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

CSU33031 Computer Networks
Assignment #1: Protocols

Róisín Ní Bhriain, Std# 18326577

October 29, 2021

Contents

1 Introduction **1**

2 Theory of Topic **2**
2.1 Broker 2
2.2 Dashboard 3
2.3 Actuator 4
2.4 Sensor 4
2.5 Communication and Packet Description 5

3 Implementation **5**
3.1 Broker 5
3.2 Dashboard 7
3.3 Sensor 8
3.4 Actuator 8
3.5 Packet Encoding 9

4 Summary **10**

5 Reflection **10**

1 Introduction

The task was to create a system using a broker and publish-subscribe techniques using UDP datagrams. The idea was that a "dashboard" could subscribe to certain topics from sensors and then these sensors would "publish" this information. The dashboard would also be able to "publish" to actuators and turn them off and on. The actuators would "subscribe" to the dashboard to receive the instructions.

2 Theory of Topic

There are several parts working together to create the system for the air conditioning. The broker receives all communication. The broker waits for other components to subscribe and then keeps them in a list of subscribers. Then when any packets are published to the broker it will send them to any subscribers to that topic or discard them. The broker sends an ack packet every time it receives a packet whether it is a publish or a subscribe packet. The dashboard is a user-controlled publisher-subscriber. The actuator is a basic subscriber. The sensor is a basic publisher.

2.1 Broker

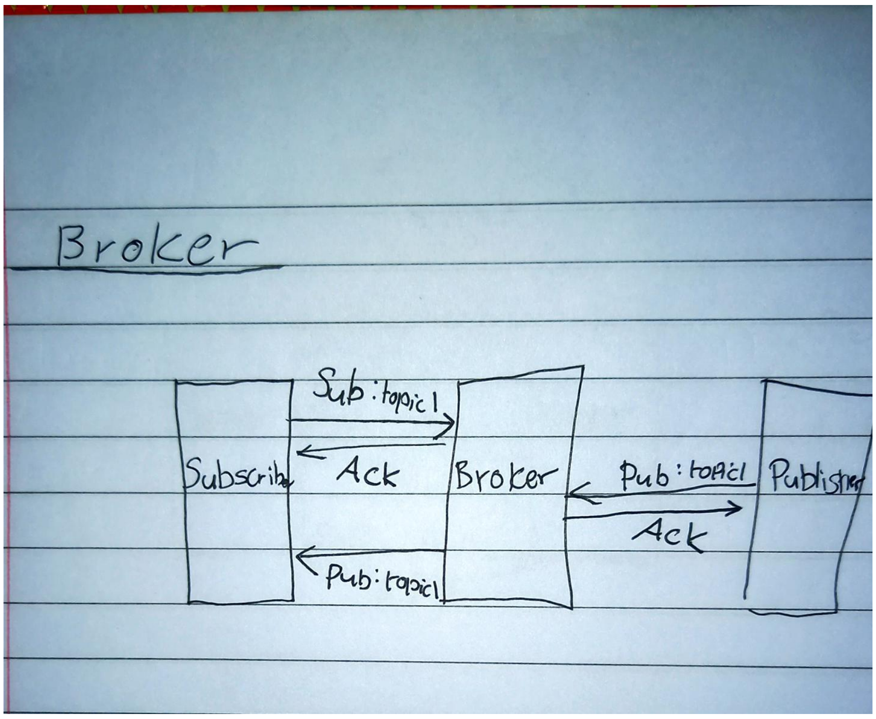

Figure 1: The Broker accepts packets from multiple subscribers and publishers. It essentially listens for packets and when it receives one it decodes what is the necessary action for that packet. Every time it receives a packet it sends an ack packet to the socket it was obtained from. Then it will parse the received packet to see if any subscribers need to be added or removed. If it is a publishing packet it will then send the information to any subscribers subscribed to that topic.

