Project Report

On

# Network Protocol Analyzer

Submitted in partial fulfilment of the requirements for the award of

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE & ENGINEERING**

(Artificial Intelligence & Machine Learning) by

**Ms. K SNEHA ( 22WH1A6604)**

**Ms. P PREETHI (22WH1A6633)**

**Ms. B SOWMYA(22WH1A6646)**

**Ms. B HARSHINI( 22WH1A6652)**

**Under the esteemed guidance of**

**Ms. P Anusha**

**Assistant Professor, CSE(AI&ML)**



**Department of Computer Science & Engineering**

**(Artificial Intelligence &Machine Learning)**

**BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN**

**(AUTONOMOUS)**

**(Approved by AICTE New Delhi and Affiliated to JNTUH, Hyderabad)**

**Accredited by NBA and NAAC with A Grade**

**Bachupally, Hyderabad – 500090**

2024-2025

# ABSTRACT

This project focuses on the development of a **Network Protocol Analyzer** using the **C programming language**, designed to monitor, capture, and analyze network traffic in real time. A network protocol analyzer is an essential tool for diagnosing and troubleshooting network issues, enhancing security, and optimizing network performance. Leveraging low-level system programming capabilities in C, the project aims to deliver an efficient and robust solution for protocol inspection and traffic analysis.

The analyzer captures live network packets, extracting crucial details such as source and destination IP addresses, protocol types, port numbers, and payload information. It categorizes packets based on protocols (e.g., TCP, UDP, ICMP) and provides detailed traffic summaries, including packet counts and bandwidth usage. Advanced filtering and parsing mechanisms enable users to inspect specific traffic patterns, facilitating the identification of suspicious activities such as unusual port usage or potential cybersecurity threats.

The tool is designed to aid network administrators, security analysts, and researchers by offering insights into traffic flow, detecting anomalies, and optimizing network performance. With its real-time analysis capabilities and resource-efficient implementation, this protocol analyzer serves as a powerful asset for maintaining secure and reliable network operations.

# PROBLEM STATEMENT

In today’s interconnected digital environment, monitoring and analyzing network protocols is vital for ensuring seamless communication, robust security, and effective troubleshooting. However, many existing network analysis tools are resource-intensive, lack flexibility, and often fail to provide efficient handling of protocol-specific traffic in real time. These limitations hinder the ability of administrators and security analysts to gain actionable insights into network behavior.

The core challenge lies in developing a lightweight, high-performance **Network Protocol Analyzer** that can effectively inspect and decode protocol-specific network traffic in real time. This tool must provide detailed information on communication protocols (e.g., TCP, UDP, ICMP) and offer the ability to identify irregularities, optimize performance, and enhance network security. Additionally, it must overcome the inherent complexity of parsing protocol headers, managing raw network data, and adhering to resource constraints in low-level programming.

This project aims to address these challenges by designing and implementing a **Network Protocol Analyzer** using the **C programming language**. By leveraging C's low-level access to system resources, the solution will achieve high performance, precise protocol analysis, and efficient resource utilization. The tool will support real-time protocol classification, anomaly detection, and traffic visualization, empowering users to optimize and secure network infrastructure effectively.

**FUNCTIONAL REQUIREMENTS:**

**Packet Capture:**

The system should capture network packets in real-time from the specified network interface.

**Packet Parsing and Decoding:**

The system should decode packets to extract information such as:

* 1. Source and destination IP addresses.
  2. Source and destination ports.
  3. Protocol type (e.g., TCP, UDP, ICMP).
  4. Packet payload and headers.

**Protocol Filtering:**

Users should be able to filter packets based on specific protocols (e.g., only TCP or UDP packets).

**Traffic Analysis:**

Provide statistics on network traffic, such as the number of packets per protocol or packet size distribution.

**NON FUNCTIONAL REQUIREMENTS:**

**Performance:**

The analyzer should capture and process packets with minimal latency, ensuring real-time analysis.

It should handle high traffic volumes without dropping packets.

**Resource Efficiency:**

The system should be lightweight, utilizing minimal CPU and memory resources.

**Scalability:**

Capable of functioning efficiently in diverse network environments, from small LANs to large-scale networks.

