

Mini Project Report Submission Format



Packet Sniffer

Guide By

Mr. Rahul Shandilya
Assistant Professor
Computer Engineering and Applications

Year of Submission

2023-2024

Submitted to

Mr. Rahul Shandilya

Department of Computer Engineering and Applications
GLA University, Mathura

Students Name

Rachita Gupta, 2115300015, VI

Shreya Roy, 2115300020, VI

Kartikey Garg, 2115300010, VI

Natansh Chauhan, 2115300014, VI

Chapter Index

Page No

Abstract

Certificate

Declaration

List of Abbreviations, Tables and Figures

1. Introduction	5
2. Literature Reviews with Table Summary	6
3. Research Gap Motivation	7
4. Proposed Methodology with Diagram	8
5. Results & Experiments	9
6. Conclusion & Future Work	10
7. References	

CERTIFICATE

This is to certify that the work titled "PACKET SNIFFER" submitted by Kartikey Garg, Natansh Chauhan, Rachita Gupta, Shreya Roy in partial fulfillment for the award of degree of B.Tech CS(CSF) of GLA University, Mathura has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or Engineering.

Signature of Supervisor:

Name of Supervisor: Mr. Rahul Shandilya

Designation: Assistant Professor

Date:

ACKNOWLEDGEMENT

We would like to express our sincere thanks and gratitude to our guide Mr. Rahul Shandilya for his valuable guidance and suggestions. We highly indebted to him for providing me an excellent opportunity to learn and present my studies in the form of this project. We take this opportunity to thank the members of the teaching and non-teaching staff of GLA for the timely help extended by them. Lastly thanking our parents, for their morale support and encouragement.

1. Introduction

Packets in computer communications can be defined as a quantity of data of limited size. In Internet all traffic travels in the form of packets, the entire file downloads, Web page retrievals, email, all these Internet communications always occur in the form of packets. In the internet, packet is a formatted unit of data carried by a packet mode in computer network. This work develop a PSniffer network analyzer that can capture network traffic and analyze it and allows user to take only the feature as needed with little memory usage for installation and store it in a file to use it later in his work, then this will reduce the memory that is used to store the data unlike available tools can only capture network traffic without analysis, while some require large memory size for installation. In addition, have a user friendly control interface.

Network sniffing is a network layer attack consisting of capturing packets from the network transmitted by other computers and reading the data content in search of sensitive information like passwords, session tokens and confidential information. This could be done using tools called network sniffers; these tools collect packets on the network and, depending on the quality of the tool, analyze the collected data like protocol decoders or stream reassembling.

Sniffer is used as an assistant of network management because of its monitoring and analyzing features which can help to troubleshoot network, detect intrusion, control traffic or supervise network contents.

Psniffer when installed in a network will help monitor network traffic and keeps log of all connections to the network, which is then analyzed for the detection of suspicious activities.

1.1 AIM

This project aims at developing a Network Packet Sniffer. Network Packet Sniffer is a piece of software that monitors all network traffic. This is unlike standard network hosts that only receive traffic sent specifically to them. As data streams flow across the network, the sniffer captures each packet and eventually decodes and analyzes its content. For network monitoring purposes it may also be desirable to monitor all data packets in a LAN and to mirror all packets passing through a shared bus.

1.2 EXISTING SYSTEM

As a network administrator who needs to identify, diagnose, and solve network problems, a company manager who wants to monitor user activities on the network and ensure that the corporation's communications assets are safe, or a consultant who has to quickly solve network problems for clients. It is difficult to identify the problems if the network traffic is not tracked, as an administrator in general we depend on the analyzer provided by the operating system (if any) or the antivirus software that is installed to provide real-time network security.

1.3 Drawbacks with the Existing System

- Administrators need to put lot of efforts to identify the traffic
- Time taking process.
- No possibility of automatic network control.
- Presence of administrator is compulsory

2. Literature Reviews (Minimum 8 research papers) with Table Summary

Packet Sniffing Tools are computer hardware/software that captures traffic traveling over any sort of digital network. As data is sent across the network, the packet sniffer intercepts the packets and decodes them accordingly. These tools can be used for a variety of reasons including monitoring network performance or stealing passwords/personal information sent. There are many packet sniffing programs available; many of which are free of charge. Here is a review of multiple packet sniffers that we were able to uncover and a description of what they do.

1. **Wireshark/Ethereal** – Most popular packet sniffing tool because of its many features and supportability. It's a Windows/Unix open source network analyzer that supports many protocols. It allows users to examine every packet individually or the network as a whole.
2. **Kismet**– Powerful wireless sniffer that identifies networks using passive sniffing. It detects network sniffing multiple types of packets such as Ip, Tcp, Udp, Arp, and Dhcp packets. Its primary use is for something called wardriving – that is finding wireless networks/hotspots in a location or while traveling.
3. **Tcpdump**– Classic sniffer tool that uses command prompt (platform for many other tools such as Wireshark). Used primarily for network monitoring and data acquisition. The advantages of a smaller tool are less security worries and it doesn't require a lot of resources.
4. **Cain and Abel** – Primary password detection/recovery for Windows systems. This software sniffs networks to recover passwords; it cracks encryptions, decodes scramblers, uncovers cached, and records VoIP. It uses multiple algorithms to accomplish these tasks.
5. **Ettercap** – Terminal based sniffer for Ethernet LANs. It is capable of dissecting almost any security protocol (https, ssh) while keeping the connection synchronized. Also determines the geometry (type) of LAN.
6. **Dsniff** – Suite of powerful examining and penetration tools used for monitoring networks. Many tools are included to sniff whatever data you are looking for through multiple protocols. Capable of recovering very confidential information and passwords.
7. **NetStumbler** – Popular windows tool for finding open wireless access points. Can be used in many handheld devices such as phones and pdas. Similar to Kismet except NetStumbler takes a much more active approach to finding wireless networks.
8. **Ntop** – Primarily used to analyze network traffic and usage. It displays network status in its terminal and can act like a web server and store many different traffic statistics.
9. **Ngrep** – Basic traffic monitoring tool that mirrors much of gnu's grep. Allows users to specify expressions to match within packets and recognizes most networking protocols.

3. Research Gap Motivation

Identified Research Gaps

1. Efficiency and Performance in High-Speed Networks

- **Current Challenges:** Many existing packet sniffers struggle to process the vast amounts of data generated in high-speed networks efficiently. This results in packet loss and inaccurate traffic analysis.
- **Research Need:** Development of optimized algorithms and hardware acceleration techniques to improve data processing rates and reduce packet loss.

