Ex5_comm:

Ohad Shirazi[redacted]

Dvir Biton-[redacted]

We will explain both parts of the exercise together:

On part A "myping" we sent a "ping" to 8.8.8.8 (Google address) and wait for a Echo Reply.
We started measure the time when we sent the ping to 8.8.8.8 and finished the measurement when we got the Echo Reply.

We calculate the RTT by milliseconds and microseconds as required.

```
ohad@ohad-VirtualBox:~/CLionProjects/Ex5_comm$ sudo ./myping
[sudo] password for ohad:
Sent one packet:
        Size: 27 bytes: ICMP header(8) + data(19)
        Data: This is the ping.

Msg #1
        Size: 47 bytes: IP header(20) + ICMP header(8) + data(19)
        Data:E
RTT time in miliseconds: 77.000000
RTT time in microseconds: 77401
ohad@ohad-VirtualBox:~/CLionProjects/Ex5_comm$
```

We can also see all this data from the Wireshark:

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
34	115.302748	10.0.2.15	31.13.92.52	TCP	54	36730 → 443 [ACK] Seq=121 Ack=3717 Win=65535 Len=0
35	127.358121	10.0.2.15	34.117.237.239	TLSv1.2	93	Application Data
36	127.358359	10.0.2.15	34.117.237.239	TLSv1.2	78	Application Data
37	127.358571	34.117.237.239	10.0.2.15	TCP	60	443 → 59846 [ACK] Seq=79 Ack=118 Win=65535 Len=0
38	127.358571	34.117.237.239	10.0.2.15	TCP	60	443 → 59846 [ACK] Seq=79 Ack=142 Win=65535 Len=0
39	127.358692	10.0.2.15	34.117.237.239	TCP	54	59846 → 443 [FIN, ACK] Seq=142 Ack=79 Win=64028 Len=0
40	127.358990	34.117.237.239	10.0.2.15	TCP	60	443 → 59846 [ACK] Seq=79 Ack=143 Win=65535 Len=0
41	127.447337	34.117.237.239	10.0.2.15	TCP	60	443 → 59846 [FIN, ACK] Seq=79 Ack=143 Win=65535 Len=0
42	127.447361	10.0.2.15	34.117.237.239	TCP	54	59846 → 443 [ACK] Seq=143 Ack=80 Win=64028 Len=0
43	128.833920	10.0.2.15	8.8.8.8	ICMP	61	Echo (ping) request id=0x1200, seq=0/0, ttl=64 (reply in 44)
44	128.911222	8.8.8.8	10.0.2.15	ICMP	61	Echo (ping) reply id=0x1200, seq=0/0, ttl=111 (request in 43)
45	141.773269	10.0.2.15	31.13.92.52	TLSv1.2	94	Application Data
46	141.773621	31.13.92.52	10.0.2.15	TCP	60	443 → 36730 [ACK] Seq=3717 Ack=161 Win=65535 Len=0
47	141.950654	31.13.92.52	10.0.2.15	TLSv1.2	101	Application Data
48	141.950683	10.0.2.15	31.13.92.52	TCP	54	36730 → 443 [ACK] Seq=161 Ack=3764 Win=65535 Len=0
49	158.655784	31.13.92.52	10.0.2.15	TLSv1.2	1514	Application Data
50	158.655849	10.0.2.15	31.13.92.52	TCP	54	36730 → 443 [ACK] Seq=161 Ack=5224 Win=65535 Len=0

> Frame 43: 61 bytes on wire (488 bits), 61 bytes captured (488 bits)
> Ethernet II, Src: PcsCompu_28:0c:62 (08:00:27:28:0c:62), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8

Wireshark · Packet 43 · Ex5.ping_sniffer.pcap

```
> Frame 43: 61 bytes on wire (488 bits), 61 bytes captured (488 bits)
> Ethernet II, Src: PcsCompu_28:0c:62 (08:00:27:28:0c:62), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xae36 [correct]
    [Checksum Status: Good]
    Identifier (BE): 4608 (0x1200)
    Identifier (LE): 18 (0x0012)
    Sequence Number (BE): 0 (0x0000)
    Sequence Number (LE): 0 (0x0000)
    [Response frame: 44]
  v Data (19 bytes)
      Data: 546869732069732074686520706696e672e0a00
      [Length: 19]
```

```
0000  52 54 00 12 35 02 08 00  27 28 0c 62 08 00 45 00   RT··5··· '(·b··E·
0010  00 2f e6 3f 40 00 40 01  38 70 0a 00 02 0f 08 08   ·/·?@·@· 8p·····
0020  08 08 08 00 ae 36 12 00  00 00 54 68 69 73 20 69   ·····6·· ··This i
0030  73 20 74 68 65 20 70 69  6e 67 2e 0a 00           s the pi ng.··
```