**Security:** Ensure secure handling of network data, preventing unauthorized access or misuse of captured packets.

# 

# 

**SOURCE CODE**

#include <pcap.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <netinet/udp.h>

#include <netinet/if\_ether.h>

/\* Callback function called for each captured packet \*/

void packet\_handler(u\_char \*user\_data, const struct pcap\_pkthdr \*pkthdr, const u\_char \*packet) {

struct ether\_header \*eth\_header;

eth\_header = (struct ether\_header \*)packet;

printf("\n=== Packet Captured ===\n");

printf("Packet Length: %d bytes\n", pkthdr->len);

// Check if the packet is IP

if (ntohs(eth\_header->ether\_type) == ETHERTYPE\_IP) {

printf("Protocol: IP\n");

// Get IP header

const struct ip \*ip\_header = (struct ip \*)(packet + sizeof(struct ether\_header));

printf("Source IP: %s\n", inet\_ntoa(ip\_header->ip\_src));

printf("Destination IP: %s\n", inet\_ntoa(ip\_header->ip\_dst));

// Check transport layer protocol

if (ip\_header->ip\_p == IPPROTO\_TCP) {

printf("Transport Protocol: TCP\n");

const struct tcphdr \*tcp\_header = (struct tcphdr \*)(packet + sizeof(struct ether\_header) + ip\_header->ip\_hl \* 4);

printf("Source Port: %d\n", ntohs(tcp\_header->source));

printf("Destination Port: %d\n", ntohs(tcp\_header->dest));

} else if (ip\_header->ip\_p == IPPROTO\_UDP) {

printf("Transport Protocol: UDP\n");

const struct udphdr \*udp\_header = (struct udphdr \*)(packet + sizeof(struct ether\_header) + ip\_header->ip\_hl \* 4);

printf("Source Port: %d\n", ntohs(udp\_header->source));

printf("Destination Port: %d\n", ntohs(udp\_header->dest));

} else {

printf("Transport Protocol: Other\n");

}

} else {

printf("Protocol: Non-IP\n");

}

}

int main() {

char errbuf[PCAP\_ERRBUF\_SIZE];

pcap\_t \*handle;

// List available devices

pcap\_if\_t \*all\_devices, \*device;

if (pcap\_findalldevs(&all\_devices, errbuf) == -1) {

fprintf(stderr, "Error finding devices: %s\n", errbuf);

return 1;

}

printf("Available devices:\n");

for (device = all\_devices; device; device = device->next) {

printf(" %s - %s\n", device->name, device->description ? device->description : "No description");

}

// Open device for capturing

char \*device\_name = all\_devices->name; // Use the first device found

printf("\nCapturing on device: %s\n", device\_name);

handle = pcap\_open\_live(device\_name, BUFSIZ, 1, 1000, errbuf);

if (handle == NULL) {

fprintf(stderr, "Could not open device %s: %s\n", device\_name, errbuf);

return 1;

}

printf("Starting packet capture...\n");

pcap\_loop(handle, 10, packet\_handler, NULL);

// Cleanup

pcap\_close(handle);

pcap\_freealldevs(all\_devices);

printf("Capture complete.\n");

return 0;

}

# 

# 

**OUTPUT:**

**Available devices:**

eth0 - Ethernet Interface

wlan0 - Wireless Network Adapter

lo - Loopback Interface

**Capturing on device:** eth0

**Starting packet capture...**

**=== Packet Captured ===**

Packet Length: 74 bytes

Protocol: IP

Source IP: 192.168.1.100

Destination IP: 172.217.10.4

Transport Protocol: TCP

Source Port: 443

Destination Port: 53000

**=== Packet Captured ===**

Packet Length: 90 bytes

Protocol: IP

Source IP: 192.168.1.100

Destination IP: 8.8.8.8

Transport Protocol: UDP

Source Port: 5353

Destination Port: 53

**=== Packet Captured ===**

Packet Length: 66 bytes

Protocol: IP

Source IP: 172.217.10.4

Destination IP: 192.168.1.100

Transport Protocol: TCP

Source Port: 443

Destination Port: 53000

**=== Packet Captured ===**

Packet Length: 64 bytes

Protocol: IP

Source IP: 8.8.8.8

Destination IP: 192.168.1.100

Transport Protocol: UDP

Source Port: 53

Destination Port: 5353

**=== Packet Captured ===**

Packet Length: 78 bytes

Protocol: Non-IP