2. Real-Time Analysis Capabilities

- **Current Challenges:** Real-time packet analysis often suffers from latency, which can delay threat detection and response times.
- **Research Need:** Implementation of low-latency processing techniques and real-time data analysis frameworks to enhance immediate detection and reaction to network anomalies and security threats.

3. Handling Encrypted Traffic

- **Current Challenges:** The increasing use of encryption in network traffic limits the ability of packet sniffers to analyze data content, leading to incomplete traffic analysis.
- **Research Need:** Development of methods for analyzing encrypted traffic through metadata, traffic patterns, and machine learning techniques without compromising data privacy.

4. Scalability in Large Networks

- **Current Challenges:** As networks grow in size and complexity, packet sniffers face difficulties in scaling their operations to monitor all network segments effectively.
- **Research Need:** Creation of scalable packet sniffing architectures, including distributed systems and cloud-based solutions, to handle large-scale network environments.

5. User Accessibility and Visualization

- **Current Challenges:** Many packet sniffers have complex interfaces that require extensive technical knowledge, making them less accessible to non-expert users. Additionally, visualization of captured data is often inadequate for quick analysis.
- **Research Need:** Design of user-friendly interfaces and advanced visualization tools that simplify data interpretation and make packet sniffers more accessible to a broader range of users.

6. Integration with Emerging Technologies

- **Current Challenges:** Packet sniffers need to adapt to monitor traffic from emerging technologies such as Internet of Things (IoT) devices and 5G networks effectively.
- **Research Need:** Development of specialized packet sniffing techniques for IoT and 5G networks, considering their unique characteristics and requirements.

Motivation for the Mini Project

1. Enhancing Network Security

- **Proactive Threat Detection:** Improving real-time analysis and handling of encrypted traffic will enhance the ability to detect and mitigate security threats promptly.
- **Comprehensive Monitoring:** Addressing scalability and efficiency will ensure comprehensive monitoring of modern, high-speed networks.

2. Improving Network Management

- **Efficient Troubleshooting:** Enhancements in packet sniffing performance and user accessibility will streamline network troubleshooting processes, reducing downtime and improving reliability.
- **Bandwidth Optimization:** Better analysis tools can help optimize network bandwidth usage, improving overall network performance.

3. Supporting Innovation and Research

- **Emerging Technologies:** Adapting packet sniffers to new technologies such as IoT and 5G will support ongoing innovation and research in these fields.
- **Educational Tools:** Creating more user-friendly and accessible packet sniffers can serve as educational tools, helping students and professionals better understand network protocols and security.

4. Ensuring Regulatory Compliance

- **Data Privacy and Protection:** Improved packet sniffing capabilities can assist organizations in ensuring data privacy and protection, helping them comply with regulatory requirements.

4. Proposed Methodology with Diagram

Installation on Windows requires WinPcap software which can be downloaded from winPcap website. Jpcap is a set of Java classes which provide an interface and system for network packet capture, it is required for packet capture in Java and built upon Libpcap which is a packet capture library in C language. Java Runtime Environment (JRE) 5.0 or higher will also be required to run this Java application. More space may be required to store the captured packets since the required space on hard disk for installation is less than 1MB.

Psniffer is a customized software application that has a number of features. These features enable:

- Administrators to show statistics of received packets.
- Administrators detect malicious IP addresses according to its number of ARP requests in previously specified time.
- Administrators to view all network interfaces and enable them to capture data from that interface and consequently save captured packets.
- Administrators generate reports that aid effective and efficient decision making.

The Psniffer is developed in Java™. This application is designed into five (5) independent modules which take care of different tasks efficiently.

1. **User Interface Module.**
2. **Packet Sniffing Module.**
3. **Analyze layers Module.**
4. **Free Memory Module.**
5. **Protocol Analysis Module.**

1.1 [MODULES](#)

User Interface Module:

Actually every application has one user interface for accessing the entire application. The user interface for the Psniffer application is designed completely based on the end users. It provides an easy to use interface to the users. This user interface has an attractive look and provides ease of navigation. Technically, the swing is used in core java for preparing this user interface.

Packet Sniffing Module:

This module takes care of capturing packets that are seen by a machine's network interface. It grabs all the packets that goes in and out of the Network Interface Card (NIC) of the machine on which the sniffer is installed. This means that, if the NIC is set to the promiscuous mode, then it will receive all the packets sent to the network.

Analyze Layers Module:

This module contains the code for analyzing the layers in the system. Mostly in this module we have to discuss about three layers Transport layer, Application Layer, Network Layer. The module shows the graphical representation of the usage of different layers in packet capturing time. It can show the graph like line graph.

Free Memory Module: This module analyzes computer memory usage at the time of packet capturing. It can show the memory size in number format as well as graphical representation.

Protocol Analysis Module: This module analyzes the protocols of the layers. Like Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Hypertext Transfer Protocol (HTTP) etc. It can show the source port, destination port and packet length of the system of each protocol.

1.2 INSTALLATION & IMPLEMENTATION OF CODE

Installation on Windows requires WinPcap software which can be downloaded from winPcap website. Jpcap is a set of Java classes which provide an interface and system for network packet capture, it is required for packet capture in Java and built upon Libpcap which is a packet capture library in C language. Java Runtime Environment (JRE) 5.0 or higher will also be required to run this Java application. More space may be required to store the captured packets since the required space on hard disk for installation is less than 1MB.

