Experiment: 5

Packet Capturing and Filtering

5.1 AIM: Program to implement packet sniffer that will capture the packets and analyze the packet header, information.

5.2 DESCRIPTION: Packet capturing is a part of packet sniffers. Packet sniffers are used to capture or sniff the network packets. The captured packets may be used to detect network traffic flow or to analyze the packet for various purposes. The sniffers are commonly used in network security. Wireshark is a very common packet sniffer/protocol analyzer. These packet sniffers can be written in C/C++, Java, Python, and PHP etc. we can also use ***libcap, pcap,*** APIs in Linux, and **Winpcap** API in Windows. The following section gives the information about various types of packets and their structures.

Ethernet Frame:

Figure 5.1 shows the header format of 802.3(Ethernet) frame.

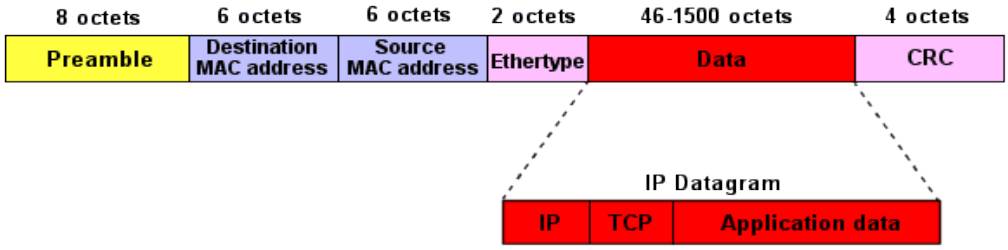


Figure 5.1: Ethernet frame header format

The structure of Ethernet frame format [1]:

```
#define ETH_P_802_3     0x0001          /* Dummy type for 802.3 frames  */
#define ETH_P_AX25      0x0002          /* Dummy protocol id for AX.25  */
#define ETH_P_ALL       0x0003          /* Every packet (be careful!!!) */
#define ETH_P_802_2     0x0004          /* 802.2 frames                 */
#define ETH_P_SNAP      0x0005          /* Internal only                */
#define ETH_P_DDCMP     0x0006          /* DEC DDCMP: Internal only     */
#define ETH_P_WAN_PPP   0x0007          /* Dummy type for WAN PPP frames*/
#define ETH_P_PPP_MP    0x0008          /* Dummy type for PPP MP frames */
#define ETH_P_LOCALTALK 0x0009          /* Localtalk pseudo type        */
#define ETH_P_PPPTALK   0x0010          /* Dummy type for Atalk over PPP*/
#define ETH_P_TR_802_2  0x0011          /* 802.2 frames                 */
#define ETH_P_MOBITEX   0x0015          /* Mobitex (kaz@cafe.net)       */
#define ETH_P_CONTROL   0x0016          /* Card specific control frames */
#define ETH_P_IRDA      0x0017          /* Linux-IrDA                   */
#define ETH_P_ECONET    0x0018          /* Acorn Econet                 */
#define ETH_P_HDLC      0x0019          /* HDLC frames                  */
#define ETH_P_ARCNET    0x001A          /* 1A for ArcNet :-)            */

/*
 *      This is an Ethernet frame header.
 */

struct ethhdr {
        unsigned char   h_dest[ETH_ALEN];       /* destination eth addr */
        unsigned char   h_source[ETH_ALEN];     /* source ether addr    */
        unsigned short  h_proto;                /* packet type ID field */
} __attribute__((packed));

#endif  /* _LINUX_IF_ETHER_H */
```

