```
Identification: 28484
RESERVED fragment flag: 0
DONT fragment flag: 0
MORE fragment flag: 0

time: Mon Oct  4 16:16:07 2004
Source:      0:0:135:175:97:65
Destination: 48:167:142:191:255:127
IP packet
header length: 20
UDP packet
  source: 128.3.26.249
  destination: 128.3.99.102
Time to live: 64
Identification: 58896
RESERVED fragment flag: 0
DONT fragment flag: 0
MORE fragment flag: 0

time: Mon Oct  4 16:16:07 2004
Source:      0:0:135:175:97:65
Destination: 48:167:142:191:255:127
IP packet
header length: 20
UDP packet
  source: 128.3.99.54
  destination: 128.3.26.152
Time to live: 62
Identification: 53340
RESERVED fragment flag: 0
DONT fragment flag: 0
MORE fragment flag: 0

time: Mon Oct  4 16:16:08 2004
Source:      0:0:136:175:97:65
Destination: 48:167:142:191:255:127

time: Mon Oct  4 16:16:08 2004
Source:      0:0:136:175:97:65
Destination: 48:167:142:191:255:127
```

```
/*********************************************
 *
 *                Lab9
 *
 *  This is an example of packet sniffing using
 *  the libpcap library.  For packet structures
 *  this program uses the definitions found in
 *  the /usr/include/netinet directory
 *
 *  Without parameters this program will use
 *  the first network interface.  The program
 *  will only work correctly if this is an
 *  ethernet interface.  It won't work for
 *  WiFi.  If there is a parameter it is the name
 *  of a PCAP file that is used for the
 *  packets instead of the network interface.
 ******************************************/
#include <stdio.h>
#include <time.h>
#include <pcap.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <stdlib.h>
#include <netinet/if_ether.h>
#include <netinet/in.h>
#include <netinet/ip.h>

void packetCallback(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet);

int main(int argc, char **argv) {
    char *device;
    char error_buffer[PCAP_ERRBUF_SIZE];
    pcap_t *handle;
    int packet_count = 25;
    int packet_timeout = 10000;

    /*
     *  determine where the packets are coming from
     */
    if(argc == 2) {
        handle = pcap_open_offline(argv[1], error_buffer);
        if(handle == NULL) {
            printf("error opening file: %s\n",error_buffer);
            exit(1);
        }
    } else {
        device = pcap_lookupdev(error_buffer);
        if(device == NULL) {
            printf("Can't find device: %s\n", error_buffer);
            exit(1);
        }
        handle = pcap_open_live(device, BUFSIZ, 0, packet_timeout,
                error_buffer);
        if(handle == NULL) {
            printf("can't open device: %s\n", error_buffer);
            exit(1);
        }
    }

    /*
     *  start the capture loop
     */
    pcap_loop(handle, packet_count, packetCallback, NULL);

    /*
     *  close the handle and exit
     */
    pcap_close(handle);
}


/*
 *  This procedure processes a single IP packet.
 *  It does the basic work that you will expand upon.
 *  For the assignment it will call procedures for
 *  handling TCP and UDP packets.
 */
void processIP(const u_char *packet) {
    struct iphdr *ip;
    u_char *payload;
    char *addr;
    unsigned int len;

    /*
     *  cast the bytes to an IP packet header
     */
    ip = (struct iphdr*) packet;
    /*
```

```
     *  check that we have an IPv4 packet
     */
    if(ip->version != 4) {
        printf("not version 4\n");
        return;
    }
    /*
     *  compute the header length and the location
     *  of the TCP or UDP packet
     */
    len = ip->ihl*4;
    printf("header length: %d\n", len);
    payload = (unsigned char*)packet+len;

    if(ip->protocol == IPPROTO_TCP) {
        printf("TCP packet\n");
        // call the TCP procedure here
    }
    if(ip->protocol == IPPROTO_UDP) {
        printf("UDP packet\n");
        // call the UDP procedure here
    }
    /*
     *  print the source and destination addresses
     */
    addr = (char*) &(ip->saddr);
    printf("  source: %hhu.%hhu.%hhu.%hhu\n",addr[0], addr[1],addr[2],
addr[3]);
    addr = (char*) &(ip->daddr);
    printf("  destination: %hhu.%hhu.%hhu.%hhu\n",addr[0],
addr[1],addr[2], addr[3]);

    printf("Time to live: %d\n", ip->ttl);
    printf("Identification: %d\n", ntohs(ip->id));

    int RF = (ip->frag_off & IP_RF);
    if (RF != 0)
    {
        printf("RESERVED Fragment Flag: %d\n", RF);
    }
    else {
        printf("RESERVED fragment flag: %d\n", RF);
    }
    int DF = (ip->frag_off & IP_DF);
    if (DF != 0)
    {
        printf("DONT Fragment Flag: %d\n", DF);
    }
    else {
        printf("DONT fragment flag: %d\n", DF);
    }
    int MF = (ip->frag_off & IP_MF);
    if (MF != 0)
    {
        printf("MORE Fragment Flag: %d\n", MF);
    }
    else {
        printf("MORE fragment flag: %d\n", MF);
    }
}


/*
 *  This is the callback procedure that processes the
 *  ethernet packets.  If the payload is an IP packet
 *  processIP() is called to process it.
 */
void packetCallback(u_char *args, const struct pcap_pkthdr *header, const
u_char *packet) {
    struct ether_header *eptr;
    short type;

    printf("time: %s", ctime((const time_t*) &header->ts.tv_sec));
    printf(
        "Source:      %d:%d:%d:%d:%d:%d\n",
        eptr->ether_shost[0], eptr->ether_shost[1], eptr->ether_shost[2],
        eptr->ether_shost[3], eptr->ether_shost[4], eptr->ether_shost[5]
    );
    printf(
        "Destination: %d:%d:%d:%d:%d:%d\n",
        eptr->ether_dhost[0], eptr->ether_dhost[1], eptr->ether_dhost[2],
        eptr->ether_dhost[3], eptr->ether_dhost[4], eptr->ether_dhost[5]
    );
    eptr = (struct ether_header *) packet;
    type = ntohs(eptr->ether_type);
    if(type == ETHERTYPE_IP) {
        printf("IP packet\n");
        processIP(packet+14);
    }
    if(type == ETHERTYPE_ARP) {
        printf("arp packet\n");
    }
    printf("\n");
}
```