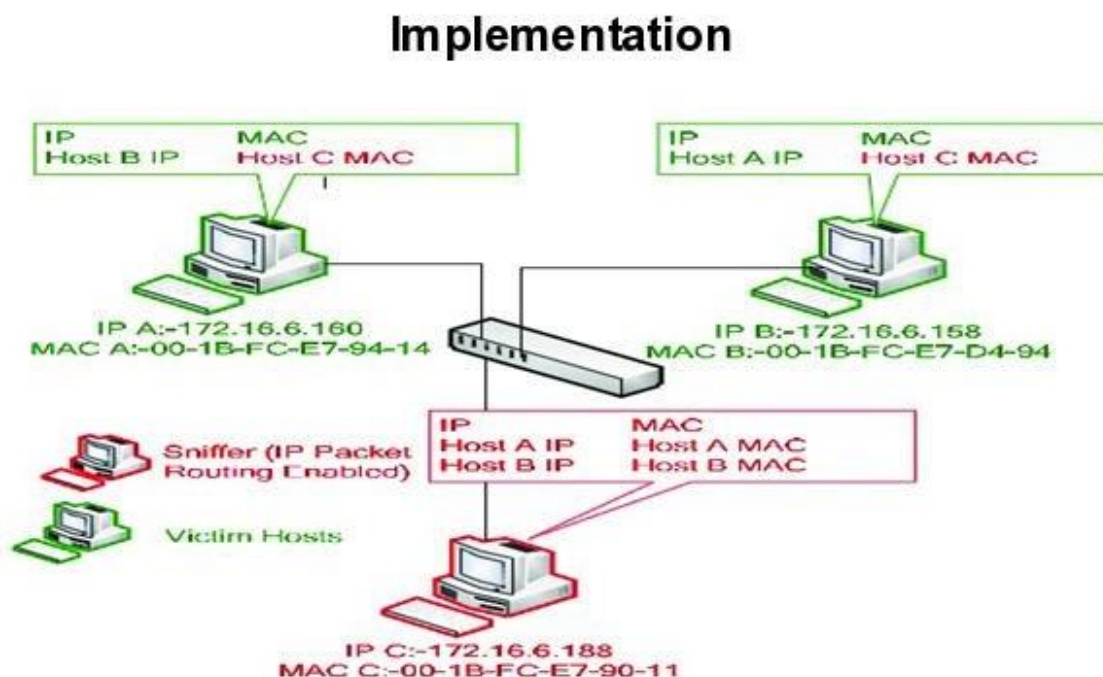


Fig.4.2.1 ARP Cache poisoning that would be used for the implementation.

Sniffers are programs that allow a host to capture any network packet illicitly. Specially, if the sniffers are active because active sniffer can alter or block network traffic while passive sniffer can only monitor network traffic. In this work the Passive Sniffer would be used. There are two ways to sniff network traffic:

A host running a sniffer sets its NIC in promiscuous mode. If any host's NIC is running in promiscuous mode, it will receive all packets either those packets targeted to it or not. This way of sniffing is effective in an environment which is broadcast in nature like hub, access point and bus Local Area Network (LAN) environments.

ARP cache poisoning is also used for sniffing. This way of sniffing is effective in an environment, which is not broadcast in nature. Address Resolution Protocol (ARP) cache poisoning depends on local ARP cache maintained by each host of network. This cache contains IP with corresponding Media Access Control (MAC) addresses of recently accessed hosts.

ARP cache poisoning process that would be used for implementation. In this diagram, 'C' host performs ARP cache poisoning attack. 'C' host sends an ARP poison packet to target host 'A' which contains host 'C' MAC address in source MAC address field and host 'B' IP address in source IP address field of ARP poison packet. When target host 'A' receives this packet, it poisons local ARP cache value either by adding false entry or updating old entry with new one. Same process is repeated with host 'B'. This process corrupts the local ARP caches of host 'A' and 'B' which are shown in Figure 5. After the completion of poisoning process, both hosts cannot communicate directly with each other. Each host sends a packet to sniffer host and sniffer host reroutes packet back to actual destination. Sniffer host must have IP packet routing enabled so that it could send packet back to actual destination after getting confidential information.

1.3 [SCREENSHOTS](#)

