CSCI 3171: ASSIGNMENT-1

Given: May 27th, 2021
Due: June 10th, 2021 (by midnight)

Objective:
With this assignment, you will start to get familiar with concepts of the Application Layer or, in other words, Layer-5 of the Internet Protocol Stack via Socket Programming.

Please read chapter 2: "Application Layer" from the textbook – "Computer Networking: A Top-Down Approach" by Kurose & Ross – and go over your lecture notes (slides and in-class discussions). Please see the course page on Brightspace for the recordings of the lectures (if you missed any) as well as the link to the textbook or the free e-book. Please also go over the related lab materials! These will help you to get started for this assignment.

Programming Assignment:
Please choose one of the options below for socket programming. Please note that you can make ***use of the sample codes given in the textbook or sample codes from publicly available*** programming language tutorials and/or web sites***. If you choose to make use of such sample codes, then please do not forget to cite the resources you use!*** Furthermore, all the examples below are given in the Python programming language (similar to the labs). If you prefer to use a different programming language of your choice, that is fine. However, debugging your code and making it work is your own responsibility. ***Please do not forget, the onus is on you.*** Last but not the least, ***the assignment will be marked out of 100% as given below***.

Option 1: Web Server (100%)

In this option, you will practice the basics of socket programming for TCP connections in Python: how to create a socket, bind it to a specific address and port, as well as send and receive an HTTP packet. You will also practice some basics of the HTTP header format.

Web Server Part-A: (80%)
You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client.

Code:
Below you will find the skeleton code for the Web server. You are to complete the skeleton code. The places where you need to fill in code are marked with #Fill in start and #Fill in end. Each place may require one or more lines of code.

Running the Server:
Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Also note the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80. Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

Skeleton Python Code for the Web Server:
```
#import socket module
from socket import *
import sys # In order to terminate the program
serverSocket = socket(AF_INET, SOCK_STREAM)
#Prepare a sever socket
#Fill in start
#Fill in end
while True:
#Establish the connection
print('Ready to serve...')
connectionSocket, addr = #Fill in start #Fill in end
try:
message = #Fill in start #Fill in end
filename = message.split()[1]
f = open(filename[1:])
outputdata = #Fill in start #Fill in end
#Send one HTTP header line into socket
#Fill in start
#Fill in end
#Send the content of the requested file to the client
for i in range(0, len(outputdata)):
connectionSocket.send(outputdata[i].encode())
connectionSocket.send("\r\n".encode())
connectionSocket.close()
except IOError:
#Send response message for file not found
#Fill in start
#Fill in end
#Close client socket
#Fill in start
#Fill in end
serverSocket.close()
sys.exit()#Terminate the program after sending the corresponding data
```

Web Server Part-B: (20%)
Currently, the web server handles only one HTTP request at a time. Implement a multithreaded server that can serve multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection

request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair.

What to Hand in:
You will hand in the complete server code along with the screenshots of your client browser, verifying that you receive the contents of the HTML file from the server. ***Please see submitting your assignment instructions below.***

Option-2: UDP Pinger (100%)

In this programming option, you will write a client ping program. Your client needs to send a simple "ping" message to a server, receive a corresponding "pong" message back from the server, and determine the delay between when the client sent the "ping" message and received the "pong" message. This delay is called the Round Trip Time (RTT). The functionality provided by the client and server is similar to the functionality provided by standard ping program available in modern operating systems. However, in this assignment, you will create a nonstandard (but simple!) UDP-based ping program. Below, you are given the complete code for the server in Python (available in the Textbook's Companion Website). Your job is to write the client code, which will be very similar to the server code. It is recommended that you first study carefully the server code.

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 *
# 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.

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 server code above injects artificial packet loss to simulate the effects of a network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.

UDP Pinger Part-A: (60%)
You need to implement the following client program.

Client Code:
The client should send 10 pings to the server. Given that 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 documentation of the programming language you choose 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 connection first, since UDP is a connectionless protocol.).
(2) Print the response message from the server, if any.
(3) Calculate and print the round trip time (RTT), in seconds, of each packet, if the server responds.
(4) Otherwise, 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 assignment 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 one and progresses to ten for each successive ping message sent by the client, and *time* is the time when the client sends the message.

UDP Pinger Part-B: (40%)
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).

What to Hand in:
You will hand in the complete client code and screenshots at the client verifying that your ping program works as required on your localhost (or, 127.0.0.1). ***Please see submitting your assignment instructions below.***

Submitting Your Assignment:
Please note that whichever option you choose to program, you can use any programming or scripting language you prefer. However, debugging your code and making it work is your own responsibility. ***Please do not forget, the onus is on you.***
When submitting your assignment, please ensure the following:

- Please upload your program using Brightspace
- Also, please prepare a README file including:
    - Your Name, your B00 number, and your e-mail address.
    - A short description of the program indicating the socket programming "option" and the programming language you chose.
    - What your program design is and how it works.
    - How the marker should run your program, including any required parameters. If there are many options to work with, please give example commands or output.
- Finally, if your program is not complete and cannot be executed, then please state it in the README file so that you could still get some partial marks.

Please submit all of the above on or before the due date of the assignment using Brightspace. If the assignment is late, the late days will be deducted from your budget days. Please remember that if you use all your budget days for the term, then "Late Policy" will apply!

Please read the "Late Policy" section in the Class Syllabus document. If you are submitting late, you will still submit using Brightspace. However, please note that no assignment submissions are accepted once the solutions for the assignment are released!

If you have any questions, please see me, preferably earlier than the day before the assignment is due ☺