Wesleyan University, Fall 2022, COMP 411

Homework 2: Measurements and distributed tic-tac-toe

Due by 5pm on September 23, 2022

---

1. Written problems (5 points)

Problem 1. *This problem looks at computing end-to-end packet delays.*

a: *Consider a packet of length L that begins at end system* **A** *and travels over three links to a destination end system* **B**. *These three links are connected by two packet switches. Let d_i, s_i, and R_i denote the length, propagation speed, and transmission rate of link i, for $i = 1, 2, 3$. The packet switches delay each packet by d_{proc}. Assuming no queueing delays, in terms of d_i, s_i, R_i ($i = 1, 2, 3$), and L, what is the total end-to-end delay for the packet?*

b: *Suppose now the packet is 1,500 bytes, the propagation speed on all three links is 2.5×10^8 m/s, the transmission rates on all three links are 2 Mbps, the packet switch processing delay is 3 msec, the length of the first link is 5,000 km, the length of the second link is 4,000 km, and the length of the last link is 1000 km. For these values, what is the end-to-end delay?*

Problem 2. *Download and read the Request For Comments (RFC) titled "Architectural Principles of the Internet," found here:* `https://tools.ietf.org/html/rfc1958`.

a: *What is the benefit of making the Internet level protocol (i.e., IP) independent of hardware?*

b: *Summarize the end-to-end argument in a few sentences in your own words. Include the benefit of having end-to-end protocols when there are network failures.*

c: *Do you think the architectural principles have stood the test of time? Explain why or why not in a few sentences.*

2. Coding and hands-on problems (15 points)

Problem 3. *The goal of this question is to use traceroute to collect some round-trip time (RTT) delays. To find more information about traceroute, type* `man traceroute` *at a terminal prompt.*

a: *Briefly describe how traceroute works.*

b: *Use traceroute to connect to* `ufrj.br`, *a university in Brazil, at 3 different times of day. Run traceroute 5 times at each time of day, to collect 15 sets of measurements of the round-trip time (RTT) delay. Give the measurements and calculate the average and standard deviation for each of the three times of day. How many routers are in the path at each time*

of day? Did the set of routers or the number of routers ever change? Do you ever see the delay to reach a closer host exceed the delay to reach a farther away host? If so, what do you hypothesize that the variation is due to?

c: *For one of your times of day in part (b), list out the names of intervening routers. Based on these names, can you identify the ISPs in the path from source to destination? For example, my traceroute to* `ufrj.br` *shows a router name that ends in* `internet2.edu`*. If I google* `internet2.edu`*, I find out that* `internet2.edu` *is part of the Internet2 network, see* `https://en.wikipedia.org/wiki/Internet2`*. There is no right or wrong answer for this question, just see what you can find out.*

PROBLEM 4. *The goal of this question is to make the tic-tac-toe game you previously implemented in homework 1 a distributed application using socket programming.*

Part 1. An introduction to socket programming. *To help you get familiar with socket programming you have been given code for a simple echo client and echo server to play around with, downloadable from the class schedule as well as the homework 2 webpage. If you recall we went over this code in class. The format of* `echo_client.py` *is similar to what you will do for* `client.py` *and the format of* `echo_server.py` *is similar to what you will do for* `server.py`*. You should run the code yourself and see what happens. To run the code, open two terminal windows. In one terminal, start the echo server with* `python3 echo_server.py`*. In the other window run the echo client with* `python3 echo_client.py`*. You should see that the "Hello world" string in* `echo_client.py` *code sent to the echo server is echoed back and appears in the terminal window where you ran the echo client. You can find additional useful references for python socket programming on the class resources page.*

Part 2. Distributing tic-tac-toe. *Your goal is to implement a distributed version of tic-tac-toe. Some starter code has been provided for you:*

- `tictactoe.py`*: This file contains two classes:* `Board` *and* `TicTacToe`*. If you'd like to use your own tic-tac-toe game code, you are free to, but you should still put it in a file called* `tictactoe.py` *that does not contain any* `main`*.*

  Any functions or methods you would like to share across the server and client, you should add to `tictactoe.py` *and then call them from the server or client. For instance, you may wish to add functions to parse the strings sent or received by the client or server. If your code is organized well, the same* `TicTacToe class` *should be useable by either the centralized or distributed versions of the game, just by changing the client and server code.*

