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

Aislinn Addison-Smyth

Computer Networks Assignment #2 – Flow Forwarding

December 3rd, 2021

Table of Contents

1. Introduction

The problem description outlined for this assignment is focused on finding a replacement for IPv4 or IPv6 by designing a protocol that forwards payloads based on a collection of strings. These strings identify the source and destination of traffic. The protocol I decided to implement was the Stop and Wait ARQ, with this protocol, it allows one Datagram packet to be sent at any one time, once a datagram packet is sent, the sender will wait for an acknowledgment, if an acknowledgement is not received after a certain period, the packet is therefore sent again. The network consists of an application, a forwarding service, a controller, multiple routers, and an end user. All network elements are connected to a network, the forwarding service is connected to all networks and so can communicate freely with all network elements. If the forwarding service is not used, the network elements would be limited to which elements they can send onto. Throughout this report, I will discuss the functionality behind my assignment and give reason to some of the implementation decisions I made along the way.

2. Overall Design

For the overall design of my report, I will be discussing in detail the idea behind my understanding and implementation of the assignment. Initially, I will begin by displaying all my network components and how they are linked to each other, I will move onto explaining my protocol of choice and finally my packet design.

2.1 Component Connection

I created the diagram in figure 2.1 to display my network elements in a way that would make sense. The assignment title is ‘Flow Forwarding’; therefore, the main purpose of this diagram is to show the forwarding mechanism and the links I have created between each network element. I created 5 subnets all different networks and connected each component to a network. Although the diagram does not display, the forwarding service is connected to every network, net1 through to net5. Therefore, when a packet is sent to the forwarding service, that packet can be sent onto any container. I referred to the diagram on the initial page of our assignment description for inspiration in what containers to make, I concluded with an application, a forwarding service which has access to all network elements, eight routers and finally an end user. The arrows show the connections between the network elements and the networks.

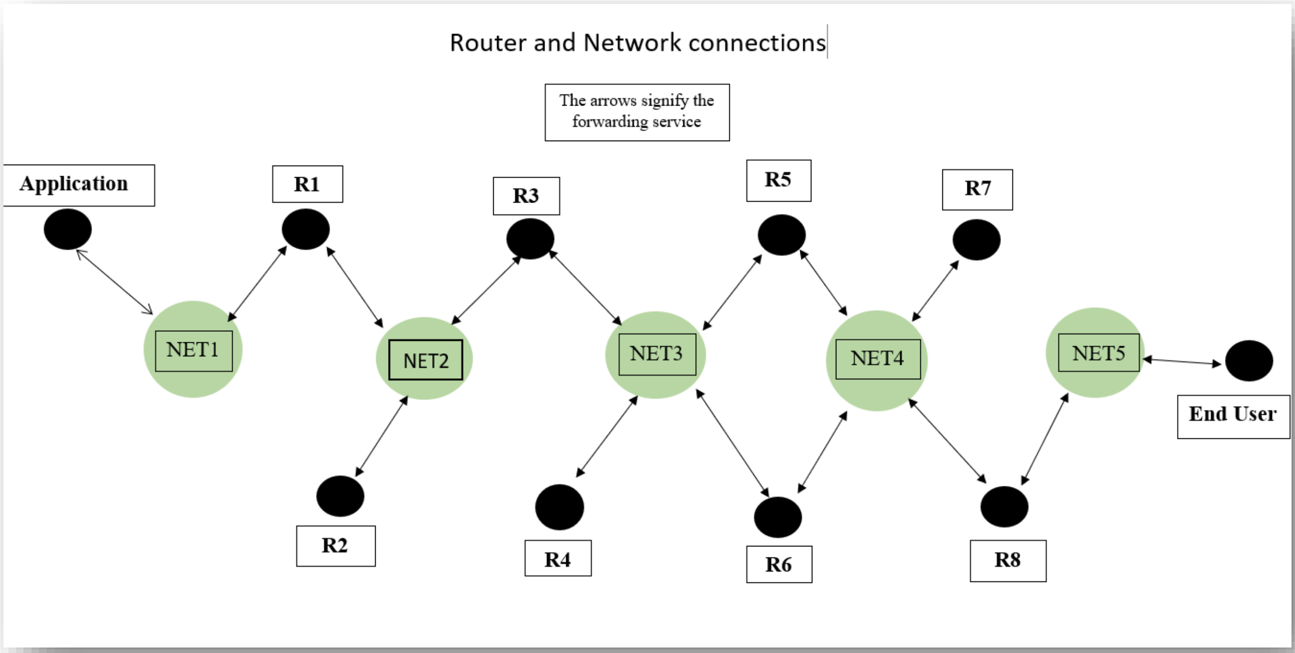

Router and Network connections
The arrows signify the forwarding service
Application
R1
R3
R5
R7
NET1
NET2
NET3
NET4
NET5
End User
R2
R4
R6
R8