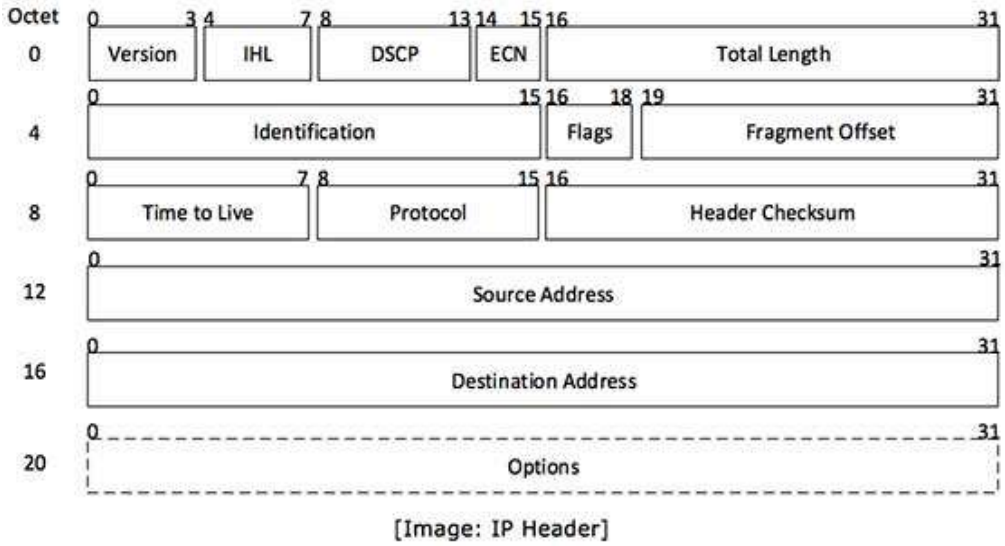


Figure 5.3: IPv4 Packet header format

16-bit	32-bit
Source Port	Destination Port
Sequence Number	
Acknowledgement Number (ACK)	
Offset Reserved | U | A | P | R | S | F	Window
Checksum	Urgent Pointer
Options and Padding	

Figure 5.4: TCP Header format

The IP header structure:

```
struct iphdr {
#if defined(__LITTLE_ENDIAN_BITFIELD)
        __u8    ihl:4,
                version:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
        __u8    version:4,
                ihl:4;
#else
#error  "Please fix <asm/byteorder.h>"
#endif
        __u8    tos;
        __u16   tot_len;
        __u16   id;
        __u16   frag_off;
        __u8    ttl;
        __u8    protocol;
        __u16   check;
        __u32   saddr;
        __u32   daddr;
        /*The options start here. */
};
```

C-Structure of TCP header:

```
struct tcphdr {
        __u16   source;
        __u16   dest;
        __u32   seq;
        __u32   ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
        __u16   res1:4,
                doff:4,
                fin:1,
                syn:1,
                rst:1,
                psh:1,
                ack:1,
                urg:1,
                ece:1,
                cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
        __u16   doff:4,
                res1:4,
                cwr:1,
                ece:1,
                urg:1,
                ack:1,
                psh:1,
                rst:1,
                syn:1,
                fin:1;
#else
#error  "Adjust your <asm/byteorder.h> defines"
#endif
        __u16   window;
        __u16   check;
        __u16   urg_ptr;
};
```

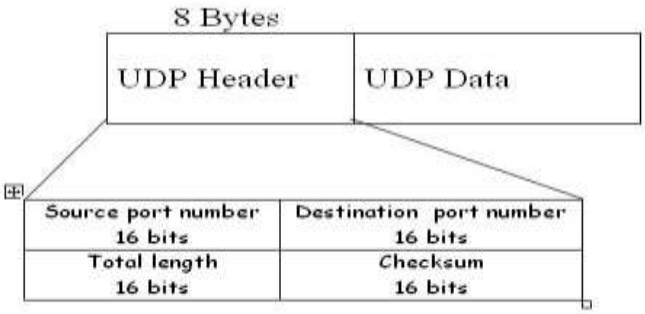


Figure 5.5: UDP header format

C-Structure of UDP header:

```
struct udphdr {
        u_short uh_sport;               /* source port */
        u_short uh_dport;               /* destination port */
        short   uh_ulen;                /* udp length */
        u_short uh_sum;                 /* udp checksum */
};
```