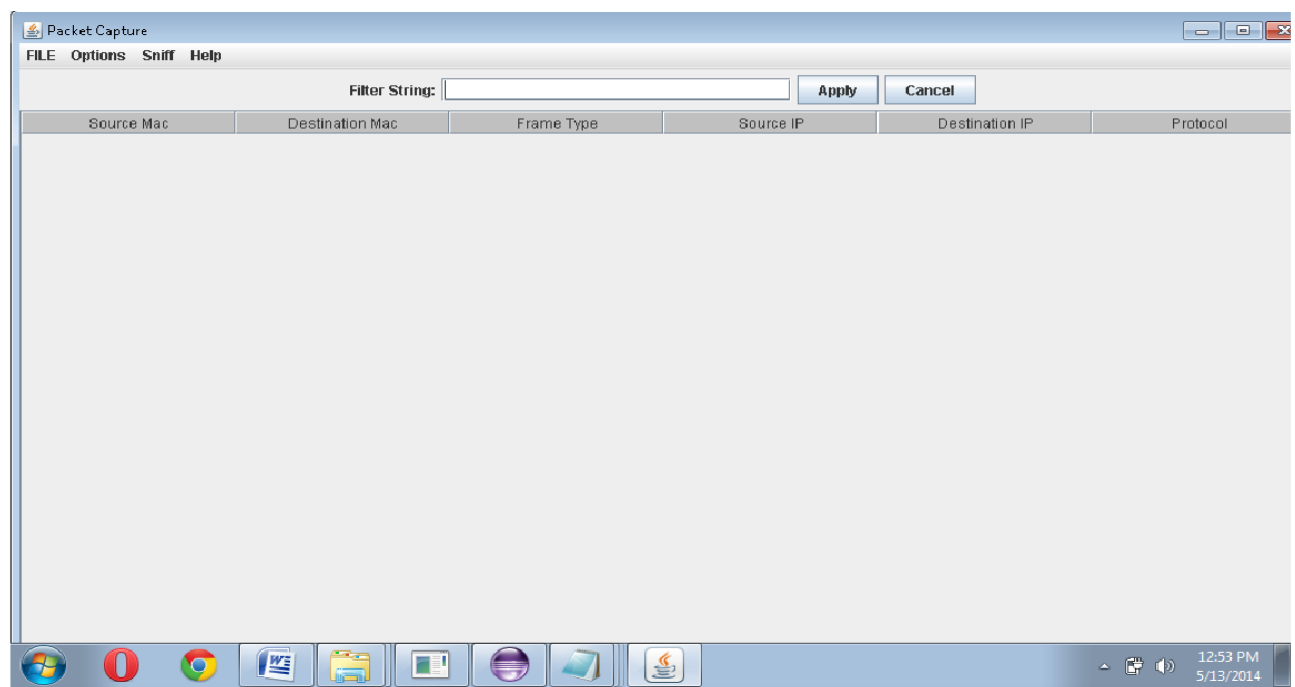


Fig.4.3.1 User Interface of Packet Sniffer

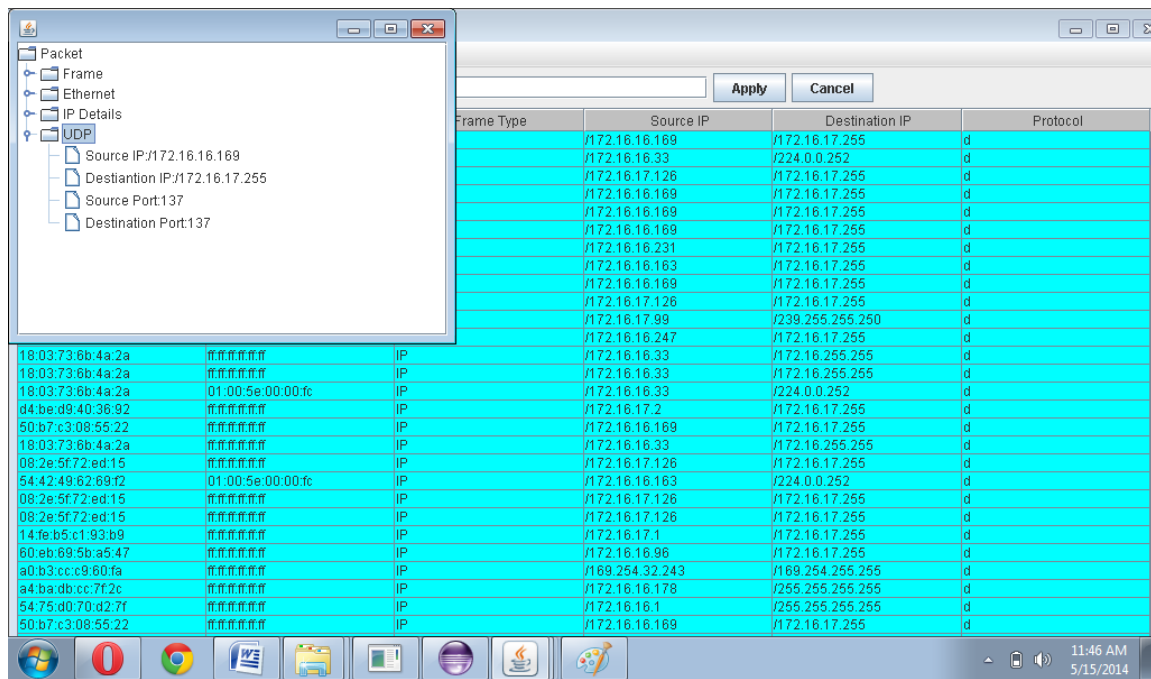


Fig.4.3.4 PacketSniffer is Sniffing UDP Packets from the Ethernet Port using udp filter

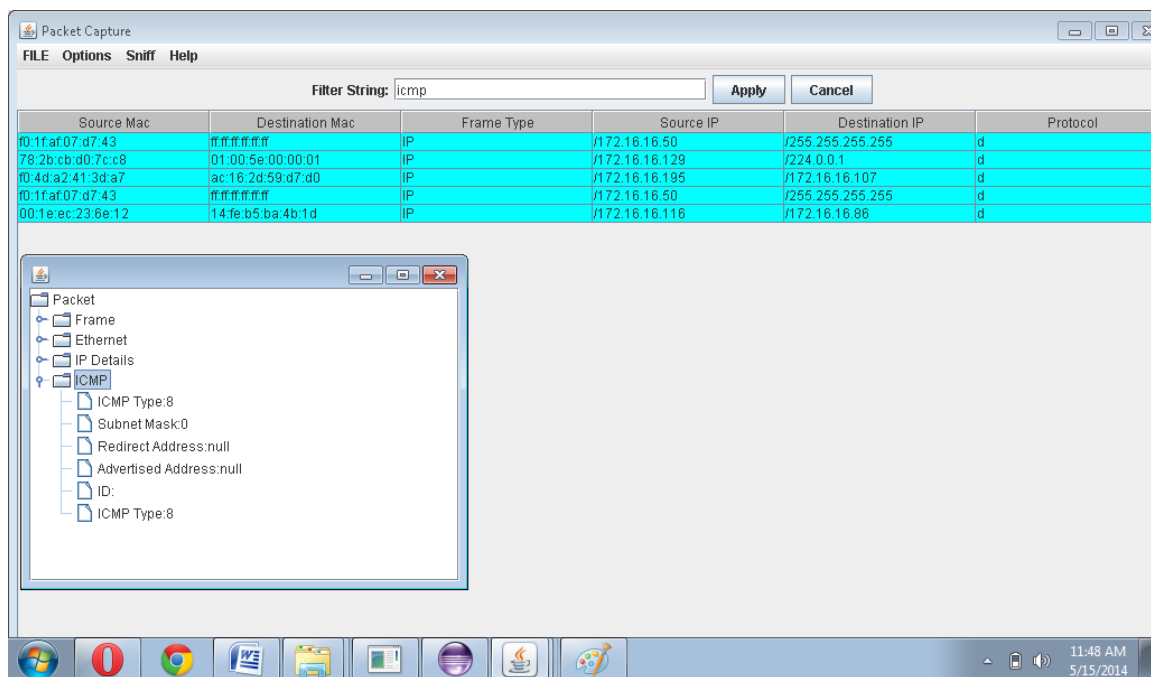


Fig.4.3.5 PacketSniffer is Sniffing ICMP Packets from the Ethernet Port using icmp filter

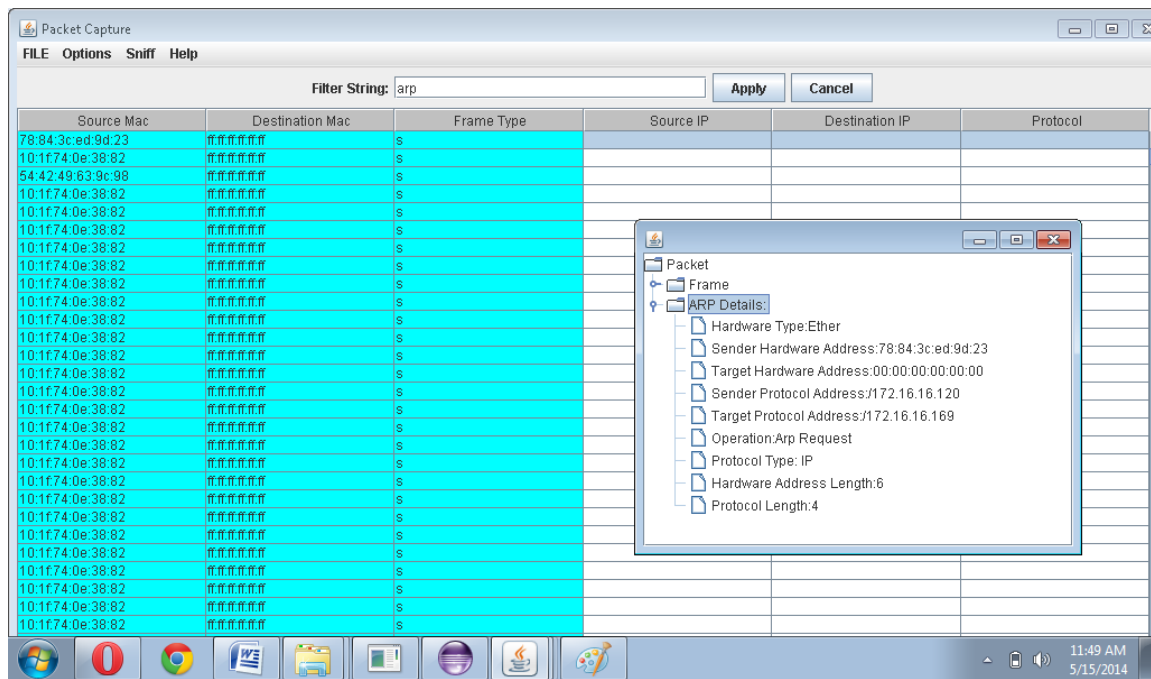


Fig.4.3.6 PacketSniffer is Sniffing ARP Packets from the Ethernet Port using arp filter

