MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report
of
Computer Networks LAB

TITLE

TCP-IP Packet Sniffer

SUBMITTED
BY

STUDENT NAME	REG. NO, ROLL NO, SECTION
Shyam Sundar Bharathi S	200905302, 53, B
Ajay Rajendra Kumar	200905390, 61, B

ABSTRACT:

A Packet Sniffer is a program that can record all network packets that travel past a given network interface, on a given computer, on a network. It can be used to troubleshoot network problems, as well as to extract sensitive information such as Credentials from unencrypted Login Session's.

Unlike telephone circuits, computer networks are shared communication channels. It is simply too expensive to dedicate local loops to the switch (hub) for each pair of communicating computers. Sharing means that computers can receive information that was intended for other machines. To capture the information going over the network is called sniffing.

Most popular way of connecting computers is through Ethernet. Ethernet protocol works by sending packet information to all the hosts on the same circuit. The packet header contains the proper address of the destination machine. Only the machine with the matching address is supposed to accept the packet. A machine that is accepting all packets, no matter what the packet header says, is said to be in promiscuous mode.

Sniffers can be used both for legitimate network management functions and for stealing information off a network. Unauthorized sniffers can be extremely dangerous to a network's security because they are virtually impossible to detect and can be inserted almost anywhere. This makes them a favourite weapon in the hacker's arsenal.

1. INTRODUCTION

1.1 General Introduction to the topic

A packet sniffer — also known as a packet analyzer, protocol analyzer or network analyzer — is a piece of hardware or software used to monitor network traffic. Sniffers work by examining streams of data packets that flow between computers on a network as well as between networked computers and the larger Internet. These packets are intended for — and addressed to — specific machines but using a packet sniffer in "promiscuous mode" allows IT professionals, end users or malicious intruders to examine any packet, regardless of destination. It's possible to configure sniffers in two ways. The first is "unfiltered," meaning they will capture all packets possible and write them to a local hard drive for later examination. Next is "filtered" mode, meaning analyzers will only capture packets that contain specific data elements.

1.2 Hardware and Software Requirements

To project has been made using the C programming language.
The software requirements include:

- GCC Compiler
- A text editor
- Linux Operating System

The Hardware requirements include:

- A computer / laptop running Linux OS
  A good internet connection

2. Problem definition

The aim of this application is to read packets that travel across various layers of the Transmission Control Protocol/Internet Protocol (TCP/IP) model of network architecture. The packet sniffer will analyze the network traffic to allow users to get a practical understanding of the flow of packets in a network.

It will be used to capture and analyze the following protocols' header information from the packets:

Application Layer: HTTP, DNS Transport Layer: TCP, UDP Network Layer: IPv4, IPv6 Data Link Layer: ARP

The packets captured will be analyzed to extract and display the header information along with other relevant parameters for the selected protocols.

3. Objectives

The project objectives are as follows:

- To gain an understanding of what is packet sniffing.
- To understand how packet sniffing works.
- What are the different types of packet sniffing.
- How to perform packet sniffing.
- Analyzing the results obtained by performing packet sniffing.
- How sniffing attacks can illegally access and read unencrypted data.

4. Methodology

The main function opens a socket and listens for packets in the process. Information about the packets is passed on to a function that processes the ethernet protocol. It strips the header, and based on data fields, passes it on to appropriate higher-level protocols (IPv4, IPv6, ARP, etc.), from where similar process is followed and further higher-level protocols are called to be processed (TCP, UDP). Functions to parse DNS messages and HTTP headers are also written in the source code. All information retrieved at each layer from all the protocols are neatly printed into the log file that is generated as "log.txt"

