



CSU33031 Computer Networks

Assignment #1: PUB/SUB Protocol

Saumya Bahuguna
21344349
2nd November
2021

Contents

1	Introduction	
2	Theory of Topic	
2.1	Transport Layer	2
2.2	Network Layer	3
2.3	Publish-Subscribe	3
2.3	Maximum Transmission Unit	3
3	Implementation	
3.1	Networks	4
3.2	Node	4
3.3	Acknowledgements.	7
3.4	Breaker.	8
3.5	Broker	8
3.6	Publisher	11
3.7	Subscriber.	14
3.8	Protocol.	16
4	Discussion	
4.1	Port Numbers and IP addresses.	17
5	Summary	17
6	Reflection	17

1 **Introduction**

This report aims at describing the implementation of the Publish-Subscribe protocol as a task for Assignment-1. I will start with exploring the topics and details related to my protocol, continuing with the approach I took to create the publish-/subscribe-mechanism for processes based on UDP (User Datagram Protocol) datagrams using Java on Docker. I will then end my report with a discussion, summary, and reflection on the assignment.

2 **Theory of Topic**

In this section, I will describe the concepts and details that were used to realize the final solution. In particular, I will briefly describe the network and transport layer from Open Systems Interconnection (OSI) model and Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). These are necessary to understand as these are used to get to design the Publish-Subscribe protocol.

2.1 **Transport Layer**

Transport Layer is the 4th layer of the Open Systems Interconnection (OSI), transport layer ensures the End-to-End Delivery of the complete message. The transport layer receives the formatted data from the upper layers, performs Segmentation, and implements error checking mechanisms and data flow controls to ensure proper data transmission.

The transport layer has two types of transmissions

(a) Connection-oriented transmissions: - the Transmission Control Protocol is the common form of connection-oriented transmission it happens in three phases First, a connection is established, then the required data is transferred, and then the connection is terminated this type of transmission is reliable and secure.

(b) Connectionless transmissions the User Datagram Protocol (UDP) is the common form of connectionless transmission, this is just a one-phase process it does not require a connection which results in faster connection, which is why UDP is used for real-time mechanisms.

A header in UDP contains 8 bytes which is less compared to 20 bytes of Header size in TCP. A UDP header is divided into 4 required fields, Source port number which represents the sender's port, destination port number which represents the receiver's port, length of data which is the total size of each datagram i.e. Including both header and data, and a checksum value inside a UDP header which is generated by the protocol sender as a mathematical technique to help the receiver detect messages that are corrupted or tampered with. The transport layer then forwards the segmented data to the Network Layer.

2.2 Network Layer

The network layer in the OSI module is similar to the Internet layer in the TCP/IP module. It is the 3rd Layer in the OSI module and is responsible for the transmission of data from one host to another located in different networks. It is also responsible for packet routing i.e selection of the shortest path to transmit the packet, from the number of routes available

network layer places the sender's and receiver's Protocol addresses on the header. A protocol is an agreed-upon way of formatting the data so that two or more devices are able to communicate with and understand each other. a number of different protocols are used, Internet Protocol (IP) is the standard for routing packets across interconnected networks. Internet Protocol (IP) is a protocol that uses datagrams to communicate over packet-switched networks. An Internet Protocol (IP) header reports the sending and receiving addresses, header length, packet length, Time to Live, and which transport packet is being used. Two versions of IP currently coexist in the global Internet; Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). an Ipv4 address is 32 bits in size whereas an IPv6 address is 128 bits in size. In my solution, java.net library is used where an IP Address is represented by an

InetAddress object, which may use either IPv4 or IPv6. IP Addresses are the only information needed to understand the role of IP in the DatagramPacket.

2.3 Publish-Subscribe

The Publish-Subscribe pattern is an architectural design that provides a framework for exchanging messages between publishers and subscribers. This pattern has a Subscriber and a Publisher that relies on a Broker that relays the messages between them. The publisher publishes messages to a channel that subscribers can sign up to. Publish-Subscribe enables the movement of messages between different components of the system without the components being aware of each other's identity.

