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

Mini Project Report
of
Computer Networks LAB

GUI BASED APPLICATION FOR GENERATION OF NETWORK PACKET PARSING LOGS

SUBMITTED
BY

Student Details:

1) Name: **Yash Mandhania** Reg no:**200905392** Roll No:62
section: **B**

2) Name: **Shourya Gupta** Reg no:**200905282** Roll No:51
section: **B**

Table Of Contents:

ACKNOWLEDGEMENT

Our team **(Yash Mandhania and Shourya Gupta)** would like to thank the Department of Computer Science and Engineering for giving us the opportunity to work on a project to help in our understanding of the coursework.

We would like to thank our teacher, **Mr. Manoj R**, for guiding us through this project and our coursework in Computer Networks. His teachings have inspired us to take up this implementation. We would also like to express our sincere gratitude to **Mr. Prashant Barla**, who also guided us during our labs.

A particular acknowledgement goes to our lab mates, who helped us by exchanging exciting ideas and information. Finally, we would like to thank our parents for their unwavering support and continuous encouragement in our educational journey.

ABSTRACT

Packet parsing and identification is an essential and integral part of networking. Most devices that are part of the networking chain in our day to day lives perform some amount of packet decoding and reassembly while passing packets. This is especially important at the lower layers of the OSI model since more devices will need to access to the encapsulated data to determine were to transfer the packet.

In this project, we have implemented a basic program that performs packet parsing and identifies various application layer protocols. To this end, we have studied and come up with algorithms to parse or identify protocols at various parts of the OSI model. These include, the Ethernet layer, the IP layer, the transport layer as well as certain application layer protocols.

1 INTRODUCTION

1.1 GENERAL INTRODUCTION

The OSI Model (Open Systems Interconnection Model) is a conceptual framework used to describe the functions of a networking system. The OSI model characterizes computing functions into a universal set of rules and requirements to support interoperability between different products and software. In the OSI reference model, the communications between a computing system are split into seven different abstraction layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application.

Physical Layer

The lowest layer of the OSI Model is concerned with electrically or optically transmitting raw unstructured data bits across the network from the physical layer of the sending device to the physical layer of the receiving device. the physical layer, one might find “physical” resources such as network hubs, cabling, repeaters, network adapters or modems.

Data Link Layer

At the data link layer, directly connected nodes are used to perform node-to-node data transfer where data is packaged into frames. The data link layer also corrects errors that may have occurred at the physical layer.

The data link layer encompasses two sub-layers of its own. The first, media access control (MAC), provides flow control and multiplexing for device transmissions over a network. The second, the logical link control (LLC), provides flow and error control over the physical medium as well as identifies line protocols.

Network Layer

The network layer is responsible for receiving frames from the data link layer and delivering them to their intended destinations among based on the addresses contained inside the frame. The network layer finds the destination by using logical addresses, such as IP (internet protocol).

Transport Layer

The transport layer manages the delivery and error checking of data packets. It regulates the size, sequencing, and ultimately the transfer of data between systems and hosts. One of the most common examples of the transport layer is TCP or the Transmission Control Protocol.

Session Layer

The session layer controls the conversations between different computers. A session or connection between machines is set up, managed, and terminated at layer 5. Session layer services also include authentication and reconnections.

Presentation Layer
The presentation layer formats or translates data for the application layer based on the syntax or semantics that the application accepts This layer can also handle the encryption and decryption required by the application layer.

Application Layer
At this layer, both the end user and the application layer interact directly with the software application. This layer sees network services provided to end-user applications. The application layer identifies communication partners, resource availability, and synchronizes communication.