5. Implementation Details

```
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/ip_icmp.h>  //Declarations for icmp header
#include <netinet/udp.h>      //Declarations for udp header
#include <netinet/tcp.h>      //Declarations for tcp header
#include <netinet/ip.h>       //Declarations for ip header
#include <netinet/ip6.h>      //Declarations for ip header
#include <netinet/if_ether.h> //ETH_P_ALL
#include <net/ethernet.h>     //Ether_header
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <time.h>
#include <signal.h>
#define zero(s) memset(s, 0, sizeof(s))
int arp = 0, ipv4 = 0, ipv6 = 0, tcp = 0, udp = 0, http = 0, dns = 0, ftp = 0, smtp = 0, num =
0;

void handleINT(int sig)
{
   char c;
   signal(sig, SIG_IGN);
   clock_t CPU_time_2 = clock();
   printf("CPU end time is : %d\n", CPU_time_2);
   printf("TCP: %d\n", tcp);
   printf("UDP: %d\n", udp);
   printf("ipv4: %d\n", ipv4);
   printf("ipv6: %d\n", ipv6);
   printf("ARP: %d\n", arp);
   printf("HTTP: %d\n", http);
   printf("DNS: %d\n", dns);
```

```
    printf("FTP: %d\n", ftp);
    printf("SMTP: %d\n", smtp);
    printf("Total: %d\n", num);
    exit(0);
}

struct headerARPPkt
{
    u_int16_t htype;
    u_int16_t ptype;
    u_int8_t hlen;
    u_int8_t plen;
    u_int16_t oper;
    u_int8_t sha[6];
    u_int8_t spa[4];
    u_int8_t tha[6];
    u_int8_t tpa[4];
};

struct headerDNSPkt
{
    unsigned short id;        // identification number
    unsigned char rd : 1;     // recursion desired
    unsigned char tc : 1;     // truncated message
    unsigned char aa : 1;     // authoritive answer
    unsigned char opcode : 4; // purpose of message
    unsigned char qr : 1;     // query/response flag
    unsigned char rcode : 4;  // response code
    unsigned char cd : 1;     // checking disabled
    unsigned char ad : 1;     // authenticated data
    unsigned char z : 1;      // its z! reserved
    unsigned char ra : 1;     // recursion available
    unsigned short q_count;   // number of question entries
    unsigned short ans_count; // number of answer entries
    unsigned short auth_count; // number of authority entries
    unsigned short add_count; // number of resource entries
};

struct headerIPv6Pkt
{
```

```
#if __BYTE_ORDER == __LITTLE_ENDIAN
   u_int8_t traffic_class_1 : 4, ip_version : 4;
   u_int8_t flow_label_1 : 4, traffic_class_2 : 4;
#elif __BYTE_ORDER == __BIG_ENDIAN
   u_int8_t ip_version : 4, traffic_class_1 : 4;
   u_int8_t traffic_class_2 : 4, flow_label : 4;
#else
#error "Please fix <bits/endian.h>"
#endif
   u_int16_t flow_label_2;
   u_int16_t payload_length;
   u_int8_t next_header;
   u_int8_t hop_limit;
   u_char src_ipv6[16];
   u_char dst_ipv6[16];
};

FILE *logfile;
struct sockaddr_in source, dest;
int total = 0;
char str[50];

void dataDisplay(unsigned char *data, int Size)
{
   int i, j, num2 = 0;
   for (i = 0; i < Size; i++)
   {
      if (i != 0 && i % 16 == 0)
      {
         for (j = i - 16; j < i; j++)
         {
            if (data[j] >= 32 && data[j] <= 128)
               fprintf(logfile, "%c", (unsigned char)data[j]);
            else
               fprintf(logfile, ".");
         }
      }

      if (i == Size - 1)
      {
```

```
        for (j = i - i % 16; j <= i; j++)
        {
          if (data[j] >= 32 && data[j] <= 128)
            fprintf(logfile, "%c", (unsigned char)data[j]);
          else
            fprintf(logfile, ".");
        }
      }
    }
}

void ethernetDisplay(struct ether_header *eth)
{
   fprintf(logfile, "\n");
   fprintf(logfile, "Ethernet Header:\n");
   fprintf(logfile, "   -->Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>ether_dhost[0], eth->ether_dhost[1], eth->ether_dhost[2], eth->ether_dhost[3], eth-
>ether_dhost[4], eth->ether_dhost[5]);
   fprintf(logfile, "   |-Source Address      : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>ether_shost[0], eth->ether_shost[1], eth->ether_shost[2], eth->ether_shost[3], eth-
>ether_shost[4], eth->ether_shost[5]);
   fprintf(logfile, "   |-Protocol           : %u \n", (unsigned short)eth->ether_type);
}

void tcpHeaderDisplay(struct tcphdr *tcph)
{
   fprintf(logfile, "\n");
   fprintf(logfile, "TCP Header:\n");
   fprintf(logfile, "   |-Source Port      : %u\n", ntohs(tcph->source));
   fprintf(logfile, "   |-Destination Port : %u\n", ntohs(tcph->dest));
   fprintf(logfile, "   |-Sequence Number    : %u\n", ntohl(tcph->seq));
   fprintf(logfile, "   |-Acknowledge Number : %u\n", ntohl(tcph->ack_seq));
   fprintf(logfile, "   |-Header Length      : %d DWORDS or %d BYTES\n", (unsigned
int)tcph->doff, (unsigned int)tcph->doff * 4);
   fprintf(logfile, "   |-Urgent Flag          : %d\n", (unsigned int)tcph->urg);
   fprintf(logfile, "   |-Acknowledgement Flag : %d\n", (unsigned int)tcph->ack);
   fprintf(logfile, "   |-Push Flag            : %d\n", (unsigned int)tcph->psh);
   fprintf(logfile, "   |-Reset Flag           : %d\n", (unsigned int)tcph->rst);
   fprintf(logfile, "   |-Synchronise Flag     : %d\n", (unsigned int)tcph->syn);
