Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Aislinn Addison-Smyth – 19337226

Computer Networks Assignment #1 - IoT Publish-/Subscribe Protocol

November 2nd, 2021

Contents:

1. Introduction

The problem description outlined for this assignment is focused on the learning, understanding and designing of an IoT publish/-subscribe protocol. It is the MQTT, The Message Queuing Telemetry Transport, protocol which I have implemented. The MQTT protocol uses a request/response communication pattern. The network consists of subscribers, publishers who traditionally have no contact with each other, there is an intermediate role called a broker which is responsible for the message response and distribution. There are two scenarios which I have been required to implement as part of the basic requirement of the protocol: 1) the reporting of sensor data to several subscribers and 2) the issuing of instructions to actuators.

In my report, I will discuss the functionality behind my assignment and give reason as to the implementation decisions I made along the way.

2 Overall Design

In the first two sections I will explain my understanding and approach to the two basic scenarios, which I listed above in my introduction. The explanation of my design will describe the interaction between the individual network elements, followed by a description of the packets that were exchanged between network elements during the protocol process.

2.1 Subscription

The first aspect of the IoT publish/-subscribe protocol is the reporting of sensor data to several subscribers. In figure 1.1 below, I outlined the easiest way to describe the design of my subscribe mechanism. The dashboard subscribes information such as a room number and a topic associated with that room number to the sensor. The sensor holds data from three separate rooms in an office block, each with its own associated temperature readings and humidity level. Once the sensor receives the information from the dashboard, it will publish the packets associated with that information to the broker. The sensor will not publish any packets to the broker unless called by the dashboard to do so.

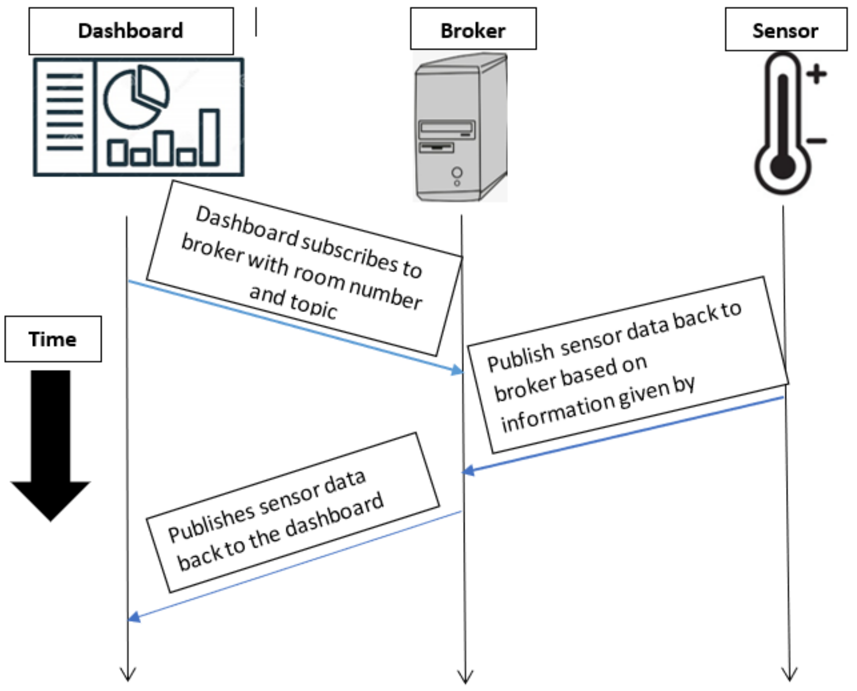

Dashboard
Broker
Sensor
Dashboard subscribes to broker with room number and topic
Time
Publish sensor data back to broker based on information given by
Publishes sensor data back to the dashboard

Figure 1.1

2.2 Publishing

The second aspect of the IoT publish/-subscribe protocol mechanism is the issuing of instructions to actuators. In figure 1.2 below, I have outlined my approach to tackling this problem. The dashboard publishes a given topic to the broker, for example, ‘AC’ or ‘laptop’, once the actuator receives this prompt of information from the dashboard it will subscribe a packet to the broker with the instructions associated with the topic received. Much like figure 1.1, the actuator will only subscribe to the broker with information once it has received a packet of data from the dashboard.

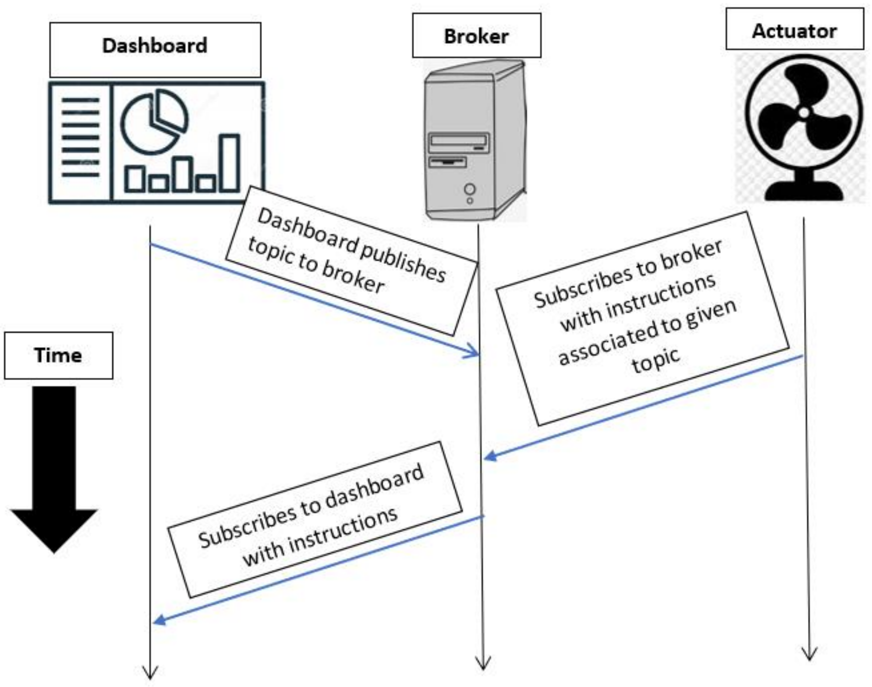

Dashboard
Broker
Actuator
Dashboard publishes topic to broker
Subscribes to broker with instructions associated to given topic
Time
Subscribes to dashboard with instructions

