ELEC 331 Computer Communications
Problem Set 2

Due date: Thursday morning, October 17, 2024, at 9 am

Problem 1

Question P5 on page 169 of the eText (Kurose and Ross, 8th edition).

Problem 2

Questions P13 and P14 on page 171 of the eText (Kurose and Ross, 8th edition).

Problem 3

In this assignment, you will develop a TCP server and a TCP client. Specifically, your TCP client will (i) collects a string variable from the keyboard input; (ii) send the string variable to the TCP server; and (iii) receive the response from the TCP server over the TCP connection. Your TCP server will (i) create a connect socket when contacted by the TCP client; (ii) receive the string variable sent by the TCP client; (iii) generate a reward based on the string variable sent by the TCP client. The rules for determining the reward are shown in Table 1; and (iv) inform the TCP client about the reward by sending a response over the TCP connection to the TCP client.

Note that, when determining the reward based on Table 1, you need to differentiate between uppercase letters and lowercase letters. For example, the uppercase 'C' should be assigned with a reward of 1 whereas the lowercase 'c' should be assigned with a reward of -1.

For this assignment, you may use `TCPClient.py` and `TCPServer.py` that are presented in Section 2.7.2 of the eText (Kurose and Ross, 8th edition) as the skeleton codes. You can also write your own codes.

What to Hand in

For this problem, you will hand in the codes for both your TCP client and TCP server. The reward received from the TCP server should be **displayed at your TCP client** (e.g., using Python `print` function). You need to provide the screenshots at the TCP client verifying that your codes works as required. There is no output requirement for your TCP server. The codes must include sufficient comment statements.

Table 1: Reward

String variable sent by the TCP Client	Reward
'A'	3
'B'	2
'C'	1
Otherwise	-1

Problem 4

In this problem, 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 problem, 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 the 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 data packet 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, some of the client's packets are simulated to be lost. You should study this code carefully, as it will help you implement 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 49
    rand = random.randint(0, 49)
    # 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 18, we consider the packet lost and do not
        respond
    if rand < 18:
        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 18, 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. The two processes normally run on two hosts on the Internet, but in this problem, you develop/debug them both running on one machine (using localhost or 127.0.0.1). Because packet loss is rare or even non-existent using localhost, the server in this problem 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.

(a) Calculate the packet loss based on the provided server code.

Client Code

(b) You need to implement the following client program.

The client should send 20 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. 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

- send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)
- print the response message from server, if any
- calculate and print the round trip time (RTT), in seconds, of each packet, if server responses
- otherwise, print "Request timed out"

Message Format

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

Ping *sequence_number time*

where *sequence_number* starts at 1 and progresses to 20 for each successive ping message sent by the client, and *time* is the time when the client sends the message.

What to Hand in

For part (b) of this problem, you will hand in the complete client code and screenshots at the client verifying that your ping program works as required, i.e., if the client receives a message from the server, it prints the received message and in the next line, it prints the RTT in seconds. Otherwise, it prints "Request time out". The code must include sufficient comment statements.

Problem 5

Wireshark Lab: Domain Name System (DNS). Details can be found in the eText and book's website, www.pearsonhighered.com/cs-resources. Please note that only DNS Wireshark lab of version 8.1 is acceptable. Also, when you hand in your assignment, provide the screenshots and annotate the output so that it is clear where in the output you are obtaining the information for your answer.