Figure 2.1: Network element connections

2.3 Protocol Design

The protocol I designed was the Stop-and-Wait ARQ, I sourced a diagram of this from a website linked below in figure 2.2. The Stop-and-Wait protocol only allows a single frame to be outstandingly sent from the sender, i.e., the sender cannot send multiple packets at any one time. This protocol works by the sender sending a packet to the receiver, it therefore waits for an acknowledgement that the receiver has received the packet, once the acknowledgement is received, another UDP datagram packet is able to be sent onto the receiver, and again the sender waits for the acknowledgement response.

How I have implemented this into my code is that when my code is initialised, depending on what container it is, it will say “Router is waiting for a packet…”. Once the sender sends the packet onto, in this case, the router, the router will print an acknowledgement message, “Router has received a packet”. This is the case for all network elements, when the code is initialised, all network elements will display a waiting message, from there once they receive a packet, an acknowledgment message is printed onto the screen.

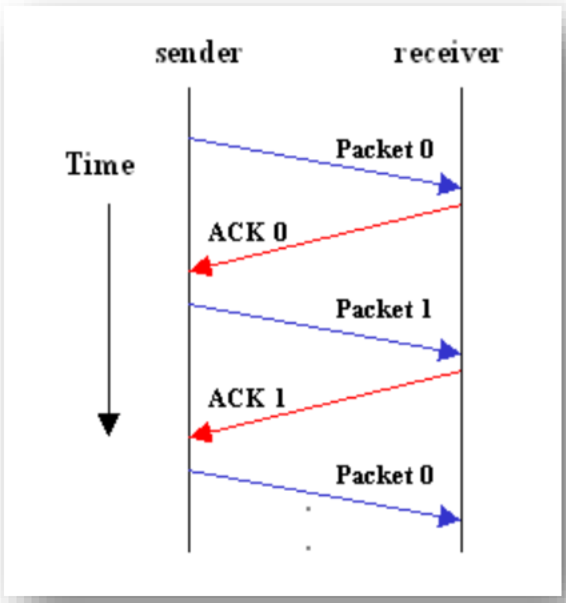
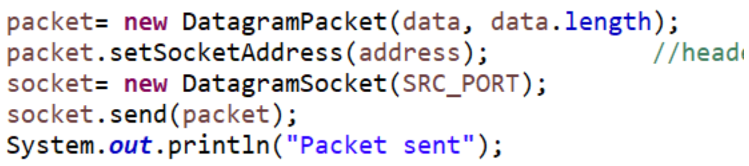


Figure 2.2: Stop and Wait ARQ diagram, taken from https://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_HYUNAH/D-Research/stop-n-wait.html

2.4 Packet Design

Figure 2.4 is a diagram taken from the assignment description; this was the anticipated design of our UDP packets. We created our UDP datagram packets in the application class using the code in figure 2.4. We also had to create a header and attach it to the UDP packet when being sent. The header was to be a encoded as a type-length-value (TLV) format. A header of a UDP packet encodes information about the destination of the UDP packet, in this case it could be 'R4', or it could be 'berkeley', which would send us to R8. I created my header based on user input. When prompted for the destination of the packet, I attach my header onto the UDP packet, this entails the destination of the packet and the header length.



```
packet= new DatagramPacket(data, data.length);
packet.setSocketAddress(address);          //head
socket= new DatagramSocket(SRC_PORT);
socket.send(packet);
System.out.println("Packet sent");
```

Figure 2.4: UDP datagram packet creation.