Figure 1.2

2.3 Packet Design

In figure 1.3 below, I have outlined how I have designed my packets, the initial example is a generalised overview of the design, and the example underneath is an example I have taken from one of the packets I have created in the sensor class.
The header is part of the data packet and contains information about the file, it precedes the actual packet data which is being sent. The header contains data which is essential for the packet being sent to reach its intended destination. From my sensor data example, you can see my header contains the protocol mechanism it is using, ‘publish’, it then holds the intended destination of the packet, ‘Broker’ and the destination port which is ‘PORT:49000’.
The packet data is the actual information we want to be sent; it is the critical information. The packet data always starts on a new line, so it is easy to differentiate the packet data from the header. As you can see from the diagrams below, the packet data in the given sensor example gives the location and temperature reading of a room, in this case the office. The date being sent in this case is ‘Location: Office Temperature: 26 degrees.

Finally, the tail, in my opinion it is important to have a tail consistent with every packet as it is easy to distinguish if data within the packet has gotten lost. In this case, I have the tail as ‘Packet End’ followed by ‘11’. The reason I chose to finish my packet off with 11 is simply as it is easier to spot than regular words. Seeing 11 at the end of a packet transmission tells me instantly that the subscribing/publishing mechanism has worked and if the double 1’s is absent it makes issue finding associated to lost packets a lot more efficient to locate and solve.

Header	Packet Data	Tail
Publish: Broker PORT:49000	Location: Office Temperature: 26 degrees	Packet End 11

Figure 1.3

3 Implementation

In this section, I will discuss in detail the different components of my implementation, the dashboard, the broker, actuators, sensor. I will also be discussing and demonstrating the use of wireshark to capture network packets and the use of docker to run code.

3.1 Dashboard

The dashboard is a key component of my network elements, all communication is done directly through the dashboard. The dashboard uses very basic code, consisting mainly of if loops and one while(true) loop to ensure the code executes until manually shut down. The dashboard acts as an interface as it is the point where the user can interact with the program without affecting the code inside the program.
The dashboard begins with a simple introduction:

```
while(true) {
    Scanner newScanner = new Scanner(System.in);
    System.out.println("Welcome to the dashboard, would you like to publish or subscribe?");
    String answer = newScanner.next();
```

The dashboard stores whether you would like to publish or subscribe in ‘answer’. As there are two options given the code uses if statements to navigate between the answer selected.

```
if(answer.equals("subscribe")) {
    System.out.println("Would you like readings from rooms 1, 2 or 3");
    String answer2= newScanner.next();
    System.out.println("Would you like information on the temperature or humidity?");
    secondAnswer = newScanner.next();
```

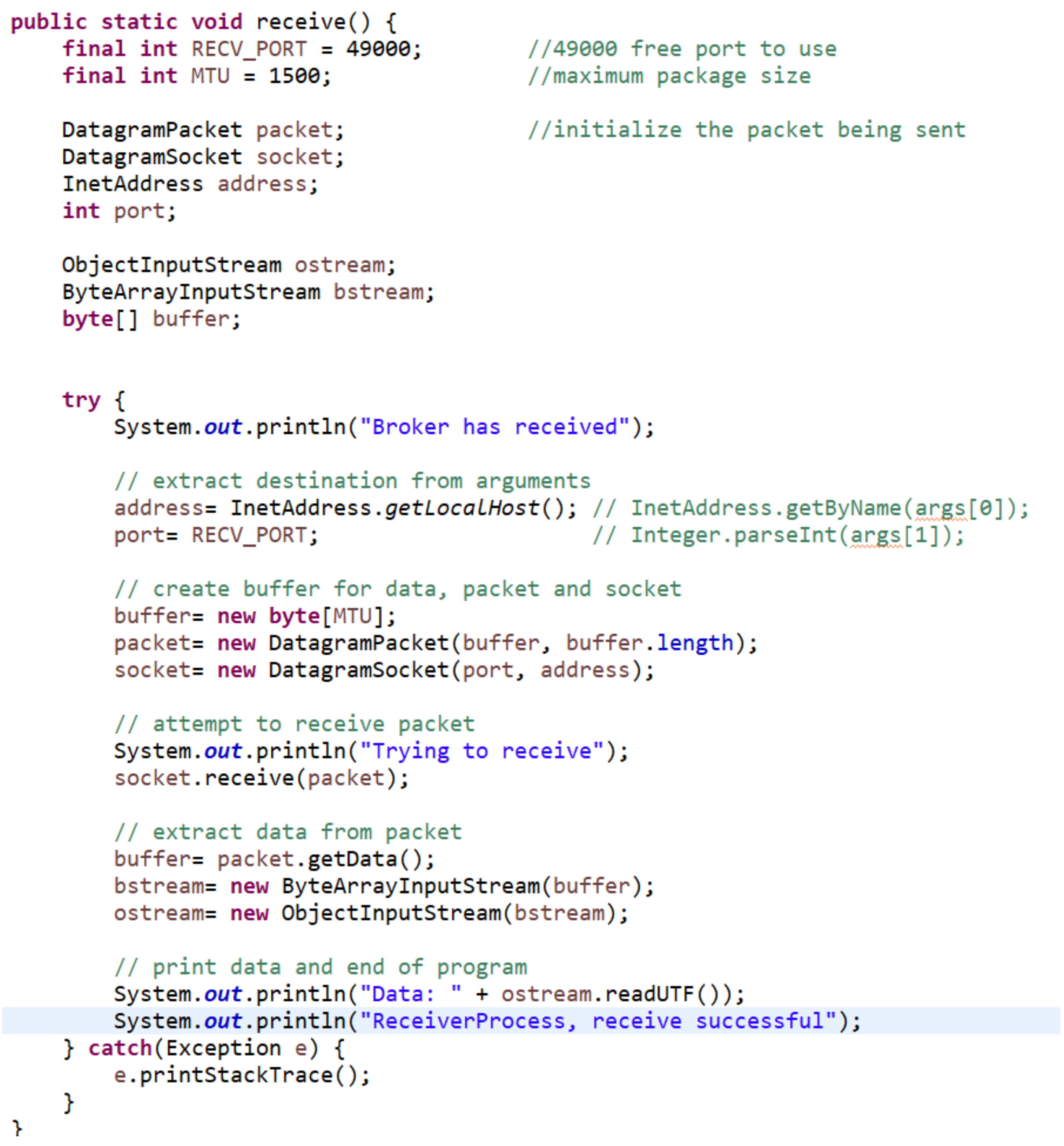
If you selected that you would like to subscribe to the dashboard the code will subscribe to the sensor class with the information you give.

```
}  else if(answer.equals("publish")) {
    System.out.println("Do you want instructions on the AC, blinds, laptop or windows");
    String answer3 = newScanner.next();
```

Finally, if you chose that you would like to publish to the dashboard, the code will prompt you to select a topic to give an instruction on and the code will publish to the actuator class.