1.4 [CODE](#)

```

/* 1:  */ import java.awt.BorderLayout;

/* 2:  */ import java.awt.Color;

/* 3:  */ import java.awt.event.ActionEvent;

/* 4:  */ import java.awt.event.ActionListener;

/* 5:  */ import java.awt.event.MouseEvent;

/* 6:  */ import java.awt.event.MouseListener;

/* 7:  */ import java.awt.event.WindowAdapter;

/* 8:  */ import java.awt.event.WindowEvent;

/* 9:  */ import java.io.IOException;

/* 10: */ import java.io.PrintStream;

/* 11: */ import java.util.ArrayList;

/* 12: */ import javax.swing.JDialog;

```



```
/* 13: */ import javax.swing.JFileChooser;

/* 14: */ import javax.swing.JFrame;

/* 15: */ import javax.swing.JLabel;

/* 16: */ import javax.swing.JMenu;

/* 17: */ import javax.swing.JMenuBar;

/* 18: */ import javax.swing.JMenuItem;

/* 19: */ import javax.swing.JOptionPane;

/* 20: */ import javax.swing.JPanel;

/* 21: */ import javax.swing.JScrollPane;

/* 22: */ import javax.swing.JTable;

/* 23: */ import javax.swing.SwingUtilities;

/* 24: */ import javax.swing.table.DefaultTableModel;

/* 25: */ import javax.swing.table.JTableHeader;

/* 26: */ import javax.swing.table.TableColumn;

/* 27: */ import javax.swing.table.TableColumnModel;

/* 28: */ import javax.swing.table.TableModel;

/* 29: */ import jpcap.JpcapCaptor;
```

```
import jpcap.packet.Packet;

/* 31:  */

/* 32:  */ public class gui

/* 33:  */ extends JPanel

/* 34:  */ implements ActionListener, MouseListener

/* 35:  */ {

/* 36:  */ JMenuBar mb;

/* 37:  */ gui.update n;

/* 38:  */ JScrollPane jsp;

/* 39: 14 */ ArrayList<Packet> arrp = new ArrayList();

/* 40:  */ JMenuItem mi1;

/* 41:  */ JMenuItem mi2;

/* 42:  */ JMenuItem mi3;

/* 43:  */ JMenuItem mi4;

/* 44:  */ JMenuItem mi5;

/* 45:  */ JMenuItem mi6;

/* 46:  */ JMenuItem mi7;

/* 47:  */ JMenu m1;

/* 48:  */ JMenu m2;

/* 49:  */ JMenu m3;

/* 50:  */ JMenu m4;

/* 51:  */ sniffer snif;
```

```
/* 52: 18 */ final RowHighlightRenderer renderer = new RowHighlightRenderer();

/* 53:  */ JpcapCaptor cap;

/* 54:  */ fstring fil;

/* 55:  */ JFrame f;

/* 56:  */ DefaultTableModel tm;

/* 57: 23 */ String[] columns = { "Source Mac", "Destination Mac", "Frame Type", "Source IP",
"Destination IP", "Protocol" };

/* 58:  */ JTable tb;

/* 59:  */

/* 60:  */ gui()

/* 61:  */ {

/* 62: 27 */ this.f = new JFrame("Packet Capture");

/* 63: 28 */ setupLayout();

/* 64: 29 */ setSize(1200, 800);

/* 65: 30 */ this.f.setSize(1200, 800);

/* 66: 31 */ this.f.setDefaultCloseOperation(0);

/* 67: 32 */ this.f.addWindowListener(

/* 68:  */

/* 69: 34 */ new WindowAdapter()

/* 70:  */ {
```

```

/* 71: */      public void windowClosing(WindowEvent w)

/* 72: */      {

/* 73: 38 */      if (!gui.this.arrp.isEmpty())

/* 74: */      {

/* 75: 40 */      int check = JOptionPane.showConfirmDialog(gui.this.f, "Do You
want to save current Data");

/* 76: 41 */      if (check == 0)

/* 77: */      {

/* 78: 43 */      JFileChooser chooser = new JFileChooser();

/* 79: 44 */      int value = chooser.showSaveDialog(gui.this.f);

/* 80: 45 */      if (value == 0)

/* 81: */      {

/* 82: 47 */      writer wr = new writer(gui.this.cap, chooser.getSelectedFile());

/* 83: 48 */      wr.save(gui.this.arrp);

/* 84: 49 */      System.out.println("Saved");

/* 85: */      }

/* 86: */      }

/* 87: 52 */      else if (check == 2)

/* 88: */      {

/* 89: 54 */      return;

/* 90: */      }

/* 91: */      }

/* 92: 57 */      if ((gui.this.n instanceof gui.update)) {

/* 93: 59 */      if (gui.this.n.isAlive())

/* 94: */      {