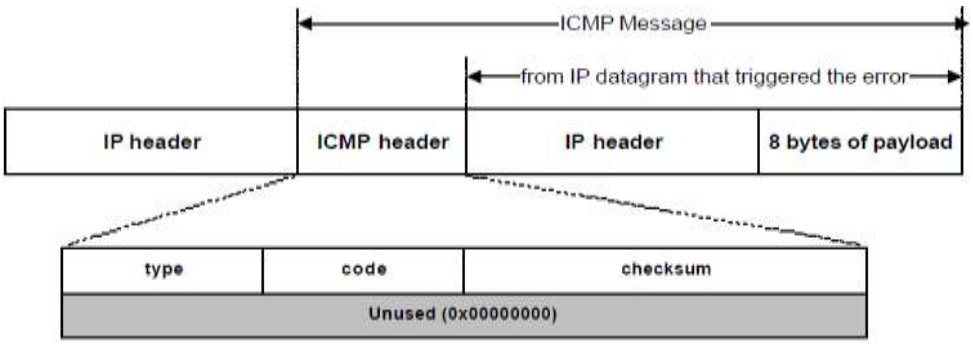
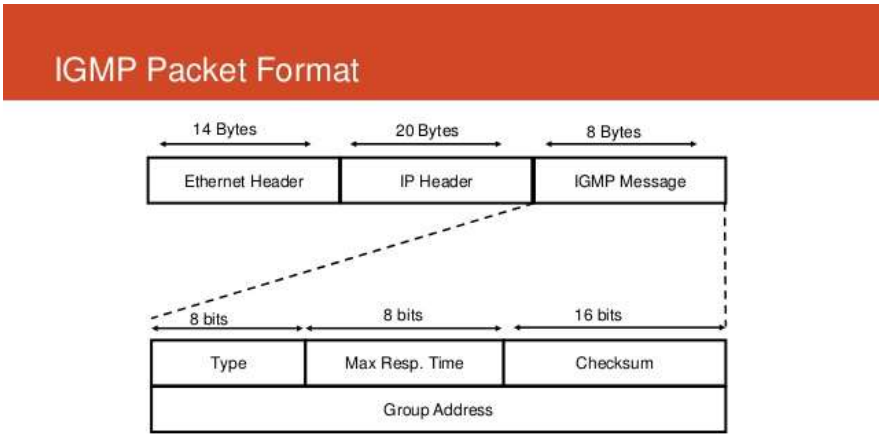


Figure 5.6: ICMP header format

C structure of ICMP header[3]:

```
struct icmphdr
{
  u_int8_t type;                /* message type */
  u_int8_t code;                /* type sub-code */
  u_int16_t checksum;
  union
  {
    struct
    {
      u_int16_t id;
      u_int16_t sequence;
    } echo;                     /* echo datagram */
    u_int32_t   gateway;        /* gateway address */
    struct
    {
      u_int16_t __unused;
      u_int16_t mtu;
    } frag;                     /* path mtu discovery */
  } un;
};

#define ICMP_ECHOREPLY          0       /* Echo Reply                   */
#define ICMP_DEST_UNREACH       3       /* Destination Unreachable      */
#define ICMP_SOURCE_QUENCH      4       /* Source Quench                */
#define ICMP_REDIRECT           5       /* Redirect (change route)      */
#define ICMP_ECHO               8       /* Echo Request                 */
#define ICMP_TIME_EXCEEDED      11      /* Time Exceeded                */
#define ICMP_PARAMETERPROB      12      /* Parameter Problem            */
#define ICMP_TIMESTAMP          13      /* Timestamp Request            */
#define ICMP_TIMESTAMPREPLY     14      /* Timestamp Reply              */
#define ICMP_INFO_REQUEST       15      /* Information Request          */
#define ICMP_INFO_REPLY         16      /* Information Reply            */
#define ICMP_ADDRESS            17      /* Address Mask Request         */
#define ICMP_ADDRESSREPLY       18      /* Address Mask Reply           */
#define NR_ICMP_TYPES           18
```



5.3: ALGORITHMS

1. Start
2. Create a raw socket
3. Receive a packet from the network
4. Analyze the packet to check whether it is TCP/UDP/IP/ICMP etc.
5. Repeat steps 3 and 4 until the required number of packets captured
6. Display the required information
7. Stop

5.4: IMPLEMENTATION

A simple Python script for capturing packet:

```
#Packet sniffer in python
#For Linux
import socket

#create an INET, raw socket
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
# receive a packet
while True:
  print s.recvfrom(65565)
```

to sun the above script use the following command:

```
$ sudo python sniffer.py
```

