



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

Mini Project Report of Computer Networks LAB

TITLE
**A tool for capturing the traffic based on the TCP
header flags**

SUBMITTED BY

| NAME | REGISTRATION NUMBER | ROLL NUMBER | SECTION |
|-------------------------|--------------------------------|--------------------|----------------|
| ABHAY MUDGIL | 200905172 | 33 | CSE-B |

ABSTRACT

- In this project, we will be designing a tool that captures the traffic based on the TCP header flag.
- In TCP connection, flags are used to indicate a particular state of the connection or to provide some additional useful information like troubleshooting purposes or to handle a control of a particular connection. The most commonly used flags are “SYN”, “ACK” and “FIN”.
- Here, we are capturing the packets from a raw socket, and classifying the packets according to the TCP flags.

CHAPTER 1: INTRODUCTION

1.1 GENERAL INTRODUCTION TO THE TOPIC

1.1.1 About TCP

The Transmission Control Protocol (TCP) is a transport protocol that is used on top of IP to ensure the reliable transmission of packets. TCP includes mechanisms to solve many of the problems that arise from packet-based messaging, such as lost packets, out of order packets, duplicate packets, and corrupted packets. Since TCP is the protocol used most commonly on top of IP, the Internet protocol stack is sometimes referred to as TCP/IP.

1.1.2 Format of a TCP Packet

When sending packets using TCP/IP, the data portion of each IP packet is formatted as a TCP segment. Each TCP segment contains a header and data. The TCP header contains many more fields than the UDP header and can range in size from 20 to 60 bytes, depending on the size of the options field. The TCP header shares some fields with the UDP header: source port number, destination port number, and checksum.

Fig. 1.1 illustrates the details of TCP segment.

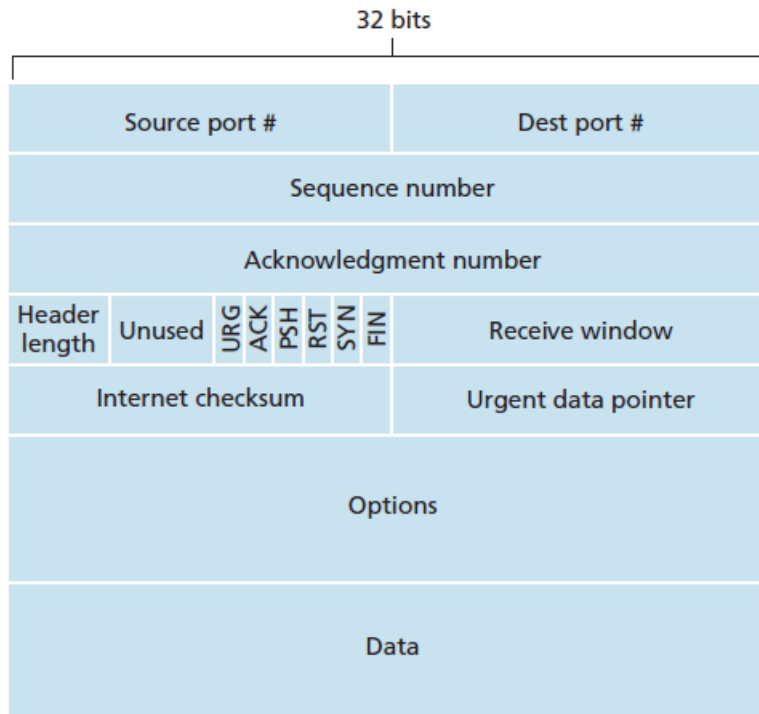


Fig. 1.1

1.1.3 Process of Transmitting a Packet with TCP/IP

When two computers want to send data to each other over TCP, they first need to establish a connection using a three-way handshake. Fig. 1.2 demonstrates the TCP three-way handshake. The first computer sends a packet with the SYN bit set to 1 (SYN = "Synchronize"). The second computer sends back a packet with the ACK bit set to 1 (ACK = "Acknowledge") plus the SYN bit set to 1. The first computer replies back with an ACK. The SYN and ACK bits are both parts of the TCP header.

