CSS Lab 1 Task 1

ROLL NO. CS-16038
NAME: UMAMA AHMED

VM1: 192.168.100.100

VM2: 192.168.100.101

VM3: 192.168.100.102

Task 1.a: Writing Packet Sniffing Program on VM1:

sniffex38.c (~/Desktop/CSS/lab1) - gedit

```
#define APP_NAME                "sniffex"
#define APP_DESC                "Sniffer example using libpcap"
#define APP_COPYRIGHT   "Copyright (c) 2005 The Tcpdump Group"
#define APP_DISCLAIMER  "THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM."

#include <pcap.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* default snap length (maximum bytes per packet to capture) */
#define SNAP_LEN 1518

/* ethernet headers are always exactly 14 bytes [1] */
#define SIZE_ETHERNET 14

/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN  6

/* Ethernet header */
struct sniff_ethernet {
```

Setting promiscous mode:

seeds_ubuntu_labs [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

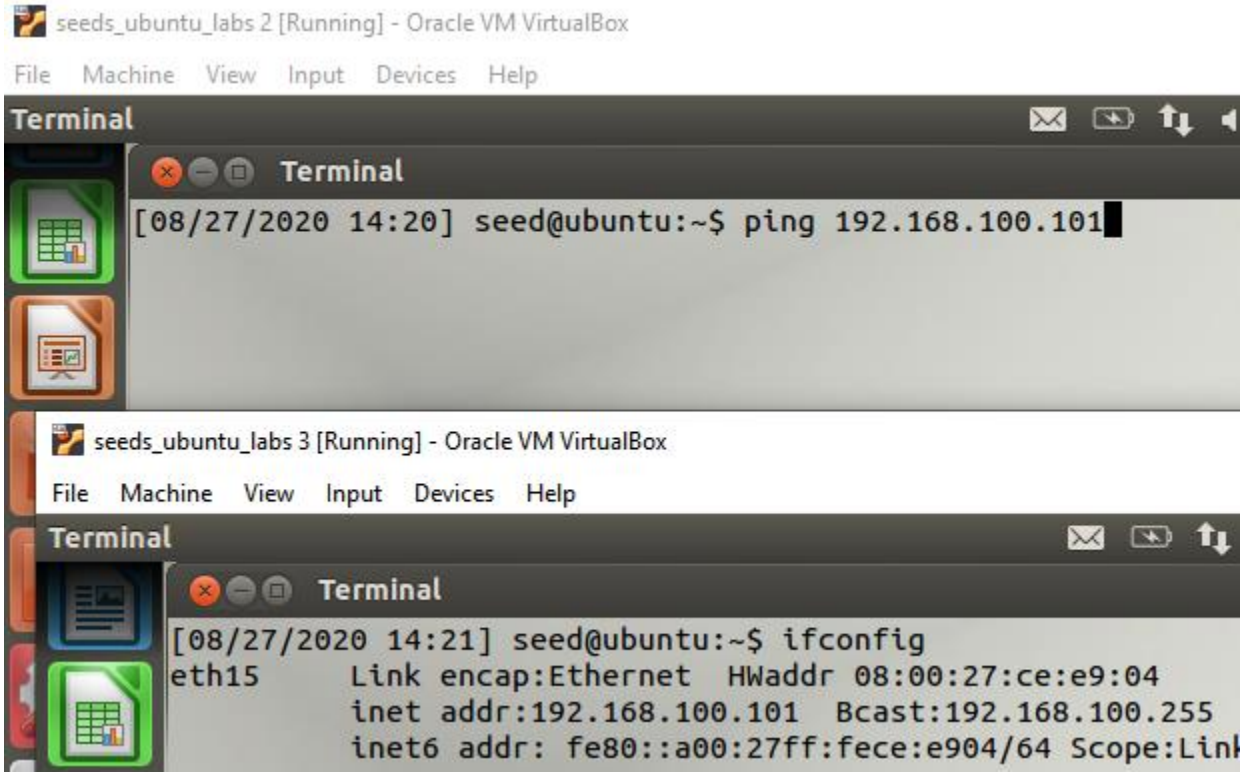
Terminal

```
[08/27/2020 14:14] seed@ubuntu:~$ sudo ifconfig eth13 promisc
[sudo] password for seed:
[08/27/2020 14:14] seed@ubuntu:~$
```