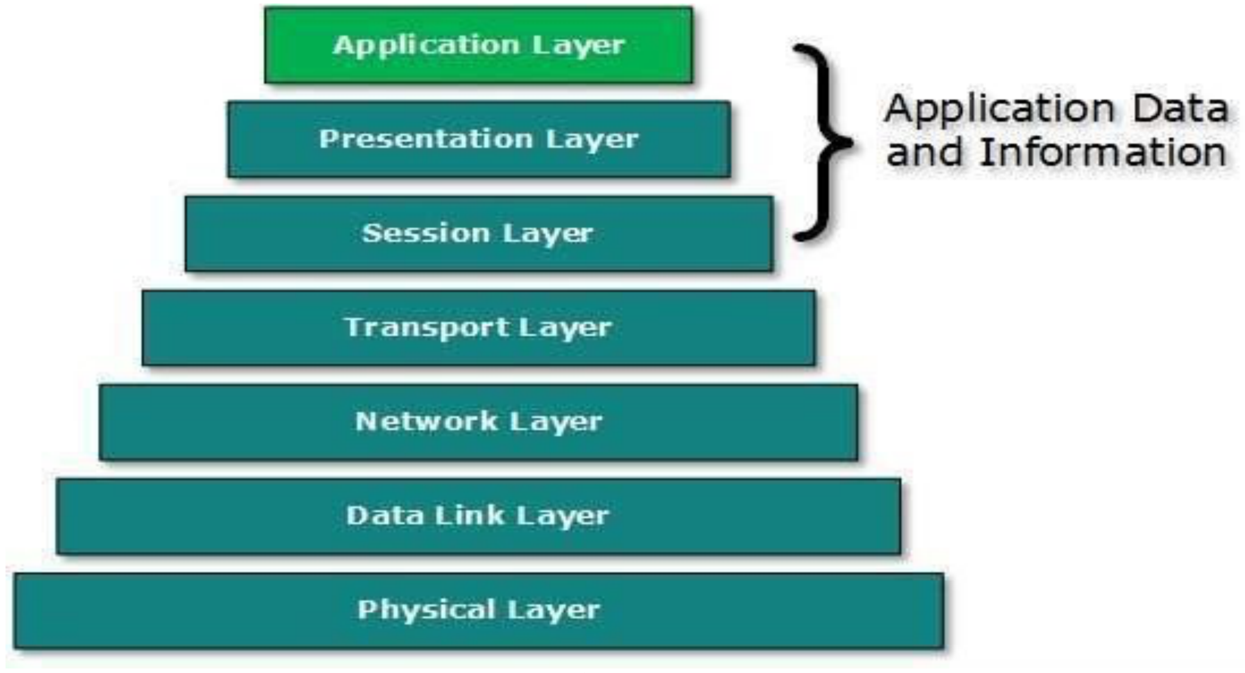


Figure 1.1

1.1.1 PROTOCOL STACK

A protocol stack is a group of protocols that all work together to allow software or hardware to perform a function. These functions are all separated into separate 'layers' of data that all require a protocol to be transferred. So, the transport layer for example, responsible for the physical transfer of data, will have a range of protocols which can be used to communicate the data. The Data Link layer has other protocols associated with its data type and is responsible for the addressing of data from the other layers. These different protocols cannot be combined because that could create sets of rules that are too complex to carry out and incompatible in function. Having different protocols in the different layers of a network is a solution but an essential part of this is to be able to communicate with each other to enable an overall function to take place (i.e., a transfer of data across a network). When protocols are able to interact in such a way so in a combined activity.

The **TCP/IP** protocol stack is a good example. It uses four layers that map to the OSI model as follows

- **Layer 1: Network Interface** - This layer combines the Physical and Data layers and routes the data between devices on the same network. It also manages the exchange of data between the network and other devices.
- **Layer 2: Internet** - This layer corresponds to the Network layer. The **Internet Protocol** (IP) uses the IP address, consisting of a **Network Identifier** and a **Host Identifier**, to determine the address of the device it is communicating with.
- **Layer 3: Transport** - Corresponding to the OSI Transport layer, this is the part of the protocol stack where the **Transport Control Protocol** (TCP) can be found. TCP works by asking another device on the network if it is willing to accept information from the local device.
- **Layer 4: Application** - Layer 4 combines the Session, Presentation and Application layers of the OSI model. Protocols for specific functions such as e-mail (**Simple Mail Transfer Protocol**, **SMTP**) and file transfer (**File Transfer Protocol**, **FTP**) reside at this level.

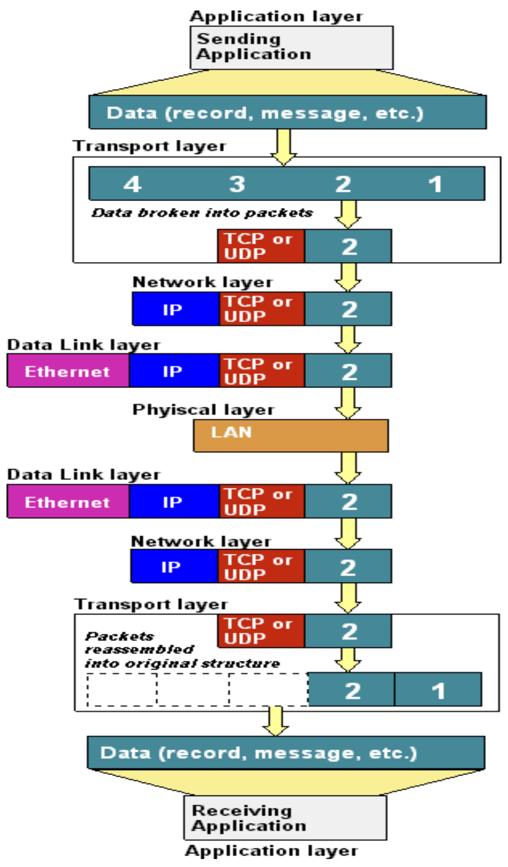


Figure 1.1.1

1.2 REQUIREMENTS

- **HARDWARE**

• At least 200 MB of available disk space.

• 64-bit AMD64/x86-64 or 32-bit x86 CPU architecture.

• At least 2GB RAM

- **SOFTWARE**

• Ubuntu 16.04+ or any Debian based Linux operating System.

• Modules and tools: GCC COMPILER 10+ version.

2 PROBLEM DEFINITION

Devise a software program to model various protocols of the network protocol stack along with generation of data logs files of the parsed packets received from the RAW_SOCKET.

3 OBJECTIVES

->To capture packets received from Linux raw socket configured to Ethernet.

->To parse various packets according to network stack protocols.

->To display the various attributes and parameters related to headers of packets of concerned protocols

->To generate data LOG files in .txt format for efficient user experience and start to end output analysis of the parsed packets.

4 METHODOLOGY

->Connect the device to a stable internet connection.

->Open the Source code provided in a Linux based Operating System have a GCC compiler

->Using the MakeFile provided compile the given Source code using the command

(CC=gcc make) on the terminal.

->Continue the packet capturing for a desired time interval by executing **(. /PacketAnalyzer)**.