Once the packet along with the header is sent to the forwarding service, the forwarding service inspects the header information using:

The packet.getData(), extracts the data from the header of the packet, packet.getOffset is the offset of the data that is to be sent or received, packet.getLength() gets the length of the packet header. Finally, standardCharsets.UTF_8 specifies the format we want our data to be extracted into. As my data is simply a string, the standard charset is suitable for this data set.

```
newData = new String(packet.getData(), packet.getOffset(), packet.getLength(), StandardCharsets.UTF_8);
```

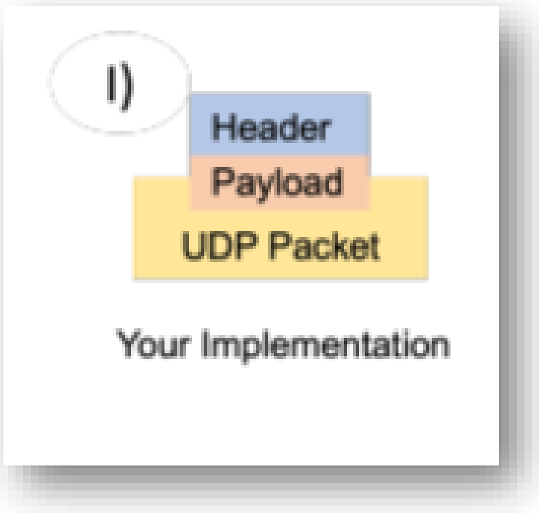


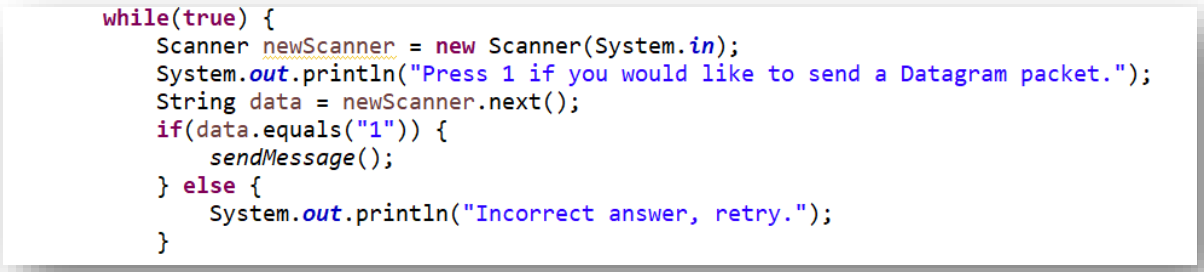
Figure 2.3: Taken from the assignment description: Packet design

3 Implementation

In this section of my report, I will discuss in detail the different components of my implementation, the application, forwarding service, router, controller, and end user. I will also be discussing and demonstrating how my code runs on an external source, I have chosen to run my code using docker. As well as this I will make use of Wireshark to capture traffic at different network elements.

3.3 Application

The application is a key component of my network elements, all datagram packets are sent from this class. I chose to have my application take user input, my reason for this is that I find it easier to follow code when I control when a datagram packet is sent, where that packet is going to and what input goes



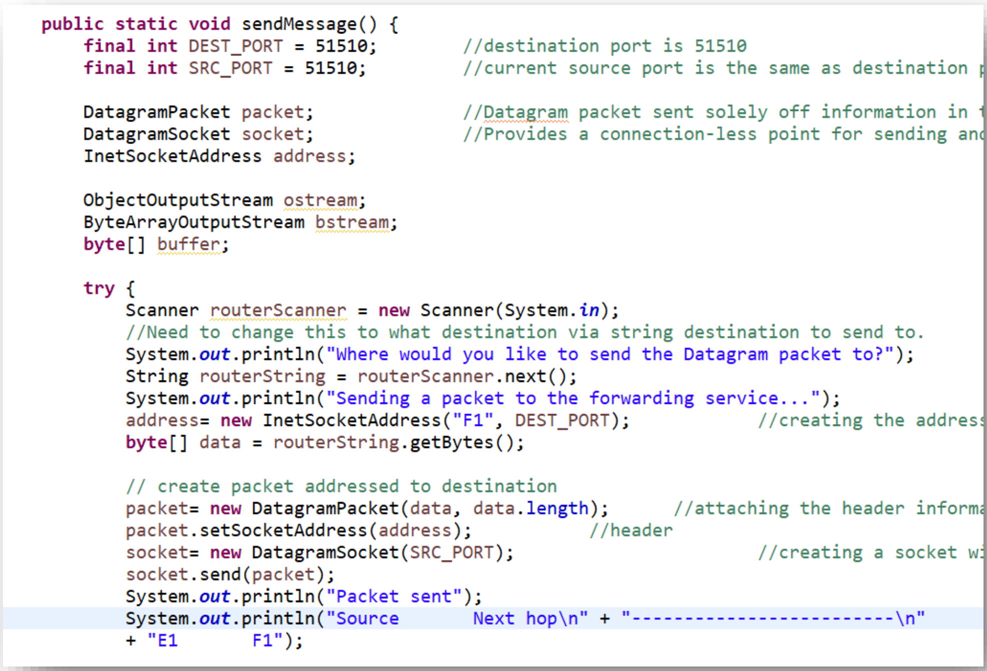
```
while(true) {
    Scanner newScanner = new Scanner(System.in);
    System.out.println("Press 1 if you would like to send a Datagram packet.");
    String data = newScanner.next();
    if(data.equals("1")) {
        sendMessage();
    } else {
        System.out.println("Incorrect answer, retry.");
    }
```

into the code.