   fprintf(logfile, "   |-Finish Flag          : %d\n", (unsigned int)tcph->fin);
```

```
   fprintf(logfile, "   |-Window        : %d\n", ntohs(tcph->window));
   fprintf(logfile, "   |-Checksum      : %d\n", ntohs(tcph->check));
   fprintf(logfile, "   |-Urgent Pointer : %d\n", tcph->urg_ptr);
   fprintf(logfile, "\n");
   return;
}

void ipv4HeaderDisplay(struct iphdr *iph)
{
   unsigned short iphdrlen;
   fprintf(logfile, "\n");
   fprintf(logfile, "IP Header:\n");
   fprintf(logfile, "   |-IP Version       : %d\n", (unsigned int)iph->version);
   fprintf(logfile, "   |-Type Of Service   : %d\n", (unsigned int)iph->tos);
   fprintf(logfile, "   |-IP Total Length   : %d  Bytes(Size of Packet)\n", ntohs(iph-
>tot_len));
   fprintf(logfile, "   |-Identification    : %d\n", ntohs(iph->id));
   fprintf(logfile, "   |-TTL      : %d\n", (unsigned int)iph->ttl);
   fprintf(logfile, "   |-Protocol : %d\n", (unsigned int)iph->protocol);
   fprintf(logfile, "   |-Checksum : %d\n", ntohs(iph->check));
}

void ipv6HeaderDisplay(struct headerIPv6Pkt *hdr)
{
   char src[50], dst[50], stemp[5], dtemp[5];
   int i;
   zero(src);
   zero(dst);
   for (i = 1; i <= 16; i++)
   {
      if (i % 2 == 0 && i < 16)
      {
         sprintf(stemp, "%02x:", hdr->src_ipv6[i - 1]);
         sprintf(dtemp, "%02x:", hdr->dst_ipv6[i - 1]);
      }
      else
      {
         sprintf(stemp, "%02x", hdr->src_ipv6[i - 1]);
         sprintf(dtemp, "%02x", hdr->dst_ipv6[i - 1]);
      }
```

```
        strcat(src, stemp);
        strcat(dst, dtemp);
    }
    fprintf(logfile, "\nIPV6 Header\n");
    fprintf(logfile, "   |-Source      : %s\n", src);
    fprintf(logfile, "   |-Destination  : %s\n", dst);
    fprintf(logfile, "\n");
}

void udpHeaderDisplay(struct udphdr *udph)
{
    fprintf(logfile, "\nUDP Header\n");
    fprintf(logfile, "   |-Source Port      : %d\n", ntohs(udph->source));
    fprintf(logfile, "   |-Destination Port : %d\n", ntohs(udph->dest));
    fprintf(logfile, "   |-UDP Length       : %d\n", ntohs(udph->len));
    fprintf(logfile, "   |-UDP Checksum     : %d\n", ntohs(udph->check));
    fprintf(logfile, "\n");
}

void icmpHeaderDisplay(struct icmphdr *icmph)
{
    fprintf(logfile, "ICMP Header:\n");
    fprintf(logfile, "   |-Type : %d", (unsigned int)(icmph->type));
    if ((unsigned int)(icmph->type) == 11)
        fprintf(logfile, "  (TTL Expired)\n");
    else if ((unsigned int)(icmph->type) == ICMP_ECHOREPLY)
        fprintf(logfile, "  (ICMP Echo Reply)\n");
    fprintf(logfile, "   |-Code : %d\n", (unsigned int)(icmph->code));
    fprintf(logfile, "   |-Checksum : %d\n", ntohs(icmph->checksum));
    fprintf(logfile, "\n");
}

void httpDisplay(unsigned char *data, int size, int hdrsize)
{
    fprintf(logfile, "\n");
    fprintf(logfile, "HTTP message\n");
    dataDisplay(data + hdrsize, size - hdrsize);
}

void dnsHeaderDisplay(struct headerDNSPkt *dnsh)
```

```
{
   fprintf(logfile, "\n");
   fprintf(logfile, "DNS Header:\n");
   fprintf(logfile, "   |-Identification Number      : %u\n", dnsh->id);
   fprintf(logfile, "   |-Recursion Desired         : %u\n", dnsh->rd);
   fprintf(logfile, "   |-Truncated Message         : %u\n", (dnsh->tc));
   fprintf(logfile, "   |-Authoritative Answer      : %u\n", (dnsh->aa));
   fprintf(logfile, "   |-Purpose of message        : %d\n", (unsigned int)dnsh->opcode);
   fprintf(logfile, "   |-Query/Response Flag       : %d\n", (unsigned int)dnsh->qr);
   fprintf(logfile, "   |-Response code             : %d\n", (unsigned int)dnsh->rcode);
   fprintf(logfile, "   |-Checking Disabled         : %d\n", (unsigned int)dnsh->cd);
   fprintf(logfile, "   |-Authenticated data        : %d\n", (unsigned int)dnsh->ad);
   fprintf(logfile, "   |-Recursion available       : %d\n", (unsigned int)dnsh->ra);
   fprintf(logfile, "   |-Number of question entries : %d\n", (dnsh->q_count));
   fprintf(logfile, "   |-Number of answer entries   : %d\n", (dnsh->ans_count));
   fprintf(logfile, "   |-Number of authority entries: %d\n", dnsh->auth_count);
   fprintf(logfile, "   |-Number of resource entries : %d\n", dnsh->add_count);
   fprintf(logfile, "\n");
   return;
}

void arpDisplay(struct headerARPPkt *hdr)
{
   fprintf(logfile, "\n");
   fprintf(logfile, "ARP Header\n");
   fprintf(logfile, "   |-Hardware type      : %d\n", ntohs(hdr->htype));
   fprintf(logfile, "   |-Protocol Type : %d\n", ntohs(hdr->ptype));
   fprintf(logfile, "   |-Hardware addr len    : %d\n", ntohs(hdr->hlen));
   fprintf(logfile, "   |-Protocol addr len   : %d\n", ntohs(hdr->plen));
   fprintf(logfile, "   |-Operation      : %d\n", ntohs(hdr->plen));
   fprintf(logfile, "\n");
}

void arpProcess(unsigned char *buffer, int size)
{
   arp++;
   struct headerARPPkt *hdr = (struct headerARPPkt *)(buffer + sizeof(struct
ether_header));
   fprintf(logfile, "Packet type: ARP\n");
   arpDisplay(hdr);
```

```
}

void tcpProcess(unsigned char *buffer, int size, int type)
{
   tcp++;
   struct tcphdr *header;
   if (type)
