CS5700 Project 1 Code Explanation

UDPPingerServer.py

```
# Semaa Amin
# Sources: Textbook: Computer Networking: A Top-Down Approach; 8th edition;
Kurose & Ross


# We will need the following module to generate randomized lost packets
import random
import random
# From textbook 8th edition, page 156:
# The socket module forms the basis of all network communications in
Python. By
# including this line, we will be able to create sockets within our
program.
from socket import *

# Create a UDP socket
# Creating an IP address using AF_INET according that is corresponds to the
socket we are creating
# according to the Python documentation
https://docs.python.org/3/library/socket.html
# Using SOCK_Dgram for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM) # complete this line

# Assign IP address and port number to socket
# using group number 7 which corresponds to Project Group H
# we are providing a string containing either the IP address of the server
(e.g., "128.138.32.126")
# or the hostname of the server, in this case the problem asks us to use
the hostname, then a
# DNS lookup will automatically be performed to get the IP address when it
is run
# we add a line into our Python program after we create the socket to
associate
# a specific port number (15007 in this case) to this UDP socket via the
socket bind() method:
serverSocket.bind(('localhost', 15007))  # binds socket to 'localhost',
port 15000+X, where X is your group number
print("Started UDP server on port 15007")
while True:
# Generate random number in the range of 0 to 10 rand =
# using the random() method imported from random above
   rand = random.randint(0, 10)

# Receive the client packet along with the address it is coming from
message, address = # complete this line
# From page 158-159 of textbook
# When a packet arrives at the server's socket, the packet's data is put
into the variable message and the
```

```
# packet’s source address is put into the variable address.
# The variable address contains both the client’s IP address and the
client’s port number.
# Here, UDPServer will make use of this address information, as it provides
a return
# address, similar to the return address with ordinary postal mail. With
this source
# address information, the server now knows to where it should direct its
reply.
# The standard/preferred UDP packet size is 1024
# The method recvfrom() also takes the buffer size 1024*0.7 as input
    message, address = serverSocket.recvfrom(int(1024*0.7))
# Capitalize the message from the client message =
# This line takes the line sent by the client and,after converting the
message to a string,
# uses the method upper() to capitalize it.
    message = message.upper()
# complete this line
# If rand is less than 4, we consider the packet lost and do not respond
    if rand < 4:
        continue
# Otherwise, the server responds and sends the message to the client
#complete this line
# This last line attaches the client’s address (IP address and port number)
to the capitalized
# message (after converting the string to bytes), and sends the resulting
packet into
# the server’s socket. Internet will then deliver the packet to this client
address.
# After the server sends the packet, it remains in the while loop, waiting
for another UDP packet to arrive

    serverSocket.sendto(message, address)


#SemaaAmin-completed-10302021
```

UDPPingerClient.py

```
# Semaa Amin
# Sources: Textbook: Computer Networking: A Top-Down Approach; 8th edition;
Kurose & Ross
# Sources: Python documentation = https://docs.python.org/3/


# importing socket
import socket
# importing time module
import time

# similar to the Server side, but we are not importing all from the socket
module
# here we are writing the socket and defining every variable
# Creating an IP address using AF_INET according that is corresponds to the
socket
# we are creating according to the Python documentation
https://docs.python.org/3/library/socket.html
# Using SOCK_Dgram for UDP packets
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Assign IP address and port number to socket
serverAddress = ('localhost', 15007)
# Here we are setting the timeout using the settimeout() method imported
from
# the time module
# the timeout applies to a single call to socket
# a note on settimeout() method: from
https://docs.python.org/3/library/socket.html#socket.socket.settimeout
# The value argument can be a nonnegative floating point number expressing
seconds, or None.
# If a non-zero value is given, subsequent socket operations will raise a
timeout exception
# if the timeout period value has elapsed before the operation has
completed.
# If zero is given, the socket is put in non-blocking mode.
# If None is given, the socket is put in blocking mode.
# Here we are putting a 1 second timeout
clientSocket.settimeout(1)
# initializing the list where we will keep all the RTTs from the the pings
pingRTT = []

# Here we use error handling method try
try:
    # Implementing a for loop that starts from 1 and goes to 10 inclusive
    for i in range(1, 11):
       #Here we are recording the time of the execution from start to
finish
       # setting execution to start
       startTime = time.time()
       # Here we are assigning to a variable message the following with the
time
       # This makes it like the format the assignment wants whereby:
       # Ping sequence_number time
       # Python time method ctime() converts to time expressed in seconds
       # since the epoch to a string representing local time
       message = "Ping #" + str(i) + " " + time.ctime(startTime)
       # We enter another Error Handling method try
       try:
           # We set the sent variable to message.
           # We use the method sendto() to send datagrams to a UDP socket.
           # We use the encode() method to put convert the message into
string
           # this line attaches the server’s address (IP address and port
number) to the
           # message (after converting the string to bytes), and sends the
resulting packet into
           # the client’s socket. Internet will then deliver the packet to
this server address.
           sent = clientSocket.sendto(message.encode(), serverAddress)
           # We are printing the sent message, which again contains the
Ping type, the sequence# out of 10 and the time
           print("Sent " + message)
           # Here we are receiving a message and the address.
           # We set the it to receive at most 4096 bytes
           # or blocking if there isn't any data that is waiting to be read
           data, server = clientSocket.recvfrom(4096)
           # Printing and stringifying the message
           print("Received " + str(data))
           # Here we are taking note of the end time
           endTime = time.time()
           # This is the calculation
           rtt = endTime - startTime
           pingRTT.append(rtt)
           # Here we are printing the RTT time in seconds
           print("RTT: " + str(rtt) + " seconds")
       # Here we are setting the program to throw an exception if there was
       # a time lag for more than 1 second
       except socket.timeout:
           # This is what will print out if an exception is thrown
           print("#" + str(i) + " Requested Time Out")
# after the success of the try error handling executing, we use the method
finally
finally:
   # We close the socket
   print("**Closing Socket** \n")
   clientSocket.close()
   # We print out the summary statements required for this Project
   print("Here are the stats from all the pings from the client: \n")
```

```
    print("The maximum RTT is: \n" + str(max(pingRTT)) + " seconds\n")
    print("The minimum RTT is: \n" + str(min(pingRTT)) + " seconds\n")
    print("The Average of RTTs is: \n" + str(sum(pingRTT)/len(pingRTT)) + "
seconds\n")
    print("The total number of RTTs are: \n" + str(len(pingRTT)) + " out of
10")
    print("The packet loss rate at the client side is: " + str((10 -
len(pingRTT))*10) + " %\n")
    print("Here are all the RTTs detected in one list for reference:")
    print(pingRTT)
    print("\n")


#SemaaAmin-completed-10302021
```