Figure 3.1: Main method in Application class

As seen in figure 3.1, my code prompts for user input of a 1 to send a Datagram packet, if any other value is given an error message will appear and the code will restart. If 1 is pressed, the code goes straight to the sendMessage() function under the main class, which we can see I figure 3.2.

My implementation of this assignment has all my network elements bound to the same port, in this case, port 51510. Once a datagram packet is sent, the application will prompt you for an acquired destination of that packet, an input of the container names for example ‘R1’ or ‘E2’ are accepted, but a string is also accepted. There are five strings which my code will accept: ‘trinity’, ‘centrallab’, ‘LG35’, ‘hamilton’, and ‘berkeley’. Once a destination has been specified that packet along with the destination header will be sent onto the forwarding service.

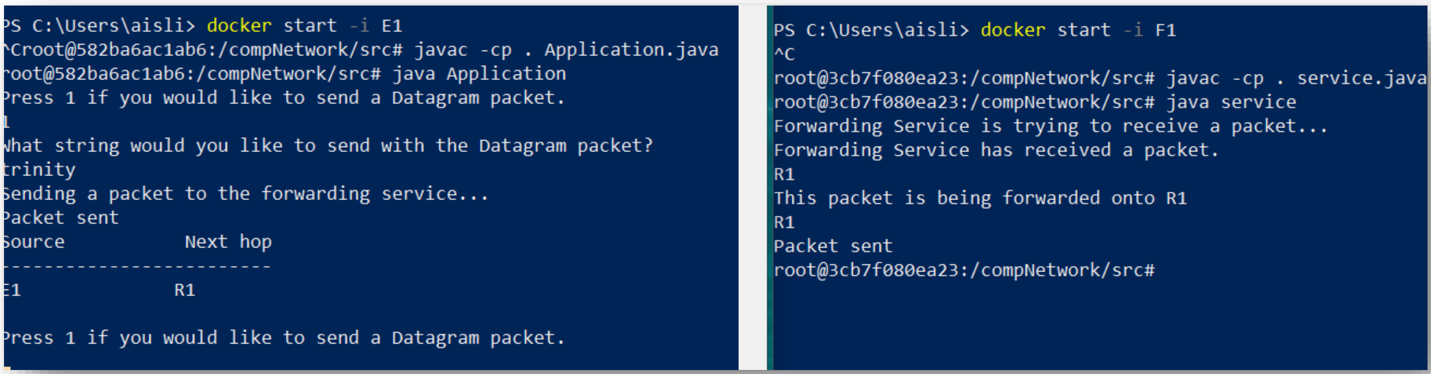


```
public static void sendMessage() {
    final int DEST_PORT = 51510;        //destination port is 51510
    final int SRC_PORT = 51510;         //current source port is the same as destination p

    DatagramPacket packet;              //Datagram packet sent solely off information in t
    DatagramSocket socket;              //Provides a connection-less point for sending and
    InetSocketAddress address;

    ObjectOutputStream ostream;
    ByteArrayOutputStream bstream;
    byte[] buffer;

    try {
        Scanner routerScanner = new Scanner(System.in);
        //Need to change this to what destination via string destination to send to.
        System.out.println("Where would you like to send the Datagram packet to?");
        String routerString = routerScanner.next();
        System.out.println("Sending a packet to the forwarding service...");
        address= new InetSocketAddress("F1", DEST_PORT);            //creating the address
        byte[] data = routerString.getBytes();

        // create packet addressed to destination
        packet= new DatagramPacket(data, data.length);      //attaching the header informa
        packet.setSocketAddress(address);           //header
        socket= new DatagramSocket(SRC_PORT);                       //creating a socket wi
        socket.send(packet);
        System.out.println("Packet sent");
        System.out.println("Source       Next hop\n" + "------------------------\n"
        + "E1       F1");
```

Figure 3.2: sendMessage() function which is used to send datagrams to the forwarding service.