      header = (struct tcphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct
iphdr));
   else
      header = (struct tcphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct
ip6_hdr));
   struct ether_header *eth = (struct ether_header *)(buffer);
   u_short src_port;
   u_short dst_port;
   u_int seq;
   u_int ack;
   src_port = ntohs(header->source);
   dst_port = ntohs(header->dest);
   seq = ntohl(header->seq);
   ack = ntohl(header->ack);
   int hdrsize;
   if (src_port == 80 || dst_port == 80)
   {
      // HTTP Packet
      http++;
      fprintf(logfile, "Packet type: HTTP\n");
      struct httphdr *shdr;
      if (type)
      {
         shdr = (struct httphdr *)(buffer + sizeof(struct tcphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
         hdrsize = sizeof(struct tcphdr) + sizeof(struct ether_header) + sizeof(struct iphdr);
         fprintf(logfile, "\n");
         fprintf(logfile, "HTTP message\n");
         dataDisplay(buffer + hdrsize, size - hdrsize);
         fprintf(logfile, "\n");
      }
      else
      {
```

```
            shdr = (struct httphdr *)(buffer + sizeof(struct tcphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
            hdrsize = sizeof(struct tcphdr) + sizeof(struct ether_header) + sizeof(struct
ip6_hdr);
            fprintf(logfile, "\n");
            fprintf(logfile, "HTTP message\n");
            dataDisplay(buffer + hdrsize, size - hdrsize);
            fprintf(logfile, "\n");
        }
    }
    else if (src_port == 25 || dst_port == 25)
    {
        // SMTP Packet
        smtp++;
        fprintf(logfile, "Packet type: SMTP\n");
        struct smtphdr *shdr;
        if (type)
            shdr = (struct smtphdr *)(buffer + sizeof(struct tcphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
        else
            shdr = (struct smtphdr *)(buffer + sizeof(struct tcphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
    }
    else if (src_port == 53 || dst_port == 53)
    {
        // DNS Packet
        dns++;
        fprintf(logfile, "Packet type: DNS\n");
        struct headerDNSPkt *shdr;
        if (type)
        {
            shdr = (struct headerDNSPkt *)(buffer + sizeof(struct tcphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
            hdrsize = sizeof(struct udphdr) + sizeof(struct ether_header) + sizeof(struct
iphdr);
            fprintf(logfile, "\n");
            fprintf(logfile, "DNS message\n");
            dnsHeaderDisplay(shdr);
            fprintf(logfile, "\n");
        }
```

```
      else
      {
         shdr = (struct headerDNSPkt *)(buffer + sizeof(struct tcphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
         hdrsize = sizeof(struct udphdr) + sizeof(struct ether_header) + sizeof(struct
ip6_hdr);
         fprintf(logfile, "\n");
         fprintf(logfile, "DNS message\n");
         dnsHeaderDisplay(shdr);
         fprintf(logfile, "\n");
      }
   }

   else if (src_port == 20 || dst_port == 20 || src_port == 21 || dst_port == 21)
   {
      // FTP Packet
      ftp++;
      fprintf(logfile, "Packet type: FTP\n");
      struct ftphdr *shdr;
      if (type)
         shdr = (struct ftphdr *)(buffer + sizeof(struct tcphdr) + sizeof(struct ether_header)
+ sizeof(struct iphdr));
      else
         shdr = (struct ftphdr *)(buffer + sizeof(struct tcphdr) + sizeof(struct ether_header)
+ sizeof(struct ip6_hdr));
   }

   else
      fprintf(logfile, "Packet type: TCP\n");

   tcpHeaderDisplay(header);
   if (type)
   {
      struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
      ipv4HeaderDisplay(iph);
   }
   else
   {
      struct headerIPv6Pkt *iph = (struct headerIPv6Pkt *)(buffer + sizeof(struct
ether_header));
```

```
        ipv6HeaderDisplay(iph);
    }
    ethernetDisplay(eth);
}

void udpProcess(unsigned char *buffer, int size, int type)
{
    udp++;
    struct udphdr *hdr;
    if (type)
        hdr = (struct udphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr));
    else
        hdr = (struct udphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct
ip6_hdr));
    struct ether_header *eth = (struct ether_header *)(buffer);
    u_short src_port;
    u_short dst_port;
    src_port = ntohs(hdr->source);
    dst_port = ntohs(hdr->dest);
    int hdrsize;
    if (src_port == 80 || dst_port == 80)
    {
        // HTTP Packet
        fprintf(logfile, "Packet type: HTTP\n");
        http++;
        struct httphdr *shdr;
        if (type)
        {
            shdr = (struct httphdr *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
            hdrsize = sizeof(struct udphdr) + sizeof(struct ether_header) + sizeof(struct
iphdr);
            fprintf(logfile, "\n");
            fprintf(logfile, "HTTP message\n");
            dataDisplay(buffer + hdrsize, size - hdrsize);
            fprintf(logfile, "\n");
        }
        else
        {
```

```
            shdr = (struct httphdr *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
            hdrsize = sizeof(struct udphdr) + sizeof(struct ether_header) + sizeof(struct