3.2 Broker

The broker in the IoT publish/-subscribe model acts as a mediator between the publishers and subscribers. The broker acts as a common ground for the publishers to publish their information whilst also allowing the subscribers to request the type of information they want to receive. The publishers and subscribers never directly communicate as it would get extremely messy.

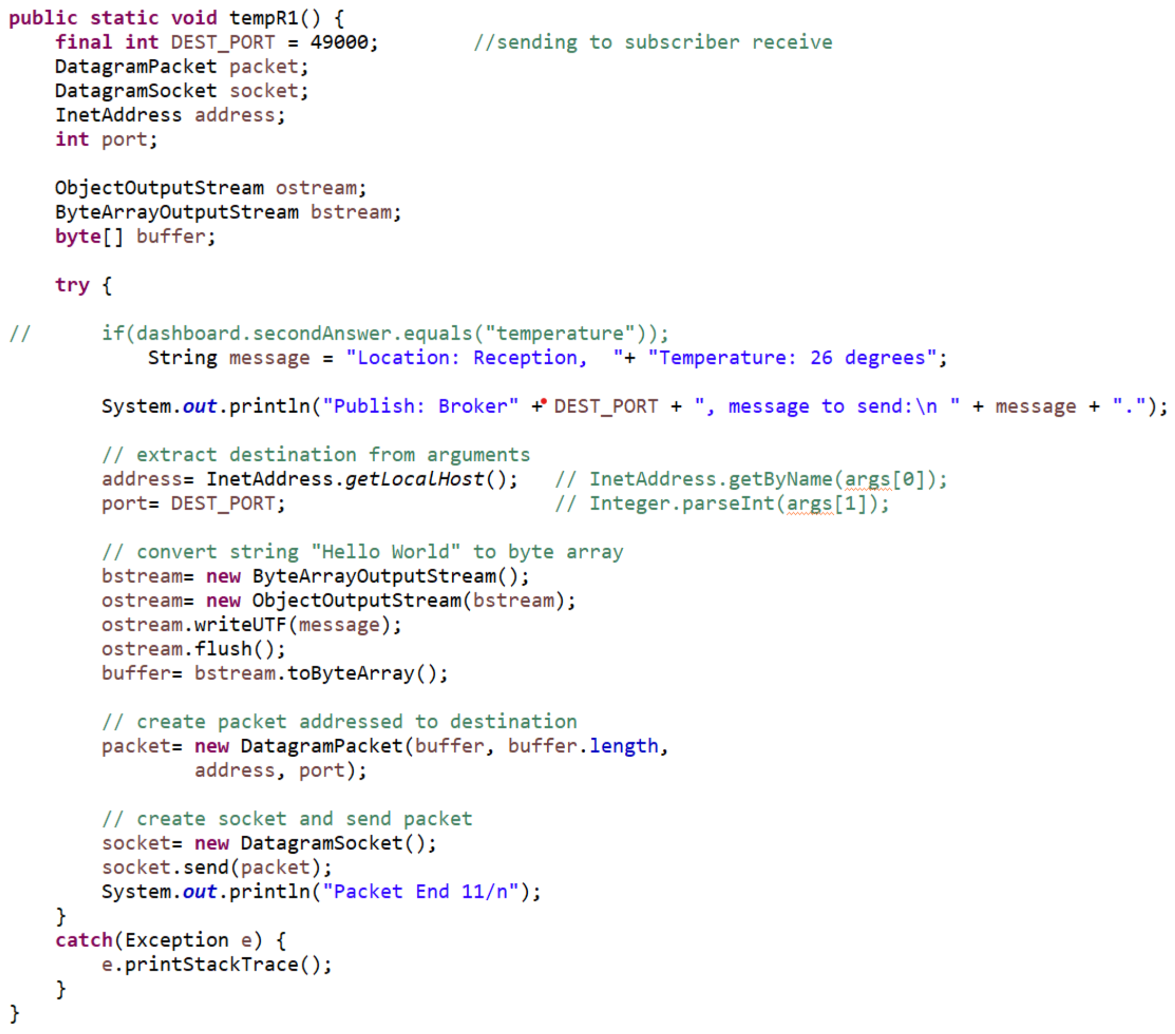


```
public static void receive() {
    final int RECV_PORT = 49000;        //49000 free port to use
    final int MTU = 1500;               //maximum package size

    DatagramPacket packet;              //initialize the packet being sent
    DatagramSocket socket;
    InetAddress address;
    int port;

    ObjectInputStream ostream;
    ByteArrayInputStream bstream;
    byte[] buffer;


    try {
        System.out.println("Broker has received");

        // extract destination from arguments
        address= InetAddress.getLocalHost(); // InetAddress.getByName(args[0]);
        port= RECV_PORT;                     // Integer.parseInt(args[1]);

        // create buffer for data, packet and socket
        buffer= new byte[MTU];
        packet= new DatagramPacket(buffer, buffer.length);
        socket= new DatagramSocket(port, address);

        // attempt to receive packet
        System.out.println("Trying to receive");
        socket.receive(packet);

        // extract data from packet
        buffer= packet.getData();
        bstream= new ByteArrayInputStream(buffer);
        ostream= new ObjectInputStream(bstream);

        // print data and end of program
        System.out.println("Data: " + ostream.readUTF());
        System.out.println("ReceiverProcess, receive successful");
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

My broker class has a basic packet receive implementation, containing a receiving port of 49000, also specifying a maximum packet size it can handle. It also has basic ACK implementations which when the data is received it will print “Data “with the data associated with the packet being sent.

3.3 Sensor

My sensor class contains the temperature and humidity reading of the room requested by the user in the dashboard. The dashboard sends data over to the sensor class which describes the room number sought after and either the temperature or the humidity reading of that room at that given time.
The class consists of six functions, each containing a send packet mechanism but with different information. Each function is called in the main line of the sensor class and whichever information I inputted in dashboard is called directly via the main line of the sensor class.
Taking one function as an example:

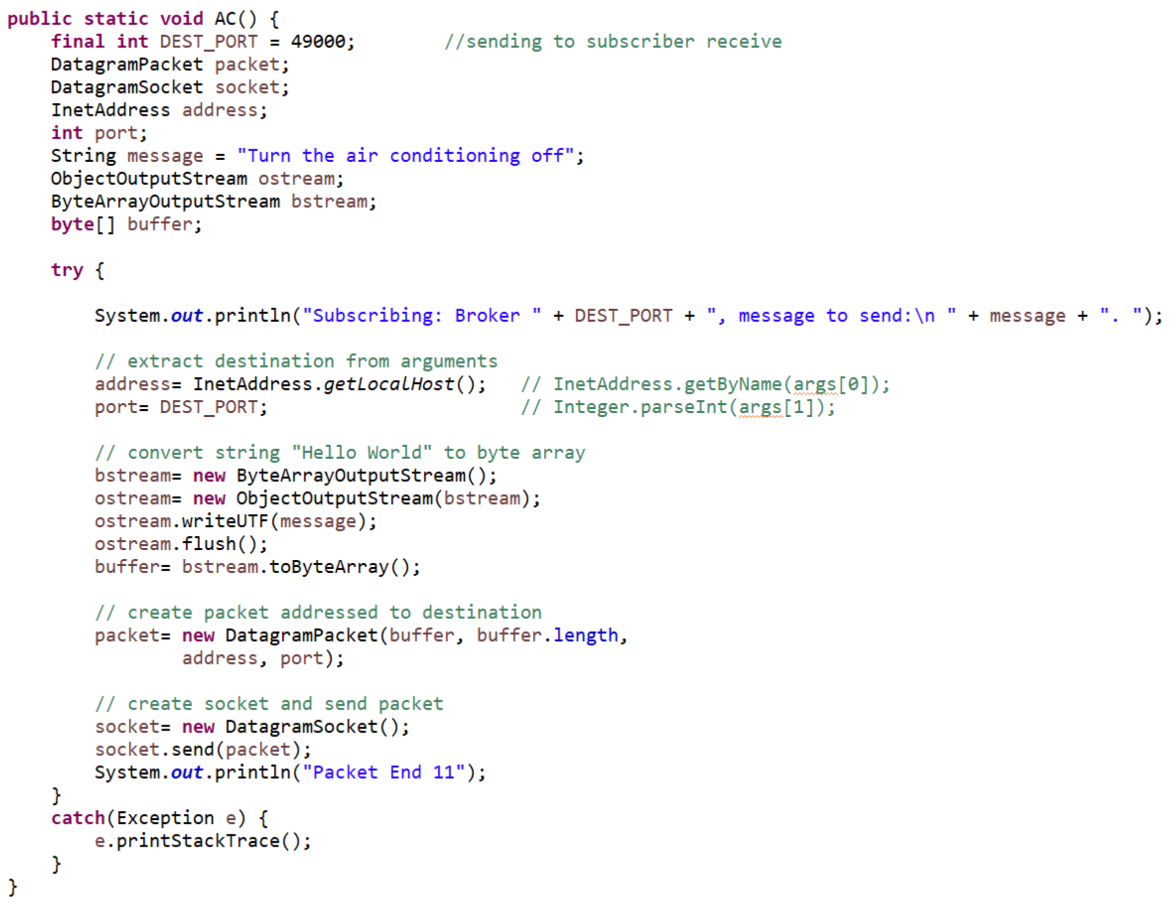


