Networking assignment 5

Yehudit Brickner 328601018

Yossi Elias 316156108

In this assignment we had 2 parts.
The first part was to use code given to us with areas to fill in, we needed to fill in code so that we would receive the pong and calculate the round-trip time it took to send and receive the pong. We also had to print out info about the pong we received. We decided to print the whole packet.
The second part was to write code to make a sniffer for ICMP packets and print out the Ips, type and code.

You can see in the screenshots below:
1. The pong printed
2. The wire shark of the ping and pong
3. The sniffer print out of the ping and pong.

```
we sent 1 packet:
Size: 30 bytes: ICMP header(8) + data(22)
Data sent: This is the ping.!!!

Msg recieved
packet size: 50 bytes: IP header(20) + ICMP header(8) + data(22)
Data: This is the ping.!!!

ip header data
version of ip: 4
length of header: 5
type of service: 96
length of the whole packet: 12800
unique identifier of the packet: 0
fragmantation flags: 0
time to live: 117
protocol: 1
checksum: 29981
ip source: 8.8.8.8
ip destionation: 10.9.13.222

icmp header data
type: 0
code: 0
checksum: 65162
id: 18
sequence: 0
icmp data: 140732892192572

packet data
This is the ping.!!!

time diff is 22.800000 milli seconds

time diff is 22800.000000 micro seconds
yehudit@yehudit-Aspire-E1-572:~/Documents/ariel university/2nd year
```

```
3402  6.455476183   10.9.13.222   8.8.8.8       ICMP   64   Echo (ping) request  id=0x1200, seq=0/0, ttl=64 (reply in 3431)
3431  6.520607632   8.8.8.8       10.9.13.222   ICMP   64   Echo (ping) reply    id=0x1200, seq=0/0, ttl=117 (request in 3402)
```

request

```
▸ Frame 3402: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: LiteonTe_bf:e9:33 (40:f0:2f:bf:e9:33), Dst: 40:b5:c1:f5:ae:3c (40:b5:c1:f5:ae:3c)
▸ Internet Protocol Version 4, Src: 10.9.13.222, Dst: 8.8.8.8
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x82fe [correct]
    [Checksum Status: Good]
    Identifier (BE): 4608 (0x1200)
    Identifier (LE): 18 (0x0012)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Response frame: 3431]
  ▸ Data (22 bytes)
```

```
0000  40 b5 c1 f5 ae 3c 40 f0  2f bf e9 33 08 00 45 00   @····<@· /··3··E·
0010  00 32 87 87 40 00 40 01  8b 4d 0a 09 0d de 08 08   ·2··@·@· ·M······
0020  08 08 08 00 82 fe 12 00  00 00 54 68 69 73 20 69   ········ ··This i
0030  73 20 74 68 65 20 70 69  6e 67 2e 21 21 21 0a 00   s the pi ng.!!!··
```

reply

```
▸ Frame 3431: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: 40:b5:c1:f5:ae:3c (40:b5:c1:f5:ae:3c), Dst: LiteonTe_bf:e9:33 (40:f0:2f:bf:e9:33)
▸ Internet Protocol Version 4, Src: 8.8.8.8, Dst: 10.9.13.222
▾ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x8afe [correct]
    [Checksum Status: Good]
    Identifier (BE): 4608 (0x1200)
    Identifier (LE): 18 (0x0012)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
    [Request frame: 3402]
    [Response time: 65.131 ms]
  ▸ Data (22 bytes)
```

```
0000  40 f0 2f bf e9 33 40 b5  c1 f5 ae 3c 08 00 45 60   @·/··3@· ···<··E`
0010  00 32 00 00 00 00 75 01  1d 75 08 08 08 08 0a 09   ·2····u· ·u······
0020  0d de 00 00 8a fe 12 00  00 00 54 68 69 73 20 69   ········ ··This i
0030  73 20 74 68 65 20 70 69  6e 67 2e 21 21 21 0a 00   s the pi ng.!!!··
```

Sniffer

```
********** message #0 ***************
SRC ip : 10.9.13.222
DST ip : 8.8.8.8
type : 8 echo (ping) request
code : 0
********** message #1 ***************
SRC ip : 8.8.8.8
DST ip : 10.9.13.222
type : 0 echo (ping) reply
code : 0
^C
yehudit@yehudit-Aspire-E1-572:~/Documen
```

Explaining the code:

Part 1

We decided to ping google 8.8.8.8 our IP is 10.9.13.222 we used 127.0.0.1 so that the computer would fill in our IP.

```
#define SOURCE_IP "127.0.0.1"
// i.e the gateway or ping to google.com for their ip-address
#define DESTINATION_IP "8.8.8.8"
```

The raw socket we created sends packets that the protocol is ICMP.

```
// Create raw socket for IP-RAW (make IP-header by yourself)

int sock = -1;

if ((sock = socket (AF_INET, SOCK_RAW, IPPROTO_ICMP)) == -1){
    fprintf (stderr, "socket() failed with error: %d",errno);
    fprintf (stderr, "To create a raw socket, the process needs to be run by Admin/root user.\n\n");
    return -1;
}
```

