CS 315: Computer Networks Lab
Spring 2023-24, IIT Dharwad
Assignment-10
Wireshark Lab: ICMP
March 14, 2024

Lab Instructions

- Login to the Ubuntu OS on your machine. The login credentials are as follows:
  - Username: user
  - Password: 123456
- Mark your attendance in the attendance sheet before leaving the lab.
- Handle the lab resources with utmost care.
- Please go through the following exercises in today's lab.
- It is recommended that you complete all the following exercises during the lab slot itself.
- If you face any difficulties, please feel free to seek help online or from your peers or TAs.
- After finishing all exercises, please carry your solutions with you (via email/pen drive) for future reference, and delete the files from the desktop.

Introduction

In this lab, we'll explore several aspects of the ICMP protocol:

- ICMP messages generating by the Ping program;
- ICMP messages generated by the Traceroute program;
- the format and contents of an ICMP message.

Part-1: ICMP and Ping

Let's begin our ICMP adventure by capturing the packets generated by the Ping program. You may recall that the Ping program is a simple tool that allows anyone (for example, a network administrator) to verify if a host is live or not. The Ping program in the source host sends a packet to the target IP address; if the target is live, the Ping program in the target host responds by sending a packet back to the source host. As you might have guessed (given that this lab is about ICMP), both of these Ping packets are ICMP packets.

Do the following:

- Let's begin this adventure by opening the Windows Command Prompt application (which can be found in your Accessories folder).
- Start up the Wireshark packet sniffer, and begin Wireshark packet capture.

- Type either "*ping -c 10 www.iitdh.ac.in*" or *"ping -n 10 www.iitdh.ac.in"* or for any other addresses.
- When the Ping program terminates, stop the packet capture in Wireshark.

At the end of the experiment, your Command Prompt Window should look something like the below Figure-1. From this window we see that the source ping program sent 10 query packets and received 10 responses. Note also that for each response, the source calculates the round-trip time (RTT).

```
sysad@sysad-OptiPlex-7080:~$ ping -c 10 www.iitdh.ac.in
PING www.iitdh.ac.in (10.250.200.15) 56(84) bytes of data.
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=1 ttl=63 time=0.606 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=2 ttl=63 time=0.550 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=3 ttl=63 time=0.531 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=4 ttl=63 time=0.430 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=5 ttl=63 time=0.423 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=6 ttl=63 time=0.459 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=7 ttl=63 time=0.495 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=8 ttl=63 time=0.404 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=9 ttl=63 time=0.531 ms
64 bytes from www.iitdh.ac.in (10.250.200.15): icmp_seq=10 ttl=63 time=0.513 ms

--- www.iitdh.ac.in ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9172ms
rtt min/avg/max/mdev = 0.404/0.494/0.606/0.061 ms
```

Figure-1

Figure 2 provides a screenshot of the Wireshark output, after "icmp" has been entered into the filter display window. Note that the packet listing shows 20 packets: the 10 Ping queries sent by the source and the 10 Ping responses received by the source. Also note that the source's IP address is a private address (behind a NAT), the destination's IP address is that of the Web server at IITDh. Now let's zoom in on the first packet (sent by the client); in the figure below, the packet contents area provides information about this packet. We see that the IP datagram within this packet has protocol number 01, which is the protocol number for ICMP. This means that the payload of the IP datagram is an ICMP packet.