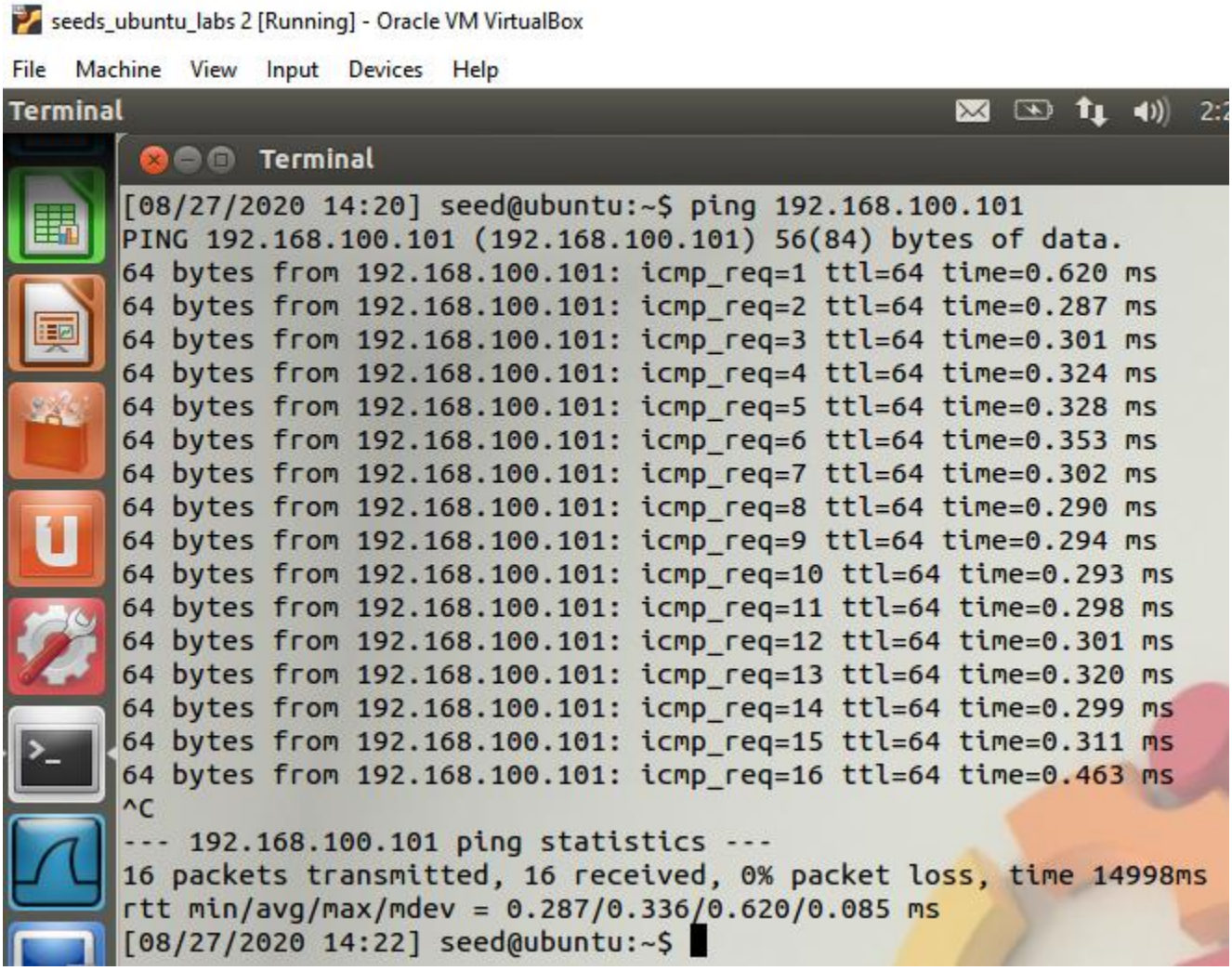
Sniffer program when promiscous mode is turned on:

```
Terminal
[08/27/2020 14:16] seed@ubuntu:~/Desktop/CSS/lab1$ gcc -o sniffex38 sniffex38.c
-lpcap
[08/27/2020 14:16] seed@ubuntu:~/Desktop/CSS/lab1$ sudo ./sniffex38
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth13
Number of packets: 10
Filter expression: ip

Packet number 1:
       From: 192.168.100.15
         To: 239.255.255.250
   Protocol: UDP

Packet number 2:
       From: 192.168.100.15
         To: 239.255.255.250
   Protocol: UDP
Packet number 3:
       From: 192.168.100.15
         To: 239.255.255.250
   Protocol: UDP

Packet number 4:
       From: 192.168.100.82
         To: 192.168.100.255
   Protocol: UDP

Packet number 5:
       From: 192.168.100.15
         To: 224.0.0.251
   Protocol: UDP

Packet number 6:
       From: 192.168.100.82
         To: 239.255.255.250
   Protocol: UDP

Packet number 7:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP
Packet number 8:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Packet number 9:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Packet number 10:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Capture complete.
[08/27/2020 14:17] seed@ubuntu:~/Desktop/CSS/lab1$
```