2.2 Dashboard

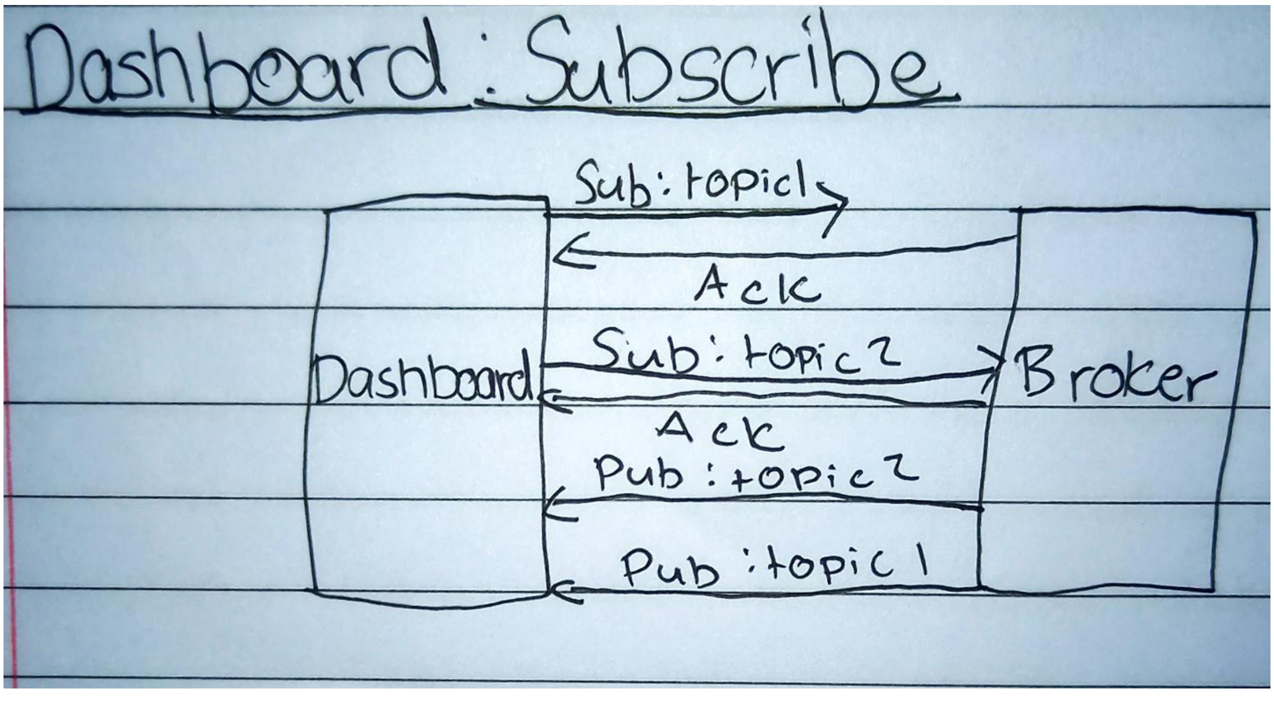

Figure 2: The dashboard only communicates with the broker. It will subscribe to topics from this broker. The way I decided to do this was to implement a user-driven system where the user will enter in the ID of the sensor that it wishes to subscribe to and also the topic. The two topic choices I have are the humidity and the temperature picked up by the sensor. The reason I decided to do it this way is that certain users will only be able to subscribe to certain sensors in a real system. The user-driven choice was just to make it easier for me to test.

