College of Engineering, Trivandrum

Network Lab Exam Report - Question 1-B

Divya Moncy

S6 CSE Roll no. 27

University Roll no. TVE16CS028

25 April 2019

Contents

1 Problem Statement 1-B **3**

2 Aim **4**

3 Theory **4**
- 3.1 User Datagram Protocol 4
- 3.2 Steps for server 4
- 3.3 Steps for client 4

4 Implementation **5**
- 4.1 Compilation and Execution 5

5 Program **5**

6 Output **6**
- 6.1 Test Run 1 6
- 6.2 Test Run 2 7
- 6.3 Test Run 3 8

7 Result **9**

1 Problem Statement 1-B

Write a UDP client program to the server program given below:

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
import sys
# 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()
        print message
        # If rand is less is than 4, we consider the packet lost and
        do not respond
        if rand < 4:
                print "Oops, dropped it!"
                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.

You need to implement the 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 wait up to two seconds for a reply; if no reply is received within two seconds, your client program should assume that the packet was lost during transmission across the network.

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 otherwise, print Request timed out.
4. Have your client report the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate the packet loss rate (in percentage).

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

Ping sequenceNumber time

Where sequenceNumber 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.

2 Aim

To write a client program to the given server program using UDP sockets, send 10 packets to the server and receive its response with a timeout of 2 seconds. If timed out, it should be printed or else the roundtrip time should be calculated. The average, minimum, maximum roundtrip time and package loss percentage should be calculated.

3 Theory

3.1 User Datagram Protocol

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

3.2 Steps for server

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

3.3 Steps for client

1. Create UDP socket.
2. Send message to server.

3. Wait until response from server is recieved.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

4 Implementation

In the client program, a UDP socket is created and the timeout is set to 2 seconds using settimeout() function. Difference between sending times and receiving times are appended to a list x. The times are obtained using functions from datetime package. The losscount is incremented whenever a timeout exception is thrown. At the end, minimum, maximum and average roundtrip times are calculated. Also the packet loss rate is calculated.

4.1 Compilation and Execution

Server and client programs are run simultaneously on two terminals.

python server.py
python client.py

5 Program

client.py

```
import socket
import sys
import datetime
UDP_IP="127.0.0.1"
UDP_PORT=12000
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.settimeout(2)    #setting the timeout
message=""
x=[]                  #list to store the roundtrip times
losscount=0
for i in range(1,11):
        sendtime=datetime.datetime.now().time().microsecond
        #getting the sending time
        message="Ping "+str(i)+" "+str(sendtime)
        sock.sendto(message,(UDP_IP,UDP_PORT)) #sending the packet
        try:

                (res,(ip,port))=sock.recvfrom(len(message))
                #receiving the packet
```

```
                rectime=datetime.datetime.now().time().microsecond-sendtime
                #calculating roundtrip time
                print("from "+str(ip)+": time="+str(rectime))
                x.append(rectime)  #adding roundtrip time to list
        except:
                losscount+=1  #updating lost packet count
                print(str(i)+": Timed out")
min=x[0]
max=x[0]
sum=x[0]
for j in range(1,len(x)):
        if x[j]<min:
           min=x[j]
        if x[j]>max:
           max=x[j]
        sum+=x[j]
print("min="+str(min)+", max="+str(max)+", avg="+str(sum/len(x)))
#calculating minimum, maximum and average
print("Package loss rate= "+str(losscount*100.0/10))
#calculating packet loss rate
sock.close()
```

6 Output

6.1 Test Run 1

Server

```
adminisrtator@pc-16:~$ python server.py
PING 1 646371
PING 2 646817
PING 3 647095
Oops, dropped it!
PING 4 649235
Oops, dropped it!
PING 5 651423
PING 6 651732
PING 7 651979
Oops, dropped it!
PING 8 654096
Oops, dropped it!
PING 9 656263
Oops, dropped it!
```

```
PING 10 658450
Oops, dropped it!
```

Client

```
adminisrtator@pc-16:~$ python client.py
from 127.0.0.1: time=388
from 127.0.0.1: time=231
3: Timed out
4: Timed out
from 127.0.0.1: time=256
from 127.0.0.1: time=196
7: Timed out
8: Timed out
9: Timed out
10: Timed out
min=196, max=388, avg=267
Package loss rate= 60.0
```

6.2 Test Run 2

Server

```
adminisrtator@pc-16:~$ python server.py
PING 1 608612
PING 2 608949
Oops, dropped it!
PING 3 611088
Oops, dropped it!
PING 4 613261
PING 5 613559
PING 6 613808
Oops, dropped it!
PING 7 613946
PING 8 614177
Oops, dropped it!
PING 9 616314
PING 10 616613
```

Client

```
adminisrtator@pc-16:~$ python client.py
from 127.0.0.1: time=277
2: Timed out
3: Timed out
```

```
from 127.0.0.1: time=248
from 127.0.0.1: time=200
6: Timed out
from 127.0.0.1: time=192
8: Timed out
from 127.0.0.1: time=248
from 127.0.0.1: time=195
min=192, max=277, avg=226
Package loss rate= 40.0
```

6.3 Test Run 3

Server

```
adminisrtator@pc-16:~$ python server.py
PING 1 571254
Oops, dropped it!
PING 2 573448
PING 3 573773
Oops, dropped it!
PING 4 575916
PING 5 576219
PING 6 576458
Oops, dropped it!
PING 7 578593
Oops, dropped it!
PING 8 580781
PING 9 581096
PING 10 581353
```

Client

```
adminisrtator@pc-16:~$ python client.py
1: Timed out
from 127.0.0.1: time=272
3: Timed out
from 127.0.0.1: time=259
from 127.0.0.1: time=197
6: Timed out
7: Timed out
from 127.0.0.1: time=254
from 127.0.0.1: time=206
from 127.0.0.1: time=207
min=197, max=272, avg=232
Package loss rate= 40.0
```

7 Result

The client program using UDP sockets was written in Python and run on Ubuntu 16.04 kernel. The packets were sent to the server and the responses were displayed. The roundtrip times were displayed in microseconds. The output was verified.