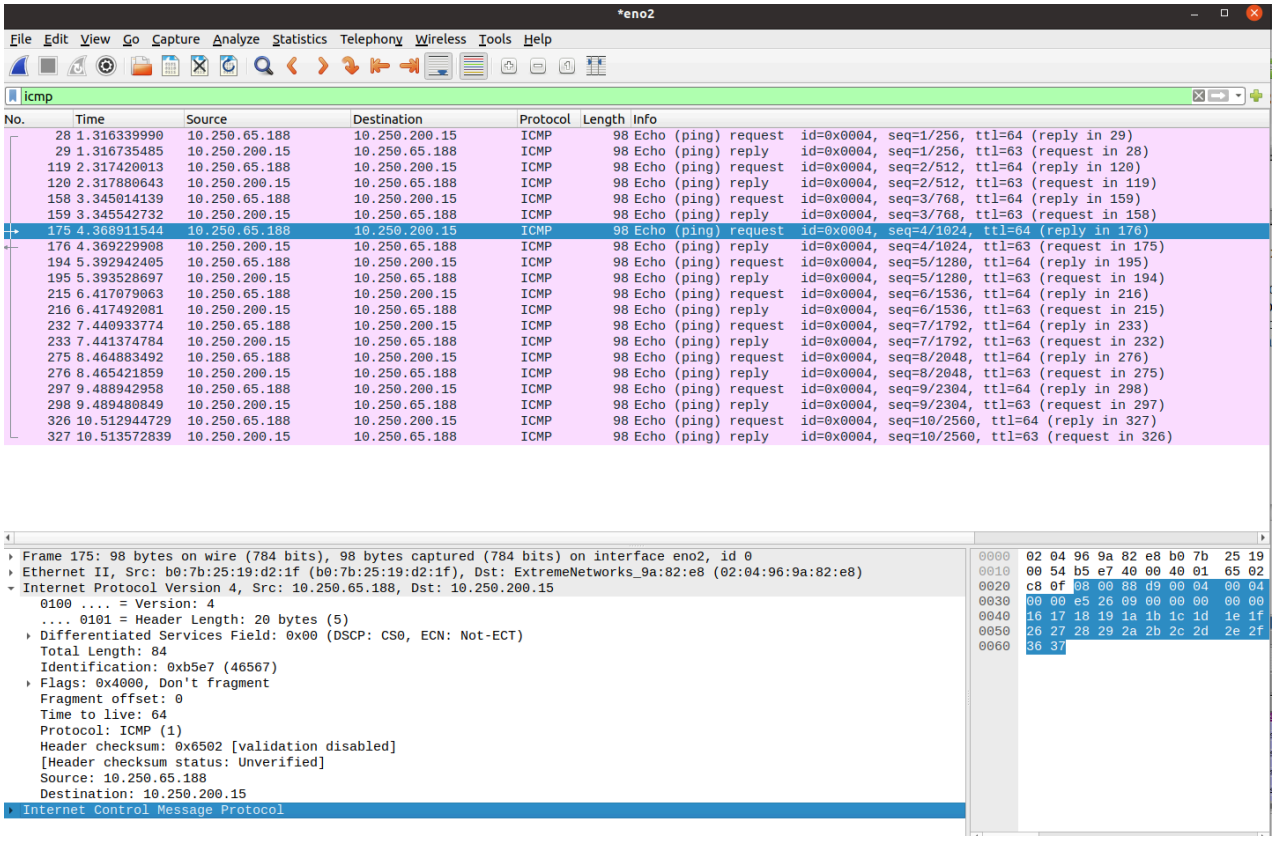

Figure-2

Figure 3 focuses on the same ICMP but has expanded the ICMP protocol information in the packet contents window. Observe that this ICMP packet is of Type 8 and Code 0 - a so-called ICMP "echo request" packet. Also note that this ICMP packet contains a checksum, an identifier, and a sequence number.