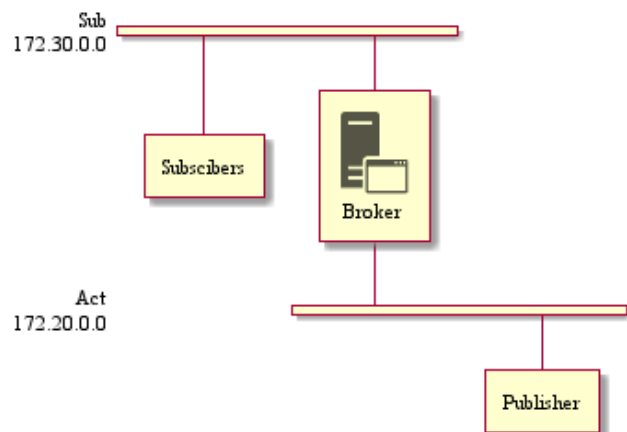
2.4 Maximum Transmission Unit

Maximum Transmission Unit (MTU) is a measurement representing the largest data packet that a network-connected data will accept. it depends on the underlying hardware architecture for e.g., Ethernet and many other types of packets MTU size is 1500 bytes, so a datagram of size 1500 or less can be transmitted with this protocol. So say if a datagram of size 2000 arrives at a router it will split their payloads into two packets that each is smaller than a size of 1500. This is not used in the assignment, but it was still important to determine a maximum packet size for the protocol so that it could accommodate relatively small MTUs.

3 Implementation

3.1 Networks

I have created 2 networks for my solution, SUB for subscribers with IP subnet address 172.30.0.0 and ACT for Publishers with IP subnet 172.20.0.0, Broker is connected to both networks and is the only connection between them. Figure 1 illustrates this.



Networks in the Protocol

Saumya Bahuguna

Figure 1- This Figure shows the number of networks in the solution. Explained in 3.1.

3.2 Node

I have used the Node class provided in Blackboard, We were provided with a Listener class that extends java.lang.Thread, it waits endlessly while listening, and notifies and calls onReciept Function. onReciept is an abstract method so each class that extends node has to override and implement it in its own way. This allows handling of the packet received in different ways as per the class's functionality. The code in **Listing 1** shows the Listener Class.

```

public abstract void onReceipt(DatagramPacket packet);

class Listener extends Thread {

    public void go(){ /* Starts Listening on the port of the class
                        where the function go was called and waits*/

        latch.countDown();
    }

    public void run() {
        try {
            latch.await();
            while (true) {
DatagramPacket packet = new DatagramPacket(new byte[PACKETSIZE],PACKETSIZE);
//Creates a Packet of Packet Size which is 500 for this code

                socket.receive(packet);

                onReceipt(packet); /* calls the abstract Class onReceipt
                                    which is defined in every sub class*/
            }
        } catch (Exception e) {
            if (!(e instanceof SocketException))
                e.printStackTrace();
        }
    }
}

```

Listing 1: The Listener Class, shows how on receive a packet the onReceipt Function is called, which is then handled by each sub-class.

Node defines all variables and functions which are commonly used by other classes that extend Node. PACKETSIZE which is used to define the size of DatagramPackets is set to 500 so that the connection could work on small MTUs and send/receive data without the router needing to divide and split the data. PUB_DST and SUB_DST are two string Variables with IP addresses of networks. The Broker and Publisher were given individual port numbers defined in PUB_PORT and BKR_PRT. Other Default variables are ACK, BRK, NEW, Pub, SUB, USUB, and Mes. which are used in Node and other Subclasses to set or verify the type of the packet.

The type of Packets used in the protocol were Datagram packets the UDP header contains the source and destination address, the packet bytes are defined in a way that Byte 0 is the type of the packet, i.e. one of the above-defined variables. The Byte 1 is always set to 0 as it was supposed to be used for implementing error control i.e Sliding Window ARQ(Automatic Speed Request) But was not implemented for this assignment due to time constraints. Byte 2 to 5 are assigned to the topic numbers i.e the agreed integer corresponding to a room in the code. the bytes from 5 to 7 contain the data(Room number).The code in Listing 2 shows how the Node class creates packets.

```

private byte[] createPacketData(int type, int sequenceNumber, int topicNumber, byte[]
message) {
    byte[] data = new byte[PACKETSIZE];
    data[0] = (byte) type;

    data[1] = (byte) sequenceNumber;
    ByteBuffer byteBuffer = ByteBuffer.allocate(4);
    byteBuffer.putInt(topicNumber);

    byteBuffer.rewind();
    byte[] topicNumberArray = byteBuffer.array();
    for (int i = 0; i < 4; i++) {
        data[i + 2] = topicNumberArray[i];
    }
    for (int i = 0; i < message.length && i < PACKETSIZE; i++) {
        data[i + 6] = message[i];
    }
    return data;
}

protected DatagramPacket[] createPackets(int type, int topicNumber, String
message, InetAddress dstAddress) throws SocketException {
    int messageSize = PACKETSIZE - 6;
    byte[] tmpArray = message.getBytes();
    byte[] messageArray = new byte[tmpArray.length];
    for (int i = 0; i < tmpArray.length; i++) {
        messageArray[i] = tmpArray[i];
    }
    int numberOfPackets = 0;
    for (int messageLength = messageArray.length; messageLength > 0;
messageLength -= messageSize) {
        numberOfPackets++;
    }
    DatagramPacket[] packets = new DatagramPacket[numberOfPackets];
    int offset = 0;
    for (int sequenceNumber = 0; sequenceNumber < numberOfPackets; sequenceNumber++) {
        byte[] dividedMessage = new byte[messageSize];
        for (int j = offset; j < offset + messageArray.length; j++) {
            dividedMessage[j] = messageArray[j + offset];
        }
        byte[] data = createPacketData(type, sequenceNumber, topicNumber, dividedMessage);
        DatagramPacket packet = new DatagramPacket(data, data.length,
dstAddress);

        packets[sequenceNumber] = packet;
        offset += messageSize;
    }
    return packets;
}

