Maryann Foley
19330904
CSU33031 Computer Networks
Assignment 1 - IoT Publish-/Subscribe Protocol

My solution demonstrates a protocol in which a broker handles addresses, allowing each address to subscribe to others, and publish to all others subscribed to it. It does so using datagrams, which are sent between the computers, with each packet being labeled as either a subscription or a message. Thus, my protocol was built on top of UDP, allowing for this transfer of information. In my protocol, the "topics" typically seen in the publish/subscribe protocol MQTT are replaced by the names of each clients, which means that clients are subscribing to others directly and will receive all messages sent by the subscribed client. The broker is in the middle of all communication, storing and managing the subscriptions and handling and forwarding all messages.

For example, the dashboard could subscribe to sensor1 and sensor 2 and sensor1 could each subscribe to dashboard. In such a small network, these direct subscriptions felt like a clearer way to demonstrate the communication between individual clients. The subscription messages will be sent to and parsed by the broker, the named clients will be matched with their addresses, and the subscriber will be noted for future message forwarding. When a computer sends out a message, the broker will determine the sender, and using the sender address, will determine which computers are subscribed to the sender. Finally, the broker will send out the message to the correct subscribers. In my demonstration, the dashboard would do a calculation based on the message content, and may send a message back to the broker to be sent to the sensors.

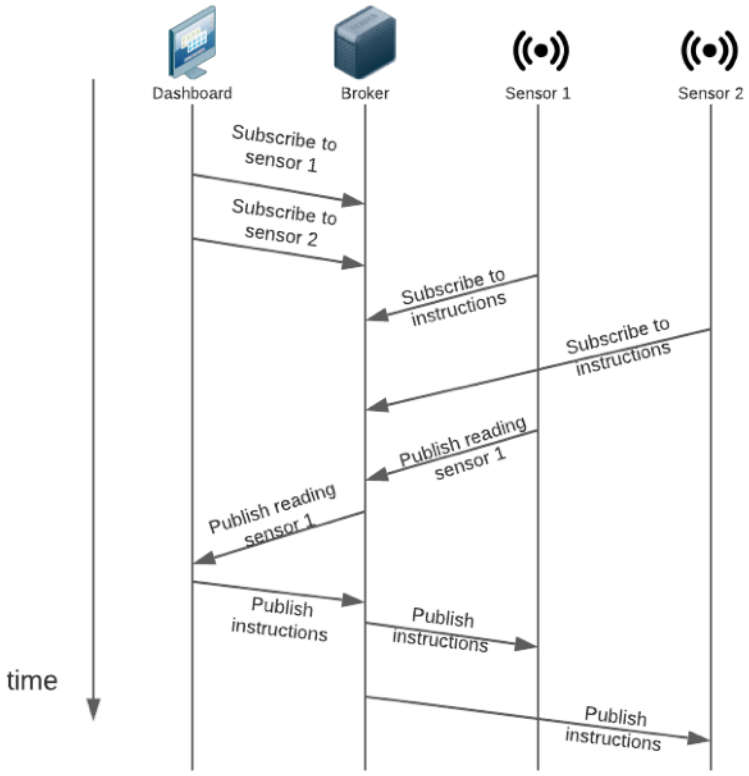

Figure 1: Dashboard subscribes to all messaged from a particular sender. Data that is published from a subscribed computer after the subscription is forwarded to the dashboard.

My solution was executed using Python, in particular the socket library, which allows for the sending of datagrams using UDP to any IP with a message. By opening the socket using, `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)` the socket library knows to use UDP. Additionally, I used Docker to run my solution. By creating four Docker images, each one is issued an individual address, which will allow me to simulate a real network. In order to run my network, I first must create the Docker images, which was accomplished by following the instructions to create images that are capable of running GUI applications (these instructions are repeated for each of sensor1, broker, and sensor2, with only the phrase "dashboard" being replaced).

```
docker create --name dashboard --cap-add=ALL -ti -v /Users/maryann/cs2031:/cs2031 python /bin/bash
docker network connect cs2031 dashboard
docker start -i dashboard
xhost + 127.0.0.1
```

In my code, some of the IP addresses are hardcoded, so in order for the code to run without modification, the dashboard must be at address 172.20.0.2, sensor1 must be 172.20.0.3, the broker must be 172.20.0.4, and sensor2 must be at 172.20.0.5. Additionally, all of the sockets are run on port 49000.

Following the setup of the images, each image has different programs to run to demonstrate the network working. The broker image is started, and if Wireshark is wanted, then `wireshark 2>/dev/null &` must be run. The broker is then run using `python broker.py`. Using another terminal tab, the dashboard image is started, and `python recievertemps.py` is run, and subscriptions can be chosen as a comma separated string (for the purpose of the demonstration I use "sensor1,sensor2"). Next, I use the image for sensor1 to run a monitor to subscribe to and monitor messages from the dashboard using `python sensor1subscribe.py`. This is repeated for sensor2, using `python sensor2subscribe.py`. Finally, to publish a message from a sensor, I must first run a second shell for the chosen sensor, as because of limitations of a basic docker container, it was a more straightforward workflow to open this additional shell. To do this, I first run `docker ps` to retrieve the correct container id. Then, using this id, I open the second shell using `docker exec -it <container id> bash`. Once this is open, I can run `python sensortemps.py`, which allows for the input of a value for the temperature. This message, once sent, can be seen going through the broker to the dashboard. If the value was numeric and less than 1 or greater than 35, it is considered to be outside of a normal range, and any computers that are subscribed to the dashboard will receive a message indicating the issue.