3.4 Forwarding service

The forwarding service is another essential network component, it is the key to forwarding packets. The forwarding service is used for any UDP datagram packets being sent from the application; it acts as the middleman. It receives a packet from the application, it then investigates the header contents and prints the destination the header of the packet contains. In figure 3.3, I have attached the communication between the application and the forwarding service. We can see from the left photo, I pressed 1 and the datagram packet is ready to be sent, I am then prompted for a destination, for the purpose of the demonstration, I used a string input of trinity.



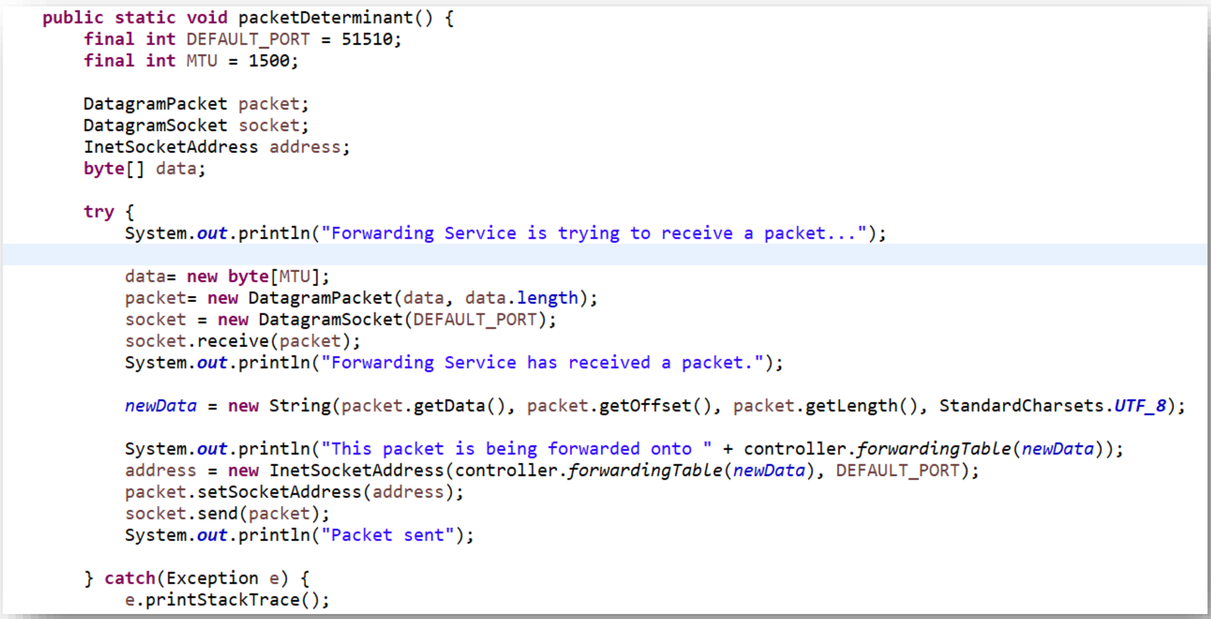
```
PS C:\Users\aisli> docker start -i E1
^Croot@582ba6ac1ab6:/compNetwork/src# javac -cp . Application.java
root@582ba6ac1ab6:/compNetwork/src# java Application
Press 1 if you would like to send a Datagram packet.
1
What string would you like to send with the Datagram packet?
trinity
Sending a packet to the forwarding service...
Packet sent
Source           Next hop
------------------------
E1              R1

Press 1 if you would like to send a Datagram packet.
```

```
PS C:\Users\aisli> docker start -i F1
^C
root@3cb7f080ea23:/compNetwork/src# javac -cp . service.java
root@3cb7f080ea23:/compNetwork/src# java service
Forwarding Service is trying to receive a packet...
Forwarding Service has received a packet.
R1
This packet is being forwarded onto R1
R1
Packet sent
root@3cb7f080ea23:/compNetwork/src#
```

Figure 3.3: Docker communication between Application and Forwarding Service

Trinity is then attached as a header to the datagram packet and sent onto the destination which the controller selects trinity to be connected to. In this case, the controller associated trinity with router 1 and then forwarded the packet onto R1.

Figure 3.2 demonstrates the code behind the forwarding service. If we look at the newData = new String…, this is reading the header information from the datagram packet, we then take newData and pass it into a function in the controller. The controller therefore determines the correct destination for the packet to be sent onto.