```

Listing 2: Creates an array of the packets which was for implementing the error control. packet is created after stating the type, message and topic number and the destination address.

The Node class also contains functions i.e. set type which sets the type of the packet based on the function, get Type which gets the type initialized earlier, get topic Number, and get Message. Node also has sendACK and send BRK functions which are explained in their respective topic.

3.3 Acknowledgements (ACK)

Node class has a function send ACK which is called whenever a message is sent, to send Acknowledgement to the address message was received from to confirm that message has been received. Each subclass waits for an acknowledgment after sending a packet. The function sendACK is shown in Listing 3.

```
protected void sendAck(DatagramPacket receivedPacket) {
    byte[] data = receivedPacket.getData();
    setType(data, ACK);

    DatagramPacket ack = new DatagramPacket(data, data.length,
        receivedPacket.getSocketAddress());
    try {
        socket.send(ack);
        System.out.println("Sent ACK.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 3: Acknowledgment, this function sets the type of the function to ACK and sends the packet to the socket address the packet was received from.

3.4 Breaker

Similar to sendACK the Node class has a function sendBRK which is used when a subscriber unsubscribes, a Break Type Packet is generated and sent to the socket address based on the data, which is then handled by the Subscriber. Listing 4 shows the send BRK function.

```
protected void sendBRK(byte[] data1) {
    String data = getMessage(data1);
    byte[] data2=data.getBytes();
    setType(data2, BRK);
    InetAddress inet=new
InetSocketAddress(SUB_DST,BKR_PORT+Integer.parseInt(data));
    DatagramPacket ack = new DatagramPacket(data2,
data2.length,inet );
    try {
        socket.send(ack);
        System.out.println("Sent BKR.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 4: This function sets the type of data to BRK , and send the data to the specified address based on the data.

Broker

The Broker class uses and listens on the port 50001 which is declared in Node class and is responsible for managing and transmitting the data between Publishers and Subscribers and manages the data of rooms created and subscribers in two HashTables. SubscribersList which stores the key room name and Inetsocketaddress of the subscriber. Only one subscriber can subscribe to one topic which can be changed by just pushing an ArrayList with the socket address of all the subscribers with room numbers to the HashTable. Due to the time constraints, it was not implemented in this solution. The other Hashtable is 'Room' which stores the data of the room numbers active with the Room name, by default I have created 3 rooms, but there is no maximum limit for creating rooms. When the broker starts it waits for a packet. upon receiving the packet the onReciept function is invoked which notifies the system and checks for the type of packet received, if an ACK Broker prints the ACK if it is a request to create a room, subscribe, unsubscribe or publish, an acknowledgment is sent to the sender and the relevant function is invoked upon the completion a message is sent to the sender stating if the task was successful or unsuccessful. The code in Listing 5 shows the functions invoked when the type is Subscribe or unsubscribe. Listing 6 illustrates the code for creating a new room or publishing the data to rooms.


```

private boolean subscribe(byte[] data, SocketAddress subscriberAddress) {
    String RoomName = getMessage(data);
    if (subscriberList.containsKey(RoomName)) {
        InetSocketAddress subscribers = subscriberList.get(RoomName);

        subscriberList.put(RoomName, (InetSocketAddress) subscriberAddress);

        System.out.println("A new subscriber subscribed to " + RoomName + ".");
        return true;
    }
    return false;
}

private boolean unsubscribe(byte[] data, SocketAddress subscriberAddress) {
    boolean unsubscribed = false;
    String RoomName = getMessage(data);
    if (subscriberList.containsKey(RoomName)) {
        InetSocketAddress subscribers = subscriberList.get(RoomName);
        if (! (subscribers==null)/*isEmpty()*/) {

            if (subscribers.equals(socket1)){

                System.out.println ("You are not Subscriberd to the room "+RoomName);

                unsubscribed=false;

            }

            else{

                sendBRK(data);
                subscriberList.put(RoomName, socket1);

                System.out.println("A subscriber unsubscribed from " + RoomName + ".");

                unsubscribed = true;
            }
        }
    }
    return unsubscribed;
}