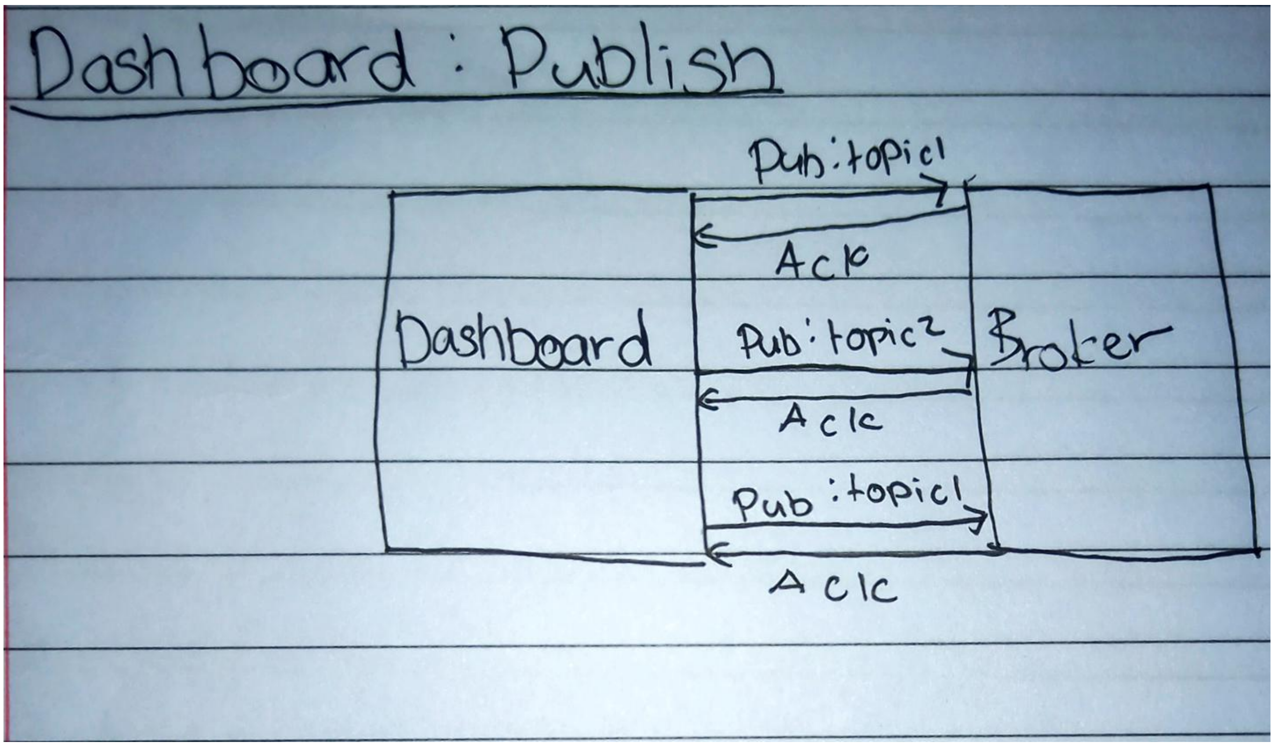

Figure 3: The dashboard also has the option to publish to actuators i.e. turn them on or off. It gives the user the option of which actuator to communicate with and then whether to turn it off or on. The dashboard also has the option to wait i.e. to wait for packets from topics subscribed to. This would have been on a timer system if I had more time.