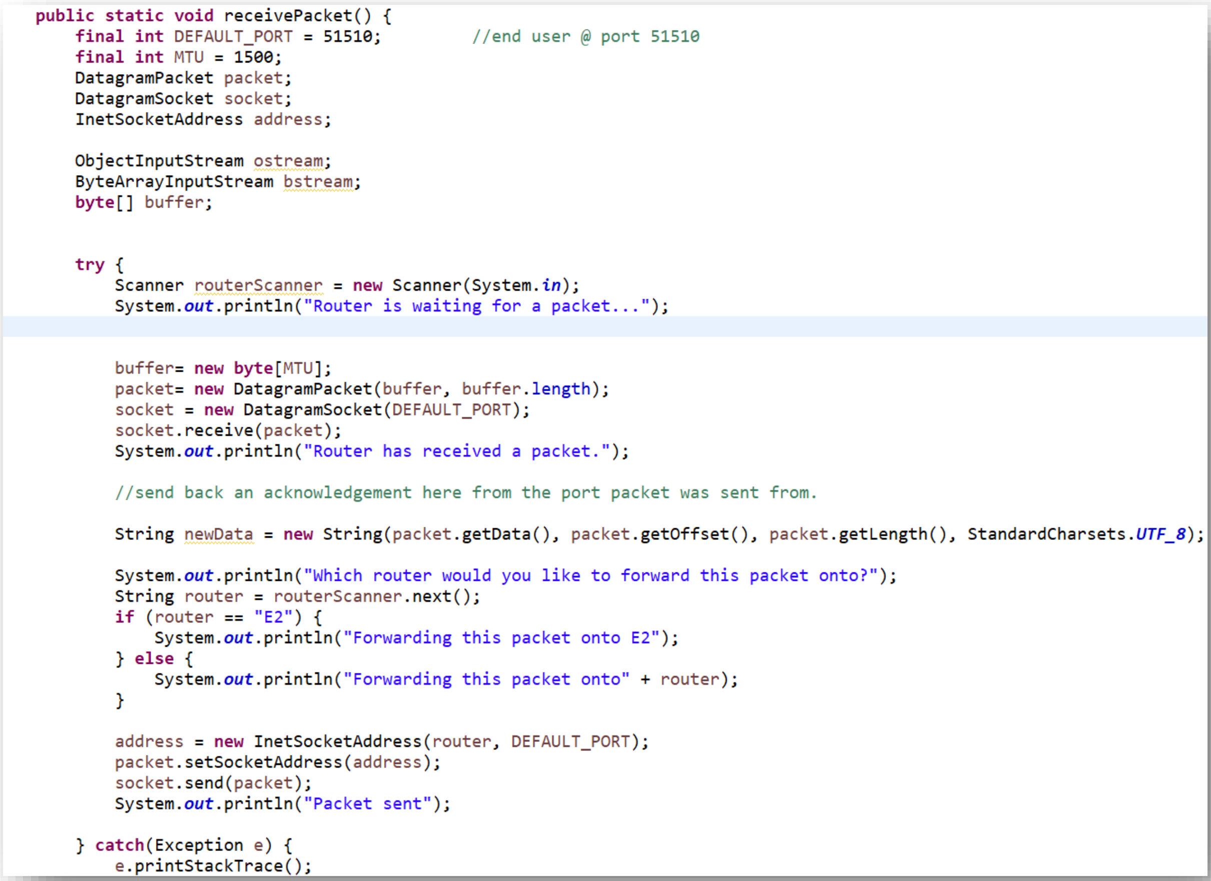


```java
public static void packetDeterminant() {
    final int DEFAULT_PORT = 51510;
    final int MTU = 1500;

    DatagramPacket packet;
    DatagramSocket socket;
    InetSocketAddress address;
    byte[] data;

    try {
        System.out.println("Forwarding Service is trying to receive a packet...");

        data= new byte[MTU];
        packet= new DatagramPacket(data, data.length);
        socket = new DatagramSocket(DEFAULT_PORT);
        socket.receive(packet);
        System.out.println("Forwarding Service has received a packet.");

        newData = new String(packet.getData(), packet.getOffset(), packet.getLength(), StandardCharsets.UTF_8);

        System.out.println("This packet is being forwarded onto " + controller.forwardingTable(newData));
        address = new InetSocketAddress(controller.forwardingTable(newData), DEFAULT_PORT);
        packet.setSocketAddress(address);
        socket.send(packet);
        System.out.println("Packet sent");

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

Figure 3.2: packetDeterminant() function in the service class.

3.5 Router

The router class is a class which account for all eight routers, it acts as a constructor for whatever router is receiving or sending a packet. I have attached the receivePacket() function from the router class in figure 3.4, this class receives packets and takes user input to send the packets onwards. Initially, the route class will wait and have “Router is waiting for a packet…” displayed, the router will then print “Router has received a packet” once the packet is received by the router class. Once a router has a packet, it will prompt you to input what network element to send the packet onto from there. We also have an end user clause which if a user chooses to send a packet to the end user, it will finish at the end user class and not prompt for the packet to be sent anywhere else. If end user is not prompted, the packet will be sent onto whatever input is given.

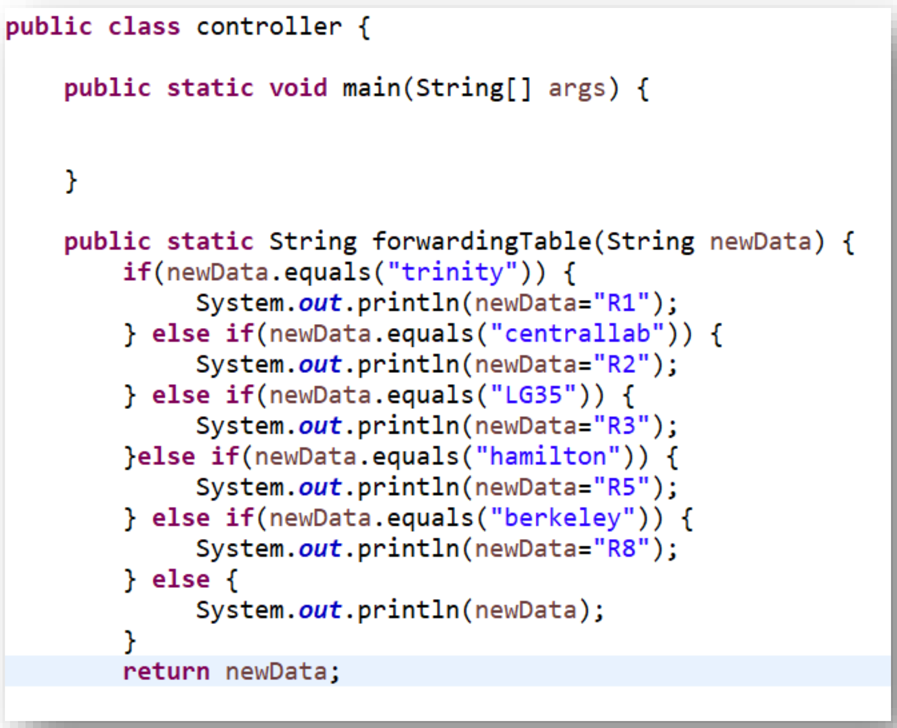