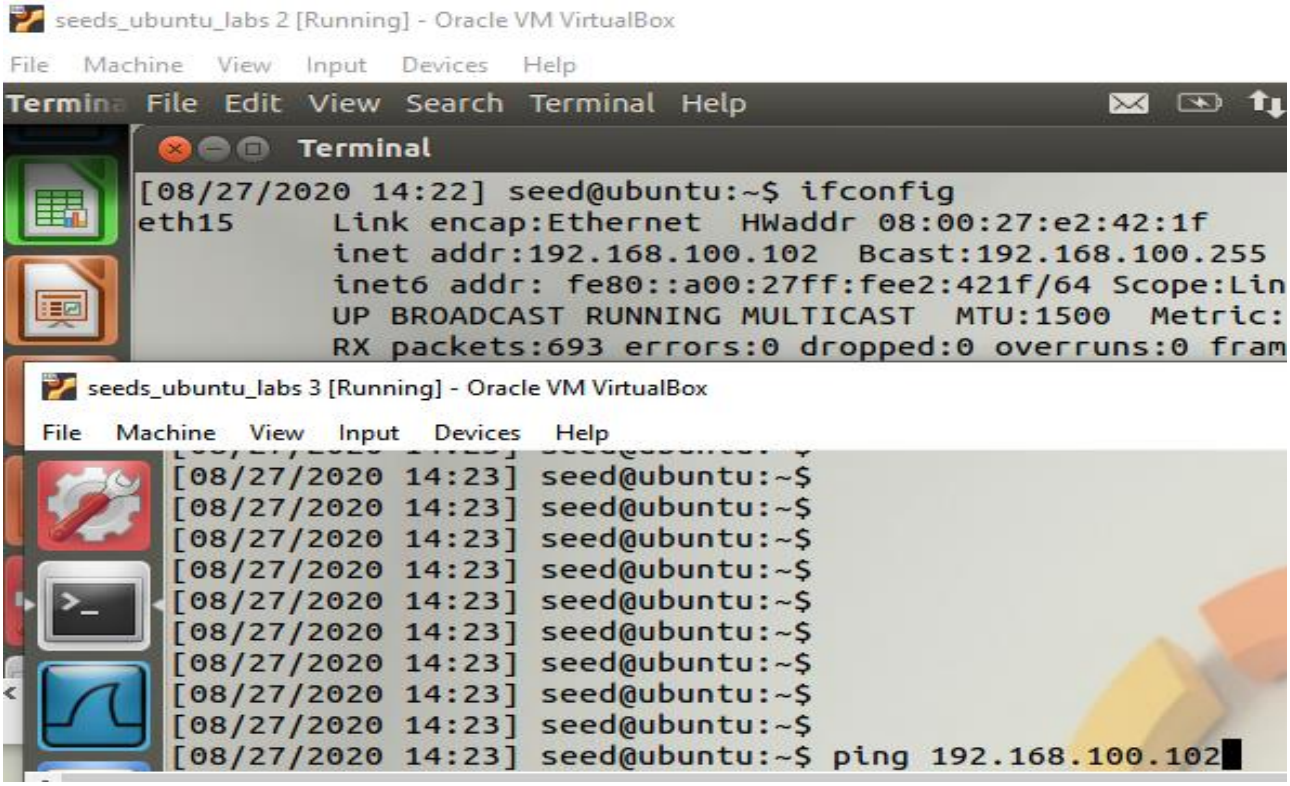
Ping VM3 from VM2:



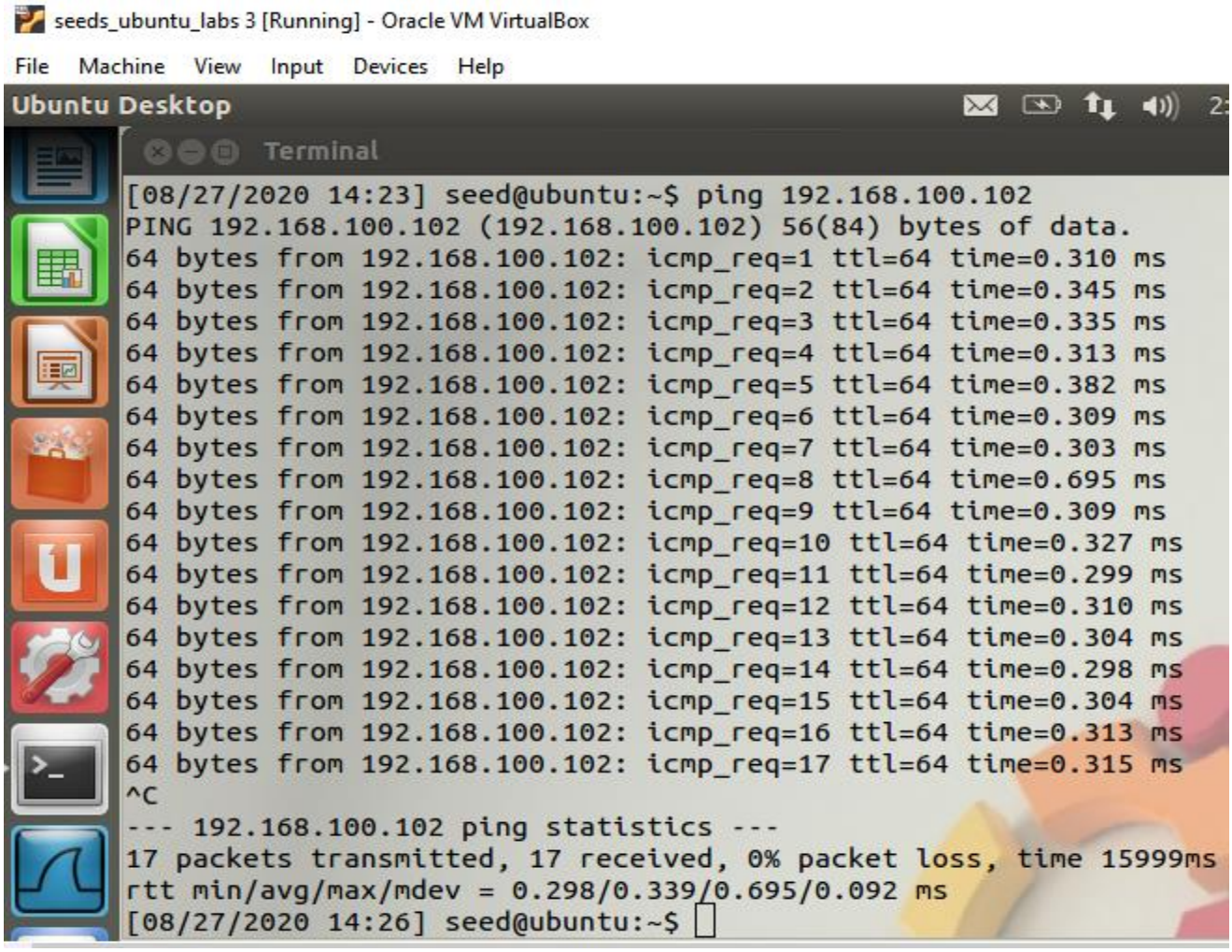
Pinging VM3 from VM2:



Ping VM2 from VM3:



Pinging VM2 from VM3:



During ping of VM2 and VM3, run sniffex program on VM1 to capture the packets of VM2 and VM3:

```
Terminal
[08/27/2020 14:30] seed@ubuntu:~/Desktop/CSS/lab1$ gcc -o sniffex38 sniffex38.c
-lpcap
[08/27/2020 14:32] seed@ubuntu:~/Desktop/CSS/lab1$ sudo ./sniffex38
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth13
Number of packets: 10
Filter expression: ip

Packet number 1:
       From: 192.168.100.82
         To: 239.255.255.250
   Protocol: UDP

Packet number 2:
       From: 192.168.100.82
         To: 192.168.100.255
   Protocol: UDP
Packet number 3:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Packet number 4:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Packet number 5:
       From: 192.168.100.82
         To: 239.255.255.250
   Protocol: UDP

Packet number 6:
       From: 192.168.100.10
         To: 224.0.0.22
   Protocol: unknown

Packet number 7:
       From: 192.168.100.10
         To: 224.0.0.251
   Protocol: UDP
Packet number 8:
       From: 192.168.100.10
         To: 224.0.0.251
   Protocol: UDP

Packet number 9:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Packet number 10:
       From: 192.168.100.1
         To: 239.255.255.250
   Protocol: UDP

Capture complete.
[08/27/2020 14:33] seed@ubuntu:~/Desktop/CSS/lab1$
```

Unsetting promiscous mode:

```
Terminal
[08/27/2020 14:36] seed@ubuntu:~$ sudo ifconfig eth13 -promisc
[sudo] password for seed:
[08/27/2020 14:36] seed@ubuntu:~$
```

Sniffer program when promiscous mode is turned off:

```
Terminal
[08/27/2020 14:39] seed@ubuntu:~/Desktop/CSS/lab1$ sudo ifconfig eth13 -promisc
[08/27/2020 14:39] seed@ubuntu:~/Desktop/CSS/lab1$ gcc -o sniffex38 sniffex38.c
-lpcap
[08/27/2020 14:39] seed@ubuntu:~/Desktop/CSS/lab1$ sudo ./sniffex38
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth13
Number of packets: 10
Filter expression: ip

Packet number 1:
       From: 192.168.100.82
         To: 192.168.100.255
   Protocol: UDP

Packet number 2:
       From: 192.168.100.82
         To: 239.255.255.250
   Protocol: UDP
Packet number 3:
       From: 192.168.100.94
         To: 192.168.100.255
   Protocol: UDP

Packet number 4:
       From: 192.168.100.94
         To: 224.0.0.251
   Protocol: UDP

Packet number 5:
       From: 192.168.100.94
         To: 224.0.0.252
   Protocol: UDP

Packet number 6:
       From: 192.168.100.94
         To: 224.0.0.252
   Protocol: UDP

Packet number 7:
       From: 192.168.100.94
         To: 192.168.100.255
   Protocol: UDP
Packet number 8:
       From: 192.168.100.94
         To: 224.0.0.251
   Protocol: UDP

Packet number 9:
       From: 192.168.100.94
         To: 192.168.100.255
   Protocol: UDP

Packet number 10:
       From: 192.168.100.82
         To: 192.168.100.255
   Protocol: UDP

Capture complete.
[08/27/2020 14:39] seed@ubuntu:~/Desktop/CSS/lab1$
```

ROLL NO. CS-16038
NAME: UMAMA AHMED

<u>Problem 1</u>: Please use your own words to describe the sequence of the library calls that are essential for sniffer programs:

- Ethernet interface that the program will utilize. (Such as eth13 in my case).
- The initialization of the PCAP to create a session, typically there is on session per device to be sniffed.
- The call to set traffic filtering rules, this ensures that the type of traffic sniffed on an interface is the type one is going for.
- The execution of the sniff.
- Termination of the session.