- `server.py`*: This file contains the tic-tac-toe server code. Method stubs have been implemented for you to get you started. The primary method that you will need to fill out is the* `play` *method, although you may want implement additional helper methods. As before the server will still query the client (as well as choose its own strategy to play), however, any information exchanged between the client and server will be sent and received from a socket.*

  The server is multi-threaded. This means that rather than blocking on a client while that client is being served, a thread is spawned off the main thread of execution of the

server process to serve each client that connects to the server. This lets the main execution thread of the server continue to listen for new client connections, while clients are being served. If multiple clients connect to the server simultaneously, then each would be served without blocking the other. You should not need to modify the threading code. As an aside, Python does not have true multi-threading in that the use of threads will actually make the code run slower rather than faster (e.g., because separate threads cannot run in parallel on separate cores). The benefit of threading here is that it allows for a simpler server implementation.

- `client.py`*: function stubs have been implemented for you to get you started. You'll want to have the client first create a connection with the server (via a socket), and then have the client send and receive (binary) data from the socket in order to communicate with the server.*

Your tic-tac-toe game should now have a user interaction like what is shown in Figure 1. You will see now that you will have to run two programs to test your code: in one terminal, start the server with `python3 server.py` *and in another terminal start the client with* `python3 client.py`*.*

Again, part of my goal is to give you some flexibility (and creativity) in architecting your now distributed TicTacToe code, so I have not included stubs for all of the methods you might want and you are free to change argument lists for the methods given. You also have some flexibility in creating additional classes to use.

Key issues to watch out for:

- *You can only send/receive binary from a socket, which means anything you wish to send or receive must first be encoded as binary or decoded from binary. An easy way to do this for a string in python3 is to do* `strvar.encode('utf-8')` *and* `strvar.decode('utf-8')`*.*

- *You will need to have some way to indicate that everything that needs to be read from the socket has been read, such as an end of message flag.*

- *You may also need to do multiple receives before receiving everything which means you will want to put any call to receive in the body of a while loop.*

- *If you stop the server and then try to restart it, you may find you get an error like* `Unable to open server socket: [Errno 48] Address already in use`*. Typically, this means that the socket wasn't shutdown cleanly, but if you wait a short time (up to a minute), the port number will be released and you can use restart the server.*

Going further:

- *Once you have submitted your assignment, you might want to try the following out with someone else in the class (who has also submitted their assignment!). Give the other person a copy of your client (or server code). You will need to do this since it is unlikely your client will be able to communicate with the other person's server. Then, use* `ifconfig` *to look up the IP address of the device on which the server is running. Now have your client connect*

not to the `localhost` *address (i.e., 127.0.0.1), but instead to the IP address you just found, and to the port on which the server is listening. Assuming both devices have their ports open to accept external connections (you may need to check your system settings), then you should be able to play a tic-tac-toe game distributed across the two devices.*

3. Submission

Upload your written work as `hw2.pdf`, and your code solution as `client.py`, `server.py`, and `tictactoe.py` to the Google Drive directory I have created for you named `comp411-f22-USERNAME/hw2/`. You should replace `USERNAME` with your Wesleyan username.

Do not forget that your written work must be submitted as a PDF! Make sure that at the top of each file you have put your name! Do not, however, change the names of the files.

```
> python3 server.py
Client with address has connected ('127.0.0.1', 64643)

==================
| TicTacToe Game |
==================


-  -  -
-  -  -
-  -  -

User
o  -  -
-  -  -
-  -  -

Server
o  -  -
x  -  -
-  -  -

User
o  -  -
x  o  -
-  -  -

Server
o  -  -
x  o  x
-  -  -

User
o  -  -
x  o  x
-  -  o

Winner:  o !
```

```
> python3 client.py

==================
| TicTacToe Game |
==================

Enter number of rows in TicTacToe board:
3

-  -  -
-  -  -
-  -  -

Choose row [0-2]: 0
Choose col [0-2]: 0
User
o  -  -
-  -  -
-  -  -

Server
o  -  -
x  -  -
-  -  -

Choose row [0-2]: 1
Choose col [0-2]: 1
User
o  -  -
x  o  -
-  -  -

Server
o  -  -
x  o  x
-  -  -

Choose row [0-2]: 2
Choose col [0-2]: 2
User
o  -  -
x  o  x
-  -  o

Winner:  o !
```

FIGURE 1. A sample session of the distributed tic-tac-toe game from the view of the server on the left and the view of the client on the right.