Here we can see all the data that we showed up before.
The kernel of the code (send and receive the ping):

```c
//*************************** Send the ICMP ECHO REQUEST packet: ****************************//
int size_sent;

struct timeval start, end; // ****** start time ********
gettimeofday(&start, NULL);

size_sent = sendto(sock, packet,  ICMP_HDRLEN + datalen, 0, (struct sockaddr *) &dest_in,sizeof(dest_in));
if (size_sent == -1) {
    fprintf(stderr, "sendto() failed with error: %d", errno);
    return -1;
}
printf("Sent one packet:\n");
printf("\tSize: %d bytes: ICMP header(%d) + data(%d)\n", size_sent, ICMP_HDRLEN, datalen);
printf("\tData: %s \n", packet + ICMP_HDRLEN);

// ********************** Receive the ICMP ECHO REPLY packet: ***********************************//
int cnt = 1;
bzero(&packet, sizeof(packet));
socklen_t len = sizeof(dest_in);
int size_recv = -1;
while (size_recv < 0) {
    size_recv = recvfrom(sock, packet, sizeof(packet), 0, (struct sockaddr *) &dest_in, &len);
}
printf("Msg #%d\n", cnt);
printf("\tSize: %d bytes: IP header(%d) + ICMP header(%d) + data(%d)\n", size_recv, IP4_HDRLEN, ICMP_HDRLEN,
       datalen);
printf("\tData:%s\n",packet);
```

On Part B "sniffer"- we created a sniffer for ICMP Packets.

We turned on the sniffer before activated the prog- "myping"
in order to sniff the Echo request and Echo reply of "myping",
Then we "ping" 8.8.8.8 from the terminal to sniff more ICMP Packets.

```
ohad@ohad-VirtualBox:~/CLionProjects/Ex5_comm$ sudo ./sniffer
[sudo] password for ohad:


****************** ICMP Packet number #1 ******************
Source IP is: 10.0.2.15
Destination IP is: 8.8.8.8
ICMP Echo type is: Type 8 - Echo Request
ICMP Echo code is: 0


****************** ICMP Packet number #2 ******************
Source IP is: 8.8.8.8
Destination IP is: 10.0.2.15
ICMP Echo type is: Type 0 - Echo Reply
ICMP Echo code is: 0
```

(myping ICMP Packets)

```
ohad@ohad-VirtualBox:~/CLionProjects/Ex5_comm$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=111 time=76.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=78.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=111 time=78.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=111 time=85.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=111 time=76.8 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 76.819/79.100/84.982/3.027 ms
```

```
****************** ICMP Packet number #3 ******************
Source IP is: 10.0.2.15
Destination IP is: 8.8.8.8
ICMP Echo type is: Type 8 - Echo Request
ICMP Echo code is: 0


****************** ICMP Packet number #4 ******************
Source IP is: 8.8.8.8
Destination IP is: 10.0.2.15
ICMP Echo type is: Type 0 - Echo Reply
ICMP Echo code is: 0


****************** ICMP Packet number #5 ******************
Source IP is: 10.0.2.15
Destination IP is: 8.8.8.8
ICMP Echo type is: Type 8 - Echo Request
ICMP Echo code is: 0
```

```
****************** ICMP Packet number #6 ******************
Source IP is: 8.8.8.8
Destination IP is: 10.0.2.15
ICMP Echo type is: Type 0 - Echo Reply
ICMP Echo code is: 0


****************** ICMP Packet number #7 ******************
Source IP is: 10.0.2.15
Destination IP is: 8.8.8.8
ICMP Echo type is: Type 8 - Echo Request
ICMP Echo code is: 0


****************** ICMP Packet number #8 ******************
Source IP is: 8.8.8.8
Destination IP is: 10.0.2.15
ICMP Echo type is: Type 0 - Echo Reply
ICMP Echo code is: 0


****************** ICMP Packet number #9 ******************
Source IP is: 10.0.2.15
Destination IP is: 8.8.8.8
ICMP Echo type is: Type 8 - Echo Request
ICMP Echo code is: 0
```