Packet Filtering[4]:

```
#Packet sniffer in python for Linux
#Sniffs only incoming TCP packet
import socket, sys
from struct import *

#create an INET, STREAMing socket
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)
```

```
except socket.error , msg:
    print 'Socket could not be created. Error Code : ' + str(msg[0]) + ' Message ' +
msg[1]
    sys.exit()

# receive a packet
while True:
    packet = s.recvfrom(65565)
    #packet string from tuple
    packet = packet[0]
    #take first 20 characters for the ip header
    ip_header = packet[0:20]

    #now unpack them :)
    iph = unpack('!BBHHHBBH4s4s' , ip_header)

    version_ihl = iph[0]
    version = version_ihl >> 4
    ihl = version_ihl & 0xF

    iph_length = ihl * 4

    ttl = iph[5]
    protocol = iph[6]
    s_addr = socket.inet_ntoa(iph[8]);
    d_addr = socket.inet_ntoa(iph[9]);

    print 'Version : ' + str(version) + ' IP Header Length : ' + str(ihl) + ' TTL : ' +
str(ttl) + ' Protocol : ' + str(protocol) + ' Source Address : ' + str(s_addr) + '
Destination Address : ' + str(d_addr)

    tcp_header = packet[iph_length:iph_length+20]

    #now unpack them :)
    tcph = unpack('!HHLLBBHHH' , tcp_header)

    source_port = tcph[0]
    dest port = tcph[1]
    sequence = tcph[2]
    acknowledgement = tcph[3]
    doff_reserved = tcph[4]
    tcph_length = doff_reserved >> 4

    print 'Source Port : ' + str(source_port) + ' Dest Port : ' + str(dest_port) + '
Sequence Number : ' + str(sequence) + ' Acknowledgement : ' + str(acknowledgement) + '
TCP header length : ' + str(tcph_length)

    h_size = iph_length + tcph_length * 4
    data_size = len(packet) - h_size

    #get data from the packet
    data = packet[h_size:]

    print 'Data : ' + data
    print
```

C code for packet sniffing:

```
#include<netinet/in.h>
#include<errno.h>
#include<netdb.h>
#include<stdio.h>              //For standard things
#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>

void ProcessPacket(unsigned char* , int);             //Process the packet
void print_ip_header(unsigned char* , int);           //Prints IP header information
void print_tcp_packet(unsigned char * , int );        //Prints TCP header information
void print_udp_packet(unsigned char * , int );        //Prints UDP header information
void print_icmp_packet(unsigned char* , int );        //Prints ICMP header information
void PrintData (unsigned char* , int);                //Prints data in the packet

FILE *logfile;
struct sockaddr_in source,dest;
int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;

int main()
{
    int saddr_size , data_size;       struct sockaddr saddr;

    unsigned char *buffer = (unsigned char *) malloc(65536); //Its Big!

    logfile=fopen("log.txt","w");
    if(logfile==NULL)
    {
        perror("nable to create log.txt file.");
        exit(-1);
    }
    printf("Starting...\n");

    int sock_raw = socket( AF_PACKET , SOCK_RAW , htons(ETH_P_ALL)) ;
    //setsockopt(sock_raw , SOL_SOCKET , SO_BINDTODEVICE , "eth0" , strlen("eth0")+ 1 );

    if(sock_raw < 0)
    {
        perror("Socket Error");         //Print the error with proper message
        exit(-1);
    }
    while(1)
    {
        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);           //process the packet and writes to log file
    }
    close(sock_raw);
    printf("Finished");
    return 0;
}                   //End of main

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));
    ++total;
    switch (iph->protocol) //Check the Protocol and do accordingly...
    {
        case 1:                                      //ICMP Protocol
            ++icmp;
            print_icmp_packet( buffer , size);
            break;

        case 2:                                      //IGMP Protocol
            ++igmp;
            break;

        case 6:                                       //TCP Protocol
            ++tcp;
            print_tcp_packet(buffer , size);
            break;

        case 17:                                     //UDP Protocol
            ++udp;
            print_udp_packet(buffer , size);
            break;

        default:                                     //Some Other Protocol like ARP etc.
            ++others;
            break;
    }
    printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d   Total : %d\r", tcp , udp ,
icmp , igmp , others , total);
}

void print_ethernet_header(unsigned char* Buffer, int Size)
{
    struct ethhdr *eth = (struct ethhdr *)Buffer;

    fprintf(logfile , "\n");
    fprintf(logfile , "Ethernet Header\n");
    fprintf(logfile , "   |-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->h_dest[0]
, eth->h_dest[1] , eth->h_dest[2] , eth->h_dest[3] , eth->h_dest[4] , eth->h_dest[5] );
    fprintf(logfile , "   |-Source Address      : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>h_source[0] , eth->h_source[1] , eth->h_source[2] , eth->h_source[3] , eth->h_source[4] , eth-
>h_source[5] );