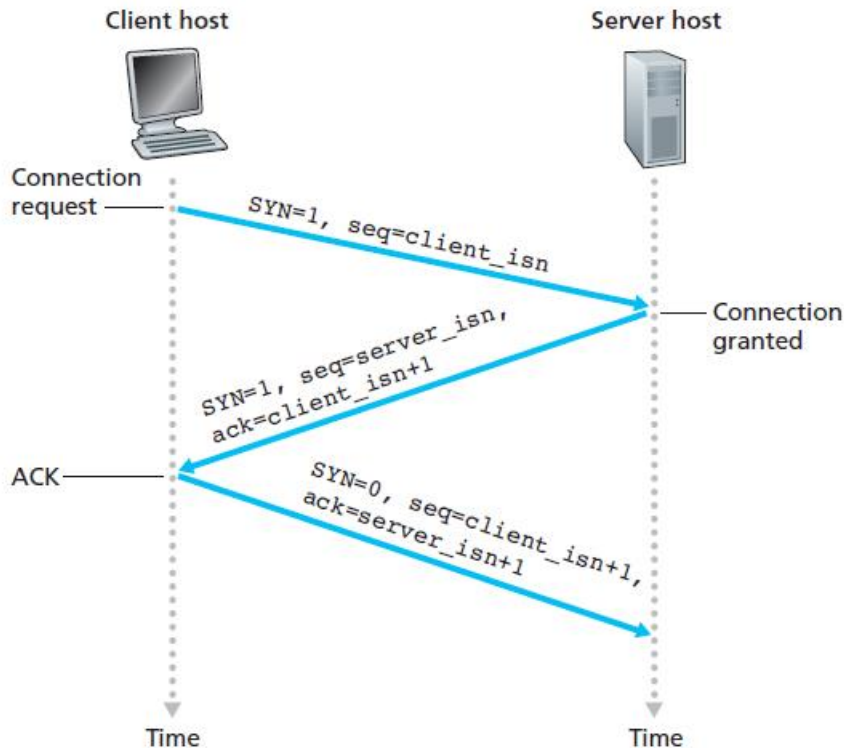


Fig. 1.2

In fact, the three packets involved in the three-way handshake do not typically include any data. Once the computers are done with the handshake, they're ready to receive packets containing actual data. When a packet of data is sent over TCP, the recipient must always acknowledge what they received.

The first computer sends a packet with data and a sequence number. The second computer acknowledges it by setting the ACK bit and increasing the acknowledgement number by the length of the received data. The sequence and acknowledgement numbers are part of the TCP header. These two numbers help the computers to keep track of which data was successfully received, which data was lost, and which data was accidentally sent twice.

Either computer can close the connection when they no longer want to send or receive data. A computer initiates closing the connection by sending a packet with the FIN bit set to 1 (FIN = finish). The other computer replies with an ACK and another FIN. After one more ACK from the initiating computer, the connection is closed. This process of closing the TCP connection is shown in Fig. 1.3.

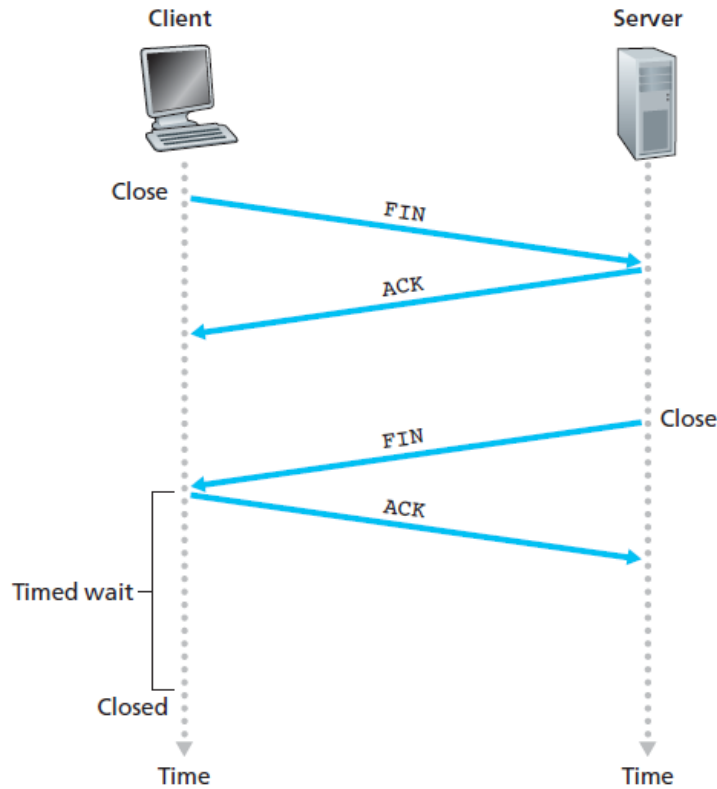


Fig. 1.3

1.2 HARDWARE AND SOFTWARE REQUIREMENTS

1.2.1 Hardware Requirements

- Standard Pentium Series Processor. (Recommended Pentium III or Advanced).
- Minimum 256 MB RAM. (Recommended 512 MB)
- HDD Storage capacity of 4 GB with 5400 rpm or more.