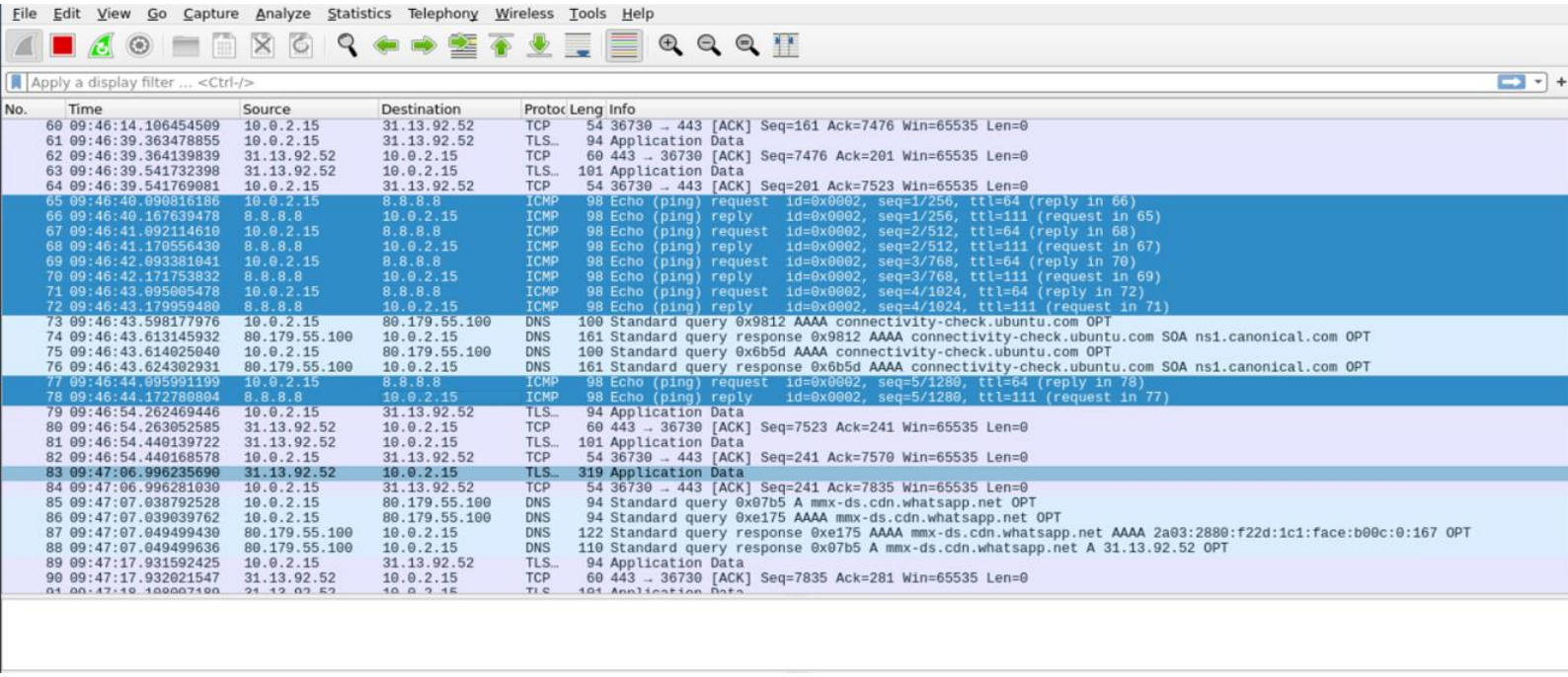
```
****************** ICMP Packet number #10 ******************
Source IP is: 8.8.8.8
Destination IP is: 10.0.2.15
ICMP Echo type is: Type 0 – Echo Reply
ICMP Echo code is: 0


****************** ICMP Packet number #11 ******************
Source IP is: 10.0.2.15
Destination IP is: 8.8.8.8
ICMP Echo type is: Type 8 – Echo Request
ICMP Echo code is: 0


****************** ICMP Packet number #12 ******************
Source IP is: 8.8.8.8
Destination IP is: 10.0.2.15
ICMP Echo type is: Type 0 – Echo Reply
ICMP Echo code is: 0
```

(all the 10 ICMP Packets – 5 of Request and 5 of Reply)

We can also see all this data from the Wireshark:



Turn on the promiscuous mode:

```
// ************* Turn on the promiscuous mode (on which interface to work) *************
    struct packet_mreq mr;
    mr.mr_type = PACKET_MR_PROMISC;
//    SOL_PACKET -->> to manipulate options at the socket api level
//                    that means: you can set up and establish connections to other users on the network
//                                send and receive data to and from other users
//                                close down connections
//    PACKET_ADD_MEMBERSHIP -->> adds a binding
    setsockopt(raw_sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mr, sizeof(mr));
```

Function to get the ICMP Packet header (as required):

```
void get_icmp_packet(unsigned char* buffer, int buffer_size){
    char *types[]={"Type 0 – Echo Reply",
                   "Type 1 – Unassigned",
                   "Type 2 – Unassigned",
                   "Type 3 – Destination Unreachable",
                   "Type 4 – Source Quench (Deprecated)",
                   "Type 5 – Redirect",
                   "Type 6 – Alternate Host Address (Deprecated)",
                   "Type 7 – Unassigned",
                   "Type 8 – Echo Request",
                   "Type 9 – Router Advertisement",
                   "Type 10 – Router Selection"};
    unsigned short ip_hdr_len;
    struct iphdr *iph = (struct iphdr *)(buffer+ETH_HLEN);
    if (iph->protocol== IPPROTO_ICMP) {
        ip_hdr_len = iph->ihl * 4;
        struct icmphdr *icmph = (struct icmphdr *) (buffer + ip_hdr_len +ETH_HLEN);

     if ((unsigned int) (icmph->type) < 11) {
         struct sockaddr_in src;
         memset(&src, 0, sizeof(src));
         src.sin_addr.s_addr = iph->saddr;

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


         printf("\n");

         printf("\n****************** ICMP Packet number #%d ******************\n", counter);
         printf("Source IP is: %s\n", inet_ntoa(src.sin_addr));
         printf("Destination IP is: %s\n", inet_ntoa(dest.sin_addr));
         printf("ICMP Echo type is: %s\n", types[icmph->type]);
         printf("ICMP Echo code is: %d\n", icmph->code);
         counter++;
     }
  }
```

This is how we deal with any packet we got,
we check if its ICMP Packet and filtered the relevants fields.