Stop the capture by performing the **CRTL ^C** operation.

->Open the GUI and click on the **GENERATE LOG REPORT BUTTON** to analyze the different parsed packets.

5 IMPLEMENTATION

We have captured traffic from the network connection of our laptop device using RAW_SOCKET on Linux. When protocol is set to **htons(ETH_P_ALL),** then all protocols are received. All incoming packets of that protocol type will be passed to the packet socket before they are passed to the protocols implemented in the kernel.**SOCK_RAW** packets are passed to and from the device driver without any changes in the packet data. When receiving a packet, the address is still parsed and passed in a standard *sockaddr_ll* address structure. When transmitting a packet, the user-supplied buffer should contain the physical-layer header. That packet is then queued unmodified to the network driver of the interface defined by the destination address.

```
socket_fd = socket (AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
```

the above piece of instruction is used to create the required socket to receive the packets from the network.

```
data_size =
        recvfrom(socket_fd, data, 65536, 0, &saddr, (socklen_t *)&sock_len);
```

recvfrom inside the while loop is used to receive the required packet data from the socket created.

```
process_packet(data, data_size);
```

The function process_packet(data,data_size) initiates the parsing process of the received packets specifying the packet number and status of current packet which is whether it is under process or whether it has been finished processing.

Further calling the **process_ethernet()** function which in turn calls **process_IPv4() or process_IPv6().**which starts the parsing of IPv4 AND IPv6 packet headers.

Based on the transport layer protocol applicable to the packet, **process_IPv4() or process_IPv6() calls process_TCP(true) or process_UDP(true)** to start tcp or udp packet header parsing.

Finally the **display_processing_stats()** function is used to display stats of the packet received.
print_UDP_header(), print_TCP_header(), print_DNS_header(), print_IPv4_header(), and **print_IPV6_header() functions** are then used to write the parsed packet header data to the**LOG.txt file.**