```
public static void receivePacket() {
    final int DEFAULT_PORT = 51510;         //end user @ port 51510
    final int MTU = 1500;
    DatagramPacket packet;
    DatagramSocket socket;
    InetSocketAddress address;

    ObjectInputStream ostream;
    ByteArrayInputStream bstream;
    byte[] buffer;


    try {
        Scanner routerScanner = new Scanner(System.in);
        System.out.println("Router is waiting for a packet...");


        buffer= new byte[MTU];
        packet= new DatagramPacket(buffer, buffer.length);
        socket = new DatagramSocket(DEFAULT_PORT);
        socket.receive(packet);
        System.out.println("Router has received a packet.");

        //send back an acknowledgement here from the port packet was sent from.

        String newData = new String(packet.getData(), packet.getOffset(), packet.getLength(), StandardCharsets.UTF_8);

        System.out.println("Which router would you like to forward this packet onto?");
        String router = routerScanner.next();
        if (router == "E2") {
            System.out.println("Forwarding this packet onto E2");
        } else {
            System.out.println("Forwarding this packet onto" + router);
        }

        address = new InetSocketAddress(router, DEFAULT_PORT);
        packet.setSocketAddress(address);
        socket.send(packet);
        System.out.println("Packet sent");

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

Figure 3.4: Router class which acts as a constructor for all eight routers

3.6 Controller

The controller class is the network element which controls the distribution of packets from the forwarding service. Once the forwarding service receives a packet, it will refer to the controller as to where to send the packet onto. The controller is a table which specifies the destination a UDP packet is to be sent onto. In figure 3.5, we can see that the controller has designated routers which our string input is assigned to. I have demonstrated my code using the trinity string input which goes to router 1. String newData is passed into the function as a constructor and is used from the forwarding service to determine the string sent. If what was sent into newData is not a string and is simply ‘R3’ or ‘R7’, the function will simply return the string as is with no changes.