2.3 Actuator

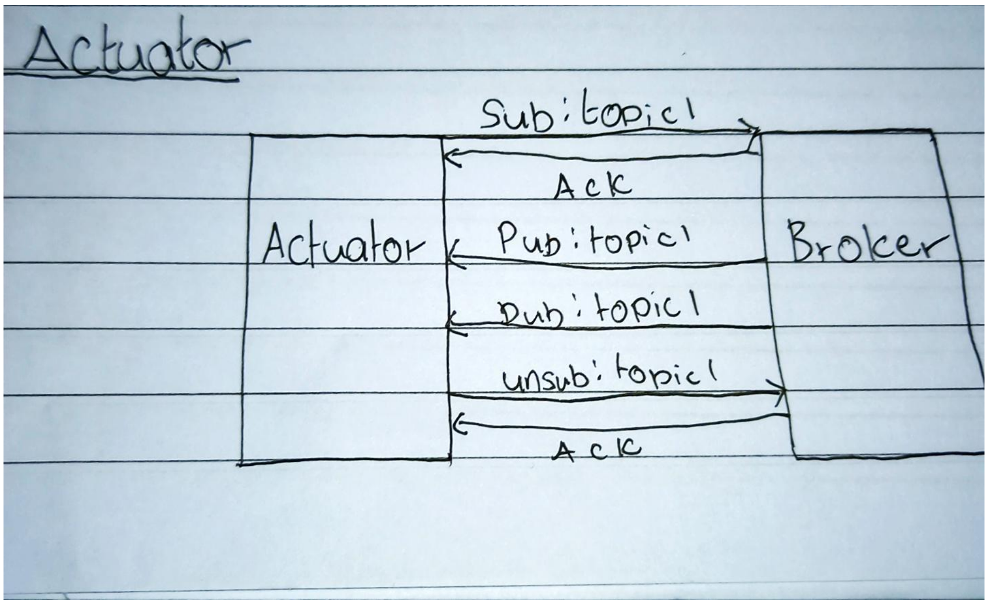

Figure 4: The Actuator is a basic subscriber. It essentially subscribes to the dashboard for “status” updates. And the has the option to either “wait” or “unsubscribe”. It only sends a subscribe packet to the broker and then waits until it receives some published information on the status.