```

```
/* 95: 61 */      gui.this.n.stop();
/* 96: 62 */      gui.this.snif.stop();
/* 97:  */      }
/* 98:  */      }
/* 99: 65 */      System.exit(0);
/* 100:  */      }
/* 101: 68 */      });
/* 102: 69 */      this.f.setVisible(true);
/* 103: 70 */      this.snif = new sniffer();
/* 104:  */      }
/* 105:  */
/* 106:  */      private void setupLayout()
/* 107:  */      {
/* 108: 74 */      this.tm = new DefaultTableModel(null, this.columns);
/* 109: 75 */      this.tb = new JTable(this.tm)
/* 110:  */      {
/* 111:  */      public boolean isCellEditable(int rowIndex, int colIndex)
/* 112:  */      {
/* 113: 79 */      return false;
/* 114:  */      }
/* 115: 87 */      };
/* 116: 88 */      TableColumn tcol = this.tb.getColumnModel().getColumn(0);
/* 117: 89 */      tcol.setCellRenderer(new RowHighlightRenderer());
/* 118: 90 */      tcol = this.tb.getColumnModel().getColumn(1);
/* 119: 91 */      tcol.setCellRenderer(new RowHighlightRenderer());
```

```
/* 120: 92 */    tcol = this.tb.getColumnModel().getColumn(2);
/* 121: 93 */    tcol.setCellRenderer(new RowHighlightRenderer());
/* 122: 94 */    tcol = this.tb.getColumnModel().getColumn(3);
/* 123: 95 */    tcol.setCellRenderer(new RowHighlightRenderer());
/* 124: 96 */    tcol = this.tb.getColumnModel().getColumn(4);
/* 125: 97 */    tcol.setCellRenderer(new RowHighlightRenderer());
/* 126: 98 */    tcol = this.tb.getColumnModel().getColumn(5);
/* 127: 99 */    tcol.setCellRenderer(new RowHighlightRenderer());
/* 128:100 */    JTableHeader header = this.tb.getTableHeader();
/* 129:101 */    header.setBackground(Color.LIGHT_GRAY);
/* 130:102 */    this.fil = new fstring(this);
/* 131:103 */    this.mb = new JMenuBar();
/* 132:104 */    this.jsp = new JScrollPane(this.tb);
/* 133:105 */    this.m1 = new JMenu("FILE");
/* 134:106 */    this.m2 = new JMenu("Options");
/* 135:107 */    this.m3 = new JMenu("Help");
/* 136:108 */    this.m4 = new JMenu("Sniff");
/* 137:109 */    this.mi1 = new JMenuItem("Open");
/* 138:110 */    this.mi2 = new JMenuItem("Save");
/* 139:111 */    this.mi3 = new JMenuItem("Exit");
/* 140:112 */    this.mi4 = new JMenuItem("Options");
/* 141:113 */    this.mi5 = new JMenuItem("Choose Device");
/* 142:114 */    this.mi6 = new JMenuItem("Start Sniffing");
/* 143:115 */    this.mi7 = new JMenuItem("Stop Sniffing");
/* 144:116 */    this.mi6.setEnabled(false);
```

```
/* 145:117 */ this.mi7.setEnabled(false);

/* 146:118 */ this.m1.add(this.mi1);

/* 147:119 */ this.m1.add(this.mi2);

/* 148:120 */ this.m1.add(this.mi3);

/* 149:121 */ this.m2.add(this.mi4);

/* 150:122 */ this.m2.add(this.mi5);

/* 151:123 */ this.m4.add(this.mi6);

/* 152:124 */ this.m4.add(this.mi7);

/* 153:125 */ this.mb.add(this.m1);

/* 154:126 */ this.mb.add(this.m2);

/* 155:127 */ this.mb.add(this.m4);

/* 156:128 */ this.mb.add(this.m3);

/* 157:129 */ this.f.setJMenuBar(this.mb);

/* 158:130 */ setLayout(new BorderLayout());

/* 159:131 */ add(this.jsp, "Center");

/* 160:132 */ this.tb.addMouseListener(this);

/* 161:133 */ this.mi1.addActionListener(this);

/* 162:134 */ this.mi2.addActionListener(this);

/* 163:135 */ this.mi3.addActionListener(this);

/* 164:136 */ this.mi4.addActionListener(this);

/* 165:137 */ this.mi5.addActionListener(this);

/* 166:138 */ this.mi6.addActionListener(this);

/* 167:139 */ this.mi7.addActionListener(this);

/* 168:140 */ this.f.setLayout(new BorderLayout());

/* 169:141 */ this.f.add(this.fil, "North");
```

```
/* 170:142 */    this.f.add(this, "Center");

/* 171:   */    }

/* 172:   */

/* 173:   */    public void mouseClicked(MouseEvent m)

/* 174:   */    {

/* 175:146 */    if (m.getSource() == this.tb)

/* 176:   */    {

/* 177:148 */    tree t = new tree((Packet)this.arrp.get(this.tb.getSelectedRow()));

/* 178:149 */    t.draw();

/* 179:   */    }

/* 180:   */    }

/* 181:   */

/* 182:   */    public void mousePressed(MouseEvent m) {}

/* 183:   */

/* 184:   */    public void mouseReleased(MouseEvent m) {}

/* 185:   */

/* 186:   */    public void mouseEntered(MouseEvent m) {}

/* 187:   */

/* 188:   */    public void mouseExited(MouseEvent m) {}

/* 189:   */

/* 190:   */    public void actionPerformed(ActionEvent e)

/* 191:   */    {

/* 192:162 */    if (e.getSource() == this.mi5)

/* 193:   */    {

/* 194:164 */    this.snif.getDeviceList();
```



```
/* 195:165 */    String[] dnames = this.snif.getDeviceNames();

/* 196:166 */    selector s = new selector();

/* 197:167 */    s.setup(dnames, this);

/* 198:168 */    s.setSize(300, 300);

/* 199:169 */    s.setDefaultCloseOperation(1);

/* 200:170 */    s.setVisible(true);

/* 201:   */    }

/* 202:172 */    else if (e.getSource() == this.mi7)

/* 203:   */    {

/* 204:174 */    this.n.stop();

/* 205:175 */    this.mi7.setEnabled(false);

/* 206:176 */    this.mi6.setEnabled(true);

/* 207:177 */    this.mi5.setLabel("Choose Device");

/* 208:178 */    this.mi5.setEnabled(true);

/* 209:179 */    repaint();

/* 210:   */    }

/* 211:181 */    else if (e.getSource() == this.mi6)

/* 212:   */    {

/* 213:183 */    this.mi6.setEnabled(false);

/* 214:184 */    this.mi7.setEnabled(true);

/* 215:185 */    this.mi5.setLabel("Stop Sniffing First");

/* 216:186 */    this.mi5.setEnabled(false);

/* 217:187 */    this.arp.clear();

/* 218:188 */    remove(this.tb);

/* 219:189 */    remove(this.jsp);
```

```

/* 220:190 */    this.tm = new DefaultTableModel(null, this.columns);

/* 221:191 */    this.tb = new JTable(this.tm)

/* 222:   */    {

/* 223:   */    public boolean isCellEditable(int rowIndex, int colIndex)

/* 224:   */    {

/* 225:195 */    return false;

/* 226:   */    }

/* 227:197 */    };

/* 228:198 */    this.jsp = new JScrollPane(this.tb);

/* 229:199 */    this.tb.addMouseListener(this);

/* 230:200 */    add(this.jsp);

/* 231:201 */    validate();

/* 232:202 */    repaint();

/* 233:203 */    this.n = new gui.update();

/* 234:204 */    this.n.setPriority(6);

/* 235:205 */    this.n.setName("PacketCapture");

/* 236:206 */    this.n.start();

/* 237:207 */    repaint();

/* 238:   */    }

/* 239:209 */    else if (e.getSource() == this.mi3)

/* 240:   */    {

/* 241:211 */    if (!this.arrp.isEmpty())

/* 242:   */    {

/* 243:213 */    int check = JOptionPane.showConfirmDialog(this, "Do You want to save current
Data");