```

Listing 5 :If the type of the received packet is SUB the onReciept function calls subscribe function with parameters as the data and the socketAddress.If the data is in the SubscriberList has room name the socket address is added to the subscribers list and Boolean value true is returned else a Boolean value of false is returned. If the type of the packet is usub the unsubscribe function is called with same parameters and check if the data that was passed is present in the Table or not. It again checks if the socket address associated with the Roomname is the default socket if yes, the requester is not subscribed. hence, false is returned else the socket number of the hashtable is changed to the default socket address.

```

private boolean createRoom(byte[] data) {
    InetAddress socketNumbers= new InetAddress(SUB_DST,BKR_PORT);

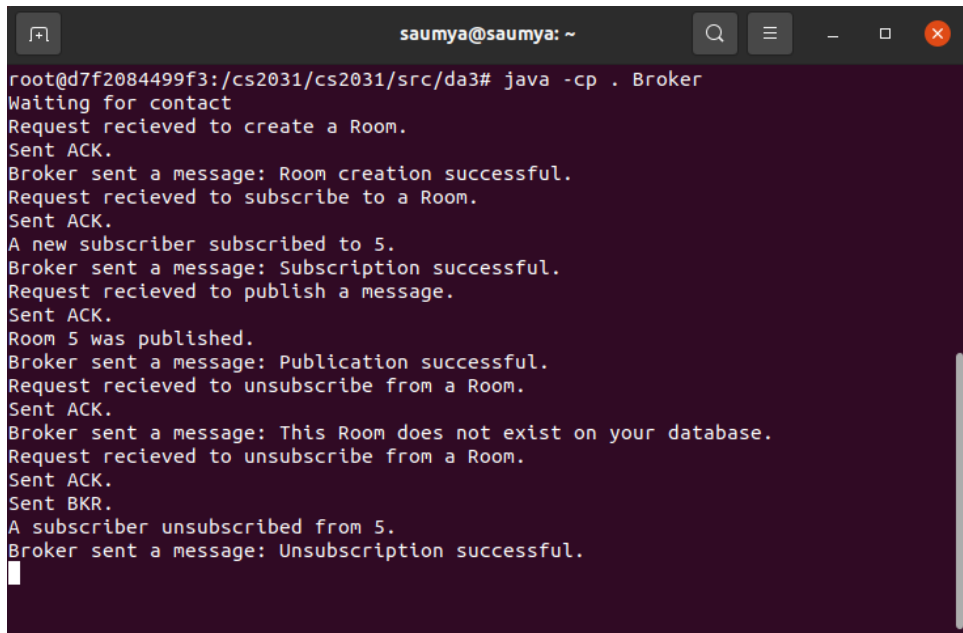
    String RoomName = getMessage(data);
    if (!subscriberList.containsKey(RoomName)) {
        subscriberList.put(RoomName, socket1);
        int RoomNumber = getRoomNumber(data);
        Room.put(RoomNumber, RoomName);
        return true;
    }
    return false;
}

private boolean publish(byte[] data) throws SocketException {
    int RoomNumber = getRoomNumber(data);
    setType(data, Pub);
    if (Room.containsKey(RoomNumber)) {
        String RoomName = Room.get(RoomNumber);
        InetAddress dstAddresses = subscriberList.get(RoomName);
        if((dstAddresses.equals(new
InetAddress/*>*/(SUB_DST,BKR_PORT)))){
            System.out.println("No one has Subscribed to this Room");
            return false;
        }
        else if (!(dstAddresses==null)) {
            try {DatagramPacket publication = new DatagramPacket(data, data.length,
dstAddresses);
                socket.send(publication);
                System.out.println("Room " + RoomName + " was published.");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return true;
    }
    return false;
}

```

Listing 6: If the type of packet is New the createroom function is invoked with the data as parameter then an if condition checks if the room is already present in the hashtable, if not, a new room is created with the default socket as the socket address. If the type of the packet is Pub the publication function called with the data(Room number) which then checks if the room is present on the hashtable or not if yes it again checks if the socket address of the room is default InetAddress if not it sends the data to the address associated with it.

While performing all the operations the broker will print the required Instructions on the docker terminal as illustrated by Figure 2.



```
saumya@saumya: ~
root@d7f2084499f3:/cs2031/cs2031/src/da3# java -cp . Broker
Waiting for contact
Request recieved to create a Room.
Sent ACK.
Broker sent a message: Room creation successful.
Request recieved to subscribe to a Room.
Sent ACK.
A new subscriber subscribed to 5.
Broker sent a message: Subscription successful.
Request recieved to publish a message.
Sent ACK.
Room 5 was published.
Broker sent a message: Publication successful.
Request recieved to unsubscribe from a Room.
Sent ACK.
Broker sent a message: This Room does not exist on your database.
Request recieved to unsubscribe from a Room.
Sent ACK.
Sent BKR.
A subscriber unsubscribed from 5.
Broker sent a message: Unsubscription successful.
```

Figure 2. The snapshot of broker. Illustration shows the instructions the broker prints on receiving different requests.

