Programming Assignment 2

CST 311-30, Introduction to Computer Networks, Online

READ INSTRUCTIONS CAREFULLY BEFORE YOU START THE ASSIGNMENT.

Assignment must be submitted electronically to Canvas by 11:59 p.m. on the due date. Late assignments will not be accepted.

This assignment requires you to work with teams assigned to you on the Team Assignment Document in Week 1. You must also adhere to the instructions on the Programming Process document.

Follow the steps below to develop and write the client part of a client server application. The naming convention of the file should be PA2_your_last_names_in_alphabetic_order.py. Put your names in the program as well. Your client program must work with the server program given to you below. Your program must have sufficient comments to clearly explain how your code works.

This assignment is worth 150 points. The grading objectives for the assignment are given below.

UDP Pinger

In this assignment, 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 assignment, 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 Starter Code

The following code fully implements a ping server. You can use the server code given below as starter code for the assignment. You need to compile and run this code before running your client program.

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.

```
# 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))

pingnum = 0

while True:

    # Count the pings received

    pingnum += 1

    # 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)

    # If rand is less is than 4, and this not the
```

```
    # first "ping" of a group of 10, consider the

    # packet lost and do not respond

    if rand < 4 and pingnum % 10 != 1:

        continue

    # Otherwise, the server responds

    serverSocket.sendto(message, address)
```

The server sits in an infinite loop listening for incoming UDP packets. When the first packet, or a succeeding packet when a randomized integer is greater than or equal to 4, comes in, 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 assignment 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

The client should send 10 pings to the server. See the Message Format below for each ping to be sent to the Server. Also print the message that is sent 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. If the packet is lost, print "Request timed out".

The program must calculate the round-trip time for each packet and print it out individually. Have the client print out the information similar to the output from doing a normal ping command – see sample output below. Your client software will need to determine and print out the minimum, maximum, and average RTTs at the end of all pings from the client along with printing out the number of packets lost and the packet

loss rate (in percentage). Then compute and print the estimated RTT, the DevRTT and the Timeout interval based on the RTT results.

The initial estimated RTT is just the first Sample RTT – the results of your first ping. The initial DevRTT is the first Sample TTT divided by 2. The initial Timeout interval is set to 1 minute. The formulas for calculating the estimated RTT, DevRTT, and Timeout interval for each succeeding ping are contained in the slides on the last page of this document and on pages 242-243 of the textbook.[1]

Execution and Testing

You are to run and test your program in a two host, one switch mininet emulation, using delay parameters that match the real world. The client program should be run on one host (h1) and the server program should be run on the second host (h2). To set up the test environment, start mininet with the basic topology, and set speed of links to 100 Mbps and delay to 25 ms per link. (By setting delay to 25 ms, the RTT will be about 100 ms – this is the approximate RTT between San Francisco and Boston.) Here is the command to set up the test environment:

mininet@mininet-vm:~$ sudo mn --link tc,bw=100,delay=25ms

Sample output of ping to Google.com

Here is a sample output from a ping of Google 4 times:

Pinging google.com [216.58.195.78] with 32 bytes of data:
Reply from 216.58.195.78: bytes=32 time=13ms TTL=54
Reply from 216.58.195.78: bytes=32 time=14ms TTL=54
Reply from 216.58.195.78: bytes=32 time=16ms TTL=54 Reply
from 216.58.195.78: bytes=32 time=13ms TTL=54 Ping
statistics for 216.58.195.78:
 Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round
trip times in milli-seconds:
 Minimum = 13ms, Maximum = 16ms, Average = 14ms

Expected output of your program

Server Output:

```
mininet@mininet-vm:~/PA1$ python Server.py
Waiting for Client....

PING 1 Received
Mesg rcvd: Ping1
Mesg sent: PING1

PING 2 Received
Mesg rcvd: Ping2
Mesg sent: PING2

PING 3 Received
Mesg rcvd: Ping3
Mesg sent: PING3

PING 4 Received
Mesg rcvd: Ping4
Mesg sent: PING4

PING 5 Received
Mesg rcvd: Ping5
Mesg sent: PING5

PING 6 Received
Mesg rcvd: Ping6
Mesg sent: PING6


Packet was lost.


PING 7 Received
Mesg rcvd: Ping8
Mesg sent: PING8


Packet was lost.


Packet was lost.