```

```

/* 244:214 */    if (check == 0)

/* 245:  */      {

/* 246:216 */      JFileChooser chooser = new JFileChooser();

/* 247:217 */      int value = chooser.showSaveDialog(this);

/* 248:218 */      if (value == 0)

/* 249:  */      {

/* 250:220 */      writer w = new writer(this.cap, chooser.getSelectedFile());

/* 251:221 */      w.save(this.arp);

/* 252:222 */      System.out.println("Saved");

/* 253:  */      }

/* 254:  */      }

/* 255:225 */      else if (check == 2)

/* 256:  */      {

/* 257:227 */      return;

/* 258:  */      }

/* 259:  */      }

/* 260:230 */      if ((this.n instanceof gui.update)) {

/* 261:232 */      if (this.n.isAlive())

/* 262:  */      {

/* 263:234 */      this.n.stop();

/* 264:235 */      this.snif.stop();

/* 265:  */      }

/* 266:  */      }

/* 267:239 */      System.exit(0);

/* 268:  */      }

```

```
/* 269:241 */    else if (e.getSource() == this.mi2)

/* 270:   */    {

/* 271:243 */    if (this.arrp.isEmpty())

/* 272:   */    {

/* 273:245 */        System.out.println("No Packets Are to save");

/* 274:   */    }

/* 275:   */    else

/* 276:   */    {

/* 277:249 */        JFileChooser chooser = new JFileChooser();

/* 278:250 */        int value = chooser.showSaveDialog(this);

/* 279:251 */        if (value == 0)

/* 280:   */        {

/* 281:253 */            writer w = new writer(this.cap, chooser.getSelectedFile());

/* 282:254 */            w.save(this.arrp);

/* 283:255 */            System.out.println("Saved");

/* 284:   */        }

/* 285:   */    }

/* 286:   */    }

/* 287:   */    else

/* 288:   */    {

/* 289:   */        JFileChooser chooser;

/* 290:259 */        if (e.getSource() == this.mi1)

/* 291:   */        {

/* 292:261 */            if (!this.arrp.isEmpty())

/* 293:   */            {
```

```

/* 294:263 */      int r = JOptionPane.showConfirmDialog(this, "Do you want to save current data?");
/* 295:264 */      if (r == 0)
/* 296:  */      {
/* 297:266 */      JFileChooser chooser1 = new JFileChooser();
/* 298:267 */      int value = chooser1.showSaveDialog(this);
/* 299:268 */      if (value == 0)
/* 300:  */      {
/* 301:270 */      writer w = new writer(this.cap, chooser1.getSelectedFile());
/* 302:271 */      w.save(this.arp);
/* 303:272 */      System.out.println("Saved");
/* 304:273 */      this.arp.clear();
/* 305:  */      }
/* 306:275 */      return;
/* 307:  */      }
/* 308:277 */      if (r == 2) {
/* 309:279 */      return;
/* 310:  */      }
/* 311:  */      }
/* 312:283 */      chooser = new JFileChooser();
/* 313:284 */      int val = chooser.showOpenDialog(this);
/* 314:285 */      if (val == 0)
/* 315:  */      {
/* 316:287 */      reader re = new reader(this.tm, chooser.getSelectedFile());
/* 317:288 */      this.arp = re.read();

```

```
/* 318:289 */      System.out.println("File has been read");

/* 319:  */      }

/* 320:  */      }

/* 321:292 */      else if (e.getSource() == this.mi4)

/* 322:  */      {

/* 323:294 */

/* 324:  */      }

/* 325:  */      }

/* 326:  */      }

/* 327:  */

/* 328:  */ public void trigger(int i)

/* 329:  */ {

/* 330:  */ try

/* 331:  */ {

/* 332:302 */      this.cap = this.snif.openDevice(i);

/* 333:303 */      this.snif.setFilter(filter.filter);

/* 334:304 */      System.out.println("Device Opened" + i);

/* 335:305 */      this.mi6.setEnabled(true);

/* 336:  */      }

/* 337:  */ catch (IOException x)

/* 338:  */ {

/* 339:309 */      System.out.println("Can't Open Device\n Error:" + x);

/* 340:310 */      JDialog err = new JDialog();

/* 341:311 */      JLabel l = new JLabel("Can not Open Device\n" + x);

/* 342:312 */      err.add(l);
```

```
/* 343:313 */    err.setVisible(true);

/* 344:  */    }

/* 345:  */    }

/* 346:  */

/* 347:  */    public void apply()

/* 348:  */    {

/* 349:  */    try

/* 350:  */    {

/* 351:320 */    this.snif.setFilter(filter.filter);

/* 352:  */    }

/* 353:  */    catch (Exception x)

/* 354:  */    {

/* 355:324 */    System.out.println("Error" + x);

/* 356:  */    }

/* 357:  */    }

/* 358:  */

/* 359:  */    public void apply(String s)

/* 360:  */    throws Exception

/* 361:  */    {

/* 362:329 */    this.snif.setFilter(s);

/* 363:  */    }

/* 364:  */

/* 365:  */    public static void main(String[] s)

/* 366:  */    {

/* 367:334 */    if (s.length > 0)
```

```

/* 368:  */ {
/* 369:336 */ if (s[0].equals("-c")) {
/* 370:338 */     if (s.length > 1) {
/* 371:339 */         new com(s);
/* 372:  */     } else {
/* 373:341 */         new com();
/* 374:  */     }
/* 375:  */ }
/* 376:  */ }
/* 377:  */ else {
/* 378:346 */     SwingUtilities.invokeLater(new Runnable()
/* 379:  */     {
/* 380:  */         public void run()
/* 381:  */         {
/* 382:350 */             new gui();
/* 383:  */         }
/* 384:  */     });
/* 385:  */ }
/* 386:  */ }
/* 387:  */
/* 388:  */ private class update
/* 389:  */     extends Thread
/* 390:  */ {
/* 391:  */     Packet p;
/* 392:  */

