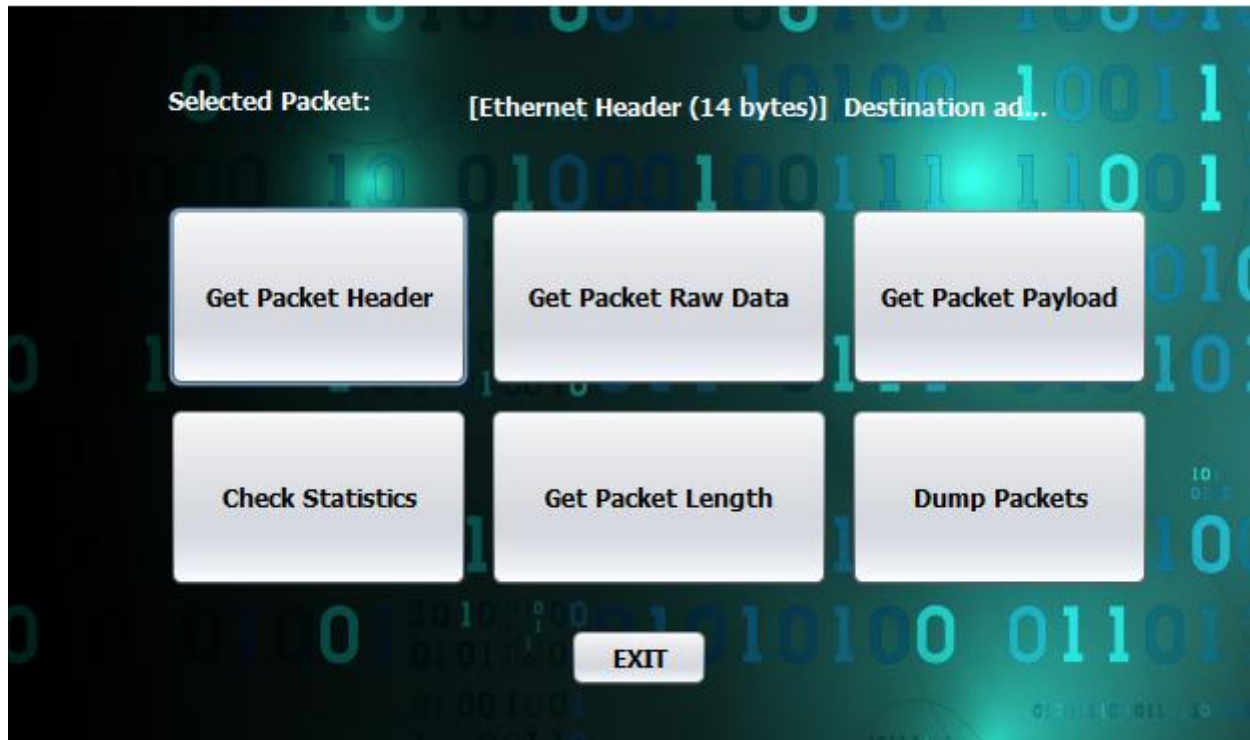


Figure 8.5.2 Packet Details Page



Figure 8.5.3 Packet Length Page