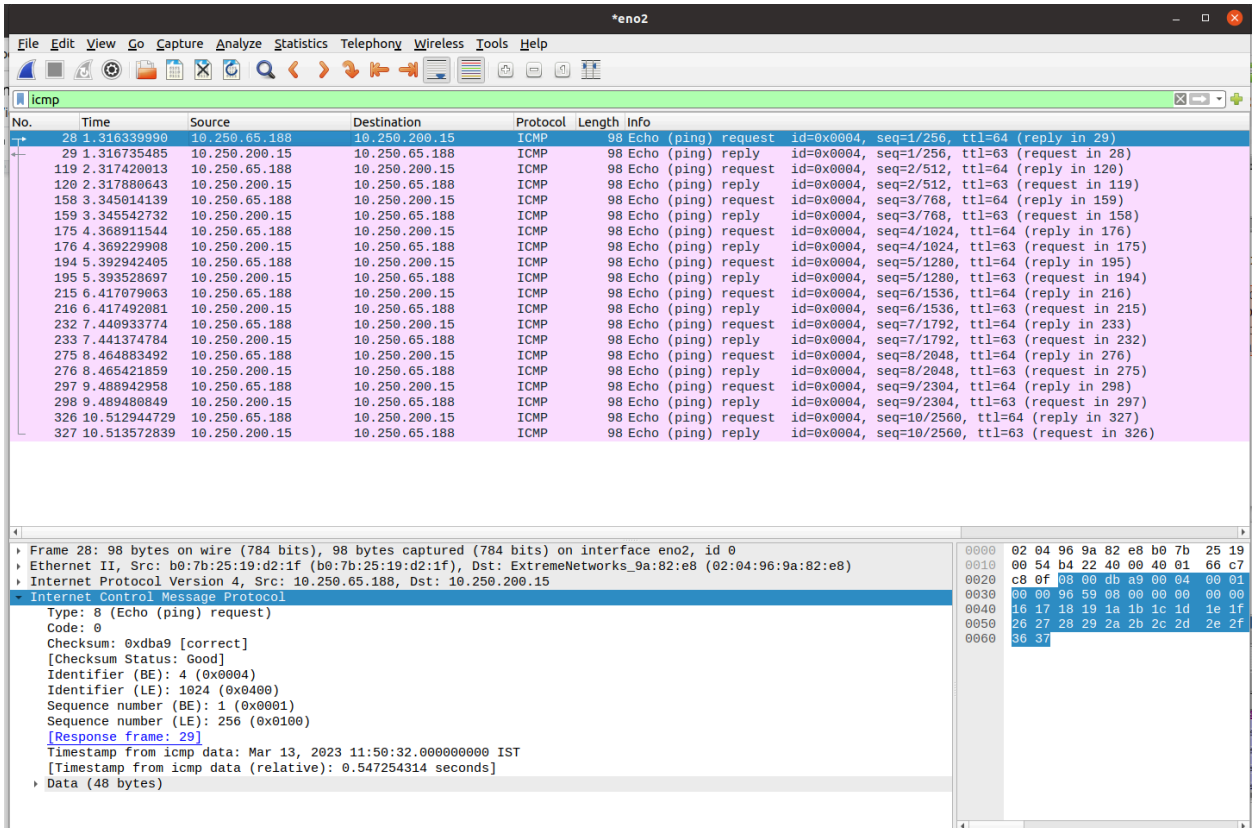

Figure-3

You should answer the following questions:

1. What is the IP address of your host? What is the IP address of the destination host?
2. Why is it that an ICMP packet does not have source and destination port numbers?
3. Examine one of the ping request packets sent by your host. What are the ICMP type and code numbers? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?
4. Examine the corresponding ping reply packet. What are the ICMP type and code numbers? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

Part-2: ICMP and Traceroute

Let's now continue our ICMP adventure by capturing the packets generated by the Traceroute program. You may recall that the Traceroute program can be used to figure out the path a packet takes from source to destination.

Traceroute is implemented in different ways in Unix/Linux/MacOS and in Windows. In Unix/Linux, the source sends a series of UDP packets to the target destination using an unlikely destination port number; in Windows, the source sends a series of ICMP packets to the target destination. For both operating systems, the program sends the first packet with TTL=1, the second packet with TTL=2, and so on. Recall that a router will decrement a packet's TTL value as the packet passes through the router. When a packet arrives at a router with TTL=1, the router sends an ICMP error packet back to the source.

Do the following:

- Let's begin by opening the Terminal.
- Start up the Wireshark packet sniffer, and begin Wireshark packet capture.
- Type traceroute -I www.google.com
- When the Traceroute program terminates, stop packet capture in Wireshark.