1.2.1 Software Requirements

- Operating System - Ubuntu (Linux)
- Programming Language - C
- Programming Tool - Sublime Text Editor and Linux Terminal
- Documentation Tool - Microsoft Word, Adobe Acrobat PDF Viewer

CHAPTER 2: PROBLEM DEFINITION

Design a Tool for capturing the traffic based on the TCP header flags. (ACK, SYN, FIN, RST, PSH, URG). This problem statement gives us an opportunity to study the TCP flags in greater depth. It also enables an individual to understand and classify the packets based on TCP flags.

CHAPTER 3: OBJECTIVE

To design a tool that simulates Wireshark to capture and analyze packets based on the TCP header flags. (ACK, SYN, FIN, RST, PSH, URG).

CHAPTER 4: METHODOLOGY

Algorithm for the required project is as follows:

Step 1: Fetch TCP Packets:

For this project, we are using C to fetch the data packets. First, we open a RAW socket so that all the incoming and outgoing data packets can be captured, thus acting as a sniffer for the data packets.

Step 2: Logging the packets:

After capturing the data packets, we open a log file where all the data packets are to be stored. We use the function `printPacketData()` (Function used in code, see 5.1) to print the packets to the log file.

Step 3: Checking if the packets are TCP:

After capturing the packets, we need to check if the packets are of the TCP protocol or not. To identify the TCP protocol, the protocol number should be 6. After identifying the TCP packet, they are stored in an array. We are capturing only 100 packets for the experiment.

Step 4: Displaying the Packets according to User's Choice:

After the packets are stored, we create a menu-driven program to display the packets according to the user's choice. The packets can be displayed according to source IP, destination IP, source port, destination port, or the TCP Flags.

CHAPTER 5: IMPLEMENTATION DETAILS

5.1 CODE

```
/*
 * @brief A script to fetch TCP packets from the network and display their
 * details according to various flags.
 * @Executing instruction : cc prog.c && sudo ./a.out
 */

// importing header files
#include <stdio.h>           //For standard functions
#include <stdlib.h>          //malloc
#include <string.h>          //strlen
#include <netinet/ip_icmp.h> //Provides declarations for icmp header
#include <netinet/udp.h>     //Provides declarations for udp header
#include <netinet/tcp.h>     //Provides declarations for tcp header
#include <netinet/ip.h>      //Provides declarations for ip header
#include <netinet/if_ether.h> //For ETH_P_ALL
#include <net/ethernet.h>    //For 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 <netinet/in.h>
#include <errno.h>
#include <netdb.h>

/**
 * @brief Function prototype declarations
 */
void ProcessPacket(unsigned char *, int);
void print_ip_header(unsigned char *, int);
void print_tcp_packet(unsigned char *, int);
void PrintData(unsigned char *, int);
void printPacketData(int);
void printAllFlags();
void initialFlags();

/**
 * @brief Structure to store the details of the packet
 */
struct packetdata
```

```

{
    char ether_dhost[128];
    char ether_shost[128];
    short ether_type;
    int ip_version;
    char src_ip[128];
    char dst_ip[128];
    int ip_header_length;
    int ip_checksum;
    unsigned short tcp_source_port;
    unsigned short tcp_dest_port;
    unsigned int tcp_checksum;
    unsigned long int acknowledgement;
    unsigned int urg_flag;
    unsigned int ack_flag;
    unsigned int push_flag;
    unsigned int reset_flag;
    unsigned int syn_flag;
    unsigned int fin_flag;
};

/**
 * @brief Global variable to store the details of the packets
 */
struct packetdata pktdata[100];

/**
 * @brief File pointer to write the output of capturing TCP packets to a
file
 */
FILE *logfile;
struct sockaddr_in source, dest;
int tcp = 0, total = 0, i, j;
int ack,syn,fin,rst,urg,push;
int count;

/**
 * @brief function to print the number of various TCP packets on the basis
of TCP flags
 */
void printAllFlags()
{
    printf("Total number of packets: %d\n", count);
    printf("Total number of SYN packets: %d\n", syn);

```



```

    printf("Total number of ACK packets: %d\n", ack);
    printf("Total number of PSH packets: %d\n", push);
    printf("Total number of RST packets: %d\n", rst);
    printf("Total number of FIN packets: %d\n", fin);
    printf("Total number of URG packets: %d\n", urg);
}

/**
 * @brief function to initialize TCP flag counters
 *
 */
void initialFlags()
{
    count = 0;
    syn = 0;
    fin = 0;
    rst = 0;
    ack = 0;
    push = 0;
    urg = 0;
}

/**
 * @brief function to check if the packet is TCP or not, and print the TCP
 * packet to logfile
 *
 * @param buffer
 * @param size
 */
void ProcessPacket(unsigned char *buffer, int size)
{
    // Get the IP Header part of this packet , excluding the ethernet
    header
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ethhdr));
    /**
     * check if the packet is TCP or not
     * TCP port 6 uses the Transmission Control Protocol.
     * TCP guarantees delivery of data packets on port 6 in the same order
     in which they were sent.
     */

    if (iph->protocol == 6)
    {
        printf("TCP Packet %d captured!\n", (tcp + 1));
        print_tcp_packet(buffer, size);
    }
}