process_packet.c

```c
#include "process_packet.h"

static void *glbl_data;
static size_t glbl_data_size;
static struct stats curr_stats;

void init_processing_stats() { memset(&curr_stats, 0, sizeof(curr_stats)); }

void display_processing_stats() {
  printf("<><><><>....Total number of packets analysed: %d\n",
         curr_stats.num_of_packets);
  txt("IPv4", curr_stats.ipv4_packets);
  txt("IPv6", curr_stats.ipv6_packets);
  txt("TCP", curr_stats.tcp_packets);
  txt("UDP", curr_stats.udp_packets);
  txt("ARP", curr_stats.arp_packets);
  txt("DNS", curr_stats.dns_packets);
  txt("HTTP", curr_stats.http_packets);
  txt("HTTPS", curr_stats.https_packets);
  txt("FTP", curr_stats.ftp_packets);
  txt("SMTP", curr_stats.smtp_packets);
}

void print_UDP_header(struct udphdr *udph) {
  write_lg("\n");
  write_lg(">>UDP Header<<");
  write_lg("<>>>>-Source Port : %d", ntohs(udph->source));
  write_lg("<>>>>-Destination Port : %d", ntohs(udph->dest));
  write_lg("<>>>>-UDP Length : %d", ntohs(udph->len));
  write_lg("<>>>>-UDP Checksum : %d", ntohs(udph->check));
  write_lg("\n");
}

void print_TCP_header(struct tcphdr *tcph) {
  write_lg("\n");
  write_lg(">>TCP Header<<");
  write_lg("<>>>>-Source Port      : %u", ntohs(tcph->source));
  write_lg("<>>>>-Destination Port : %u", ntohs(tcph->dest));
  write_lg("<>>>>-Sequence Number    : %u", ntohl(tcph->seq));
  write_lg("<>>>>-Acknowledge Number : %u", ntohl(tcph->ack_seq));
  write_lg("<>>>>-Header Length      : %d DWORDS or %d BYTES",
           (unsigned int)tcph->doff, (unsigned int)tcph->doff * 4);
  write_lg("<>>>>-Urgent Flag          : %d", (unsigned int)tcph->urg);
  write_lg("<>>>>-Acknowledgement Flag : %d", (unsigned int)tcph->ack);
  write_lg("<>>>>-Push Flag            : %d", (unsigned int)tcph->psh);
  write_lg("<>>>>-Reset Flag           : %d", (unsigned int)tcph->rst);
  write_lg("<>>>>-Synchronise Flag     : %d", (unsigned int)tcph->syn);
  write_lg("<>>>>-Finish Flag          : %d", (unsigned int)tcph->fin);
  write_lg("<>>>>-Window         : %d", ntohs(tcph->window));
  write_lg("<>>>>-Checksum       : %d", ntohs(tcph->check));
  write_lg("<>>>>-Urgent Pointer : %d", tcph->urg_ptr);
  write_lg("\n");

  return;
}

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

void process_DNS_header(bool ipv4_type) {
  struct dnshdr *shdr;
  if (ipv4_type) {
    shdr =
        (struct dnshdr *)(glbl_data + sizeof(struct tcphdr) +
                          sizeof(struct ether_header) + sizeof(struct iphdr));
  } else {
    shdr =
        (struct dnshdr *)(glbl_data + sizeof(struct tcphdr) +
                          sizeof(struct ether_header) + sizeof(struct ip6_hdr));
  }
  print_DNS_header(shdr);
}

void process_tls() {

}

void process_TCP(bool ipv4_type) {
  curr_stats.tcp_packets++;
  struct tcphdr *hdr;
  if (ipv4_type) {
    hdr = (struct tcphdr *)(glbl_data + sizeof(struct ether_header) +
                            sizeof(struct iphdr));
  } else {
    hdr = (struct tcphdr *)(glbl_data + sizeof(struct ether_header) +
                            sizeof(struct ip6_hdr));
  }
  print_TCP_header(hdr);
  if (port(80)) {
    write_lg(">>>HTTP Protocol<<<");
    curr_stats.http_packets++;
  } else if (port(443)) {
    write_lg(">>>HTTP/TLS Protocol<<<");
    process_tls();
    curr_stats.https_packets++;
  } else if (port(25) || port(587)) {
    write_lg(">>>SMTP Protocol<<<");
    curr_stats.smtp_packets++;
  } else if (port(20) || port(21)) {
    write_lg(">>>FTP Protocol<<<");
    curr_stats.ftp_packets++;
  }
}

void process_UDP(bool ipv4_type) {
  curr_stats.udp_packets++;
  struct udphdr *hdr;
  if (ipv4_type) {
    hdr = (struct udphdr *)(glbl_data + sizeof(struct ether_header) +
                            sizeof(struct iphdr));
  } else {
    hdr = (struct udphdr *)(glbl_data + sizeof(struct ether_header) +
                            sizeof(struct ip6_hdr));
```

```c
  }
  print_UDP_header(hdr);
  if (port(53)) {
    curr_stats.dns_packets++;