At the end of the experiment, your Command Prompt Window should look something like Figure 4.

```
sysad@sysad-OptiPlex-7080:~$ traceroute -I www.google.com
traceroute to www.google.com (142.250.183.164), 30 hops max, 60 byte packets
 1  _gateway (10.250.65.250)  0.659 ms  0.613 ms  0.602 ms
 2  firewall.iitdh.ac.in (10.250.209.251)  0.282 ms  0.249 ms  0.235 ms
 3  14.139.150.65 (14.139.150.65)  0.698 ms  0.691 ms  0.733 ms
 4  * * *
 5  * * *
 6  * * *
 7  10.119.73.122 (10.119.73.122)  38.107 ms  40.256 ms  40.224 ms
 8  72.14.195.128 (72.14.195.128)  43.128 ms  42.826 ms  42.791 ms
 9  142.251.227.213 (142.251.227.213)  40.892 ms  38.294 ms  38.199 ms
10  74.125.242.147 (74.125.242.147)  36.795 ms  36.823 ms  35.965 ms
11  72.14.232.50 (72.14.232.50)  53.025 ms  52.979 ms  53.155 ms
12  216.239.50.23 (216.239.50.23)  52.433 ms  52.199 ms  52.200 ms
13  108.170.248.193 (108.170.248.193)  54.819 ms  54.968 ms  54.927 ms
14  142.251.64.13 (142.251.64.13)  61.838 ms  61.722 ms  61.721 ms
15  bom07s32-in-f4.1e100.net (142.250.183.164)  57.357 ms  57.461 ms  57.367 ms
sysad@sysad-OptiPlex-7080:~$
```

Figure-4

Figure 5 displays the Wireshark window for an ICMP packet returned by a router.