3.5 Publisher

The publisher listens on the port 50000 which is declared in the node class and a destination address to the broker as it only needs to interact with the broker. The Publisher has a HashTable to create and store, room numbers, which enables to check-before hand if the room has been created or not. The publisher can create and publish as many topics as required. to make the process fast as the task of the assignment was to create and implement a protocol I didn't focus much on the data part that's why I have used room and temperatures as the data for the transmission to make sense. When the publication function is invoked, the function calls a function data to get the temperature reading which then invokes another function Random temp which uses java.util.Random package to generate a String of two digits which is then converted to byte and sent to the broker. When the socket at which publisher is listening receives a packet the onReciept function is invoked which checks if the type of the package received is ACK or Mes and prints them accordingly. After sending a packet the publisher waits for the broker to send an ACK and a message with status of the request, If it receives a message the publisher sends back an ACK. Listing 7 shows the code for creating a new topic and publishing. Listing 8 shows the Random temp function.

```

        private void createRoom() throws SocketException {
            System.out.println("Please enter a room to create: ");
            String Room = sc.next();
            System.out.println("Sending packet...");
            DatagramPacket[] packets = createPackets(NEW, RoomNumbers.size(), Room,
dstAddress);
            RoomNumbers.put(RoomNumbers.size()+1, Room);
            try {
                socket.send(packets[0]);
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.out.println("Packet sent");
        }
        private boolean publishMessage() throws SocketException {

System.out.println("Please enter the name of the room you want to publish the details
for: ");
            String Room = sc.next();
            String message=data();

            int RoomNumber = Integer.MAX_VALUE;
            for (int i = 0; i < RoomNumbers.size(); i++) {
                if ((RoomNumbers.get(i)).equals(Room)) {
                    RoomNumber = i;
                }
            }
            if (RoomNumber == Integer.MAX_VALUE) {
                System.out.println("This Room does not exist.");
            } else {
                DatagramPacket[] packets = createPackets(Pub, RoomNumber, message, dstAddress);
                try {
                    System.out.println("Sending packet...");
                    socket.send(packets[0]);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                System.out.println("Packet sent");
                return true;
            }
            return false;
        }
    }

```

Listing 7 :- The Create and Publish function of Publisher which is

Invoked after the user enters the directions with keyboard in the Start function which then either creates a room and pushes it to the hashtable room and send it to broker, or, it publishes the data sending the room number and the room temperature in the header file

```
public static String RandomTemp(){
    String arr[]= new String [10];
    int i=0;
    while(i<10) {
        Random rand = new Random();

        Integer random=rand.nextInt(90)+10;
        arr[i]=random.toString();

        i=i+1;
    }
    Random r=new Random();
    int randomNumber=r.nextInt(arr.length);

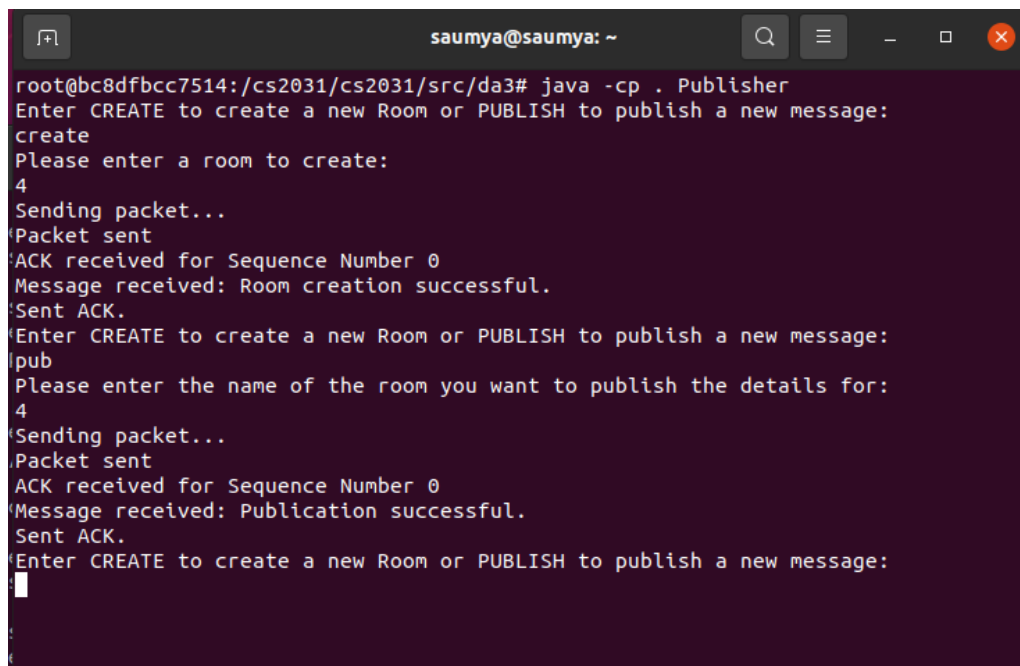
    return arr[randomNumber];

}

public static String data()
{
    return "The Temp is "+RandomTemp()+" degree ";
}
```

Listing 8:- Using the Random class of java.util.Random this function returns a String to send to the broker.