    fprintf(logfile , "   |-Protocol            : %u \n",(unsigned short)eth->h_proto);
}

void print_ip_header(unsigned char* Buffer, int Size)
{
    print_ethernet_header(Buffer , Size);

    unsigned short iphdrlen;
```

```
    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr) );
    iphdrlen =iph->ihl*4;

    memset(&source, 0, sizeof(source));
    source.sin_addr.s_addr = iph->saddr;

    memset(&dest, 0, sizeof(dest));
    dest.sin_addr.s_addr = iph->daddr;

    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 , "   |-Reserved ZERO Field   : %d\n",(unsigned int)iphdr->ip_reserved_zero);
    //fprintf(logfile , "   |-Dont Fragment Field   : %d\n",(unsigned int)iphdr->ip_dont_fragment);
    //fprintf(logfile , "   |-More Fragment Field   : %d\n",(unsigned int)iphdr->ip_more_fragment);
    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));
}

void print_tcp_packet(unsigned char* Buffer, int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)( Buffer  + sizeof(struct ethhdr) );
    iphdrlen = iph->ihl*4;
    struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen + sizeof(struct ethhdr));
    int header_size =  sizeof(struct ethhdr) + iphdrlen + tcph->doff*4;
    fprintf(logfile , "\n\n**********************TCP Packet*************************\n");
    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 , "   |-CWR Flag : %d\n",(unsigned int)tcph->cwr);
    //fprintf(logfile , "   |-ECN Flag : %d\n",(unsigned int)tcph->ece);
    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");

    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###########################################################");
}

void print_udp_packet(unsigned char *Buffer , int Size)
{
    unsigned short iphdrlen;
    struct iphdr *iph = (struct iphdr *)(Buffer +  sizeof(struct ethhdr));
    iphdrlen = iph->ihl*4;
    struct udphdr *udph = (struct udphdr*)(Buffer + iphdrlen  + sizeof(struct ethhdr));
    int header_size =  sizeof(struct ethhdr) + iphdrlen + sizeof udph;

    fprintf(logfile , "\n\n***********************UDP Packet*************************\n");

    print_ip_header(Buffer,Size);
    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");
    fprintf(logfile , "IP Header\n");
    PrintData(Buffer , iphdrlen);
    fprintf(logfile , "UDP Header\n");
    PrintData(Buffer+iphdrlen , sizeof udph);
    fprintf(logfile , "Data Payload\n");

              //Move the pointer ahead and reduce the size of string
    PrintData(Buffer + header_size , Size - header_size);
    fprintf(logfile , "\n###########################################################");
}

void print_icmp_packet(unsigned char* Buffer , int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr));
    iphdrlen = iph->ihl * 4;
    struct icmphdr *icmph = (struct icmphdr *)(Buffer + iphdrlen  + sizeof(struct ethhdr));
    int header_size =  sizeof(struct ethhdr) + iphdrlen + sizeof icmph;
    fprintf(logfile , "\n\n***********************ICMP Packet*************************\n");
    print_ip_header(Buffer , Size);
    fprintf(logfile , "\n");
    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 , "   |-ID       : %d\n",ntohs(icmph->id));
```

```
    //fprintf(logfile , "   |-Sequence : %d\n",ntohs(icmph->sequence));
    fprintf(logfile , "\n");

    fprintf(logfile , "IP Header\n");
    PrintData(Buffer,iphdrlen);

    fprintf(logfile , "UDP Header\n");
    PrintData(Buffer + iphdrlen , sizeof icmph);

    fprintf(logfile , "Data Payload\n");

    //Move the pointer ahead and reduce the size of string
    PrintData(Buffer + header_size , (Size - header_size) );

    fprintf(logfile , "\n###########################################################");
}

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" );
        }
    }
}
```

TASKS to be performed

Read and understand the header formats/structures of Ethernet, TCP, UDP, IP and ICMP. Run the above programs and interpret the results. After understanding the results and structures implement the following tasks.

1. Develop a program that capture 10 Ethernet frames from the network and display the source and destination MAC addresses along with the data in each of the frame?
2. Develop a program that capture n packets and displays the complete packet details including the headers?
3. Write a program to extract and print the TCP segment information including the header
4. Write a program to extract and print the UDP segment information?
5. Write a program that captures 100 packets and display the count of ICMP, IGMP, TCP, UDP, and IP packets?

5.6 CONCLUSIONS

1. Implement all the task and write your conclusions
2. Also write the report for each task and submit through your lab report.

References:

1. http://yuba.stanford.edu/~nickm/papers/ancs48-gibb.pdf
2. http://www.binarytides.com/packet-sniffer-code-c-linux/
3. http://www.scs.stanford.edu/histar/src/uinc/linux/if_ether.h
4. http://www.cse.scu.edu/~dclark/am_256_graph_theory/linux_2_6_stack/linux_2ip_8h-source.html
5. https://www.cymru.com/Documents/ip_icmp.h