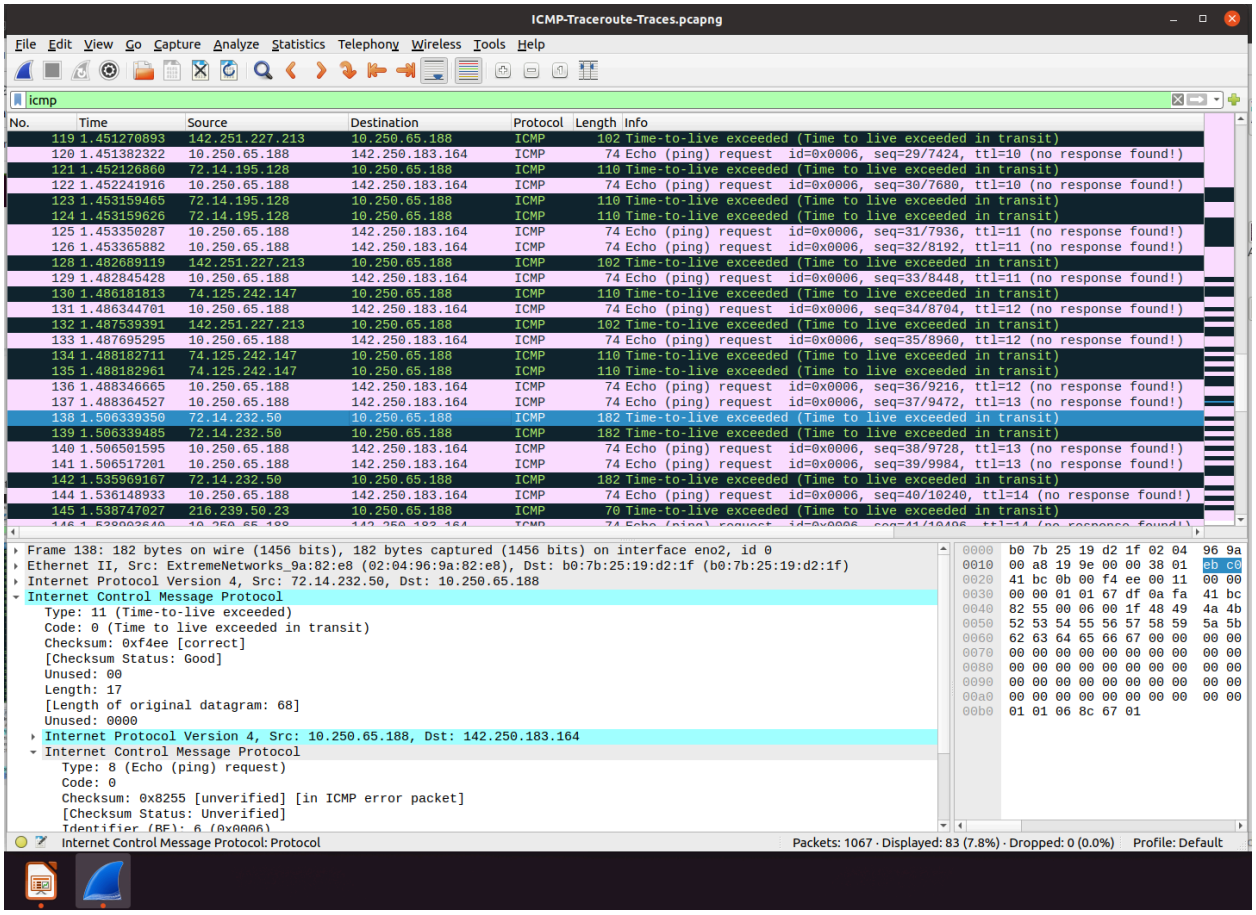

Figure-5

Answer the following questions:

1. What is the IP address of your host? What is the IP address of the target destination host?
2. If traceroute sent UDP packets, would the IP protocol number still be 01 for the probe packets? If not, what would it be?
3. Examine the ICMP echo packet in your screenshot. Is this different from the ICMP ping query packets in the first half of this lab? If yes, how so?
4. Examine the ICMP error packet in your screenshot. It has more fields than the ICMP echo packet. What is included in those fields?
5. Examine the last three ICMP packets received by the source host. How are these packets different from the ICMP error packets? Why are they different?
6. Within the traceroute measurements, is there a link whose delay is significantly longer than others?

Part-3: UDP Pinger

In this lab, you will learn the basics of socket programming for UDP in Python. You will learn how to send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout. Throughout the lab, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

You will first study a simple Internet ping server written in Python, and implement a corresponding client. The functionality provided by these programs is similar to the functionality provided by standard ping programs available in modern operating systems. However, these programs use a simpler protocol, UDP, rather than the standard Internet Control Message Protocol (ICMP) to communicate with each other. The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

You are given the complete code for the Ping server below. Your task is to write the Ping client.

Server Code

The following code fully implements a ping server. You need to compile and run this code before running your client program. *You do not need to modify this code.*

In this server code, 30% of the client's packets are simulated to be lost. You should study this code carefully, as it will help you write your ping client.

```
# UDP_pinger_server.py

# We will need the following module to generate randomized lost packets
import random
from socket import *

# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Generate random number in the range of 0 to 10
```

```
    rand = random.randint(0, 10)

    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)

    # Capitalize the message from the client
    message = message.upper()

    # If rand is less is than 4, we consider the packet lost and do not
respond
    if rand < 4:
        continue

    # Otherwise, the server responds
    serverSocket.sendto(message, address)
```

The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server simply capitalizes the encapsulated data and sends it back to the client.

Packet Loss

UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Because packet loss is rare or even non-existent in typical campus networks, the server in this lab injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.

Client code

You need to implement the following client program.
The client should send 10 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client to wait up to one second for a reply; if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network. You will need to look up the Python documentation to find out how to set the timeout value on a datagram socket.

Specifically, your client program should

1. send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection

first, since UDP is a connectionless protocol.)

2. print the response message from server, if any
3. calculate and print the round trip time (RTT), in seconds, of each packet, if server responses
4. otherwise, print "Request timed out"

During development, you should run the `UDPPingerServer.py` on your machine, and test your client by sending packets to *localhost* (or, 127.0.0.1). After you have fully debugged your code, you should see how your application communicates across the network with the ping server and ping client running on different machines.

Message Format:

The ping messages in this lab are formatted in a simple way. The client message is one line, consisting of ASCII characters in the following format:

Ping *sequence_number time*

where *sequence_number* starts at 1 and progresses to 10 for each successive ping message sent by the client, and *time* is the time when the client sends the message.

Submission Details

- Write your answers in a single doc/tex file, and submit its PDF named after your IIT Dharwad roll number, which contains all answers (with screenshots, if necessary).
- Submit the client code with your roll number prefixed as <Roll_number>_client.py
- Screenshots at the client verifying that your ping program works as required.