Mid Term Assignment

Computer Networks 2022 – Johanna Catharina Smit (12220070)

# Goal

My goal is to implement a UDP socket program with the Python language. I am going to write a listening server program, and a client program that will send ping sensor data messages to this targeted server. The program will contain the necessary error handling.

# Tools

When it comes to the IDE, I will use Visual Studio Code . For version control I will use a GitHub repository.

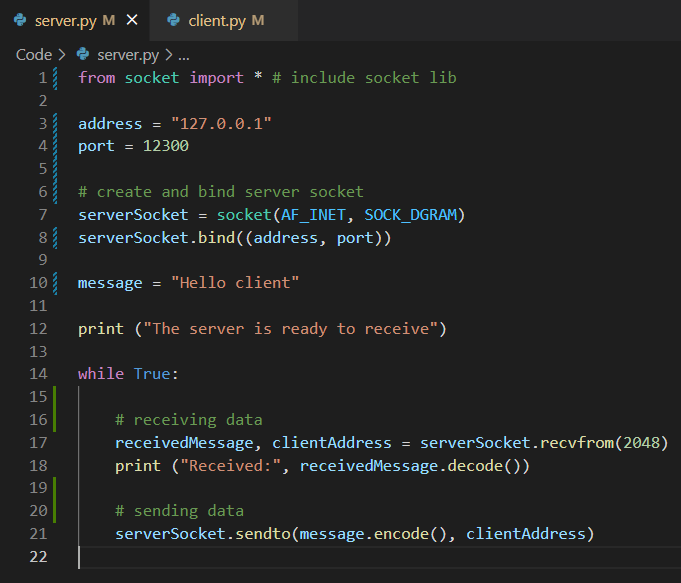
# Steps

I will implement the different functionalities in order of the given requirements. For each of these functionalities I will share screenshots and some information about how I approached this in the form of different steps.

## Step 1: Your client will send a simple ping message to a server and receive a corresponding pong message back from the server.

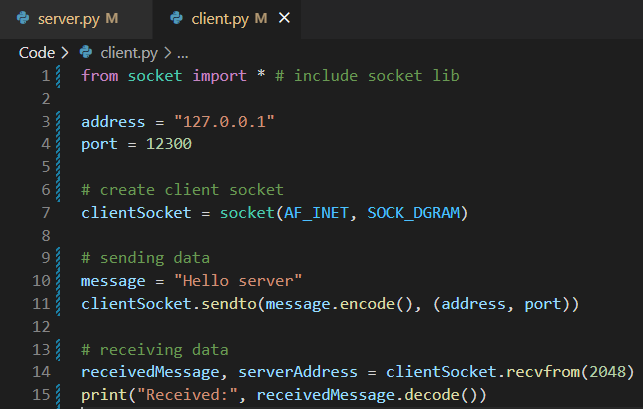
Using the given practical/lecture material, I tried to understand the code and write a separate server and client program.

Server side



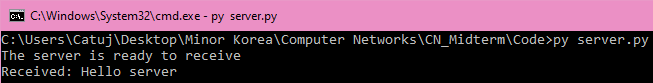
First the socket is created using the Python socket library. Upon creation we need to specify two parameters: family address and type of socket. In this case we use Internet Protocol v4 addresses to communicate, so we specify “AF\_INET” for family address. Because we want to use UDP and work with datagrams, we specify “SOCK\_DGRAM” for type of socket. To make the address complete we have to bind a port number to the socket. Here I used 12300. The server receives the message alongside the address of the client with the recvfrom() method, with a buffer size of 2048. The server prints this message and sends a message back to the client with the sendto() method, specifying the stored address of the client.

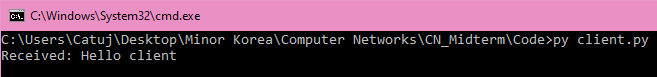
Client side



The client side is pretty similar. First the UDP socket is created, a message is send into the socket and then a reply is read and printed.