While the publisher is performing all these functions it also prints the status on the Docker terminal as shown in Figure 3.

A screenshot of a Docker terminal window titled 'saumya@saumya: ~'. The terminal shows the execution of a Java program. The user enters 'create' to create a new room. The program sends a packet, receives an ACK, and prints 'Message received: Room creation successful.' The user then enters 'pub' to publish a message. The program sends a packet, receives an ACK, and prints 'Message received: Publication successful.' The terminal text is as follows:

```
root@bc8dfbcc7514:/cs2031/cs2031/src/da3# java -cp . Publisher
Enter CREATE to create a new Room or PUBLISH to publish a new message:
create
Please enter a room to create:
4
Sending packet...
Packet sent
ACK received for Sequence Number 0
Message received: Room creation successful.
Sent ACK.
Enter CREATE to create a new Room or PUBLISH to publish a new message:
pub
Please enter the name of the room you want to publish the details for:
4
Sending packet...
Packet sent
ACK received for Sequence Number 0
Message received: Publication successful.
Sent ACK.
Enter CREATE to create a new Room or PUBLISH to publish a new message:
```

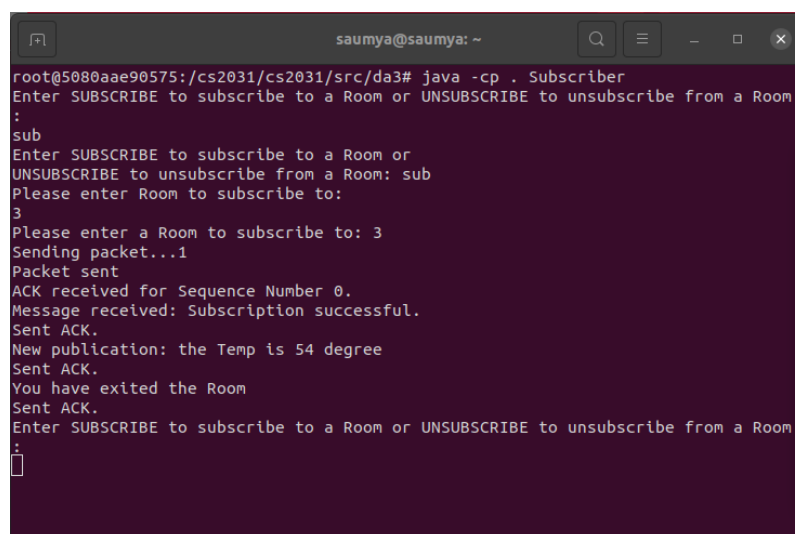
Figure 3:- shows the instructions publisher prints on the docker terminal .

3.6 Subscriber

The subscriber uses the destination address as the broker's address as subscribers similar to publishers have to communicate only with the broker. This class has a HashTable: subscriberList, which has room number and the port number as an integer as its keys. The port number is determined by the room number entered. The subscriber class has two main functions Subscribe and Unsubscribe which is entered on the terminal by the user, but as I have previously stated, my solution did not implement multiple subscriptions to a single topic or multiple topics to a single subscriber as I realized that late and it was too late to implement threads or switch to multicast sockets. as my solution, either way, implemented the basic requirements for a publish-subscribe protocol, I didn't implement multicast sockets or threads. If the user enters 'subscribe' the system asks for the room the user wants to subscribe to the subscribe function is called the system asks for the room number to subscribe which is then added with the broker port to get a unique port for the topic. Once the subscriber has subscribed to a topic it will continuously listen to the port and wait for a message with a condition notified which is set to true. As the port numbers depend on the room number subscribed to, a subscriber can unsubscribe from a topic using another instance of subscriber class on another container. the onReciept function in subscriber checks for the type of the packet, if an ACK it prints the Acknowledgement if it is a publication or message it prints the data and sends the acknowledgment, if the type of the packet is BRK which was initialized in the node, the function prints a message sets the notified to false and stops waiting and then creates a new object which calls the function start again.

The code for subscribing and unsubscribing can be seen in Listing 9.

