Computer Networks Fall 2017
Project 1: Implementing a Ping Client

1 Project Overview

In this lab, you will learn the basics of socket programming for User Datagram Protocol (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 the 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.

2 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 clients packets are simulated to be lost. You should study this code carefully, as it will help you write your ping client.

```
# UDPPingerServer.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.

3 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.

4 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 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 milliseconds, of each packet, if server responses
4. otherwise, print Request timed out. Please use this exact string.

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.

5 Testing

You may do your work on any machines you like, but you should make sure to test on the CSIF machines as this is where we will do our grading. If you receive an error that 'address is already in use' when launching the server, there is already a process running that is bound to the same port as the server. In that case, you should adjust serverPort on both your client and server to a different value between 10000 and 65535. Don't forget to change it back before submitting.

6 Message Format

The ping messages that your client sends should be formatted as follows:

```
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 (as measured in milliseconds). Make sure to use this exact string.

7 Deliverables

You will hand in the complete client code for this program in a file named UDPPingClient.py. Please include the names of your group members in a comment on the second line of the program.

7.1 How To Hand In

Assignments should be submitted via handin on the CSIF machines. The command to submit is:

```
$ handin cs152a hw1 UDPPingClient.py
Sample Output
The output should look like the following example:
Ping 1 2013-10-18 T 10:45 UTC
PING 1 2013-10-18 T 10:45 UTC
RTT: 0.454

Ping 2 2013-10-18 T 10:46 UTC
Request timed out
```

The above is an example output. The first string Ping 1 2013-10-18 T 10:45 UTC is the message that client sends to the server. The second string PING 1 2013-10-18 T 10:45 UTC is the returned capitalized message from the server. RTT is the round-trip time and is give in milliseconds.

For abbreviation of the days of the week you can use the following:

```
Monday - M
Tuesday - T
Wednesday - W
Thursday - R
Friday - F
Saturday - S
Sunday - U
```

7.2 Group Size

For this assignment you will work in groups of size of 2.

7.3 Deadline

The project is due Tuesday 10/31 by 5:00 PM. Late submissionsa will not be accepted.