Lab 2: UDP Pinger Lab

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 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.

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.

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *
import time
import hashlib
import sys

def serve(port):
  # 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(('', port))

  while True:
    # Generate random number in the range of 0 to 10
    try:
      rand = random.randint(0, 10)

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

      s_time = time.time()

      # If rand is less is than 4, we consider the packet lost and do not
respond
```

```
        if rand < 4:
          continue

        m = message.decode().split()
        seq = m[1]
        c_time = m[2]
        h = hashlib.md5('seq:{0},c_time:{1},s_time:{2},key:{3}'.format(seq,
c_time, str(s_time), 'randomkey').encode()).hexdigest()

        print(m)
        resp = 'Reply {0} {1} {2} {3}\n'.format(seq, c_time, str(s_time), h)

        # Otherwise, the server responds
        serverSocket.sendto(resp.encode(), address)
      except KeyboardInterrupt:
        serverSocket.close()
        sys.exit()
      except:
        continue

if __name__ == '__main__':
  serve(12000)
```

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 generates a reply message 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 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) collect and print the response message from server, if any
(3) calculate and print the round trip time (RTT), in seconds, of each packet, if server responds
(4) otherwise, log and 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 should be 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 (in floating-point seconds) when the client sends the message.

The reply messages from the server includes the data it received from the client, along with some additional data. The server reply messages consist of ASCII characters in the following format:

Reply *sequence_number client_time server_time message_digest*

where *sequence_number* and *client_time* are the same strings as received from the client ping message, the *server_time* is the time (in floating-point seconds) when the server received the ping message, and the *message_digest* is an MD5 hash (encoded into hex) generated from all the above data and a secret key. Note that the key that will be used during grading will be different from the key used in the sample server code above.

Client Skeleton Code

```
from socket import *
import time
import sys

def ping(host, port):
  resps = []

  for seq in range(1,11):
    # Send ping message to server and wait for response back
    # On timeouts, you can use the following to add to resps
    # resps.append((seq, 'Request timed out', 0))
   # On successful responses, you should instead record the server
response and the RTT (must compute server_reply and rtt properly)
    # resps.append((seq, server_reply, rtt))

    #Fill in start
```

```
    #Fill in end

  return resps

if __name__ == '__main__':
  resps = ping('127.0.0.1', 12000)
  print(resps)
```

What to Hand in

You will hand in the complete client code in a file called client.py and a PDF report. Your report should contain a line-by-line explanation of your code, accompanied by appropriate code snippets or screenshots of your code.

Optional Exercises

1. Currently, the program calculates the round-trip time for each packet and prints it out individually. Modify this to correspond to the way the standard ping program works. You will need to 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).
2. Another similar application to the UDP Ping would be the UDP Heartbeat. The Heartbeat can be used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped. Implement the UDP Heartbeat (both client and server). You will need to modify the given `UDPPingerServer.py`, and your UDP ping client.

Considerations for Gradescope

In order for your submission to work with the Gradescope autograder:

1. Make sure you submit your client code in a file called client.py in the top-level directory of your submission
2. client.py must contain a function called ping that matches the signature used in the skeleton code above
3. The ping function must return a list of triples. The list type should be [(integer, string, float)], which represent sequence number, server response, and RTT respectively.
4. If you are doing Optional Exercise #1, write this version of the client to a **different** file than client.py and indicate in your report what file to check.
5. If you are doing Optional Exercise #2, write your versions of the server and client to **different** files than client.py, UDPPingerServer.py, and your file for Optional Exercise #1 (if any). Indicate in your report which files contain your heartbeat client and server.