2.4 Sensor

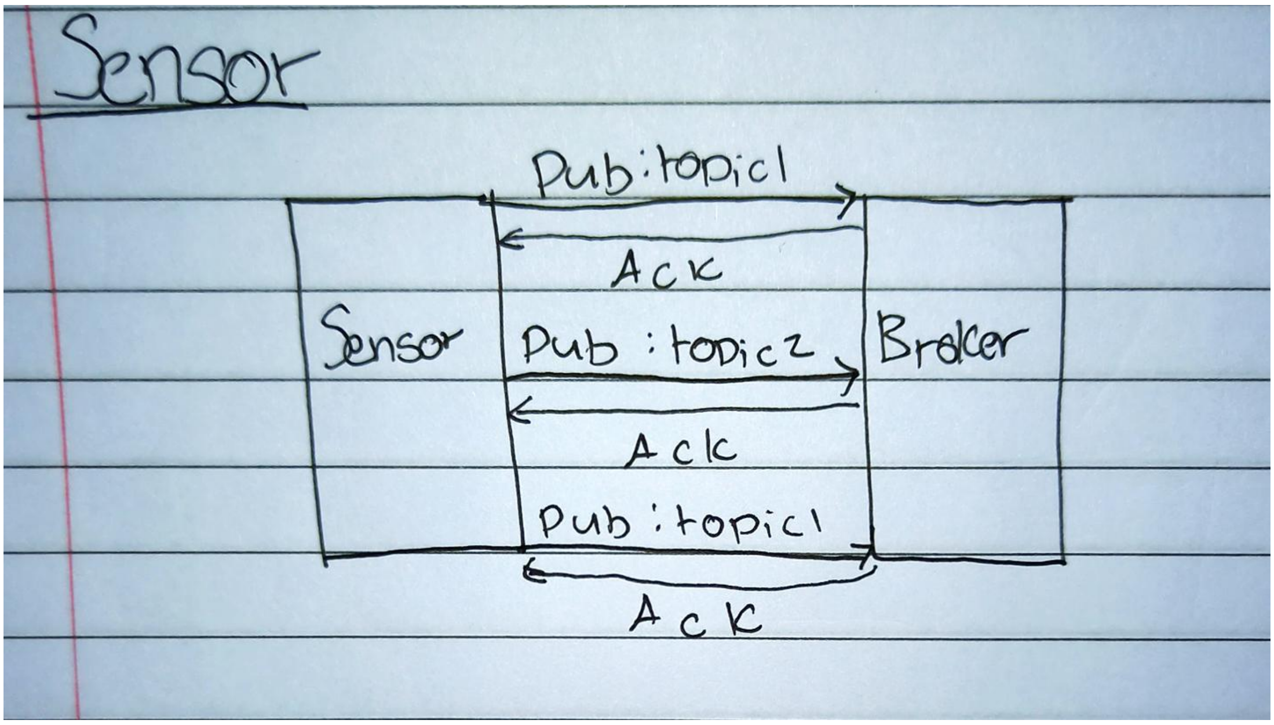

Figure 5: The Sensor is a basic publisher. It sends packets to the broker based on whether the user wants to send the topic of humidity or temperature.

2.5 Communication and Packet Description

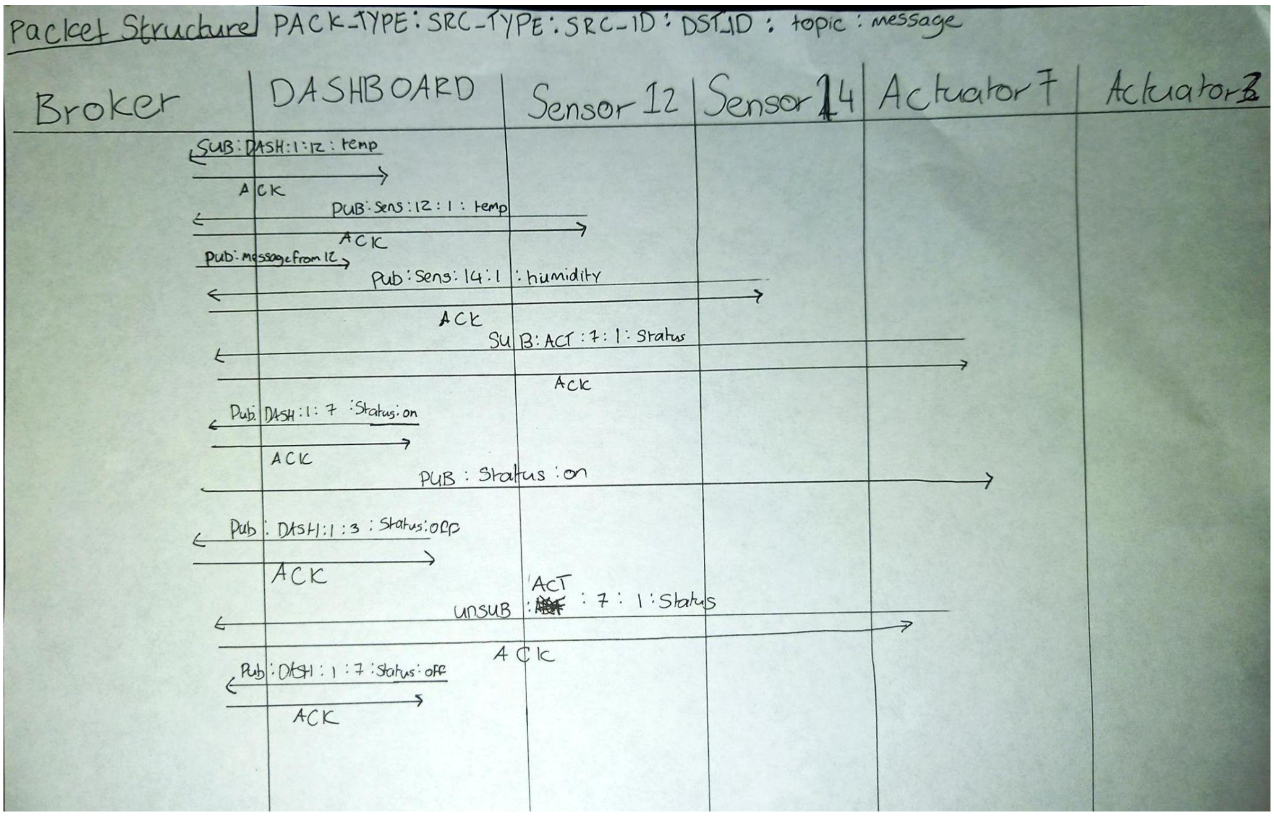

Figure 6: As you can see here everything goes through the broker. The broker controls the entire system. So it relies