Figure 4 and figure 5 show the terminal screen for subscriber container.



```
saumya@saumya: ~  
root@5080aae90575:/cs2031/cs2031/src/da3# java -cp . Subscriber  
Enter SUBSCRIBE to subscribe to a Room or UNSUBSCRIBE to unsubscribe from a Room  
:  
sub  
Enter SUBSCRIBE to subscribe to a Room or  
UNSUBSCRIBE to unsubscribe from a Room: sub  
Please enter Room to subscribe to:  
3  
Please enter a Room to subscribe to: 3  
Sending packet...1  
Packet sent  
ACK received for Sequence Number 0.  
Message received: Subscription successful.  
Sent ACK.  
New publication: the Temp is 54 degree  
Sent ACK.  
You have exited the Room  
Sent ACK.  
Enter SUBSCRIBE to subscribe to a Room or UNSUBSCRIBE to unsubscribe from a Room  
:  
[ ]
```

Figure 4: shows the screen of subscriber the instruction it prints

```

saumya@saumya: ~
^Croot@20dc4463d0c5:/cs2031/cs2031/src/da3# java -cp . Subscriber
Enter SUBSCRIBE to subscribe to a Room or UNSUBSCRIBE to unsubscribe from a Room
:
unsub
Enter SUBSCRIBE to subscribe to a Room or
UNSUBSCRIBE to unsubscribe from a Room: unsub
Please enter a Room to unsubscribe from:
3
Sending packet...
Packet sent
ACK received for Sequence Number 0.
You have exited the Room
Sent ACK.
Enter SUBSCRIBE to subscribe to a Room or UNSUBSCRIBE to unsubscribe from a Room
:

```

Figure 5: Is another instance of subscriber class running on different container which was used to unsubscribe a subscriber

```

public synchronized void subscribe() throws SocketException {
    System.out.println("Please enter Room to subscribe to: ");
    String data = sc.next();
    port(data);
    subscriberList.put(data, BKR_PORT+Integer.parseInt(data));
    SUB_POR=subscriberList.get(data);
    if(SUB_POR!=0) {
        socket = new DatagramSocket(SUB_POR);
        listener.go();
    }
    Set(data);
    int seq= getseq(data);
    System.out.println("Please enter a Room to subscribe to: " + data);
    System.out.println("Sending packet..." + seq);
    DatagramPacket packet = createPackets(SUB, seq, data, dstAddress)[0];
    try {
        socket.send(packet);
    } catch (IOException e) {}
    System.out.println("Packet sent");}

public synchronized void unsubscribe() throws SocketException {
    System.out.println("Please enter a Room to unsubscribe from: ");
    String data = sc.next();
    subscriberList.put(data, BKR_PORT+Integer.parseInt(data));
    SUB_POR=subscriberList.get(data);
    if(SUB_POR!=0) {
        socket = new DatagramSocket(SUB_POR);

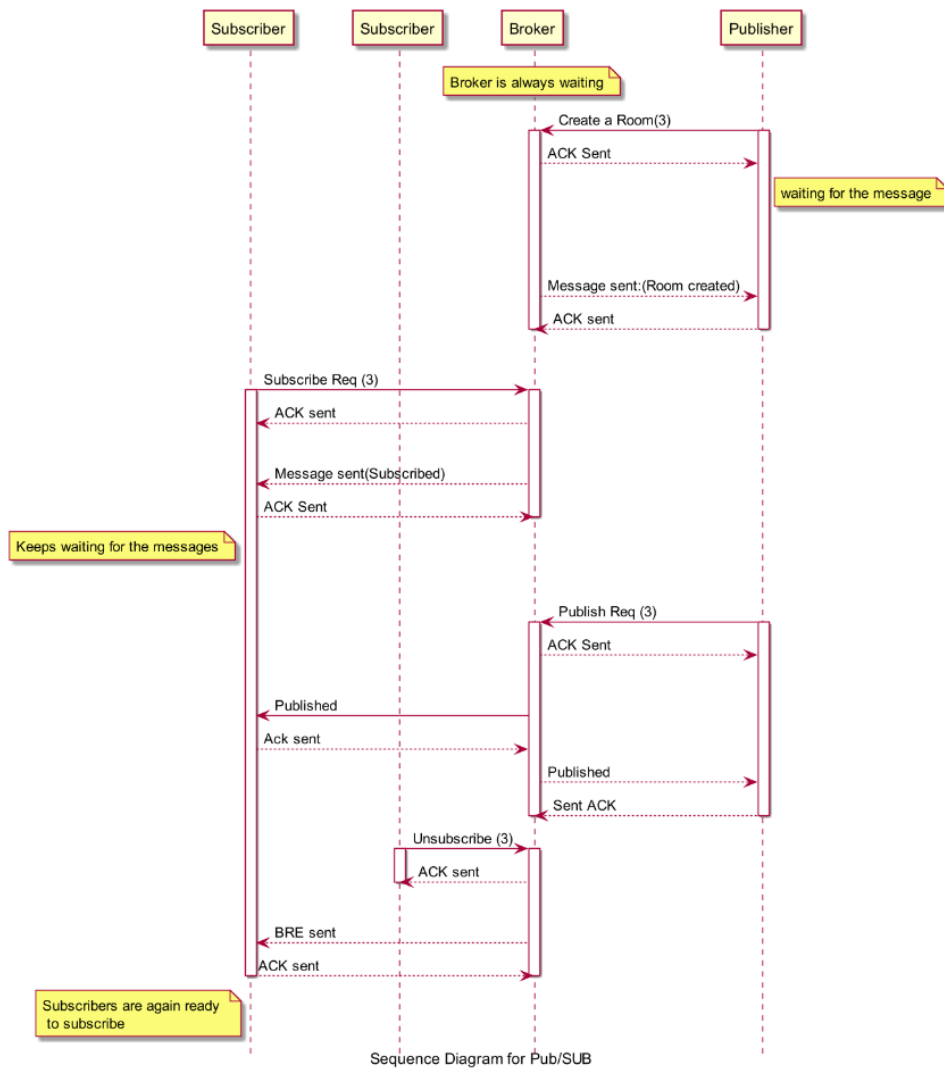
        listener.go();
    }
    System.out.println("Sending packet..." );
    DatagramPacket packet = createPackets(USUB, 0, data, dstAddress)[0];
    try {
        socket.send(packet);
    } catch (IOException e) {
    }
    System.out.println("Packet sent");}