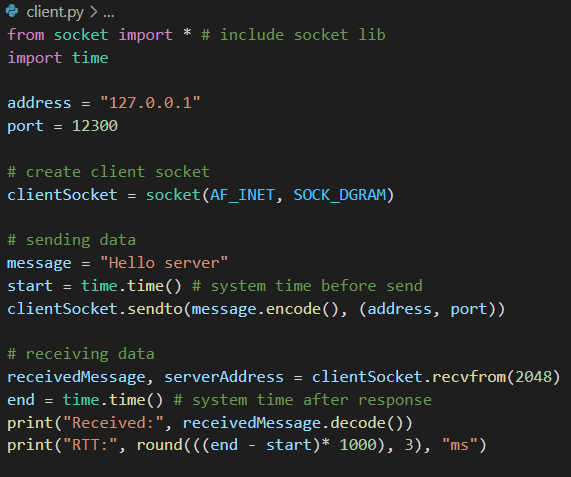
Result

Running the client program and the server program in two terminals, with the following result:

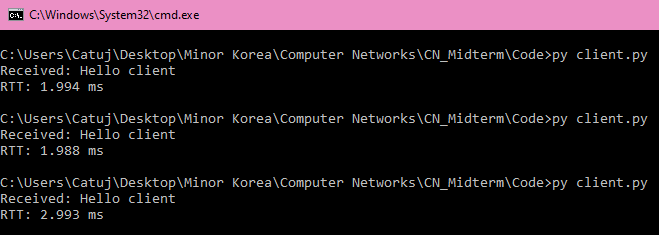


## Step 2: Your client will determine the delay between when the client sent the ping message and received the pong message. This delay is called the Round Trip Time (RTT).

For this I used the time function from the time module. This function gives back the current system time, or more specific the elapsed seconds since the point where times starts (epoch). I store this value in “start” before sending out the data, and a second value after receiving the data in “end”. Subtracting “start” from “end” will give us the delay in seconds. To convert this to ms I multiplied this value with 1000 and using round() to keep it uncluttered and print it with only 3 decimals.

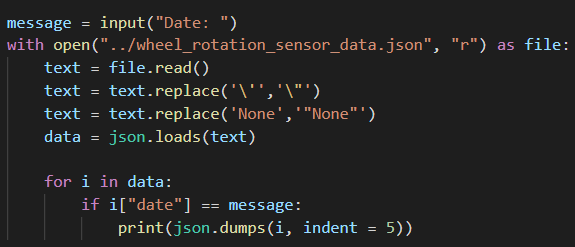


With the following result:

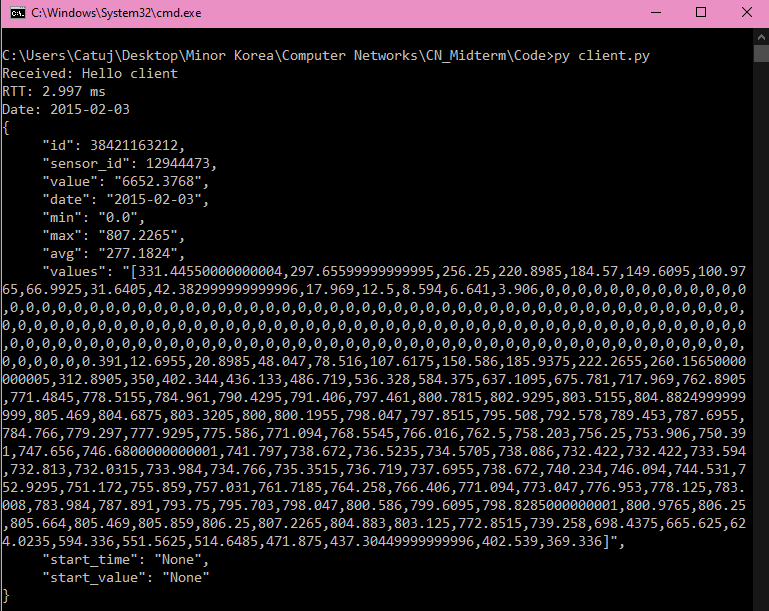


## Step 3: Your client will input the enclosed wheel rotation sensor data from an json file for a specific day.

First I just want to be able to read the json file and print the corresponding sensor data from a specific day. I tried to realize this with the following code:

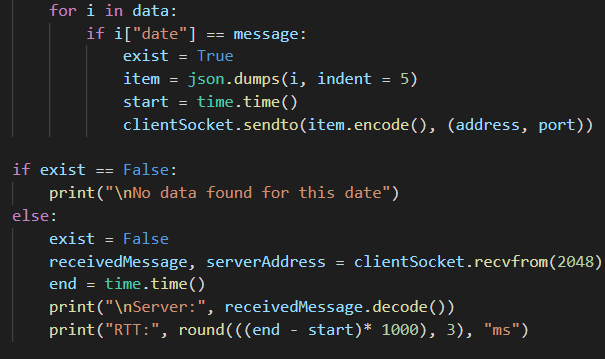


The client program will take a date as user input and compare this with the date of the items of the json file to get the corresponding wheel rotation sensor data of that day. I use the “with” statement to open the json file for correct release of the used resources. Because the format of the json file wasn’t completely right(single quotes, no quotes), I use the “replace()” statement to fix these issues without having to change anything from the real json file. I will adjust some of these values in a later step. For now I only want to print the data, so I implement this by using a for loop to go over all the data and find the right day data to print. To make the data organized I use the json.dumps() function with an indent of 5(start line of text after 5 spaces). Running this results in:

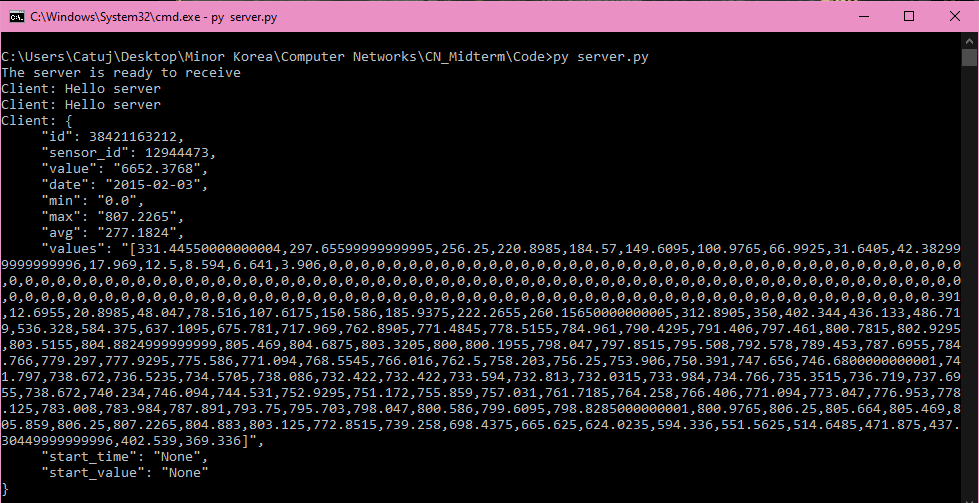


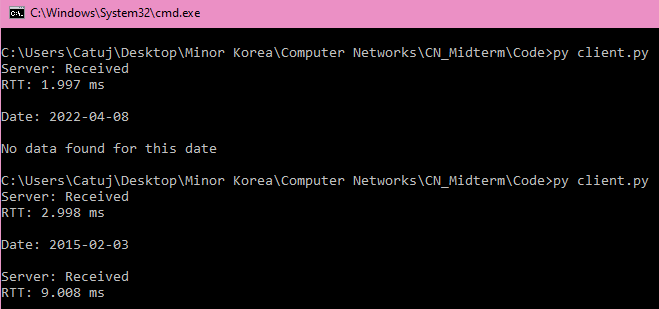
## Step 4: Your ping program is to read the time series data for a specific day and send all ping sensor data messages to the target server over UDP + for each message, your client is to determine and print the RTT when the corresponding pong message is returned.

Now I’m going to implement actually sending the data to the server over UDP. I replaced the print() method with sendto() method to do this. A boolean variable had to be added to check if the date actually exists in the json file. If the date does correspond, , the client waits for a return(pong) message from the server and prints the RTT again.