<u>Problem 2:</u> Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?

Pcap_lookupdev() function needs root access because it wants to access network interfaces and it is impossible without root access in linux. Sniffer programs need raw sockets that allow direct sending of packets by the applications bypassing all applications in network software of operating system. And we need to be a root to create raw socket as we can't discover NIC until we are root.

<u>Problem 3:</u> Please turn on and turn off the promiscuous mode in the sniffer program:

Screenshots are attached above.

Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this.

Promiscuous mode is one in which all the packets are sent to a computer or sniffed by sniffer and not only those which are addressed to it whereas in a non-promiscuous mode only those packets are send to the computer or sniffed by sniffer which are addressed to it.

CSS Lab 1 Task 2

Roll No. CS-16038
Name: Umama Ahmed

On VM1, making program rawudp.c

seeds_ubuntu_labs [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

rawudp38.c (~/Desktop/CSS/lab1) - gedit 4:03 PM

Open Save Undo

rawudp38.c

```c
// ----rawudp.c------
// Must be run by root lol! Just datagram, no payload/data
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>

// The packet length
#define PCKT_LEN 8192

// Can create separate header file (.h) for all headers' structure
// The IP header's structure
struct ipheader {
 unsigned char      iph_ihl:4, iph_ver:4;
 unsigned char      iph_tos;
 unsigned short int iph_len;
 unsigned short int iph_ident;
 unsigned char      iph_flag:3;
 unsigned short int iph_offset:13;
 unsigned char      iph_ttl;
 unsigned char      iph_protocol;
```

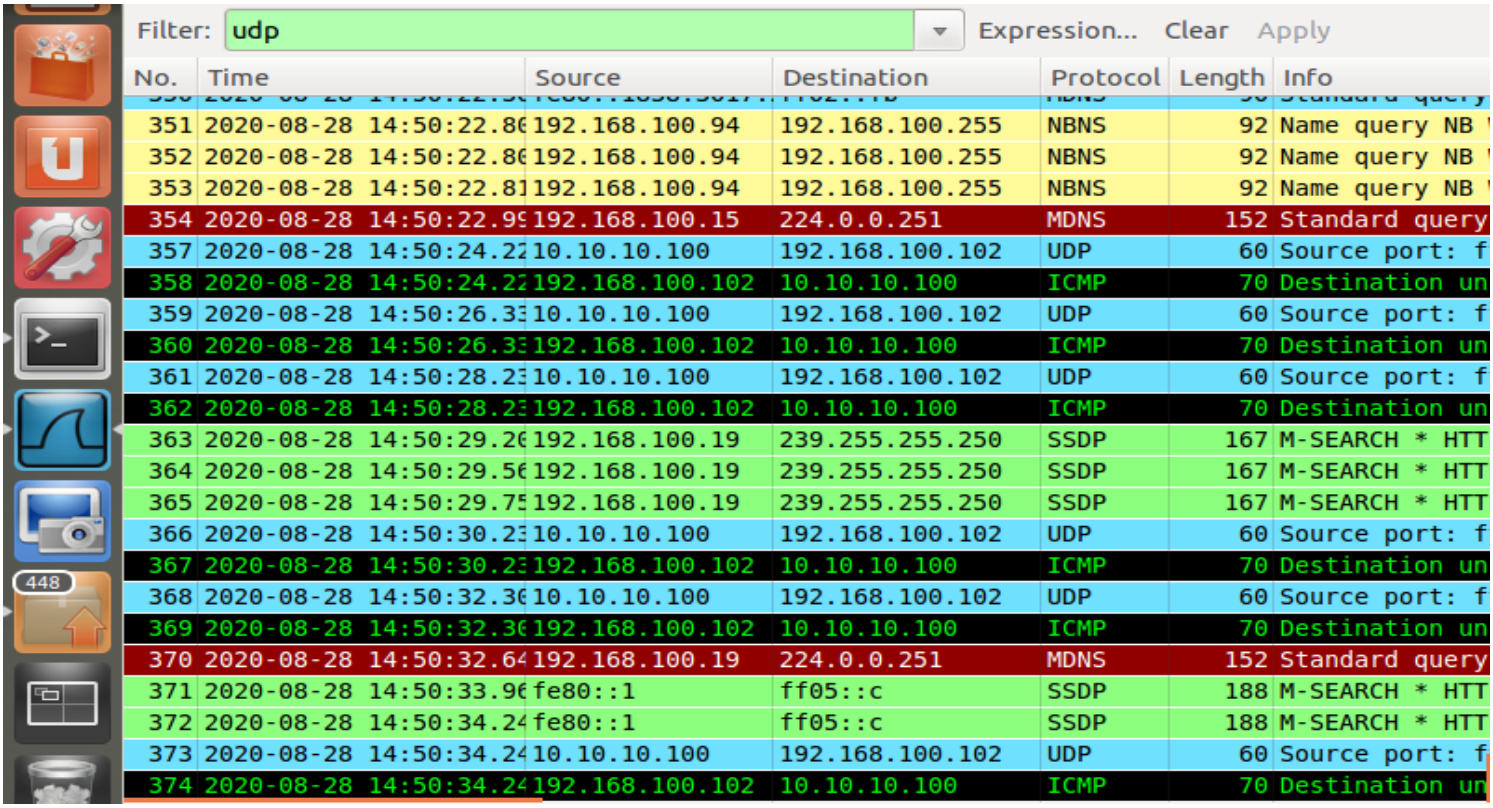
Run the program rawudp.c on VM 1:

