EECS 563 Homework 4

Due: Friday, October 4, 2024, 11:59 pm.

Problem 1: TCP Client-Server

Objective. In this assignment, you'll write a client that will use TCP sockets to communicate with a server that you will also write. The goal is to build a Network Time Protocol (NTP) Client-Server. You need to implement an NTP-like service where a client requests the current date and time from a server.

TCP Server. The server should listen for incoming TCP packets on a specified port. Upon receiving a request from a client, the server should:

- Retrieve the current timestamp (including both date and time in UTC format).
- Construct a response packet containing the timestamp. A simple string format (e.g., "2024-09-28T12:34:56.789Z") to capture date and time in UTC (or local time zone) is acceptable.
- Send the response back to the requesting client using TCP.

Name your server application `TCP-server.py`. Run the server application using the following command, where the local port number is provided as user input:

```
python TCP-server.py <server-port-number>
```

TCP Client: The client sends a time request to the server. Upon receiving a response, it parses the timestamp from the server's packet. Then, it calculates the time difference between the server's time and the client's local time. The client displays the received timestamp, the local time, and the difference (latency) in milliseconds. The client should establish a connection to the server on a specified IP address and port, which is received as user input. Name your TCP client as `TCP-client.py` and execute it using the following command:

```
python TCP-client.py <server-IP-address> <server-port-number>
```

Testing your code: So far, you've got the basic assignment done. The last part of the assignment includes testing your code. To this end, you have three options:

(a) You may be running the clients and senders on the same machine (e.g., by starting up the server and running it in the background, then starting up the client. This is fine; since you are using sockets for communication, these processes can run on the same machine or different machines without modification.

(b) You can test the server and client applications on two different machines in the EECS lab or between your laptop and a lab machine.

(c) [**Recommended**] You can team up with someone from the class and get your client to interact with their server or vice versa. If you've got a server up and running, you can advertise your server's services to anyone in the class, and if you've got a client, all you need to do is interact with such an advertised server. Note that if you've got your own client and server running, there isn't any more programming involved – just running your client and server with someone else's. This shouldn't actually be very hard at all! Recall that when we discussed RFC standards for protocols we noted that the IETF requires that two independent implementations of a protocol must interoperate; that's what you are doing here.

Problem 2: Change the previous programs to work with UDP sockets. Therefore, you need to provide two programs called `UDP-server.py` and `UDP-client.py`. The rest of the problem remains the same. For UDP, implement basic error handling to manage network issues, such as packet loss or timeouts.

Questions and Deliverables.

(a) You need to submit your Python codes (4 .py files) along with the output from your clients and servers. You should program your client and server to each print an informative statement whenever it takes an action (e.g., sends or receives a message, detects termination of input, etc.), so that you can see that your processes are working correctly (or not!). This also allows the GTA to also determine from this output if your processes are working correctly.

(b) You should hand in screen shots (or file content, if your process is writing to a file) of these informative messages as well as the required output of the client and server.

(c) By comparing the TCP and UDP client-server applications, do you notice any differences in terms of delays? Explain your observations.

(d) Modify your client application to record the time at which the request is sent to the server and the time the response is received. Use this information to calculate the network round-trip time (RTT). Use the RTT to calculate the client-to-server delay. Are the network delay values the same for the client-to-server and server-to-client connections? Discuss your observations and provide possible explanations.

(e) Run your NTP client-server applications at various times of the day and when the client and server are located with physical distances from each other. Do you observe any changes in the delay values? Discuss your observations and provide possible explanations for the variations.

Programming notes

Here are a few tips/thoughts to help you with the assignment:

- You must choose a server port number greater than 1023 (to be safe, choose a server port number larger than 5000). If you want to explicitly choose you client-side port, also choose a number larger than 5000.

- You may need to know your machine's IP address, when one process connects to another. On Windows, see the `ipconfig` utility. On a Mac, you can run the terminal program and use the `ifconfig` command.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use.