    process_DNS_header(ipv4_type);
  }
}

void print_IPv4_header(struct iphdr *iph) {
 write_lg("\n");
  write_lg(">>IP HEADER<<");
  write_lg("<>>>>-IP Version    : %d", (unsigned int)iph->version);
  write_lg("<>>>>-Type Of Service : %d", (unsigned int)iph->tos);
  write_lg("<>>>>-IP Total Length : %d  Bytes(Size of Packet)",
           ntohs(iph->tot_len));
  write_lg("<>>>>-Identification    : %d", ntohs(iph->id));
  write_lg("<>>>>-TTL : %d", (unsigned int)iph->ttl);
  write_lg("<>>>>-Protocol : %d", (unsigned int)iph->protocol);
  write_lg("<>>>>-Checksum : %d", ntohs(iph->check));
  write_lg("\n");
}

void print_IPV6_header(struct ipv6_header *hdr) {
  char src[50], dst[50], stemp[5], dtemp[5];
  int i;

  memset(src, 0, sizeof(src));
  memset(dst, 0, sizeof(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);
  }
  write_lg("\n");
  write_lg(">>IPV6 HEADER<<");
  write_lg("<>>>>>-Source   :   %s",   src);
  write_lg("<>>>>>-Destination : %s", dst);
  write_lg("\n");
```

```c
}

void process_IPv4() {
  curr_stats.ipv4_packets++;
  struct iphdr *iph = (struct iphdr *)(glbl_data + sizeof(struct ether_header));
  print_IPv4_header(iph);
  switch (iph->protocol) {
  case 6:
    process_TCP(true);
    break;
  case 17:
    process_UDP(true);
    break;
  default:
    break;
  }
}

void process_ARP() {
  curr_stats.arp_packets++;
  struct arp_header *hdr =
      (struct arp_header *)(glbl_data + sizeof(struct ether_header));
 write_lg("\n");
  write_lg(">>ARP HEADER<<");
  write_lg("<>>>>>-Hardware type : %d", ntohs(hdr->htype));
  write_lg("<>>>>>-Protocol Type : %d", ntohs(hdr->ptype));
  write_lg("<>>>>>-Hardware addr len: %d", ntohs(hdr->hlen));
  write_lg("<>>>>>-Protocol addr len: %d", ntohs(hdr->plen));
  write_lg("<>>>>>--Operation : %d", ntohs(hdr->plen));
  write_lg("\n");
}

void process_IPv6() {
  curr_stats.ipv6_packets++;
  struct ipv6_header *ip6h =
      (struct ipv6_header *)(glbl_data + sizeof(struct ether_header));

  switch (ntohs(ip6h->next_header)) {
  case 6:
    process_TCP(false);
    break;
  case 17:
    process_UDP(false);
    break;
  default:
    print_IPV6_header(ip6h);
    break;
  }
}

void printEther(struct ether_header *eth) {
 write_lg("\n");
  write_lg(">>ETHERNET HEADER<<");
  write_lg("<>>>>>-Destination Address: %.2X-%.2X-%.2X-%.2X-%.2X-%.2X",
           eth->ether_dhost[0], eth->ether_dhost[1], eth->ether_dhost[2],
           eth->ether_dhost[3], eth->ether_dhost[4], eth->ether_dhost[5]);
  write_lg("<>>>>>-Source Address: %.2X-%.2X-%.2X-%.2X-%.2X-%.2X",
           eth->ether_shost[0], eth->ether_shost[1], eth->ether_shost[2],
           eth->ether_shost[3], eth->ether_shost[4], eth->ether_shost[5]);
  write_lg("<>>>>>>>-Protocol: %u", (unsigned short)eth->ether_type);
  write_lg("\n");
}

void process_ethernet() {
  struct ether_header *eth = (struct ether_header *)glbl_data;
  printEther(eth);
  switch (ntohs(eth->ether_type)) {
  case 0x0800: // IPv4 Protocol
    process_IPv4();
    break;
  case 0x0806: // ARP Protocol
    process_ARP();
    break;
  case 0x86dd: // IPv6 Protocol
    process_IPv6();
    break;
  default: // Some Other Protocol
    break;
  }
}

void process_packet(void *data, size_t data_size) {
  glbl_data = NULL;
  glbl_data_size = 0;
  curr_stats.num_of_packets++;
  printf("_________________________________________________________________\n");
  printf("<><>--Packet %d recieved, processing has started ...\n", curr_stats.num_of_packets);
  write_lg("<><>____________________________________________________________<><>\n");
  write_lg("Packet Detials:\npacket NO.: %d\npacket Size %zu", curr_stats.num_of_packets,
           data_size);
  write_lg("<><>-----------------------------------------------------------<><>\n");
  glbl_data = data;
  glbl_data_size = data_size;
  process_ethernet();
  printf("<><>--Packet %d has been processed\n", curr_stats.num_of_packets);
  printf("-----------------------------------------------------------------\n");
}
}
```