Terminal

```
[08/28/2020 14:33] seed@ubuntu:~/Desktop/CSS/lab1$ sudo ./rawudp38 10.10.10.100
21 192.168.100.102 8080
socket() - Using SOCK_RAW socket and UDP protocol is OK.
setsockopt() is OK.
Trying...
Using raw socket and UDP protocol
Using Source IP: 10.10.10.100 port: 21, Target IP: 192.168.100.102 port: 8080.
Count #1 - sendto() is OK.
Count #2 - sendto() is OK.
Count #3 - sendto() is OK.
Count #4 - sendto() is OK.
Count #5 - sendto() is OK.
Count #6 - sendto() is OK.
Count #7 - sendto() is OK.
Count #8 - sendto() is OK.
Count #9 - sendto() is OK.
Count #10 - sendto() is OK.
Count #11 - sendto() is OK.
Count #12 - sendto() is OK.
Count #13 - sendto() is OK.
Count #14 - sendto() is OK.
Count #15 - sendto() is OK.
Count #16 - sendto() is OK.
Count #17 - sendto() is OK.
Count #18 - sendto() is OK.
Count #19 - sendto() is OK.
Count #20 - sendto() is OK.
```

Verification on VM 2:

```
[08/28/2020 14:48] seed@ubuntu:~$ sudo tcpdump -vv
[sudo] password for seed:
tcpdump: listening on eth15, link-type EN10MB (Ethernet), capture size 65535 byt
es
14:48:55.039013 IP (tos 0x10, ttl 63, id 54321, offset 0, flags [none], proto UD
P (17), length 28)
    10.10.10.100.fsp > ubuntu-2.local.http-alt: [no cksum] UDP, length 0
14:48:55.039039 IP (tos 0xd0, ttl 64, id 24434, offset 0, flags [none], proto IC
MP (1), length 56)
    ubuntu-2.local > 10.10.10.100: ICMP ubuntu-2.local udp port http-alt unreach
able, length 36
        IP (tos 0x10, ttl 63, id 54321, offset 0, flags [none], proto UDP (17),
length 28)
    10.10.10.100.fsp > ubuntu-2.local.http-alt: [no cksum] UDP, length 0
14:48:55.050586 IP (tos 0x0, ttl 64, id 20103, offset 0, flags [DF], proto UDP (
17), length 74)
4.0.0.251 to_ex { }]
14:48:56.584892 IP (tos 0x0, ttl 255, id 5276, offset 0, flags [none], proto UDP
 (17), length 122)
    192.168.100.10.mdns > 224.0.0.251.mdns: [udp sum ok] 1 [2q] PTR (QU)? _%9E5E
7C8F47989526C9BCD95D24084F6F0B27C5ED._sub._googlecast._tcp.local. PTR (QU)? _goo
glecast._tcp.local. (94)
14:48:56.584909 IP (tos 0x0, ttl 255, id 5277, offset 0, flags [none], proto UDP
 (17), length 122)
    192.168.100.10.mdns > 224.0.0.251.mdns: [udp sum ok] 1 [2q] PTR (QU)? _%9E5E
7C8F47989526C9BCD95D24084F6F0B27C5ED._sub._googlecast._tcp.local. PTR (QU)? _goo
glecast._tcp.local. (94)
14:48:57.042087 IP (tos 0x10, ttl 63, id 54321, offset 0, flags [none], proto UD
P (17), length 28)
```



