CSCI 156 Term Project

Author: Scott McCoy

Instructor: Dr. Li

Course: CSCI 156

Date: 1 December 2017

Summary

The main point of this project was to practice using sockets to send data over a network. By completing this project students were able to gain a greater understanding of some of the finer points of network programming including synchronization across networks and server request handling. Basic socket programming skills were also developed such as the development of an awareness of packet size when sending transmissions and library specific details in connecting between between client and server.

Design Decisions/Architecture

For this project I decided to write in Python 3.5 for the purpose of improving my skill with the language. The price list for this program was taken by downloading the closing price for a list of stocks over the course of the last five years. The same amount of prices were taken for each stock so that when there are no more prices left for the program to read the program will know to exit. The program executes in a series of stages that are described as follows. For the purpose of simplicity all packets without a predetermined size were set to be 35 bytes long. If a packet was not long enough then zeros were added to the end of the string that was being transmitted until it was long enough. These zeros were stripped off when the packet arrived at its destination before processing. If the data was the same every time then the expected packet size for the recv function was just set to the size of the of the expected transmission. This concept of packet length was very important when it game to not scrambling packet data because if the recv buffer was larger than the packet size then in some cases the recv buffer would also catch part of the next packet in the sequence which would completely derail the processing of packet data.

Stage 1
In stage 1 the server reads all of the prices from their .csv files into memory. The server then sets it IP address which will always be localhost (127.0.0.1) and it's port number. The port number may change depending on if that particular port is already being used by the system or not. Once the port and IP address are set the server waits for the expected number clients to connect (in this case this is two). This allows all of the clients to start trading at the same time which helps to determine what trading strategy is working the best when it comes time for the test. For the clients, stage one is setting all of their nondeterministic variables such as x, y, z, port, and IP address of the server. Once all of these variables are set then the clients attempt to connect to the server. The clients also start a thread for their monitor function in this stage.

Stage 2
In stage two the first communications with the server take place. The client sends it's name, what it wants to do (buy/sell), and what stock it is interested in.

Stage 3
In stage three the server receives the request from the client and returns a message informing the client if its request was successful or not (there is a 10 percent chance of failure as required by the assignment), the stock name, and the price of the stock.

Stage 4
In stage four, the client receives the request and if it was successful processes the request. This processing entails performing the entire process of performing the transaction. When this processing is done the client notifies the server that it is ready to proceed.

Stage 5
In stage five the server receives the ready signal from each client. When every client notifies the server that they are ready to proceed the server deletes the old price for every stock in the stock list and notifies the clients that it is safe to proceed with the next transaction.

Stage 6
In stage six the clients receive the proceed signal from the server. When this signal is received the client changes whatever state it is that they are in (buy/sell) and returns to stage 2.

This staged execution cycle continues until the server runs out of prices.

Results

All results were generated using the same code. The data here represents the final balance of each client's account after the program had been executing for five minutes. The starting balance of each account was 10,000,000 to protect against exhausting funds. In trial 1, client 1 exhibits a more aggressive approach by selling if the price of the stock rises or falls by 10 percent and a more conservative