process_packet.h

```
#pragma once
#include "common.h"
#include "logger.h"

#define port(A) ntohs(hdr->source) == A || ntohs(hdr->dest) == A
#define txt(A, B)                                                                   \
  printf("<><><><><><>Total number of packets that used the %s protocol: %d\n", A, B);

struct stats {
  int num_of_packets;
  int tcp_packets;
  int udp_packets;
  int dns_packets;
  int http_packets;
  int smtp_packets;
  int ssh_packets;
  int ftp_packets;
  int https_packets;
  int ipv4_packets;
  int ipv6_packets;
  int arp_packets;
  int gopher_packets;
};

struct tls_header {
    uint8_t content_type;
    uint16_t version;
    uint16_t length;
};

struct arp_header {
  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 ipv6_header {
#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;

  unsigned char src_ipv6[16];
  unsigned char dst_ipv6[16];
};

struct dnshdr {
  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
};

void init_processing_stats();
void process_packet(void *data, size_t data_size);
void display_processing_stats();
```

main.c

```
#include "common.h"
#include "logger.h"
#include "process_packet.h"

static unsigned char *data;
static fd_t socket_fd;

void sig_handler() {
  display_processing_stats();
  close(socket_fd);
  free(data);
  exit(EXIT_SUCCESS);
}

int main() {
  size_t data_size = -1, sock_len = sa_size;
  struct sockaddr saddr;

  signal(SIGINT, sig_handler);
  signal(SIGABRT, sig_handler);

  data = (unsigned char *)malloc(65536);

  socket_fd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));

  int num_of_packets = 0;

  init_logger("LOG.txt");
  init_processing_stats();

  if ((signed int)socket_fd < 0) {
    perror("socket");
    exit(EXIT_FAILURE);
  }
  while (true) {
    data_size =
        recvfrom(socket_fd, data, 65536, 0, &saddr, (socklen_t *)&sock_len);
    if ((signed long)data_size < 0) {
      perror("recvfrom");
      exit(EXIT_FAILURE);
    }

    process_packet(data, data_size);
  }

  display_processing_stats();
  destroy_logger();
  close(socket_fd);
  free(data);
}
```

common.h

```
#pragma once


#include <arpa/inet.h>
#include <errno.h>
#include <net/ethernet.h>
#include <netdb.h>
#include <netinet/if_ether.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip6.h>
#include <netinet/ip_icmp.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <signal.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
```

```
#include <sys/time.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

#define fd_t unsigned int
#define sa_size sizeof(struct sockaddr)
```

logger.c

```
#include "logger.h"

static FILE *fp = NULL;

void write_lg(const char *format, ...) {
  va_list args;
  va_start(args, format);
  vfprintf(fp, format, args);
  va_end(args);
  fprintf(fp, "\n");
}

void init_logger(char *file_name) { fp = fopen(file_name, "w"); }

void destory_logger() { fclose(fp); }
```