```

Listing 9: Illustrates the Subscribe and unsubscribe function used in subscriber class upon being called both the classes pass the data to set it a public variable then the SubscriberList class is inserted with the data and a socket number associated to it. Which is then used for further communication with this room number.

3.7 Protocol

All the classes and functions implemented to make a pub/sub protocol that sends randomly generated room temperatures to subscribers subscribed to the room. Which can unsubscribe from the rooms. Figure 6 shows the sequence diagram between the three classes.



Saumya Bahuguna

Figure 6: Shows the communication between three classes and how the containers get activated when the socket receives a packet and how ack is sent each time a packet is received.

4 Discussion

This section will be a discussion of my protocol and how it uses the material taught to us in the lectures.

4.1 Port Numbers and IP Addresses

My protocol is a simple version of a publish subscribe protocol. Though all the classes

listen on different sockets the publisher has only one address even for multiple instances

unlike subscribe which assigns a port number according to the room number is entered.

Networks, as illustrated in Figure 1, 2 different networks are used and have no direct way of communication with each other, other than Broker class which stores and forwards data. It also uses UDP which is a protocol in transport layer and shows the basic functions of Transport layer, it also modifies the UDP header to store values as per the classes' need.

5 Summary

This report has described the basic solution for the assignment 1, which was my implementation of pub/sub protocol, which consists of an abstract Node class which is extended by a Broker a Subscriber and a Publisher class. The features of the overall system can be summarized as the following:

Abstract Node Class: Stores all the common variables and functions uses through the protocol by different classes i.e. SendACK(), getType() etc, listen on a socket and notifies the extending class when one packet is received.

Broker: Getting packet types, stores the room as they are created, This Class handles almost each request from other classes by using functions defined in Node class.

Subscriber: This function accepts an input from a user to subscribe and unsubscribe to a room and assign a socket to the user based on the room they have entered and passes the request to broker, printing the message received.

Publisher: Handles user input, creates a room and stores the value of room create and forwards the requests as received to broker and prints the messages received.

6 Reflection:-

Upon reflection, I found assignment 1 to be quite informative and interesting. I have divided them into advantages and disadvantages as follows:

Advantages: -

I learned a lot about the network and transport layer of the OSI module and its functionality.

I learned a lot about other features that were implemented in my Publish/Subscribe protocol, such as how to design a packet layout and how to implement simple acknowledgments.

The research for java implementation and for making the report helped me to reinforce my knowledge of the subject.

Disadvantages: -

I didn't initially research enough before making the model I wanted to work on which led to wastage of time in finding new ways of implementing things and trying to implement them and then at the end not having enough time to implement processes such as segmentation and using threads to run different functions consequently which would have solved the problem of multiple subscribers and unsubscribing from same terminal, or using multicast sockets even after having enough knowledge to implement them.

Overall, making the protocol and the report made me learn a lot about computer networks only not planning the implementation led to some setbacks, which I will make sure is not continued in Assignment 2.

References

Computer Networks Fifth Edition Andre S. Tanenbaum, David J. Wetherall