```



```
/* 393: */ private update() {}

/* 394: */

/* 395: */ public void run()

/* 396: */ {

/* 397:360 */ int i = 0;

/* 398: */

/* 399:362 */ TableColumn tcol = gui.this.tb.getColumnModel().getColumn(0);

/* 400:363 */ tcol.setCellRenderer(new RowHighlightRenderer());

/* 401:364 */ tcol = gui.this.tb.getColumnModel().getColumn(1);

/* 402:365 */ tcol.setCellRenderer(new RowHighlightRenderer());

/* 403:366 */ tcol = gui.this.tb.getColumnModel().getColumn(2);

/* 404:367 */ tcol.setCellRenderer(new RowHighlightRenderer());

/* 405:368 */ tcol = gui.this.tb.getColumnModel().getColumn(3);

/* 406:369 */ tcol.setCellRenderer(new RowHighlightRenderer());

/* 407:370 */ tcol = gui.this.tb.getColumnModel().getColumn(4);

/* 408:371 */ tcol.setCellRenderer(new RowHighlightRenderer());

/* 409:372 */ tcol = gui.this.tb.getColumnModel().getColumn(5);

/* 410:373 */ tcol.setCellRenderer(new RowHighlightRenderer());

/* 411:374 */ JTableHeader header = gui.this.tb.getTableHeader();

/* 412:375 */ header.setBackground(Color.LIGHT_GRAY);

/* 413: */ for (;;)

/* 414: */ {

/* 415: */ try

/* 416: */ {

/* 417:380 */ this.p = gui.this.snif.getPacket();
```

```

/* 418:381 */      gui.this.arp.add(this.p);

/* 419:382 */      i++;

/* 420:383 */      pac.arrange(gui.this.tm, this.p);

/* 421:  */      }

/* 422:  */      catch (IOException x)

/* 423:  */      {

/* 424:387 */      System.out.println("Exception>>>" + x);

/* 425:  */      }

/* 426:  */      catch (Exception nx)

/* 427:  */      {

/* 428:391 */      System.out.println("Exception>>>" + nx);

/* 429:392 */      nx.printStackTrace();

/* 430:393 */      gui.this.arp.remove(i - 1);

/* 431:394 */      i--;

/* 432:  */      }

/* 433:396 */      System.out.println("No of Packets Loaded:" + i);

/* 434:  */      }

/* 435:  */      }

/* 436:  */      }

/* 437:  */      }

```

SNIFFER:

```

/* 1:  */ import java.io.IOException;

/* 2:  */ import java.io.PrintStream;

/* 3:  */ import jpcap.JpcapCaptor;

```

```

/* 4: */ import jpcap.NetworkInterface;

/* 5: */ import jpcap.packet.Packet;

/* 6: */

/* 7: */ class sniffer

/* 8: */ {

/* 9: */     private JpcapCaptor capt;

/* 10: */     private NetworkInterface[] devs;

/* 11: */     private Thread t;

/* 12: */

/* 13: */     public void getDeviceList()

/* 14: */     {

/* 15:16 */         this.devs = JpcapCaptor.getDeviceList();

/* 16: */     }

/* 17: */

/* 18: */     public NetworkInterface[] getDeviceList(boolean b)

/* 19: */     {

/* 20:20 */         this.devs = JpcapCaptor.getDeviceList();

/* 21:21 */         return this.devs;

/* 22: */     }

/* 23: */

/* 24: */     public String[] getDeviceNames()

/* 25: */     {

/* 26:25 */         String[] dname = new String[this.devs.length];

/* 27:26 */         for (int i = 0; i < this.devs.length; i++) {

/* 28:27 */             dname[i] = this.devs[i].description;

/* 29: */         }

/* 30:28 */         return dname;

```

```

/* 31: */ }

/* 32: */

/* 33: */ public JpcapCaptor openDevice(int i)

/* 34: */ throws IOException

/* 35: */ {

/* 36:33 */ System.out.println("You Choose device No.>" + i);

/* 37:34 */ this.capt = JpcapCaptor.openDevice(this.devs[i], 65535, true, 1000);

/* 38:35 */ return this.capt;

/* 39: */ }

/* 40: */

/* 41: */ public void setFilter(String filt)

/* 42: */ throws IOException

/* 43: */ {

/* 44:39 */ this.capt.setFilter(filt, true);

/* 45:40 */ System.out.println("Filter is set to>>" + filt);

/* 46: */ }

/* 47: */

/* 48: */ public Packet getPacket()

/* 49: */ {

/* 50:45 */ Packet pac = this.capt.getPacket();

/* 51:46 */ return pac;

/* 52: */ }

/* 53: */

/* 54: */ public void stop()

/* 55: */ {

/* 56:50 */ this.capt.close();

/* 57: */ }

```

5. Results & Experiments

Compared to similar works this application shows the layer involved in sniffing and the protocols. This Passive Sniffer would be installed on a collision domain that makes use of the switch rather than the broadcast domain (HUB). The collision domain would be used since the use of HUBS in network setting is gradually reducing due to its broadcasting nature. Psniffer has a very rich and user friendly GUI developed in Java Swing Technology. Thus it is totally easy to use. With Java, the most considerable advantage is platform independence; therefore Psniffer is also platform independent. The installation file for Psniffer is only 587 KB, so it is highly economical in terms of memory use and because it is based on object-oriented design, any further changes can be easily adaptable.

6. Conclusion and Future Work

Conclusion

The mini project on packet sniffers has successfully developed an enhanced tool for network monitoring, analysis, and security. Through rigorous experimentation and testing, the packet sniffer has demonstrated its effectiveness in capturing packets accurately, analyzing network traffic in real-time, handling encrypted traffic, scaling to large network environments, and providing a user-friendly interface. These achievements represent significant progress in addressing the identified research gaps and advancing the capabilities of packet sniffers.

Future Work

It is not possible to develop a system that meets all the requirements of the user. User requirements keep changing as the system is being used. Some of the future enhancements that can be done to this system are:

- As the technology emerges, it is possible to upgrade the system that can be adaptable to desired environment.
- The present application is a standalone application, i.e. only in intranet. So we have chance to extend this in internet.
- Based on the future security issues, security can be improved using emerging technologies.

References

1. Ryan Spangler “Packet Sniffing on Layer 2 Switched Local Area Networks”University of Wisconsin – Whitewater Department of Computer and Network Administration(December 2003).
2. Suhas A Desai “ Packet Sniffing: Sniffing Tools Detection Prevention Methods” University of California Department of Network Administration.(April 2004).
3. Ryan Spangler University of Wisconsin –“Packet Sniffer Detection with Anti Sniff”.
4. A. Ornaghi, M. Valleri, “Man in the middle attacks Demos” Blackhat [Online Document], 2003.