```
public static void tempR1() {
    final int DEST_PORT = 49000;        //sending to subscriber receive
    DatagramPacket packet;
    DatagramSocket socket;
    InetAddress address;
    int port;

    ObjectOutputStream ostream;
    ByteArrayOutputStream bstream;
    byte[] buffer;

    try {

//      if(dashboard.secondAnswer.equals("temperature"));
            String message = "Location: Reception,  "+ "Temperature: 26 degrees";

        System.out.println("Publish: Broker" + DEST_PORT + ", message to send:\n " + message + ".");

        // extract destination from arguments
        address= InetAddress.getLocalHost();   // InetAddress.getByName(args[0]);
        port= DEST_PORT;                       // Integer.parseInt(args[1]);

        // convert string "Hello World" to byte array
        bstream= new ByteArrayOutputStream();
        ostream= new ObjectOutputStream(bstream);
        ostream.writeUTF(message);
        ostream.flush();
        buffer= bstream.toByteArray();

        // create packet addressed to destination
        packet= new DatagramPacket(buffer, buffer.length,
                address, port);

        // create socket and send packet
        socket= new DatagramSocket();
        socket.send(packet);
        System.out.println("Packet End 11/n");
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

The above function contains the information regarding Room 1, which from above we can see is the reception of the building, also the temperature reading of that room. You can see the header of the packet above as “Publish: broker” + DEST_PORT and the data packet being message.
You can see above that the destination port of this packet is 49000, therefore the packet will be sent to the receiving port of 49000, which is the receiving port of the broker receive function.

3.4 Actuator

My actuator class follows similar implementation to my sensor class, the dashboard sends the information it gets from the user to the actuator, from there depending on which topic is chosen an instruction is subscribed to the broker.
This class consists of four different functions, connected to four different topics from the dashboard and are all associated with a given instruction.