Result:

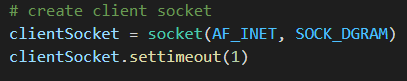




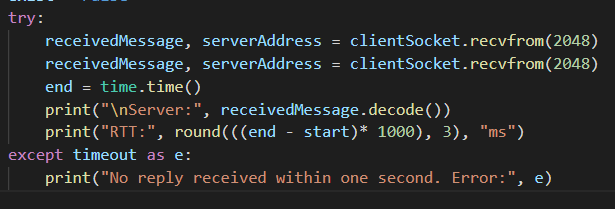
Right now the entire data collection is send as a whole. In a later step I will expand this functionality by sending the list items separately with a specific time interval.

## Step 5: You should have the client wait up to one second for a reply from the server; if no reply is received, the client should assume that the packet was lost and print a message accordingly

Implementing a timeout:



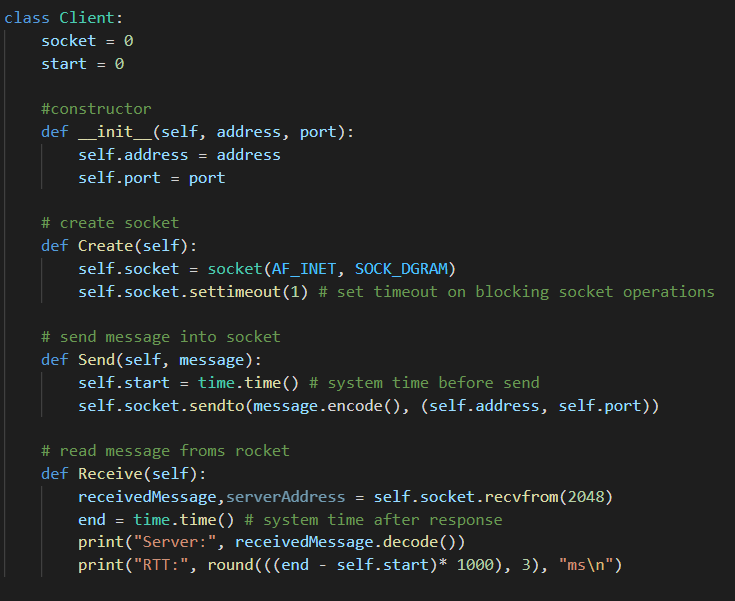
First the settimeout() method is used to set a timeout on the blocking socket operations. As a parameter I give the value ‘1’, because we want the client to wait up to one second when carrying out recvfrom() methods. After this I moved the receiving functionality into try and except blocks. This way the exceptions can be catched and handled accordingly (in this case printing an error message). To test if this worked I let the code try to carry out 2 recvfrom() methods directly one after another. The expected response in printed in the terminal.