on the correct parsing of the packets. The packets all have a similar structure which is shown at the top of the diagram. I attempted to get the header restructured with the type in the header but this was giving me errors when the information was being unpacked. So I decided to give them all the same structure in the payload. This part was then unpacked and parsed. This figure also shows an example of the system working with the dashboard subscribing and publishing. When a component is not subscribed to a topic the published message never reaches that component.

3 Implementation

This section will focus on the code used to implement the functionality of the system. There were some similar functions in each of the components such as the connect function, the send and receive function and so I decided not to explain these. I also didn't include the constructors as they are not complicated.

3.1 Broker

```
	public static void listen () {

		String message ;

		try {
			// Endless loop: attempt to receive packet, notify receivers, etc
			while(true) {
				DatagramPacket packet;

				ObjectInputStream ostream;
				ByteArrayInputStream bstream;
				byte[] buffer;

				try {
					System.out.println("Broker is receiving");

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

					// 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);

					message = ostream.readUTF();
					Scanner scanner = new Scanner(message);
					String mess = scanner.next();

					clientAdd = packet.getAddress();
					clientPort = packet.getPort();
					InetSocketAddress dstaddress = new InetSocketAddress(clientAdd,
clientPort);

					// send an ack
					send("", TYPE_ACK, dstaddress);

					parse(message);

				}
				catch(Exception e) {
					e.printStackTrace();
				}

			}
		} catch (Exception e) {if (!(e instanceof SocketException)) e.printStackTrace();}
```

```
        }

        public static void getSub (String [] data) {
                Subscriber sub = null;

                for (int i = 0; i < subscribers.size(); i++) {
                        Subscriber current = subscribers.get(i);
                        if (current.id == Integer.parseInt(data[3]) && current.info.equals(data[4])) {
                                String string = "";
                                for (int j = 3; j < data.length; j++) {
                                        string += data[j] + ":";
                                }
                                send(string, TYPE_PUB, sub.dstAddress);
                        }
                }
        }

        public static void parse (String message) {
                System.out.println("Received: " + message);
                InetSocketAddress add = new InetSocketAddress(clientAdd, clientPort);

                String [] data = message.split(":");
                if (data[0].equals(Integer.toString(TYPE_UNKNOWN))) {
                        System.out.println("Error");
                } else if (data[0].equals(Integer.toString(TYPE_PUB))) {
                        getSub(data);
                } else if (data[0].equals(Integer.toString(TYPE_SUB))) {
                        Subscriber sub = new Subscriber(Integer.parseInt(data[2]), add, data[4]);
                        System.out.println("This is where the problem is: " + Integer.parseInt(data[2]));
                        subscribers.add(sub);
                } else if (data[0].equals(Integer.toString(TYPE_UNSUB))) {
                        Subscriber sub = new Subscriber(Integer.parseInt(data[2]), add, data[3]);
                        subscribers.remove(sub);
                }
        }

        public static void main(String[] args) {
                try {
                        socket= new DatagramSocket(DEFAULT_BROKER_PORT);
                }
                catch (SocketException e) {
                        e.printStackTrace();
                }

                Broker broker = new Broker(socket);
                subscribers = new ArrayList<Subscriber>();

                broker.listen();
        }
}
```

Listing 1: This component does not use any user input and is essentially just a listener. The listen function waits until it receives a packet and then uses the parse function to deal with the packet. The parse function is used to check what kind of packet has been received. This function will print an error message if the packet is not of a normal type. If it is a subscribe packet then the subscriber will be added to the subscriber ArrayList. If it is an unsubscribe packet it will be removed from the subscriber list. If the packet is a publish packet then it will check if there are any subscribers for that topic by using the getSub function. The getSub function will search through the ArrayList of subscribers for subscribers that are subscribed to the published topic and then it will send the message to these subscribers. The main function in the broker just does the setup for the broker class and then calls the listen function.