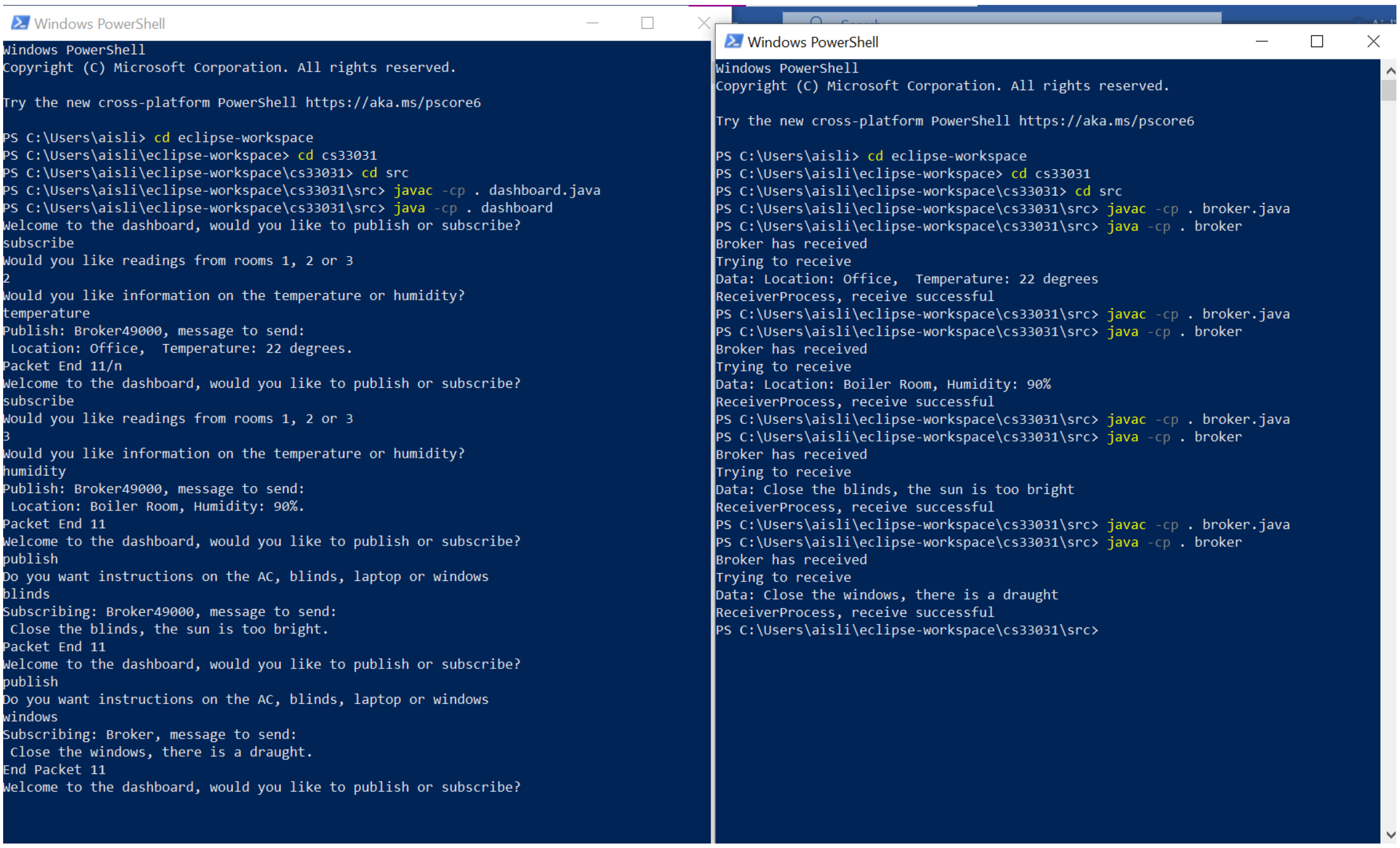
```java
public static void AC() {
    final int DEST_PORT = 49000;        //sending to subscriber receive
    DatagramPacket packet;
    DatagramSocket socket;
    InetAddress address;
    int port;
    String message = "Turn the air conditioning off";
    ObjectOutputStream ostream;
    ByteArrayOutputStream bstream;
    byte[] buffer;

    try {

        System.out.println("Subscribing: Broker " + DEST_PORT + ", message to send:\n " + message + ". ");

        // extract destination from arguments
        address= InetAddress.getLocalHost();   // InetAddress.getByName(args[0]);
        port= DEST_PORT;                       // Integer.parseInt(args[1]);

        // convert string "Hello World" to byte array
        bstream= new ByteArrayOutputStream();
        ostream= new ObjectOutputStream(bstream);
        ostream.writeUTF(message);
        ostream.flush();
        buffer= bstream.toByteArray();

        // create packet addressed to destination
        packet= new DatagramPacket(buffer, buffer.length,
                address, port);

        // create socket and send packet
        socket= new DatagramSocket();
        socket.send(packet);
        System.out.println("Packet End 11");
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

From the snippet of code above, exactly like the sensor class, the DEST_PORT of the send in the AC() function is also 49000, straight to the receiving function of the broker. The packet header above is '"Subscribing: Broker " + DEST_PORT', similarly the tail of the packet is the same as the tail of the packets also in sensor, finishing with the double 1 to notify the end of the transmission.

4 Discussion

The strengths of this program are the ability to send and receive packets through the broker without any disturbances within the code. The dashboard also loops back once the program is executed, making it faster to send packets continuously. The functionality of the code is implemented to a high standard and there is error handling within the dashboard which if a word is typed incorrectly or is not recognised the dashboard will give an error and repeat.

Although the flow control in my program is quick and implemented to a high standard, it could be improved. Unfortunately, I was unable to implement a loop within the broker to ensure that the code would loop, instead the code must be recompiled after a packet is received. As the deadline came closer, I decided to ignore this issue as I had more important problems within my code which needed attending to.

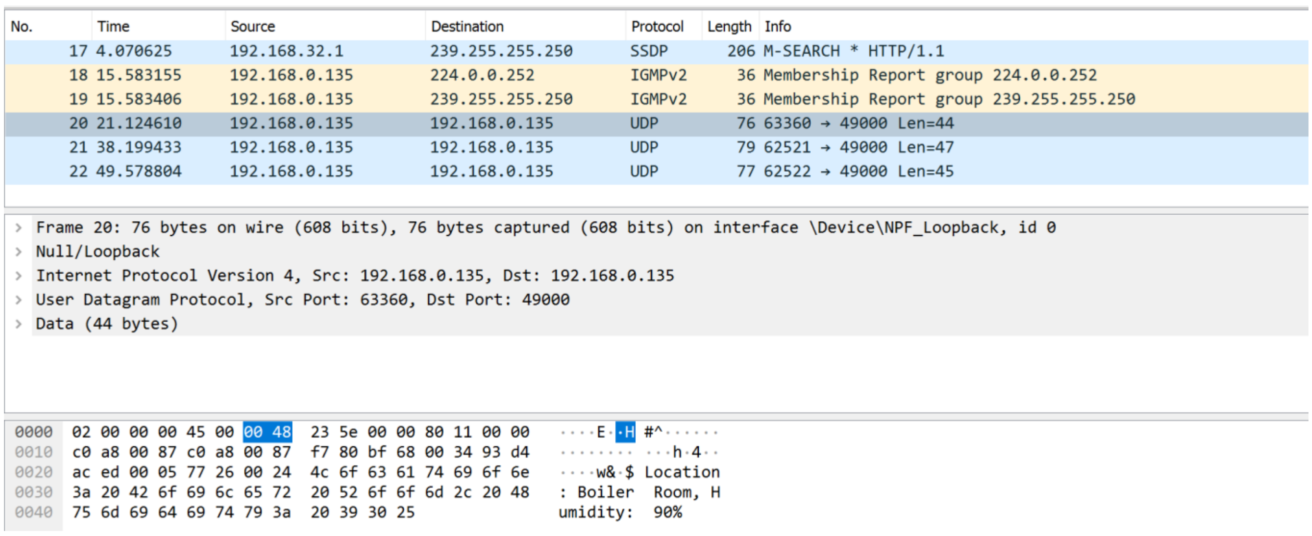


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\aisli> cd eclipse-workspace
PS C:\Users\aisli\eclipse-workspace> cd cs33031
PS C:\Users\aisli\eclipse-workspace\cs33031> cd src
PS C:\Users\aisli\eclipse-workspace\cs33031\src> javac -cp . dashboard.java
PS C:\Users\aisli\eclipse-workspace\cs33031\src> java -cp . dashboard
Welcome to the dashboard, would you like to publish or subscribe?
subscribe
Would you like readings from rooms 1, 2 or 3
2
Would you like information on the temperature or humidity?
temperature
Publish: Broker49000, message to send:
 Location: Office,  Temperature: 22 degrees.
Packet End 11/n
Welcome to the dashboard, would you like to publish or subscribe?
subscribe
Would you like readings from rooms 1, 2 or 3
3
Would you like information on the temperature or humidity?
humidity
Publish: Broker49000, message to send:
 Location: Boiler Room, Humidity: 90%.
Packet End 11
Welcome to the dashboard, would you like to publish or subscribe?
publish
Do you want instructions on the AC, blinds, laptop or windows
blinds
Subscribing: Broker49000, message to send:
 Close the blinds, the sun is too bright.
Packet End 11
Welcome to the dashboard, would you like to publish or subscribe?
publish
Do you want instructions on the AC, blinds, laptop or windows
windows
Subscribing: Broker, message to send:
 Close the windows, there is a draught.
End Packet 11
Welcome to the dashboard, would you like to publish or subscribe?
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\aisli> cd eclipse-workspace
PS C:\Users\aisli\eclipse-workspace> cd cs33031
PS C:\Users\aisli\eclipse-workspace\cs33031> cd src
PS C:\Users\aisli\eclipse-workspace\cs33031\src> javac -cp . broker.java
PS C:\Users\aisli\eclipse-workspace\cs33031\src> java -cp . broker
Broker has received
Trying to receive
Data: Location: Office,  Temperature: 22 degrees
ReceiverProcess, receive successful
PS C:\Users\aisli\eclipse-workspace\cs33031\src> javac -cp . broker.java
PS C:\Users\aisli\eclipse-workspace\cs33031\src> java -cp . broker
Broker has received
Trying to receive
Data: Location: Boiler Room, Humidity: 90%
ReceiverProcess, receive successful
PS C:\Users\aisli\eclipse-workspace\cs33031\src> javac -cp . broker.java
PS C:\Users\aisli\eclipse-workspace\cs33031\src> java -cp . broker
Broker has received
Trying to receive
Data: Close the blinds, the sun is too bright
ReceiverProcess, receive successful
PS C:\Users\aisli\eclipse-workspace\cs33031\src> javac -cp . broker.java
PS C:\Users\aisli\eclipse-workspace\cs33031\src> java -cp . broker
Broker has received
Trying to receive
Data: Close the windows, there is a draught
ReceiverProcess, receive successful
PS C:\Users\aisli\eclipse-workspace\cs33031\src>
```