logger.h

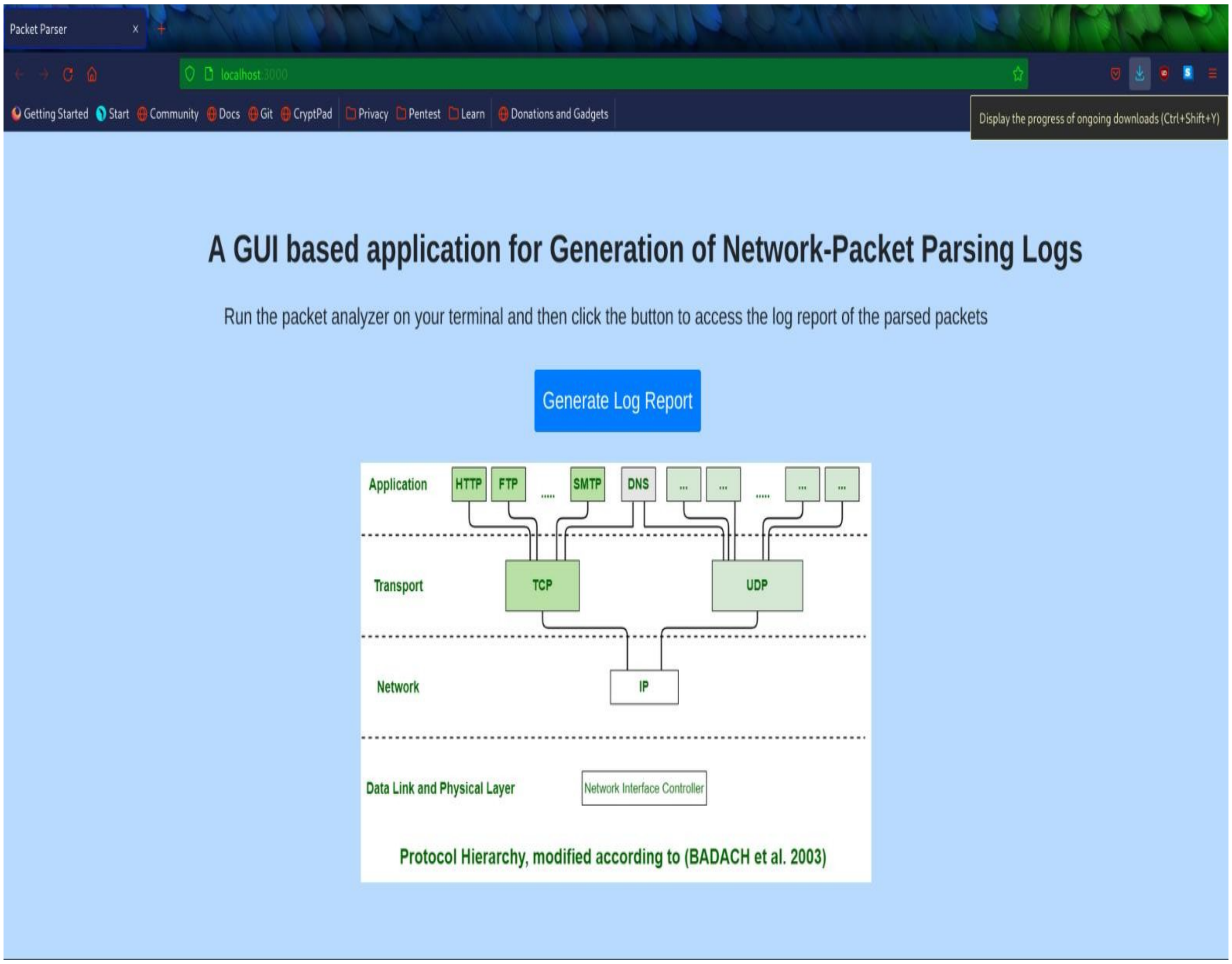
```
#pragma once
#include "common.h"

void write_lg(const char *, ...);
void init_logger(char *);
void destroy_logger();
```

Makefile

```
all: PacketAnalyzer

logger.o: logger.c
    $(CC) -Wall -c logger.c

process_packet.o: process_packet.c
```

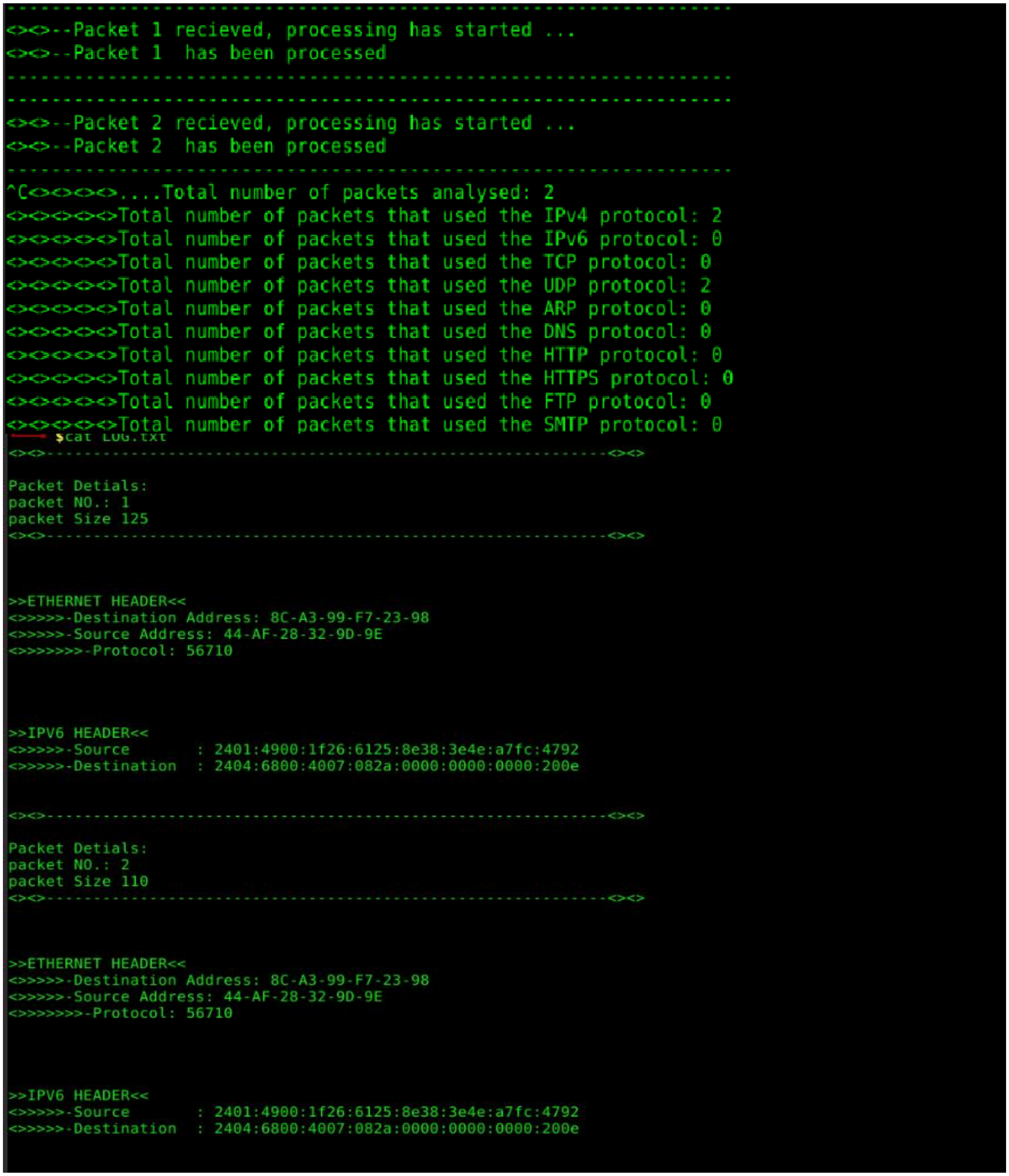
```
    $(CC) -Wall -c process_packet.c

sniffer: main.c logger.o process_packet.o
    $(CC) main.c logger.o process_packet.o -ggdb -o PacketAnalyzer
    sudo chown root PacketAnalyzer
    sudo chmod +s PacketAnalyzer

clean:
    rm -f *.o sniffer
```