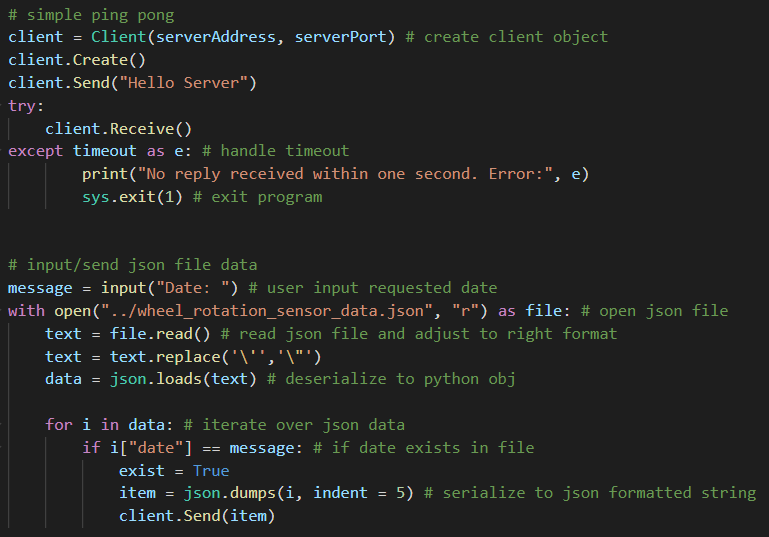


# Result

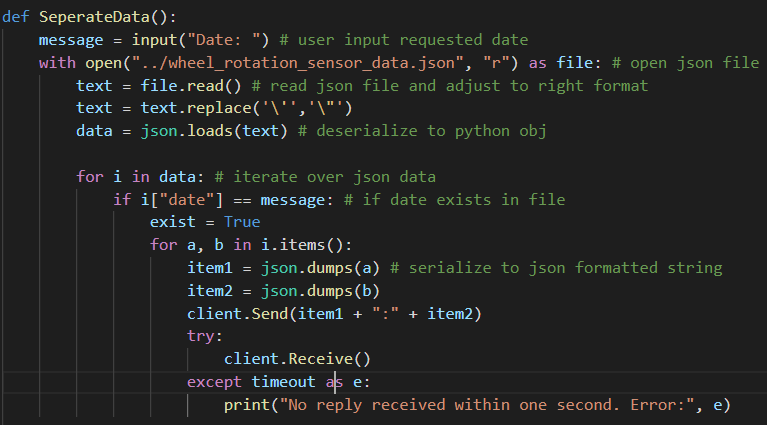
As for the final touches I changed some values in the json file to make more sense: defined a start\_time/start\_value and changed the date to today. Next to that I created a client class and added comments to keep the code a little bit more readable:



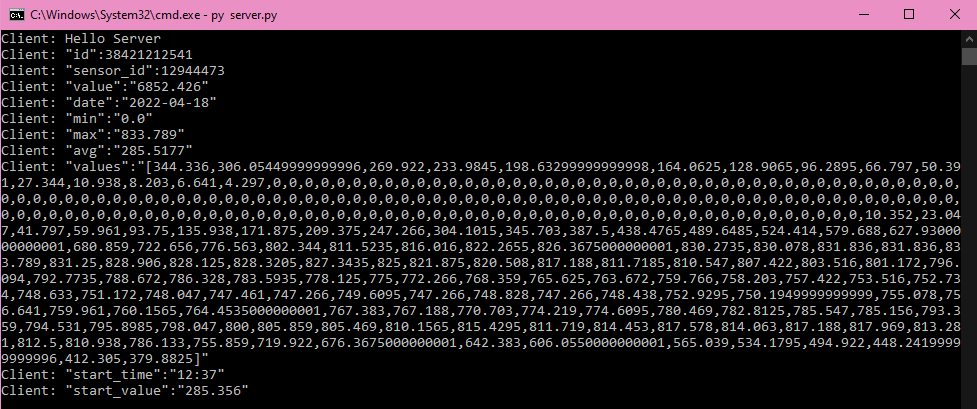
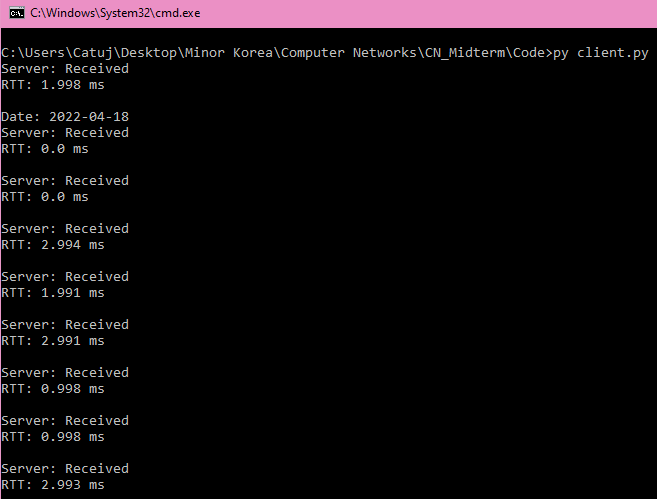
Initializing and using the object as follows:



To also test sending the sensor data separately(per key + value pair) I created a fast and simple SeperateData() method. This was just an extra test, so this block of code will not exist in the final source code.



With following result:



Refer to the demo for final result.