```

```

        ++tcp;
        ++total;
    }
}

/**
 * @brief prints packet data to terminal
 *
 * @param i
 */
void printPacketData(int i)
{
    printf("\n");
    printf("PACKET %d\n", i);
    printf(" |-Source IP : %s\n", pktdata[i].src_ip);
    printf(" |-Destination IP : %s\n", pktdata[i].dst_ip);
    printf(" |-Source Port : %d\n", pktdata[i].tcp_source_port);
    printf(" |-Destination Port : %d\n", pktdata[i].tcp_dest_port);
    printf(" |-TCP Flags: \n");
    if (pktdata[i].syn_flag == 1)
    {
        syn++;
        printf(" |-SYN\n");
    }
    if (pktdata[i].ack_flag == 1)
    {
        ack++;
        printf(" |-ACK\n");
    }
    if (pktdata[i].push_flag == 1)
    {
        push++;
        printf(" |-PSH\n");
    }
    if (pktdata[i].reset_flag == 1)
    {
        rst++;
        printf(" |-RST\n");
    }
    if (pktdata[i].fin_flag == 1)
    {
        fin++;
        printf(" |-FIN\n");
    }
    if (pktdata[i].urg_flag == 1)

```

```

    {
        urg++;
        printf(" URG\n");
    }
    printf("\n");
}

/**
 * @brief function to print the ethernet header of TCP packet to logfile
 * and store the ethernet header data in pktdata
 *
 * @param Buffer
 * @param Size
 */
void print_ethernet_header(unsigned char *Buffer, int Size)
{
    struct ethhdr *eth = (struct ethhdr *)Buffer;
    char dest_addr[64];
    char src_addr[64];

    /**
     * A media access control address (MAC address) is a unique identifier
     * assigned to a network interface controller (NIC)
     * for use as a network address in communications within a network
     * segment.
     */

    // formatting the destination address of the packet to string
    sprintf(dest_addr, "%02X:%02X:%02X:%02X:%02X:%02X", eth->h_dest[0],
eth->h_dest[1], eth->h_dest[2], eth->h_dest[3], eth->h_dest[4], eth-
>h_dest[5]);
    strcpy(pktdata[tcp].ether_dhost, dest_addr);

    // formatting the source address of the packet to string
    sprintf(src_addr, "%02X:%02X:%02X:%02X:%02X:%02X", eth->h_source[0],
eth->h_source[1], eth->h_source[2], eth->h_source[3], eth->h_source[4],
eth->h_source[5]);
    strcpy(pktdata[tcp].ether_shost, src_addr);

    // copying the protocol type of the packet to pktdata
    pktdata[tcp].ether_type = eth->h_proto;
    fprintf(logfile, "\n");
    fprintf(logfile, "Ethernet Header\n");
}

```

```

// prints destination address, source address and protocol type of the
packet to logfile
    fprintf(logfile, " |-Destination Address : %s \n", dest_addr);
    fprintf(logfile, " |-Source Address : %s \n", src_addr);
    fprintf(logfile, " |-Protocol : %u \n", (unsigned short)eth->h_proto);
}

/**
 * @brief function to print the IP header of TCP packet to logfile and
store the IP header data in pktdata
 *
 * @param Buffer
 * @param Size
 */
void print_ip_header(unsigned char *Buffer, int Size)
{
    // Print ethernet header to logfile
    print_ethernet_header(Buffer, Size);
    unsigned short iphdrlen;

    // initialise the iphdr structure with the IP header information
    struct iphdr *iph = (struct iphdr *) (Buffer + sizeof(struct ethhdr));
    iphdrlen = iph->ihl * 4;

    // set source and destination IP address
    memset(&source, 0, sizeof(source));
    source.sin_addr.s_addr = iph->saddr;
    memset(&dest, 0, sizeof(dest));
    dest.sin_addr.s_addr = iph->daddr;

    // print the IP header to logfile
    fprintf(logfile, "\n");
    fprintf(logfile, "IP Header\n");
    fprintf(logfile, " |-IP Version : %d\n", (unsigned int)iph->version);
    fprintf(logfile, " |-IP Header Length : %d DWORDS or %d
Bytes\n", (unsigned int)iph->ihl, ((unsigned int)(iph->ihl)) * 4);
    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));
    fprintf(logfile, " |-Source IP : %s\n", inet_ntoa(source.sin_addr));
}