OUTPUT

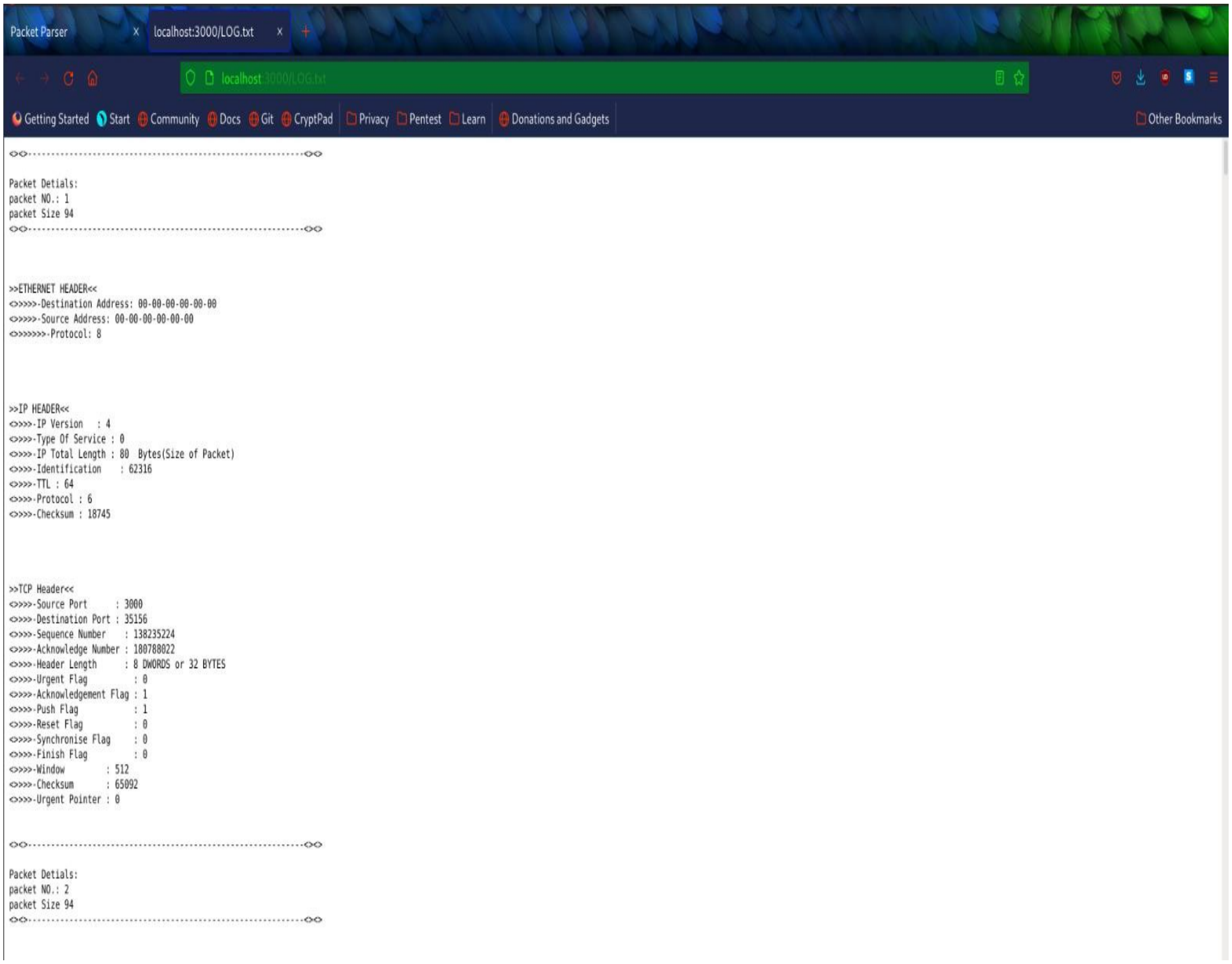


A GUI based application for Generation of Network-Packet Parsing Logs

Run the packet analyzer on your terminal and then click the button to access the log report of the parsed packets



```
-----------------------------------------------------------------
<><>--Packet 1 recieved, processing has started ...
<><>--Packet 1  has been processed
-----------------------------------------------------------------
-----------------------------------------------------------------
<><>--Packet 2 recieved, processing has started ...
<><>--Packet 2  has been processed
-----------------------------------------------------------------
^C<><><><>....Total number of packets analysed: 2
<><><><><>Total number of packets that used the IPv4 protocol: 2
<><><><><>Total number of packets that used the IPv6 protocol: 0
<><><><><>Total number of packets that used the TCP protocol: 0
<><><><><>Total number of packets that used the UDP protocol: 2
<><><><><>Total number of packets that used the ARP protocol: 0
<><><><><>Total number of packets that used the DNS protocol: 0
<><><><><>Total number of packets that used the HTTP protocol: 0
<><><><><>Total number of packets that used the HTTPS protocol: 0
<><><><><>Total number of packets that used the FTP protocol: 0
<><><><><>Total number of packets that used the SMTP protocol: 0
    $cat LOG.txt
<><>----------------------------------------------------------------<><>

Packet Detials:
packet NO.: 1
packet Size 125
<><>----------------------------------------------------------------<><>



>>ETHERNET HEADER<<
<>>>>>-Destination Address: 8C-A3-99-F7-23-98
<>>>>>-Source Address: 44-AF-28-32-9D-9E
<>>>>>>>-Protocol: 56710



>>IPV6 HEADER<<
<>>>>>-Source       : 2401:4900:1f26:6125:8e38:3e4e:a7fc:4792
<>>>>>-Destination  : 2404:6800:4007:082a:0000:0000:0000:200e


<><>----------------------------------------------------------------<><>

Packet Detials:
packet NO.: 2
packet Size 110
<><>----------------------------------------------------------------<><>



>>ETHERNET HEADER<<
<>>>>>-Destination Address: 8C-A3-99-F7-23-98
<>>>>>-Source Address: 44-AF-28-32-9D-9E
<>>>>>>>-Protocol: 56710



>>IPV6 HEADER<<
<>>>>>-Source       : 2401:4900:1f26:6125:8e38:3e4e:a7fc:4792
<>>>>>-Destination  : 2404:6800:4007:082a:0000:0000:0000:200e
```



```
<><>------------------------------------------------------------<><>

Packet Detials:
packet NO.: 1
packet Size 94
<><>------------------------------------------------------------<><>



>>ETHERNET HEADER<<
<>>>>>-Destination Address: 00-00-00-00-00-00
<>>>>>-Source Address: 00-00-00-00-00-00
<>>>>>>>-Protocol: 8



>>IP HEADER<<
<>>>>-IP Version   : 4
<>>>>-Type Of Service : 0
<>>>>-IP Total Length : 80  Bytes(Size of Packet)
<>>>>-Identification    : 62316
<>>>>-TTL : 64
<>>>>-Protocol : 6
<>>>>-Checksum : 18745



>>TCP Header<<
<>>>>-Source Port      : 3000
<>>>>-Destination Port : 35156
<>>>>-Sequence Number    : 138235224
<>>>>-Acknowledge Number : 180788022
<>>>>-Header Length      : 8 DWORDS or 32 BYTES
<>>>>-Urgent Flag          : 0
<>>>>-Acknowledgement Flag : 1
<>>>>-Push Flag            : 1
<>>>>-Reset Flag           : 0
<>>>>-Synchronise Flag     : 0
<>>>>-Finish Flag          : 0
<>>>>-Window        : 512
<>>>>-Checksum      : 65092
<>>>>-Urgent Pointer : 0


<><>------------------------------------------------------------<><>

Packet Detials:
packet NO.: 2
packet Size 94
<><>------------------------------------------------------------<><>
```

CONTRIBUTION FROM EACH STUDENT

1. **Yash Mandhania-** RESPONSIBLE FOR COMBINING ALL THE PROTOCOLS AS WELL AS WRITING IPv4, IPv6,DNS PART OF THE CODE FOR THE PROJECT AND FOR MAKING THE FINAL REPORT
2. **Shourya Gupta-** RESPONSIBLE FOR WRITING THE TCP, UDP, SMTP, HTTP PROTOCOLS OF THE CODE AND CORRECTING ANY CODE BUGS AND DOCUMENTATION

REFERENCES

1) http://yuba.stanford.edu/~nickm/papers/ancs48-gibb.pdf
2) https://www.javatpoint.com/tcp
3) https://www.javatpoint.com/udp-protocol
4) https://www.thousandeyes.com/learning/techtorials/ipv4-vs-ipv6
5) https://www.thousandeyes.com/learning/techtorials/ipv4-vs-ipv6
6) https://www.javatpoint.com/simple-mail-transfer-protocol
7) https://www.javatpoint.com/ssh-meaning
8) https://en.wikipedia.org/wiki/Application_layer#:~:text=An%20application%20layer%20is%20an,IP)%20and%20the%20OSI%20model.
9) https://www.geeksforgeeks.org/protocols-application-layer/