ip6_hdr);
            fprintf(logfile, "\n");
            fprintf(logfile, "HTTP message\n");
            dataDisplay(buffer + hdrsize, size - hdrsize);
            fprintf(logfile, "\n");
        }
    }
    else if (src_port == 25 || dst_port == 25)
    {
        // SMTP Packet
        smtp++;
        fprintf(logfile, "Packet type: SMTP\n");
        struct smtphdr *shdr;
        if (type)
            shdr = (struct smtphdr *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
        else
            shdr = (struct smtphdr *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
    }
    else if (src_port == 53 || dst_port == 53)
    {
        // DNS Packet
        dns++;
        fprintf(logfile, "Packet type: DNS\n");
        struct headerDNSPkt *shdr;
        if (type)
        {
            shdr = (struct headerDNSPkt *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
            hdrsize = sizeof(struct udphdr) + sizeof(struct ether_header) + sizeof(struct
iphdr);
            fprintf(logfile, "\n");
            fprintf(logfile, "DNS message\n");
            dnsHeaderDisplay(shdr);
            fprintf(logfile, "\n");
        }
```

```
      else
      {
         shdr = (struct headerDNSPkt *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
         hdrsize = sizeof(struct udphdr) + sizeof(struct ether_header) + sizeof(struct
ip6_hdr);
         fprintf(logfile, "\n");
         fprintf(logfile, "DNS message\n");
         dnsHeaderDisplay(shdr);
         fprintf(logfile, "\n");
      }
   }
   else if (src_port == 20 || dst_port == 20 || src_port == 21 || dst_port == 21)
   {
      // FTP Packet
      ftp++;
      struct ftphdr *shdr;
      if (type)
         shdr = (struct ftphdr *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct iphdr));
      else
         shdr = (struct ftphdr *)(buffer + sizeof(struct udphdr) + sizeof(struct
ether_header) + sizeof(struct ip6_hdr));
      fprintf(logfile, "Packet type: FTP\n");
   }

   else
      fprintf(logfile, "Packet type: UDP\n");

   udpHeaderDisplay(hdr);
   if (type)
   {
      struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
      ipv4HeaderDisplay(iph);
   }
   else
   {
      struct headerIPv6Pkt *iph = (struct headerIPv6Pkt *)(buffer + sizeof(struct
ether_header));
      ipv6HeaderDisplay(iph);
```

```
    }
    ethernetDisplay(eth);
}

void ipv4Process(unsigned char *buffer, int size)
{
    ipv4++;
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
    struct ether_header *eth = (struct ether_header *)(buffer);
    switch (iph->protocol)
    {
    case 6:
        tcpProcess(buffer, size, 1);
        break;

    case 17:
        udpProcess(buffer, size, 1);
        break;

    case 1:
        break;

    default:
        fprintf(logfile, "Packet type: IPv4\n");
        ipv4HeaderDisplay(iph);
        ethernetDisplay(eth);
        break;
    }
}

void ipv6Process(unsigned char *buffer, int size)
{
    ipv6++;
    struct headerIPv6Pkt *ipv6 = (struct headerIPv6Pkt *)(buffer + sizeof(struct
ether_header));
    struct ether_header *eth = (struct ether_header *)(buffer);
    switch (ntohs(ipv6->next_header))
    {
    case 6:
        tcpProcess(buffer, size, 0);
```

```
        break;

    case 17:
        udpProcess(buffer, size, 0);
        break;

    default:
        fprintf(logfile, "Packet type: IPv6\n");
        ipv6HeaderDisplay(ipv6);
        ethernetDisplay(eth);
        break;
    }
}

void ethernetProcess(unsigned char *buffer, int size)
{
    struct ether_header *eth = (struct ether_header *)(buffer);
    ++total;
    switch (ntohs(eth->ether_type))
    {
    case 0x0800: // Ethernet Protocol
        ipv4Process(buffer, size);
        break;

    case 0x0806: // ARP Protocol
        ethernetDisplay(eth);
        arpProcess(buffer, size);
        break;

    case 0x86dd: // IPv6 Protocol
        ipv6Process(buffer, size);
        break;

    default:
        fprintf(logfile, "Packet type: Ethernet\n");
        ethernetDisplay(eth);
        break;
    }
}
int main()
```

```
{
   printf("\nCN Mini Project by Ajay Rajendra Kumar and Shyam Sundar Bharathi.\n\n");
   int saddr_size, data_size;
   struct sockaddr saddr;
   unsigned char *buffer = (unsigned char *)malloc(65536);
   logfile = fopen("log.txt", "w+");
   if (!logfile)
      printf("Unable to create log.txt file.");
   int sock_raw = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
   if (sock_raw < 0)
   {
      perror("Socket Error");
      return 1;
   }
   clock_t CPU_time_1 = clock();
   printf("CPU start time: %d \n", CPU_time_1);
   signal(SIGINT, handleINT);
   while (1)
   {
      saddr_size = sizeof saddr;
      data_size = recvfrom(sock_raw, buffer, 65536, 0, &saddr, (socklen_t
*)&saddr_size);
      if (data_size < 0)
      {
         printf("Failed to get packets.\n");
         return 1;
      }
      num++;
      fprintf(logfile, "\nPacket number %d\n", num);
      ethernetProcess(buffer, data_size);
   }
   close(sock_raw);
   return 0;
}
```