```

```

        fprintf(logfile, " |-Destination IP : %s\n",
inet_ntoa(dest.sin_addr));

// copy source and destination address to pktdata
strcpy(pktdata[tcp].src_ip, inet_ntoa(source.sin_addr));
strcpy(pktdata[tcp].dst_ip, inet_ntoa(dest.sin_addr));
}

/**
 * @brief function to print the TCP header of TCP packet to logfile and
store the TCP header data in pktdata
 *
 * @param Buffer
 * @param Size
 */
void print_tcp_packet(unsigned char *Buffer, int Size)
{
    unsigned short iphdrlen;

// initialise the iphdr structure with the IP header information
    struct iphdr *iph = (struct iphdr *) (Buffer + sizeof(struct ethhdr));
    iphdrlen = iph->ihl * 4;

// initialise the tcphdr structure with the TCP header information
    struct tcphdr *tcph = (struct tcphdr *) (Buffer + iphdrlen +
sizeof(struct ethhdr));
    int header_size = sizeof(struct ethhdr) + iphdrlen + tcph->doff *
4; // doff - data offset, tcp header is doff*4 bytes

// print the TCP header to logfile
    fprintf(logfile, "\n\n*****TCP
Packet#%d*****\n", (tcp + 1));
    print_ip_header(Buffer, Size);
    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");
    fprintf(logfile, " DATA Dump");
    fprintf(logfile, "\n");

// print header data to logfile
    fprintf(logfile, "IP Header\n");
    PrintData(Buffer, iphdrlen);
    fprintf(logfile, "TCP Header\n");
    PrintData(Buffer + iphdrlen, tcph->doff * 4);
    fprintf(logfile, "Data Payload\n");
    PrintData(Buffer + header_size, Size - header_size);
    fprintf(logfile, "\n#####");
    fprintf(logfile, "#####");

// copy source, destination ports, and flags to pktdata
    pktdata[tcp].tcp_source_port = ntohs(tcph->source);
    pktdata[tcp].tcp_dest_port = ntohs(tcph->dest);
    pktdata[tcp].push_flag = tcph->psh;
    pktdata[tcp].ack_flag = tcph->ack;
    pktdata[tcp].syn_flag = tcph->syn;
    pktdata[tcp].fin_flag = tcph->fin;
    pktdata[tcp].reset_flag = tcph->rst;
    pktdata[tcp].urg_flag = tcph->urg;
    pktdata[tcp].acknowledgement = tcph->ack_seq;
}

/**
 * @brief function to format data in the logfile
 *
 * @param data
 * @param Size
 */
void PrintData(unsigned char *data, int Size)
{
    int i, j;
    for (i = 0; i < Size; i++)
    {

```

```

        if (i != 0 && i % 16 == 0) // if one line of hex printing is
complete...
        {
            fprintf(logfile, " ");
            for (j = i - 16; j < i; j++)
            {
                if (data[j] >= 32 && data[j] <= 128)
                    fprintf(logfile, "%c", (unsigned char)data[j]); // if
its a number or alphabet
                else
                    fprintf(logfile, "."); // otherwise print a dot
            }
            fprintf(logfile, "\n");
        }
        if (i % 16 == 0)
            fprintf(logfile, " ");
        fprintf(logfile, " %02X", (unsigned int)data[i]);
        if (i == Size - 1) // print the last spaces
        {
            for (j = 0; j < 15 - i % 16; j++)
            {
                fprintf(logfile, " "); // extra spaces
            }
            fprintf(logfile, " ");
            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, ".");
                }
            }
            fprintf(logfile, "\n");
        }
    }
}

/**
 * @brief Main function to execute the script
 * @return int
 */
int main()
{

```

```

/**
 * @brief Declaring the local variables
 *
 */
int saddr_size, data_size;
struct sockaddr saddr; // address of the source
int flag = 0; // variable to check if the TCP packets have been
fetched or not
unsigned char *buffer = (unsigned char *)malloc(65536);
// opening the log file
logfile = fopen("log.log", "w");
if (logfile == NULL)
{
    printf("Unable to create log.log file.");
}
printf("Starting...\n");

// creating a raw socket that allows us to capture raw packets from
the network.
int sock_raw = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
/**
 * 1) AF_PACKET means "send packets as is, without any transport
protocol".
 *
 * 2) Raw Sockets provide users access to the underlying communication
protocols, which support socket abstractions.
 * They are not intended for the general user; they have been provided
mainly for those interested in developing new communication protocols,
 * or for gaining access to some of the more cryptic facilities of an
existing protocol.
 *
 * 3) 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.
 */
if (sock_raw < 0)
{
    // Print the error with proper message
    perror("Socket Error");
    return 1;
}
while (flag == 0)
{
    saddr_size = sizeof(saddr);