Filter: udp Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
351	2020-08-28 14:50:22.86	192.168.100.94	192.168.100.255	NBNS	92	Name query NB
352	2020-08-28 14:50:22.86	192.168.100.94	192.168.100.255	NBNS	92	Name query NB
353	2020-08-28 14:50:22.81	192.168.100.94	192.168.100.255	NBNS	92	Name query NB
354	2020-08-28 14:50:22.95	192.168.100.15	224.0.0.251	MDNS	152	Standard query
357	2020-08-28 14:50:24.22	10.10.10.100	192.168.100.102	UDP	60	Source port: f
358	2020-08-28 14:50:24.22	192.168.100.102	10.10.10.100	ICMP	70	Destination un
359	2020-08-28 14:50:26.33	10.10.10.100	192.168.100.102	UDP	60	Source port: f
360	2020-08-28 14:50:26.33	192.168.100.102	10.10.10.100	ICMP	70	Destination un
361	2020-08-28 14:50:28.23	10.10.10.100	192.168.100.102	UDP	60	Source port: f
362	2020-08-28 14:50:28.23	192.168.100.102	10.10.10.100	ICMP	70	Destination un
363	2020-08-28 14:50:29.26	192.168.100.19	239.255.255.250	SSDP	167	M-SEARCH * HTT
364	2020-08-28 14:50:29.56	192.168.100.19	239.255.255.250	SSDP	167	M-SEARCH * HTT
365	2020-08-28 14:50:29.75	192.168.100.19	239.255.255.250	SSDP	167	M-SEARCH * HTT
366	2020-08-28 14:50:30.23	10.10.10.100	192.168.100.102	UDP	60	Source port: f
367	2020-08-28 14:50:30.23	192.168.100.102	10.10.10.100	ICMP	70	Destination un
368	2020-08-28 14:50:32.36	10.10.10.100	192.168.100.102	UDP	60	Source port: f
369	2020-08-28 14:50:32.36	192.168.100.102	10.10.10.100	ICMP	70	Destination un
370	2020-08-28 14:50:32.64	192.168.100.19	224.0.0.251	MDNS	152	Standard query
371	2020-08-28 14:50:33.96	fe80::1	ff05::c	SSDP	188	M-SEARCH * HTT
372	2020-08-28 14:50:34.24	fe80::1	ff05::c	SSDP	188	M-SEARCH * HTT
373	2020-08-28 14:50:34.24	10.10.10.100	192.168.100.102	UDP	60	Source port: f
374	2020-08-28 14:50:34.24	192.168.100.102	10.10.10.100	ICMP	70	Destination un

Making program rawtcp.c on VM 1:

rawtcp38.c (~/Desktop/CSS/lab1) - gedit

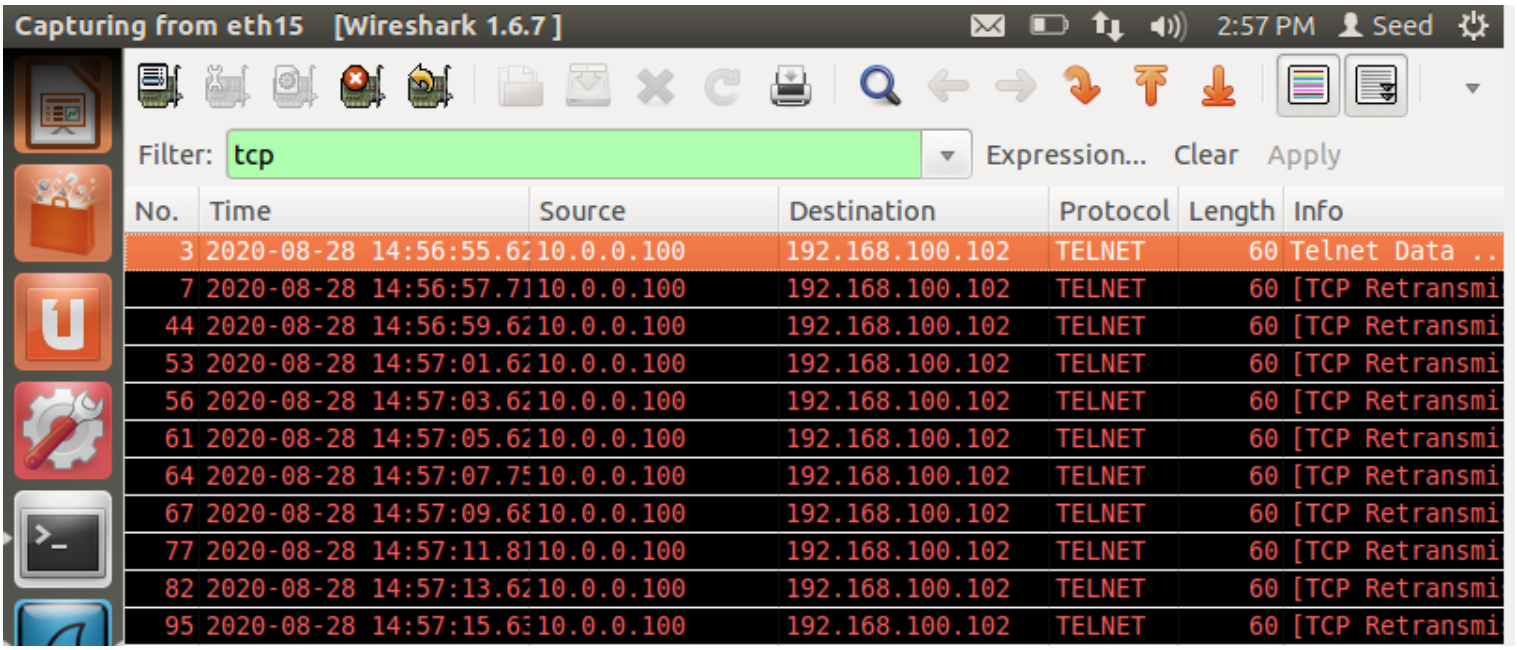
```
//---cat rawtcp.c---
// Run as root or SUID 0, just datagram no data/payload
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
// Packet length
#define PCKT_LEN 8192

// May create separate header file (.h) for all
// headers' structures
// IP header's structure
struct ipheader {
 unsigned char      iph_ihl:4, /* Little-endian */
                    iph_ver:4;
 unsigned char      iph_tos;
 unsigned short int iph_len;
 unsigned short int iph_ident;
 unsigned char      iph_flag:3;
 unsigned short int iph_offset:13;
 unsigned char      iph_ttl;
 unsigned char      iph_protocol;
 unsigned short int iph_chksum;
```