Output:

Packet number 1
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 54604
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 0
|-Response code : 0
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 0
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 39396
|-Destination Port : 53
|-UDP Length : 43
|-UDP Checksum : 15808

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 47443
|-TTL : 64
|-Protocol : 17
|-Checksum : 44182

Ethernet Header:
-->Destination Address : 30-E3-7A-10-DD-D5
|-Source Address : 48-BF-6B-DF-21-D8
|-Protocol : 8

Packet number 2
Packet type: DNS

DNS message

DNS Header:
  |-Identification Number   : 6650
  |-Recursion Desired   : 1
  |-Truncated Message   : 0
  |-Authoritative Answer   : 0
  |-Purpose of message   : 0
  |-Query/Response Flag   : 0
  |-Response code   : 0
  |-Checking Disabled   : 0
  |-Authenticated data   : 0
  |-Recursion available   : 0
  |-Number of question entries : 256
  |-Number of answer entries   : 0
  |-Number of authority entries: 0
  |-Number of resource entries : 0

UDP Header
  |-Source Port   : 34148
  |-Destination Port : 53
  |-UDP Length   : 43
  |-UDP Checksum   : 65342

IP Header:
  |-IP Version   : 4
  |-Type Of Service   : 0
  |-IP Total Length   : 63 Bytes(Size of Packet)
  |-Identification   : 45566
  |-TTL   : 64
  |-Protocol : 17
  |-Checksum : 35245

Ethernet Header:
  -->Destination Address : 00-00-00-00-00-00
  |-Source Address   : 00-00-00-00-00-00
  |-Protocol   : 8

Packet number 3
Packet type: DNS

DNS message

DNS Header:
  |-Identification Number   : 6650
  |-Recursion Desired   : 1
  |-Truncated Message   : 0
  |-Authoritative Answer   : 0
  |-Purpose of message   : 0
  |-Query/Response Flag   : 0
  |-Response code   : 0
  |-Checking Disabled   : 0
  |-Authenticated data   : 0
  |-Recursion available   : 0
  |-Number of question entries : 256
  |-Number of answer entries   : 0
  |-Number of authority entries: 0
  |-Number of resource entries : 0

UDP Header
  |-Source Port   : 34148
  |-Destination Port : 53
  |-UDP Length   : 43
  |-UDP Checksum   : 65342