```



```

        // Receive a packet
        data_size = recvfrom(sock_raw, buffer, 65536, 0, &saddr,
(socklen_t*)&saddr_size);
        if (data_size < 0)
        {
            printf("Recvfrom error , failed to get packets\n");
            return 1;
        }
        // Now process the packet
        ProcessPacket(buffer, data_size);
        if (tcp == 100)
            flag = 1;
    }
    // close the socket
    close(sock_raw);

    // menu driven program to print the data according to the user's choice
    int choice;
    do
    {
        printf("\nFilters: \n");
        printf("1. Source IP\n");
        printf("2. Destination IP\n");
        printf("3. Source Port\n");
        printf("4. Destination Port\n");
        printf("5. TCP Flags\n");
        printf("6. All\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                /**
                 * @brief print data based on source IP
                 */
                printf("Enter the source IP: ");
                char src_ip[128];
                initialFlags();
                scanf("%s", src_ip);
                for (i = 0; i < 100; i++)
                {
                    if (strcmp(src_ip, pktdata[i].src_ip) == 0)
                    {
                        count++;
                        printPacketData(i);
                    }
                }
            }
        }
    } while (choice != 7);
}

```

```

    }
}
printAllFlags();
if (count == 0)
{
    printf("No such IP found.\n");
}
initialFlags();
break;

case 2:
/**
 * @brief print data based on destination IP
 */
printf("Enter the destination IP: ");
char dst_ip[128];
initialFlags();
scanf("%s", dst_ip);
for (i = 0; i < 100; i++)
{
    if (strcmp(dst_ip, pktdata[i].dst_ip) == 0)
    {
        count++;
        printPacketData(i);
    }
}
printAllFlags();
if (count == 0)
{
    printf("No such IP found.\n");
}
initialFlags();
break;

case 3:
/**
 * @brief print data based on source port
 */
printf("Enter the source port: ");
int src_port;
initialFlags();
scanf("%d", &src_port);
for (i = 0; i < 100; i++)
{
    if (src_port == pktdata[i].tcp_source_port)
    {

```

```

        count++;
        printPacketData(i);
    }
}
printAllFlags();
if (count == 0)
{
    printf("No such port found.\n");
}
initialFlags();
break;

case 4:
/**
 * @brief print data based on destination port
 */
printf("Enter the destination port: ");
int dst_port;
initialFlags();
scanf("%d", &dst_port);
for (i = 0; i < 100; i++)
{
    if (dst_port == pktdata[i].tcp_dest_port)
    {
        count++;
        printPacketData(i);
    }
}
printAllFlags();
if (count == 0)
{
    printf("No such port found.\n");
}
initialFlags();
break;

case 5:
// print packets based on flag entered by user
printf("Enter the flag: [ACK, SYN, FIN, RST, PSH, URG]\n");
char flag[10];
scanf("%s", flag);
count = 0;
if (strcasecmp(flag, "ACK") == 0)
{
    for (i = 0; i < 100; i++)
    {

```

```

        if (pktdata[i].ack_flag == 1)
        {
            count++;
            printPacketData(i);
        }
    }
    printf("Total number of packets: %d\n", count);
}
else if (strcasecmp(flag, "SYN") == 0)
{
    for (i = 0; i < 100; i++)
    {
        if (pktdata[i].syn_flag == 1)
        {
            count++;
            printPacketData(i);
        }
    }
    printf("Total number of packets: %d\n", count);
}
else if (strcasecmp(flag, "FIN") == 0)
{
    for (i = 0; i < 100; i++)
    {
        if (pktdata[i].fin_flag == 1)
        {
            count++;
            printPacketData(i);
        }
    }
    printf("Total number of packets: %d\n", count);
}
//In TCP, packets with the "Reset" (RST) flag are sent to
abort a connection.
else if (strcasecmp(flag, "RST") == 0)
{
    for (i = 0; i < 100; i++)
    {
        if (pktdata[i].reset_flag == 1)
        {
            count++;
            printPacketData(i);
        }
    }
    printf("Total number of packets: %d\n", count);
}