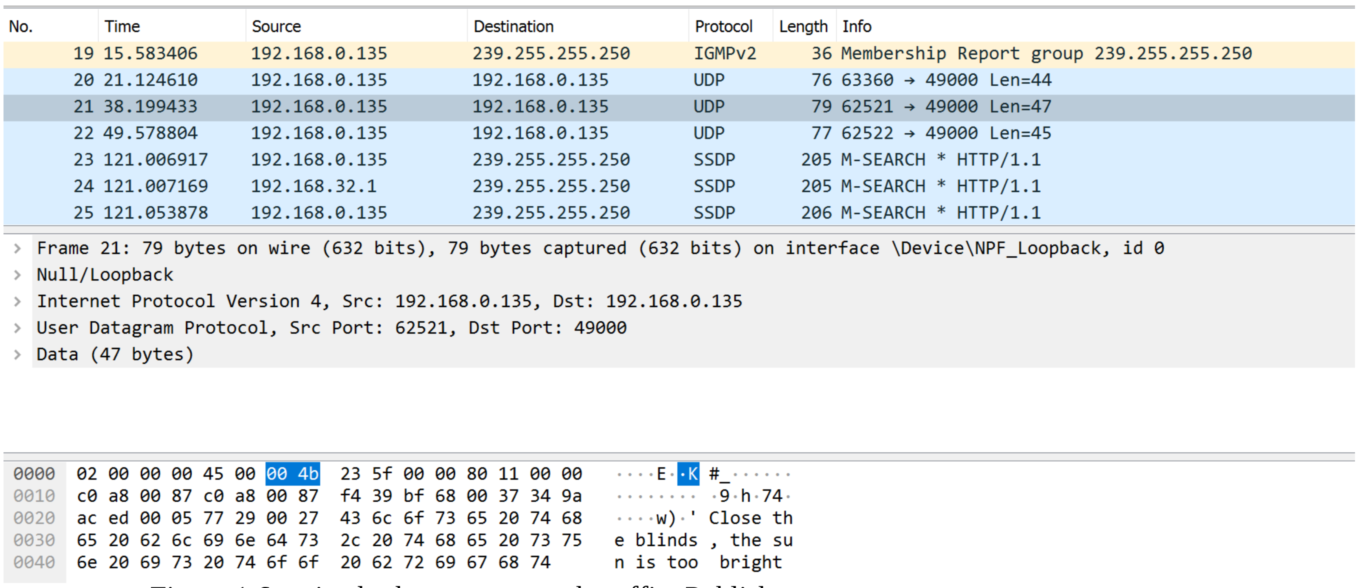
Figure 1.4: This shows the sending and receiving of packets from the dashboard to the broker.