IP Header:
  |-IP Version   : 4
  |-Type Of Service   : 0
  |-IP Total Length   : 63 Bytes(Size of Packet)
  |-Identification   : 45566
  |-TTL   : 64
  |-Protocol : 17
  |-Checksum : 35245

Ethernet Header:
  -->Destination Address : 00-00-00-00-00-00
  |-Source Address   : 00-00-00-00-00-00
  |-Protocol   : 8

Packet number 4
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 6650
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 1
|-Response code : 5
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 1
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 53
|-Destination Port : 34148
|-UDP Length : 43
|-UDP Checksum : 65342

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 14192
|-TTL : 64
|-Protocol : 17
|-Checksum : 1084

Ethernet Header:
-->Destination Address : 00-00-00-00-00-00
|-Source Address : 00-00-00-00-00-00
|-Protocol : 8

Packet number 5
Packet type: DNS

DNS message

DNS Header:

|-Identification Number : 6650
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 1
|-Response code : 5
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 1
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 53
|-Destination Port : 34148
|-UDP Length : 43
|-UDP Checksum : 65342

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 14192
|-TTL : 64
|-Protocol : 17
|-Checksum : 1084

Ethernet Header:
-->Destination Address : 00-00-00-00-00-00
|-Source Address : 00-00-00-00-00-00
|-Protocol : 8

Packet number 6
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 54604
|-Recursion Desired : 1

|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 1
|-Response code : 5
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 1
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 53
|-Destination Port : 39396
|-UDP Length : 43
|-UDP Checksum : 48442

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 13894
|-TTL : 64
|-Protocol : 17
|-Checksum : 61347

Ethernet Header:
-->Destination Address : 48-BF-6B-DF-21-D8
|-Source Address : 30-E3-7A-10-DD-D5
|-Protocol : 8

Packet number 7
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 20058
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0

|-Purpose of message : 0
|-Query/Response Flag : 0
|-Response code : 0
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 0
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 46146
|-Destination Port : 53
|-UDP Length : 43
|-UDP Checksum : 5609

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 55249
|-TTL : 64
|-Protocol : 17
|-Checksum : 36376

Ethernet Header:
-->Destination Address : 30-E3-7A-10-DD-D5
|-Source Address : 48-BF-6B-DF-21-D8
|-Protocol : 8

Packet number 8
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 24238
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 0

|-Response code : 0
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 0
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 25561
|-Destination Port : 53
|-UDP Length : 43
|-UDP Checksum : 65342

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 45567
|-TTL : 64
|-Protocol : 17
|-Checksum : 35244

Ethernet Header:
-->Destination Address : 00-00-00-00-00-00
|-Source Address : 00-00-00-00-00-00
|-Protocol : 8

Packet number 9
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 24238
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 0
|-Response code : 0
|-Checking Disabled : 0

|-Authenticated data : 0
|-Recursion available : 0
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 25561
|-Destination Port : 53
|-UDP Length : 43
|-UDP Checksum : 65342

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 45567
|-TTL : 64
|-Protocol : 17
|-Checksum : 35244

Ethernet Header:
-->Destination Address : 00-00-00-00-00-00
|-Source Address : 00-00-00-00-00-00
|-Protocol : 8

Packet number 10
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 24238
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 1
|-Response code : 5
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 1

|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 53
|-Destination Port : 25561
|-UDP Length : 43
|-UDP Checksum : 65342

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 14193
|-TTL : 64
|-Protocol : 17
|-Checksum : 1083

Ethernet Header:
-->Destination Address : 00-00-00-00-00-00
|-Source Address : 00-00-00-00-00-00
|-Protocol : 8

Packet number 11
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 24238
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 1
|-Response code : 5
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 1
|-Number of question entries : 256
|-Number of answer entries : 0

|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
|-Source Port : 53
|-Destination Port : 25561
|-UDP Length : 43
|-UDP Checksum : 65342

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 63 Bytes(Size of Packet)
|-Identification : 14193
|-TTL : 64
|-Protocol : 17
|-Checksum : 1083

Ethernet Header:
-->Destination Address : 00-00-00-00-00-00
|-Source Address : 00-00-00-00-00-00
|-Protocol : 8

Packet number 12
Packet type: DNS

DNS message

DNS Header:
|-Identification Number : 20058
|-Recursion Desired : 1
|-Truncated Message : 0
|-Authoritative Answer : 0
|-Purpose of message : 0
|-Query/Response Flag : 1
|-Response code : 5
|-Checking Disabled : 0
|-Authenticated data : 0
|-Recursion available : 1
|-Number of question entries : 256
|-Number of answer entries : 0
|-Number of authority entries: 0
|-Number of resource entries : 0

UDP Header
   |-Source Port   : 53
   |-Destination Port : 46146
   |-UDP Length   : 43
   |-UDP Checksum   : 38243

IP Header:
   |-IP Version   : 4
   |-Type Of Service   : 0
   |-IP Total Length   : 63 Bytes(Size of Packet)
   |-Identification   : 13895
   |-TTL   : 64
   |-Protocol : 17
   |-Checksum : 61346

Ethernet Header:
   -->Destination Address : 48-BF-6B-DF-21-D8
   |-Source Address   : 30-E3-7A-10-DD-D5
   |-Protocol   : 8

Packet number 13
Packet type: HTTP

HTTP message
.............3.).....