3.2 Dashboard

```
	public static void main(String[] args) {

		try {
			socket= new DatagramSocket();
		}
		catch (SocketException e) {
			e.printStackTrace();
		}

		Dashboard dash = new Dashboard(socket);
		Scanner scanner = new Scanner(System.in);

		dash.connect();

		String message = "";

		while (true) {
			System.out.println("would you like to subscribe or publish or wait?");
			message = scanner.next();
			if (message.equals("subscribe")) {
				System.out.println("Which sensor would u like to see?");
				int sensor = scanner.nextInt();
				System.out.println("Would you like to see humidity or temperature?");
				message = scanner.next();
				if (message.equals("humidity")) {
					send("Dash :1:" + sensor + ":humidity", TYPE_SUB, dstAddress);
					dash.receive();
				} else if (message.equals("temperature")) {
					send("Dash :1:" + sensor + ":temperature", TYPE_SUB, dstAddress);
					dash.receive();
				}
				dash.receive();
			} else if (message.equals("publish")){
				System.out.println("Which actuator would u like to communicate with?");
				int actuator = scanner.nextInt();
				System.out.println("Would you like to turn it on or off?");
				message = scanner.next();
				if (message.equals("on")) {
					send("Dash :1:" + actuator + ":status:on", TYPE_PUB, dstAddress);
				} else if (message.equals("off")) {
					send("Dash :1:" + actuator + ":status:off", TYPE_PUB, dstAddress);
				}
				dash.receive();
			} else if (message.equals("wait")) {
				dash.receive();
			}
		}

	}
}
```

Listing 2: The dashboard has the same connect and send function as the other components. The receive is slightly different as it prints the received message. The main is where most of the dashboard runs from. It takes user input to see if the user wants to subscribe, wait or publish. If the dash is subscribing it will ask the user to choose a sensor and a topic: temperature or humidity. Then it will send a subscription packet to the broker. If the user decides to publish then they will be asked what actuator to publish to and what they would like the status to be: on or off. The other option is to wait for any subscriptions to come through.

3.3 Sensor

```
	public String toString () {

		return " in" + " Room number : " + roomNo + " Floor number : " + floor;
	}

	public static void main(String[] args) {

		connect();
		try {
			socket= new DatagramSocket();
		} catch (SocketException e) {
			e.printStackTrace();
		}

		Sensor sensor = new Sensor(socket);
		Scanner scanner = new Scanner(System.in);

		System.out.println("What is the ID of this sensor?");
		id = scanner.nextInt();
		System.out.println("What is the room number of this sensor?");
		roomNo = scanner.nextInt();
		System.out.println("What is the floor number of this sensor?");
		floor = scanner.nextInt();

		while (true) {
			System.out.println("Which would you like to publish? humidity or temperature");
			String received = scanner.next();
			if (received.equals("humidity")) {
				System.out.println("What is the humidity?");
				humidity = scanner.nextDouble();
				sensor.send("Sensor " + ":" + id + ":" + "1" + ":" + "humidity: " + humidity + "%"
+ sensor.toString());

				// to receive ack
				sensor.receive();
			} else if (received.equals("temperature")) {
				System.out.println("What is the temperature?");
				temp = scanner.nextDouble();
				sensor.send("Sensor "+ ":" + id + ":" + "1" + ":" + "temperature: "+ temp + "c" +
sensor.toString());

				// to receive ack
				sensor.receive();
			}
		}

	}
}
```

Listing 3: The sensor is a very simple publisher. It takes in some information from the command line and inputs it into some variables such as the id and the room number etc. Otherwise, it is a simple endless while loop asking the command line whether to publish humidity or temperature. It will then send this information to the broker.