Figure1.4 is a demonstration of multiple data packets being sent from the dashboard, via the sensor/actuator classes to the broker.



```
No.   Time         Source          Destination       Protocol  Length Info
   17 4.070625     192.168.32.1    239.255.255.250   SSDP         206 M-SEARCH * HTTP/1.1
   18 15.583155    192.168.0.135   224.0.0.252       IGMPv2        36 Membership Report group 224.0.0.252
   19 15.583406    192.168.0.135   239.255.255.250   IGMPv2        36 Membership Report group 239.255.255.250
   20 21.124610    192.168.0.135   192.168.0.135     UDP           76 63360 → 49000 Len=44
   21 38.199433    192.168.0.135   192.168.0.135     UDP           79 62521 → 49000 Len=47
   22 49.578804    192.168.0.135   192.168.0.135     UDP           77 62522 → 49000 Len=45

> Frame 20: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.0.135, Dst: 192.168.0.135
> User Datagram Protocol, Src Port: 63360, Dst Port: 49000
> Data (44 bytes)

0000  02 00 00 00 45 00 00 48  23 5e 00 00 80 11 00 00   ····E··H #^······
0010  c0 a8 00 87 c0 a8 00 87  f7 80 bf 68 00 34 93 d4   ········ ···h·4··
0020  ac ed 00 05 77 26 00 24  4c 6f 63 61 74 69 6f 6e   ····w&·$ Location
0030  3a 20 42 6f 69 6c 65 72  20 52 6f 6f 6d 2c 20 48   : Boiler  Room, H
0040  75 6d 69 64 69 74 79 3a  20 39 30 25               umidity:  90%
```

Figure 1.5: wireshark pcapp network traffic: Subscribe
Figure 1.5: This figure shows the packets that make up the message exchange for the data packet “Location: Boiler Room, Humidity: 90%”.



```
No.   Time          Source          Destination       Protocol  Length Info
   19 15.583406     192.168.0.135   239.255.255.250   IGMPv2        36 Membership Report group 239.255.255.250
   20 21.124610     192.168.0.135   192.168.0.135     UDP           76 63360 → 49000 Len=44
   21 38.199433     192.168.0.135   192.168.0.135     UDP           79 62521 → 49000 Len=47
   22 49.578804     192.168.0.135   192.168.0.135     UDP           77 62522 → 49000 Len=45
   23 121.006917    192.168.0.135   239.255.255.250   SSDP         205 M-SEARCH * HTTP/1.1
   24 121.007169    192.168.32.1    239.255.255.250   SSDP         205 M-SEARCH * HTTP/1.1
   25 121.053878    192.168.0.135   239.255.255.250   SSDP         206 M-SEARCH * HTTP/1.1

> Frame 21: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 192.168.0.135, Dst: 192.168.0.135
> User Datagram Protocol, Src Port: 62521, Dst Port: 49000
> Data (47 bytes)

0000  02 00 00 00 45 00 00 4b  23 5f 00 00 80 11 00 00   ····E··K #_······
0010  c0 a8 00 87 c0 a8 00 87  f4 39 bf 68 00 37 34 9a   ········ ·9·h·74·
0020  ac ed 00 05 77 29 00 27  43 6c 6f 73 65 20 74 68   ····w)·' Close th
0030  65 20 62 6c 69 6e 64 73  2c 20 74 68 65 20 73 75   e blinds , the su
0040  6e 20 69 73 20 74 6f 6f  20 62 72 69 67 68 74      n is too  bright
```

Figure 1.6 : wireshark pcapp network traffic: Publish
Figure 1.6: This figure shows the packets that make up the message exchange for the data packet “Close the blinds, the sun is too bright”.