```

```

else if (strcasecmp(flag, "PSH") == 0)
{
    for (i = 0; i < 100; i++)
    {
        if (pktdata[i].push_flag == 1)
        {
            count++;
            printPacketData(i);
        }
    }
    printf("Total number of packets: %d\n", count);
}
else if (strcasecmp(flag, "URG") == 0)
{
    for (i = 0; i < 100; i++)
    {
        if (pktdata[i].urg_flag == 1)
        {
            count++;
            printPacketData(i);
        }
    }
    printf("Total number of packets: %d\n", count);
}
break;

case 6:
/**
 * @brief print all packets
 */
initialFlags();
for (i = 0; i < 100; i++)
{
    count++;
    printPacketData(i);
}
printAllFlags();
initialFlags();
break;

case 7:
// exit condition
break;

default:
printf("Invalid choice\n");

```

```

        break;
    }
}
while (choice != 7);
printf("Finished\n");
return 0;
}

```

5.2 OUTPUT

5.2.1 Log File

NOTE: This is a sample of the log file. There are hundred such packets in the log file.

*****TCP Packet#1*****

Ethernet Header

```

|-Destination Address : 00:50:56:FB:C4:6F
|-Source Address : 00:0C:29:7C:0D:8B
|-Protocol : 8

```

IP Header

```

|-IP Version : 4
|-IP Header Length : 5 DWORDS or 20 Bytes
|-Type Of Service : 0
|-IP Total Length : 60 Bytes(Size of Packet)
|-Identification : 20748
|-TTL : 64
|-Protocol : 6
|-Checksum : 5834
|-Source IP : 192.168.18.128
|-Destination IP : 34.107.221.82

```

TCP Header

```

|-Source Port : 40114
|-Destination Port : 80
|-Sequence Number : 2660478223
|-Acknowledge Number : 0
|-Header Length : 10 DWORDS or 40 BYTES
|-Urgent Flag : 0
|-Acknowledgement Flag : 0
|-Push Flag : 0
|-Reset Flag : 0
|-Synchronise Flag : 1
|-Finish Flag : 0

```

| -Window : 64240
| -Checksum : 54036
| -Urgent Pointer : 0

DATA Dump

IP Header

00 50 56 FB C4 6F 00 0C 29 7C 0D 8B 08 00 45 00 .PV..o..)|....E.
00 3C 51 0C .<Q.

TCP Header

40 00 40 06 16 CA C0 A8 12 80 22 6B DD 52 9C B2 @.@.....€"k.R..
00 50 9E 93 AD 0F 00 00 00 00 A0 02 FA F0 D3 14 .P.....
00 00 02 04 05 B4 04 02

Data Payload

#####

*****TCP Packet#2*****

Ethernet Header

| -Destination Address : 00:50:56:FB:C4:6F
| -Source Address : 00:0C:29:7C:0D:8B
| -Protocol : 8

IP Header

| -IP Version : 4
| -IP Header Length : 5 DWORDS or 20 Bytes
| -Type Of Service : 0
| -IP Total Length : 60 Bytes(Size of Packet)
| -Identification : 59067
| -TTL : 64
| -Protocol : 6
| -Checksum : 28787
| -Source IP : 192.168.18.128
| -Destination IP : 34.117.237.239

TCP Header

| -Source Port : 48854
| -Destination Port : 443
| -Sequence Number : 1726396753
| -Acknowledge Number : 0
| -Header Length : 10 DWORDS or 40 BYTES
| -Urgent Flag : 0
| -Acknowledgement Flag : 0
| -Push Flag : 0
| -Reset Flag : 0
| -Synchronise Flag : 1

```
| -Finish Flag : 0
| -Window : 64240
| -Checksum : 58299
| -Urgent Pointer : 0
```

DATA Dump

IP Header

```
00 50 56 FB C4 6F 00 0C 29 7C 0D 8B 08 00 45 00 .PV..o..)|....E.
00 3C E6 BB .<..
```