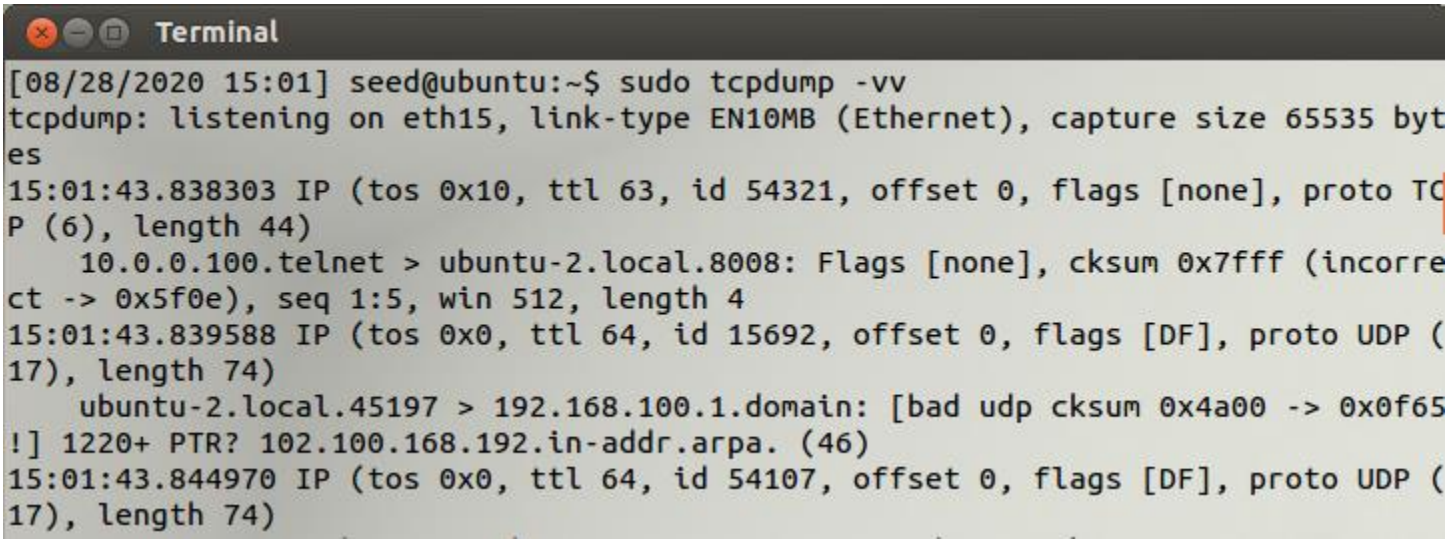
Run the program rawtcp.c on VM 1:

Terminal

```
[08/28/2020 14:55] seed@ubuntu:~/Desktop/CSS/lab1$ sudo ./rawtcp38 10.0.0.100 23
 192.168.100.102 8008
socket()-SOCK_RAW and tcp protocol is OK.
setsockopt() is OK
Using:::::Source IP: 10.0.0.100 port: 23, Target IP: 192.168.100.102 port: 8008.
Count #0 - sendto() is OK
Count #1 - sendto() is OK
Count #2 - sendto() is OK
Count #3 - sendto() is OK
Count #4 - sendto() is OK
Count #5 - sendto() is OK
Count #6 - sendto() is OK
Count #7 - sendto() is OK
Count #8 - sendto() is OK
Count #9 - sendto() is OK
Count #10 - sendto() is OK
Count #11 - sendto() is OK
Count #12 - sendto() is OK
Count #13 - sendto() is OK
Count #14 - sendto() is OK
Count #15 - sendto() is OK
Count #16 - sendto() is OK
Count #17 - sendto() is OK
Count #18 - sendto() is OK
Count #19 - sendto() is OK
[08/28/2020 14:57] seed@ubuntu:~/Desktop/CSS/lab1$
```

Verification on VM 2:





<u>Question 1</u>: Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

When sending a packet larger than its actual size, the additional data in the payload is a chunk of zeroes. Let say the TCP packet to google.com is (178.60.128.48). The payload is "ABC...XYZ", but the IP's *total_length* has been manually increased. The result is zero padding in the payload until completing the total length of the packet. So, the problem can be of *sendto* system call. This is the call that actually sends a packet on the socket. But this call also sets the *total_length* of the packet. If the *len* parameter on the *sendto* call has not been modified, so the packet's total length is overwritten to its original size when is sent.

<u>Question 2</u>: Using the raw socket programming, do you have to calculate the checksum for the IP header?

No, the computer generally the system automatically does this, or rather it fills it in.

<u>Question 3:</u> Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

In short this is how it is defined by the authorities who set networking rules. Due to the fact one can create custom packets that could prove detrimental to a network configuration