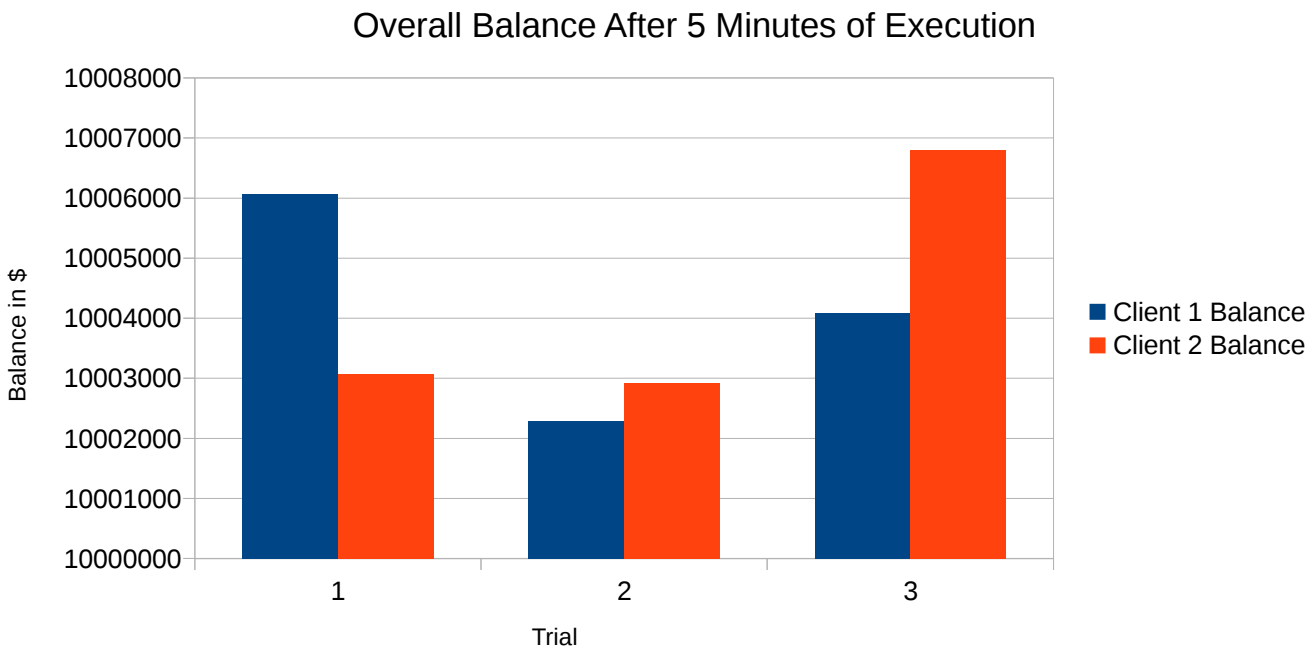
buying approach by buying only 25 percent of the time. Conversely, client 2 buys 75 percent of the time but only sells if a stock price has risen or fallen by 20 percent. For the second test client 1 was given a balanced approach by buying 50 percent of the time and selling if the price of the stock rose or fell by 10 percent. Client 2 was given a very aggressive strategy in this case by buying 80 percent of the time and selling if the stock price rose or fell by 5 percent. For the third test the buy percentage was set to be only slightly aggressive at 60 percent chance to buy and the sell chance was set to be slightly more conservative at 15 percent increase or decrease to sell. For client 2 the chance to buy was set to be slightly conservative at 40 percent and the percent increase/decrease was set to be 15 percent decrease to sell and 5 percent increase. These graphs for the results are located in the final page of the report.

Bonus

For the bonus section of the project I decided that the best option would be to move the majority of the processing for the buy and sell requests to the server rather than have each client handle the requests. This was necessary in order to facilitate the data exchange needed to determine who was selling and buying which stock each round. This approach mostly yielded success. However, there is a synchronization bug between the two clients that will sometimes allow a client to buy stocks when they should not be able to. This leads to a slowly growing number of stocks available to each client but does not cause the server/clients to crash or otherwise affect the execution of the program.

Conclusion

This project helped to develop my skills both as a python programmer and as a network programmer. Without this project I would not likely be confident programming with sockets or over a network. Based on the graphs the best approach appears to be to have a more conservative approach when it comes to buying (buy less often) and to have a more aggressive approach when it comes to selling. In particular, if the percent increase needed to sell is low the client appears to do well. As far as possible extensions to this project, I think that it would be most interesting to have a higher level of statistical analysis in the project. For example it might be interesting to look at the correlation between how many transactions actually took place and the overall profitability of the strategy. Another interesting possible extension would be to develop a method of self adapting strategy so that if in general prices are low (bear market) then buying is done more and if in general prices are high (bull market) then selling is done more.


Overall Balance After 5 Minutes of Execution
10008000
10007000
10006000
10005000
10004000
10003000
10002000
10001000
10000000
Balance in $
1
2
3
Trial
Client 1 Balance
Client 2 Balance