TCP Header

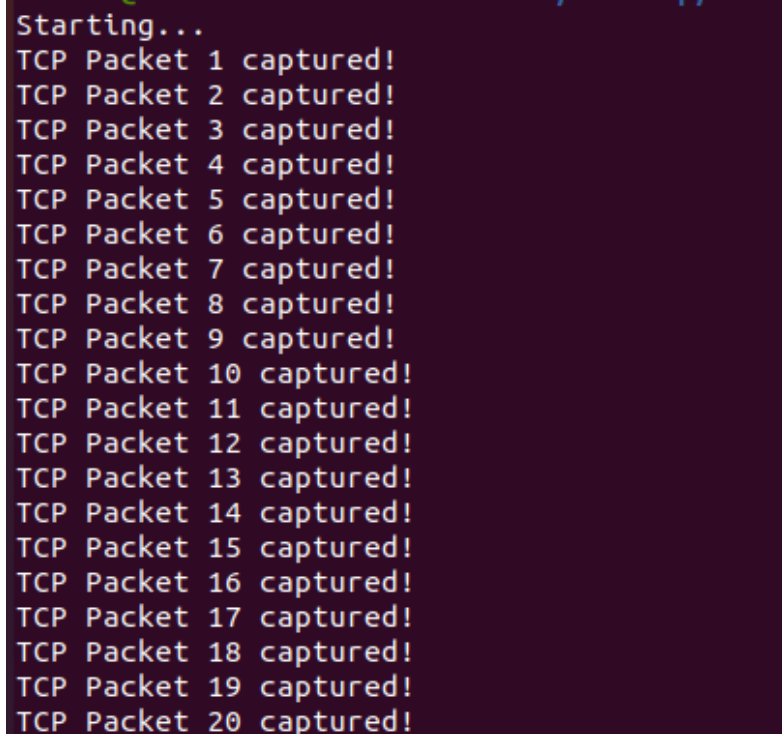
```
40 00 40 06 70 73 C0 A8 12 80 22 75 ED EF BE D6 @.@.ps...€"u....
01 BB 66 E6 B9 51 00 00 00 00 A0 02 FA F0 E3 BB ..f..Q.....
00 00 02 04 05 B4 04 02 .....
```

Data Payload

```
#####
```

5.2.2 Terminal

NOTE: Execution command: `cc prog.c && sudo ./a.out`



```
Starting...
TCP Packet 1 captured!
TCP Packet 2 captured!
TCP Packet 3 captured!
TCP Packet 4 captured!
TCP Packet 5 captured!
TCP Packet 6 captured!
TCP Packet 7 captured!
TCP Packet 8 captured!
TCP Packet 9 captured!
TCP Packet 10 captured!
TCP Packet 11 captured!
TCP Packet 12 captured!
TCP Packet 13 captured!
TCP Packet 14 captured!
TCP Packet 15 captured!
TCP Packet 16 captured!
TCP Packet 17 captured!
TCP Packet 18 captured!
TCP Packet 19 captured!
TCP Packet 20 captured!
```

Fig. 5.1


```
PACKET 98
|-Source IP : 192.168.18.128
|-Destination IP : 10.93.0.2
|-Source Port : 49520
|-Destination Port : 443
|-TCP Flags:
|-ACK
|-PSH

PACKET 99
|-Source IP : 10.93.0.2
|-Destination IP : 192.168.18.128
|-Source Port : 443
|-Destination Port : 49520
|-TCP Flags:
|-ACK

Total number of packets: 100
Total number of SYN packets: 14
Total number of ACK packets: 93
Total number of PSH packets: 32
Total number of RST packets: 3
Total number of FIN packets: 13
Total number of URG packets: 0

Filters:
1. Source IP
2. Destination IP
3. Source Port
4. Destination Port
5. TCP Flags
6. All
7. Exit
Enter your choice: █
```

Fig. 5.2

CHAPTER 6: CONTRIBUTION SUMMARY

- I was successfully able to capture TCP packets in a log file using a raw socket.
- After capturing, I was able to distinguish the various TCP packets based on TCP flags (ACK, SYN, FIN, RST, PSH, URG) and find the frequency of each.
- On executing this code for various sets of input it was found that the RST and FIN are captured infrequently whereas the URG flag is found rarely.
- The overall objective of this project was successfully achieved.

CHAPTER 7: REFERNCES

7.1 BOOK

- James F. Kurose, Keith W. Ross, “Transport Layer”, in Computer Networking: A Top-down Approach, 6th ed. New Jersey, (only U.S. State), The United States of America, Pearson Education, Inc., 2013, ch. 3, sec. 3.5, pp. 230-258.

7.2 WEBSITE

- The Regents of the University of California. “The GNU C Library.” sites.uclouvain.be. <https://sites.uclouvain.be/SystInfo/usr/include/netinet/ip.h.html> (accessed Oct. 22, 2022)
- IEEE Periodicals. “IEEE REFERENCE GUIDE” [ieeauthorcenter.ieee.org. https://ieeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf](https://ieeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf) (accessed Oct. 22, 2022)
- GeeksforGeeks. “TCP Flags. ” [geeksforgeeks.org. https://www.geeksforgeeks.org/tcp-flags/](https://www.geeksforgeeks.org/tcp-flags/) (accessed Oct. 22, 2022)
- Nevil Brownlee. “ Transport Layer Decodes.” [cs.auckland.ac.nz. https://www.cs.auckland.ac.nz/~nevil/pypy-libtrace/TCP.html](https://www.cs.auckland.ac.nz/~nevil/pypy-libtrace/TCP.html) (accessed Oct. 22, 2022)