```

Client Output :

(Values in sample below are smaller than your expected output)

```
mininet@mininet-vm:~/PA1$ python Client.py

Mesg sent: Ping1
Mesg rcvd: PING1
Start time: 1.58863081852e+12
Return time: 1.58863081852e+12
PONG 1 RTT: 1.61328125 ms

Mesg sent: Ping2
Mesg rcvd: PING2
Start time: 1.58863081852e+12
Return time: 1.58863081852e+12
PONG 2 RTT: 1.41088867188 ms

Mesg sent: Ping3
Mesg rcvd: PING3
Start time: 1.58863081852e+12
Return time: 1.58863081853e+12
PONG 3 RTT: 1.01196289062 ms

Mesg sent: Ping4
Mesg rcvd: PING4
Start time: 1.58863081853e+12
Return time: 1.58863081853e+12
PONG 4 RTT: 0.6279296875 ms

Mesg sent: Ping5
Mesg rcvd: PING5
Start time: 1.58863081853e+12
Return time: 1.58863081853e+12
PONG 5 RTT: 0.220947265625 ms

Mesg sent: Ping6
Mesg rcvd: PING6
Start time: 1.58863081853e+12
Return time: 1.58863081853e+12
PONG 6 RTT: 0.68798828125 ms

Mesg sent: Ping7
No Mesg rcvd
PONG 7 Request Timed out

Mesg sent: Ping8
Mesg rcvd: PING8
Start time: 1.58863082053e+12
Return time: 1.58863082053e+12
PONG 8 RTT: 0.35595703125 ms

Mesg sent: Ping9
No Mesg rcvd
PONG 9 Request Timed out

Mesg sent: Ping10
No Mesg rcvd
PONG 10 Request Timed out

Min RTT:         0.220947265625 ms
Max RTT:         1.61328125 ms
Avg RTT:         0.592895507812 ms
Packet Loss:     30.0%
Estimated RTT:   0.448582238518 ms
Dev RTT:         0.283880939009 ms
Timeout Interval:1.58410599455 ms
mininet@mininet-vm:~/PA1$
```

Grading Objectives

1. **(30 points)** You must complete this program in the Mininet VM. The screenshots for the running code and the results in the Output checker file must come from executing your code on the mininet VM.
2. **(5 points)** Ping messages must be sent using UDP. Server must change the received message to uppercase before sending the message back to the client.
3. **(5 + 5 points)** Print the request from client and response from server messages on both the client and server machines.
4. **(8 x 5 points)** Calculate and print the following on the client side in milliseconds:
    a. Round trip time (RTT) - If the server doesn't respond, print "Request timed out".
    b. Minimum RTT
    c. Maximum RTT
    d. Average RTT (Leave out lost packets from average calculation)
    e. Estimated RTT. Consider alpha = 0.125. (Look at slides at the end for formulae.)
    f. Deviation RTT. Consider beta = 0.25. (Look at slides at the end for formulae.)
    g. Timeout Interval (Look at slides at the end for formulae.)
    h. Packet loss percentage
5. **(5 points)** You must write the client code to do the assignment with the calculations in 4 above without the use of a list (or array). Find an efficient and effective use of storage and program speed.
6. **(5 points)** Programs must be well documented.
7. **(5 points)** Submission files are in order. (Look at the "What to hand in" section.)
8. **(50 points)** Teamwork grade: (50 points) Each team member will grade each other teammate out of 10 points during peer evaluation. I will average all team members' grades and scale it to get your teamwork grade out of 50 points. Note that 30% of your grade will come from your teamwork and team member evaluations.

What to Hand in

1. You will hand in the complete client and server code to iLearn. (Server starter is provided. You must modify the server code to print messages as shown in the Expected output section.)
2. Minutes of the 3 meetings.

3. Screenshots of server and client-side output. Please provide screenshots in one pdf file.
4. Fill in columns B and C with RTTs and lost packets as indicated in the file - Output Checker. Your outputs in your screenshots must match the outputs calculated in the Output Checker.
5. One submission per Team.

Footnotes:

1. The estimated RTT, DevRTT, and Timeout intervals are actually only used with TCP. These calculations are included in this programming assignment to give you experience working with these parameters.
2. **RTT Slides**:

TCP estimated round trip time (RTT)

$$\text{EstimatedRTT} = (1-\alpha)*\text{EstimatedRTT} + \alpha*\text{SampleRTT}$$

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$
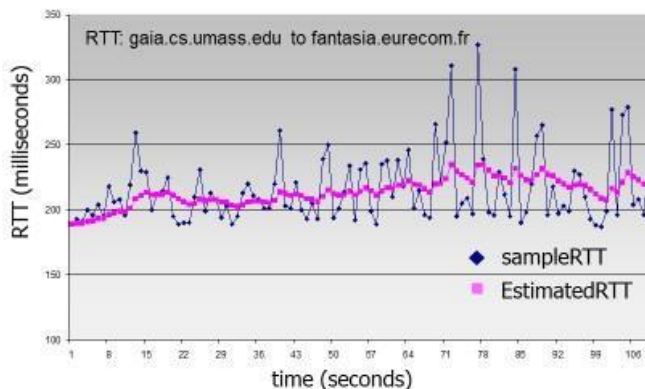
TCP estimated RTT deviation, timeout

- **timeout interval:** **EstimatedRTT** plus "safety margin"
  - large variation in **EstimatedRTT** -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

  `DevRTT = (1-β)*DevRTT +`
               `β*|SampleRTT-EstimatedRTT|`
               `(typically, β = 0.25)`

  `TimeoutInterval = EstimatedRTT + 4*DevRTT`

                             estimated RTT   "safety margin"