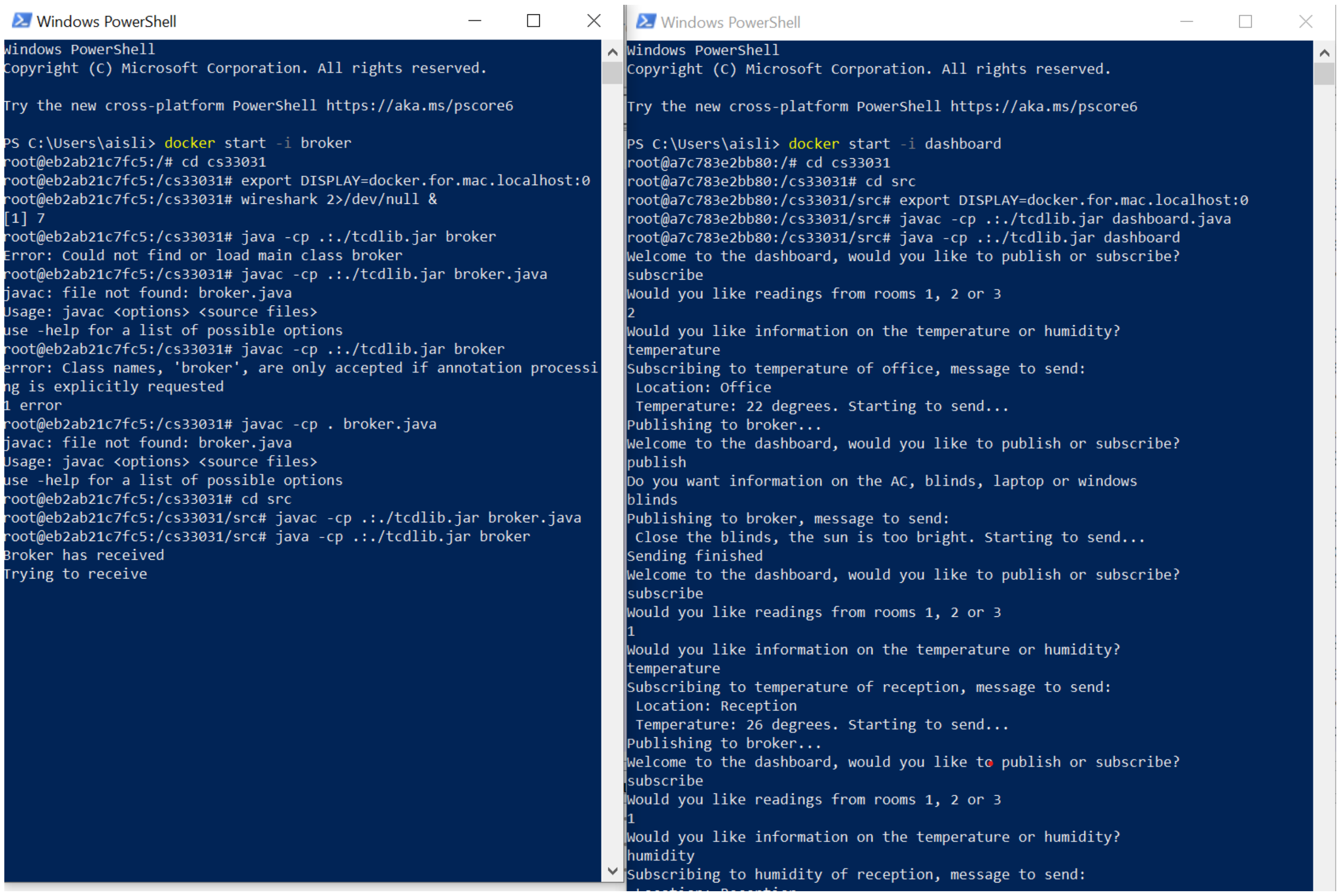
I tested my code implementation via my local machine, unfortunately I was getting the error listed below in figure 1.7:

I only managed to fix my code using rm *.class within the last two hours of submission and therefore although I have set up docker, my containers and connected them via a network they do not seem to be working up to the standard I would like them to work to. My fixed implementation of my code using docker is below under figure 1.8.

…

```
root@a7c783e2bb80:/cs33031/src# javac -cp . dashboard.java
dashboard.java:21: error: cannot access sensor sensor.tempR1(); ^ bad
class file: ./sensor.class class file has wrong version 55.0, should be
52.0 Please remove or make sure it appears in the correct subdirectory
of the classpath. 1 error
```

Figure 1.7



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\aisli> docker start -i broker
root@eb2ab21c7fc5:/# cd cs33031
root@eb2ab21c7fc5:/cs33031# export DISPLAY=docker.for.mac.localhost:0
root@eb2ab21c7fc5:/cs33031# wireshark 2>/dev/null &
[1] 7
root@eb2ab21c7fc5:/cs33031# java -cp .:./tcdlib.jar broker
Error: Could not find or load main class broker
root@eb2ab21c7fc5:/cs33031# javac -cp .:./tcdlib.jar broker.java
javac: file not found: broker.java
Usage: javac <options> <source files>
use -help for a list of possible options
root@eb2ab21c7fc5:/cs33031# javac -cp .:./tcdlib.jar broker
error: Class names, 'broker', are only accepted if annotation processi
ng is explicitly requested
1 error
root@eb2ab21c7fc5:/cs33031# javac -cp . broker.java
javac: file not found: broker.java
Usage: javac <options> <source files>
use -help for a list of possible options
root@eb2ab21c7fc5:/cs33031# cd src
root@eb2ab21c7fc5:/cs33031/src# javac -cp .:./tcdlib.jar broker.java
root@eb2ab21c7fc5:/cs33031/src# java -cp .:./tcdlib.jar broker
Broker has received
Trying to receive
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\aisli> docker start -i dashboard
root@a7c783e2bb80:/# cd cs33031
root@a7c783e2bb80:/cs33031# cd src
root@a7c783e2bb80:/cs33031/src# export DISPLAY=docker.for.mac.localhost:0
root@a7c783e2bb80:/cs33031/src# javac -cp .:./tcdlib.jar dashboard.java
root@a7c783e2bb80:/cs33031/src# java -cp .:./tcdlib.jar dashboard
Welcome to the dashboard, would you like to publish or subscribe?
subscribe
Would you like readings from rooms 1, 2 or 3
2
Would you like information on the temperature or humidity?
temperature
Subscribing to temperature of office, message to send:
 Location: Office
 Temperature: 22 degrees. Starting to send...
Publishing to broker...
Welcome to the dashboard, would you like to publish or subscribe?
publish
Do you want information on the AC, blinds, laptop or windows
blinds
Publishing to broker, message to send:
 Close the blinds, the sun is too bright. Starting to send...
Sending finished
Welcome to the dashboard, would you like to publish or subscribe?
subscribe
Would you like readings from rooms 1, 2 or 3
1
Would you like information on the temperature or humidity?
temperature
Subscribing to temperature of reception, message to send:
 Location: Reception
 Temperature: 26 degrees. Starting to send...
Publishing to broker...
Welcome to the dashboard, would you like to publish or subscribe?
subscribe
Would you like readings from rooms 1, 2 or 3
1
Would you like information on the temperature or humidity?
humidity
Subscribing to humidity of reception, message to send:
```

Figure 1.8

5. Summary

This report has described my attempt at a solution in understanding, implementing and designing an IoT -subscribe/publish protocol, in a small topology with a MQTT, The Message Queuing Telemetry Transport approach. Whilst the code I used throughout this program is simple and repetitive, it shows the simplicity that can be taken to design and implement a protocol with good flow control. The description of the implementation throughout this report highlights all the essential components which I used as part of my solution.

6 Reflection

After looking back at the last six weeks of attempting this assignment I am shocked at how much progress I have made from Part one until now. At the beginning of the assignment, although I successfully completed the docker walkthrough, I struggled to understand the basic concepts we had to implement for our IoT publish/-subscribe protocol to successfully send and receive data packets. I have a much clearer understanding of protocol development and the process behind designing a protocol.
Apart from this report, this code took approximately 25 hours to complete.