We started counting the time before we sent the ping and finished counting the time after we printed the data from the pong that was replied.
To print the data, we split the packet into 3, the IP header the ICMP header and the data.

```
206     // Send the packet using sendto() for sending datagrams.
207     // starting the clock
208     clock_t start, end, diff;
209     start = clock();
210
211     int packet_size;
212     packet_size=sendto (sock, packet, ICMP_HDRLEN + datalen, 0, (struct sockaddr *) &dest_in, sizeof (dest_in));
213     if (packet_size==-1){
214         printf("sendto failed with error:%d",errno);
215         return -1;
216     }
217     printf("We sent 1 packet:\n");
218     printf("Size: %d bytes: ICMP header(%d) + data(%d)\n", packet_size, ICMP_HDRLEN, datalen);
219     printf("Data sent: %s \n", packet + ICMP_HDRLEN);
220
221     socklen_t len = sizeof(dest_in);
222     int recieve_size=-1;
223     // making sure we recieved a packet
224     recieve_size=recvfrom(sock, &packet, sizeof(packet), 0, (struct sockaddr *) &dest_in, &len);
225     if (recieve_size>0){
226         //printing all the packet data
227         printf("Msg recieved\n");
228         printf("packet size: %d bytes: IP header(%d) + ICMP header(%d) + data(%d)\n", recieve_size, IP4_HDRLEN, ICMP_HDRLEN,datalen);
229         printf("Data: %s \n", packet + IP4_HDRLEN+ICMP_HDRLEN);
230
231         struct ip  *myheader=(struct ip  *)packet;
232         printf("ip header data\n");
233         printf("version of ip: %d\n",(myheader->ip_v));
234         printf("length of header: %d\n",(myheader->ip_hl));
235         printf("type of service: %d\n",(myheader->ip_tos));
236         printf("length of the whole packet: %d\n",(myheader->ip_len));
237         printf("unique identifier of the packet: %d\n",(myheader->ip_id));
238         printf("fragmentation flags: %d\n",(myheader->ip_off));
239         printf("time to live: %d\n",(myheader->ip_ttl));
240         printf("protocol: %d\n",(myheader->ip_p));
241         printf("checksum: %d\n",(myheader->ip_sum));
242         printf("ip source: %s\n",inet_ntoa(myheader->ip_src));
243         printf("ip destionation: %s\n",inet_ntoa(myheader->ip_dst));
244
```

```
245         struct icmp *myicmpheader=(struct icmp *)(packet+IP4_HDRLEN);
246         printf("\nicmp header data\n");
247         printf("type: %d\n",(myicmpheader->icmp_type));
248         printf("code: %d\n",(myicmpheader->icmp_code));
249         printf("checksum: %d\n",(myicmpheader->icmp_cksum));
250         printf("id: %d\n",(myicmpheader->icmp_id));
251         printf("sequence: %d\n",(myicmpheader->icmp_seq));
252         printf("icmp data: %ld\n",(myicmpheader->icmp_data));
253 
254         printf ("\npacket data\n");
255         int startp= IP4_HDRLEN+ ICMP_HDRLEN;
256         int endp= IP4_HDRLEN+ICMP_HDRLEN + datalen;
257         for( int i=startp;i<endp;i++){
258             printf("%c",packet[i]);
259         }
260     }
261     //ending the clock
262     end=clock();
263     diff =(end-start);
264     printf("\ntime diff is %f milli seconds\n", diff/10.0);
265     printf("\ntime diff is %f micro seconds\n", diff*100.0);
```

Part 2

we created a raw socket that accepts packets with all types of protocols.

```
//creating raw socket:
//the third paramter enables the socket to accept all packets that were sent over all protocals
raw_socket = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
if(raw_socket < 0){
    printf("Error creating socket\n");
}
```

we then created a while loop that is always true and started receiving packets.
we made sure that we received the packet and that the protocol type is ICMP.

```
// Receive all packets:
while(1){
    bzero(buff, sizeof(buff)); // init buffer.
    // receve data from all ip sorce and dest
    data_size = recvfrom(raw_socket, buff,PACKET_LEN, 0, NULL, NULL);
    // check if we gwt a packet
    if(data_size < 0){
        printf("failed to get the packet\n");
    }
    else{
        //reading from icmp:
        struct ethhdr *eth = (struct ethhdr *)buff;
        struct iphdr *iph;
        if (ntohs(eth->h_proto) == 0x0800) { // 0x0800 is IP type
            iph = (struct iphdr *)(buff + sizeof(struct ethhdr));

            //getting the length of the ip header length field in bits in case of icmp protocol.
            int iph_len = iph->ihl *4;
            // check if the protocal is icmp id so print the wanted info
            if(iph->protocol == IPPROTO_ICMP){
```

If the protocol is ICMP we printed the requested data, IPs, type, and code.
We printed the data from the IP and ICMP headers.

```
printf("************ message #%d ***************\n",count);
// Starts after IP HEADER so we head there.
struct icmphdr *icmph = (struct icmphdr *)(buff + sizeof(struct ethh

// Getting the ip data :
bzero(&src,sizeof(src));
bzero(&dst,sizeof(dst));
src.sin_addr.s_addr = iph->saddr;
dst.sin_addr.s_addr = iph->daddr;
```

```
printf("SRC ip : %s\n",inet_ntoa(src.sin_addr)); //convert back
printf("DST ip : %s\n",inet_ntoa(dst.sin_addr)); // convert back
int type=icmph->type;
if (type==0){
    printf("type : %d echo (pin) reply\n",type);
}
else if(type==8){
    printf("type : %d echo (ping) request\n",type);
}
else{
    printf("type : %d\n",type);
}
printf("code : %d\n",(icmph->code));
```

The rest of the code for part 1 that isn't shown was either given to us or is defining variables. the rest of the code for part 2 that isn't shown here is defining variables, headers and closing the socket.

All the code is in the zip file that was handed in, and it can be found at: https://github.com/Yehudit-Brickner/networkingEx5