TCP Header:
   |-Source Port   : 51173
   |-Destination Port : 80
   |-Sequence Number   : 2836641336
   |-Acknowledge Number : 0
   |-Header Length   : 11 DWORDS or 44 BYTES
   |-Urgent Flag   : 0
   |-Acknowledgement Flag : 0
   |-Push Flag   : 0
   |-Reset Flag   : 0
   |-Synchronise Flag   : 1
   |-Finish Flag   : 0
   |-Window   : 65535
   |-Checksum   : 44408
   |-Urgent Pointer : 0

IP Header:
  |-IP Version   : 4
  |-Type Of Service   : 0
  |-IP Total Length   : 64 Bytes(Size of Packet)
  |-Identification   : 62322
  |-TTL   : 64
  |-Protocol : 6
  |-Checksum : 43167

Ethernet Header:
  -->Destination Address : 30-E3-7A-10-DD-D5
  |-Source Address   : 48-BF-6B-DF-21-D8
  |-Protocol   : 8

Packet number 14
Packet type: TCP

TCP Header:
  |-Source Port   : 51168
  |-Destination Port : 443
  |-Sequence Number   : 3910365137
  |-Acknowledge Number : 0
  |-Header Length   : 11 DWORDS or 44 BYTES
  |-Urgent Flag   : 0
  |-Acknowledgement Flag : 0
  |-Push Flag   : 0
  |-Reset Flag   : 0
  |-Synchronise Flag   : 1
  |-Finish Flag   : 0
  |-Window   : 65535
  |-Checksum   : 45689
  |-Urgent Pointer : 0

IP Header:
  |-IP Version   : 4
  |-Type Of Service   : 0
  |-IP Total Length   : 64 Bytes(Size of Packet)
  |-Identification   : 38687
  |-TTL   : 64
  |-Protocol : 6
  |-Checksum : 1267

Ethernet Header:
  -->Destination Address : 30-E3-7A-10-DD-D5

|-Source Address : 48-BF-6B-DF-21-D8
|-Protocol : 8

Packet number 15
Packet type: HTTP

HTTP message
.....3.....

TCP Header:
|-Source Port : 51173
|-Destination Port : 80
|-Sequence Number : 2836641336
|-Acknowledge Number : 0
|-Header Length : 11 DWORDS or 44 BYTES
|-Urgent Flag : 0
|-Acknowledgement Flag : 0
|-Push Flag : 0
|-Reset Flag : 0
|-Synchronise Flag : 1
|-Finish Flag : 0
|-Window : 65535
|-Checksum : 43408
|-Urgent Pointer : 0

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 64 Bytes(Size of Packet)
|-Identification : 17442
|-TTL : 64
|-Protocol : 6
|-Checksum : 22512

Ethernet Header:
-->Destination Address : 30-E3-7A-10-DD-D5
|-Source Address : 48-BF-6B-DF-21-D8
|-Protocol : 8

Packet number 16
Packet type: TCP

TCP Header:
|-Source Port : 51168
|-Destination Port : 443

|-Sequence Number : 3910365137
|-Acknowledge Number : 0
|-Header Length : 11 DWORDS or 44 BYTES
|-Urgent Flag : 0
|-Acknowledgement Flag : 0
|-Push Flag : 0
|-Reset Flag : 0
|-Synchronise Flag : 1
|-Finish Flag : 0
|-Window : 65535
|-Checksum : 44689
|-Urgent Pointer : 0

IP Header:
|-IP Version : 4
|-Type Of Service : 0
|-IP Total Length : 64 Bytes(Size of Packet)
|-Identification : 7059
|-TTL : 64
|-Protocol : 6
|-Checksum : 32895

Ethernet Header:
-->Destination Address : 30-E3-7A-10-DD-D5
|-Source Address : 48-BF-6B-DF-21-D8
|-Protocol : 8

6. Contribution Summary

Ajay
Application layer: DNS
Transport layer: UDP
Network layer: IPv4

Shyam
Application layer: HTTP
Transport layer: TCP
Network layer: IPv6

Common
Coding and ARP (data link layer protocol)

7. REFERENCES

- https://www.tutorialandexample.com/scan-disk-scheduling-algorithm
- https://www.paessler.com/it-explained/packet-sniffing
- https://www.geeksforgeeks.org/what-is-packet-sniffing/
- https://www.netscout.com/what-is/sniffer