3.4 Actuator

```
	public String toString () {
		String data = "Actuator \nRoom Number: " + roomNo + "\nFloor Number: " + floor + "\nI.D.
Number: " + id + "\nStatus: ";

		if (status) data += "on";
		else data += "off";
```

```
                return data;

        }

        public static void main(String[] args) {

                try {
                        socket= new DatagramSocket();
                }
                catch (SocketException e) {
                        e.printStackTrace();
                }

                Actuator act = new Actuator(socket);

                act.connect();

                Scanner scanner = new Scanner(System.in);

                System.out.println("What is the ID of this actuator?");
                id = scanner.nextInt();
                System.out.println("What is the room number of this actuator?");
                roomNo = scanner.nextInt();
                System.out.println("What is the floor number of this actuator?");
                floor = scanner.nextInt();
                act.send(TYPE_SUB, "Actuator :" + id + ":" + "1" + ":status");
                act.receive();
                boolean finished = false;

                while (!finished) {
                        System.out.println("Would you like to wait or unsubscribe?");
                        String received = scanner.next();
                        if (received.equals("wait")) {
                                act.receive();
                        } else if (received.equals("unsubscribe")){
                                act.send(TYPE_UNSUB, "Actuator :" + id + ":" + "1" + " :status");
                                act.receive();
                                finished = true;
                        }

                }

        }
}
```

Listing 4: The Actuator subscribes to the dashboard for updates on status. This was implemented so that it could only subscribe to one dashboard but the option is there to expand this by giving each dashboard an id. The default id, in this case, is 1. There is an option to unsubscribe or wait for new subscriptions. The wait will simply wait for a new packet and the unsubscribe will send an unsubscribe packet to the broker and it will no longer receive updates from the dashboard.

3.5 Packet Encoding

```
public class SenderReceiver {

        static DatagramSocket socket;

        public static final int DEFAULT_BROKER_PORT= 49000;

        static final byte TYPE_UNKNOWN = 0;
        static final byte TYPE_ACK = 1;
        static final byte TYPE_PUB = 2;
        static final byte TYPE_SUB = 3;
        static final byte TYPE_UNSUB = 4;


        public static byte [] packPacket (int type, String message) {
                byte [] data = null;
                DatagramPacket packet = null;
                ObjectOutputStream ostream;
                ByteArrayOutputStream bstream;
                byte[] buffer = null;
                String finalData = type + ":" + message;
                try {
                        bstream= new ByteArrayOutputStream();
                        ostream= new ObjectOutputStream(bstream);
                        ostream.writeUTF(finalData);
                        ostream.flush();
                        buffer= bstream.toByteArray();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
                return buffer;
        }

}
```

Listing 5: This shows all the packet types that can be encoded into a packet as numbers. The constructor takes a socket and that is the socket used for the subclass. The packPacket function takes the packet type and the packet message and encodes it into a byte array that can be used in a UDP Datagram packet.

4 Summary

This code implemented a very basic system for air-conditioning in a building controlled by a dashboard. The broker is the component through which all the packets are sent. The dashboard is user-controlled and can turn actuators on and off by publishing packets. It can also subscribe to information from sensors. The sensor or sensors will publish either humidity or temperature to the broker. The actuator will subscribe to the dashboard so the dashboard can turn it on and off.

5 Reflection

A lot of this system relies on the user entering information. While this was solely for testing purposes I would have preferred to have the sensor on a timer system for example. Sending humidity and/or temperature every 30 seconds or so. The ArrayList for keeping track of subscribers could also be split into different types so that you can have more than one dashboard with a different dashboard ID etc. It might also be easier if the dashboard was able to access each sensor and actuator by room number and floor number rather than by ID. This would require a more rigid structure to each actuator and sensor rather than using the command line to assign rooms and floors or using random numbers.