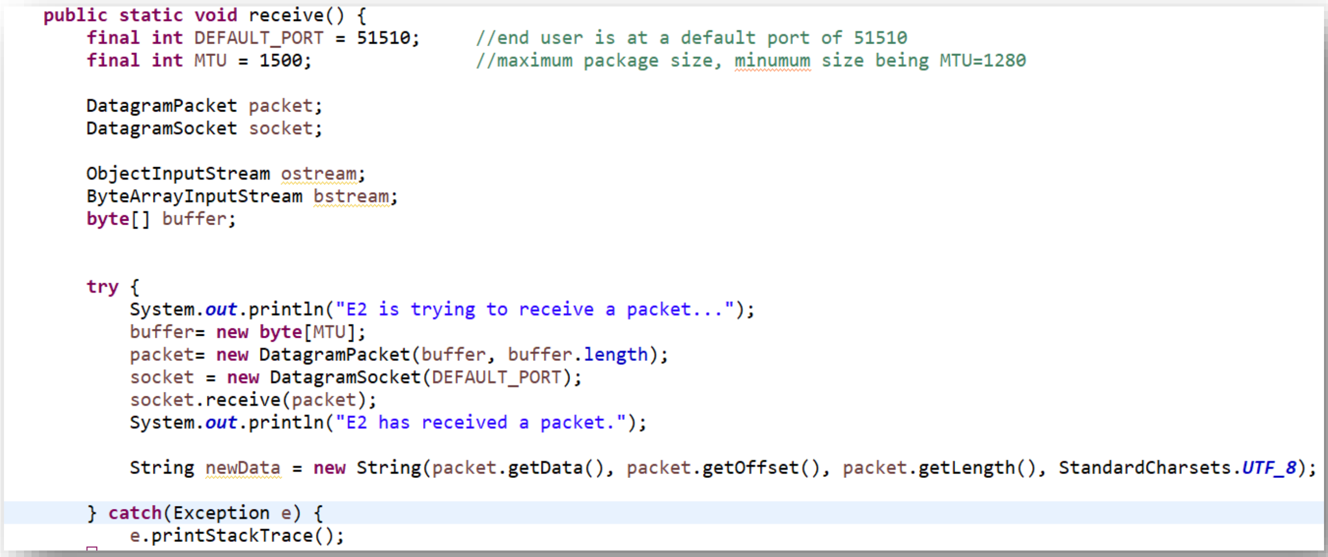


```java
public class controller {

    public static void main(String[] args) {


    }

    public static String forwardingTable(String newData) {
        if(newData.equals("trinity")) {
             System.out.println(newData="R1");
        } else if(newData.equals("centrallab")) {
             System.out.println(newData="R2");
        } else if(newData.equals("LG35")) {
             System.out.println(newData="R3");
        }else if(newData.equals("hamilton")) {
             System.out.println(newData="R5");
        } else if(newData.equals("berkeley")) {
             System.out.println(newData="R8");
        } else {
             System.out.println(newData);
        }
        return newData;
```

Figure 3.5: Controller table which determines the destination location of a packet.

3.7 End User

The end user is the destination of our UDP datagram packets. It acts much like the routers however it does not send any datagram packets on; it receives a packet from either R8 or the forwarding service and keeps that packet as shown in figure 3.6. The class receives the datagram and prints that E2 has



```java
public static void receive() {
    final int DEFAULT_PORT = 51510;     //end user is at a default port of 51510
    final int MTU = 1500;               //maximum package size, minumum size being MTU=1280

    DatagramPacket packet;
    DatagramSocket socket;

    ObjectInputStream ostream;
    ByteArrayInputStream bstream;
    byte[] buffer;


    try {
        System.out.println("E2 is trying to receive a packet...");
        buffer= new byte[MTU];
        packet= new DatagramPacket(buffer, buffer.length);
        socket = new DatagramSocket(DEFAULT_PORT);
        socket.receive(packet);
        System.out.println("E2 has received a packet.");

        String newData = new String(packet.getData(), packet.getOffset(), packet.getLength(), StandardCharsets.UTF_8);

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

received a packet, from there it will extract the data and the class will terminate as its job is to simply receive datagram packets. I could have implemented this class to be able to send datagram packets back toward R8 or to the forwarding service, however, for the purpose of the implementation having a start and end seemed a lot easier to showcase.

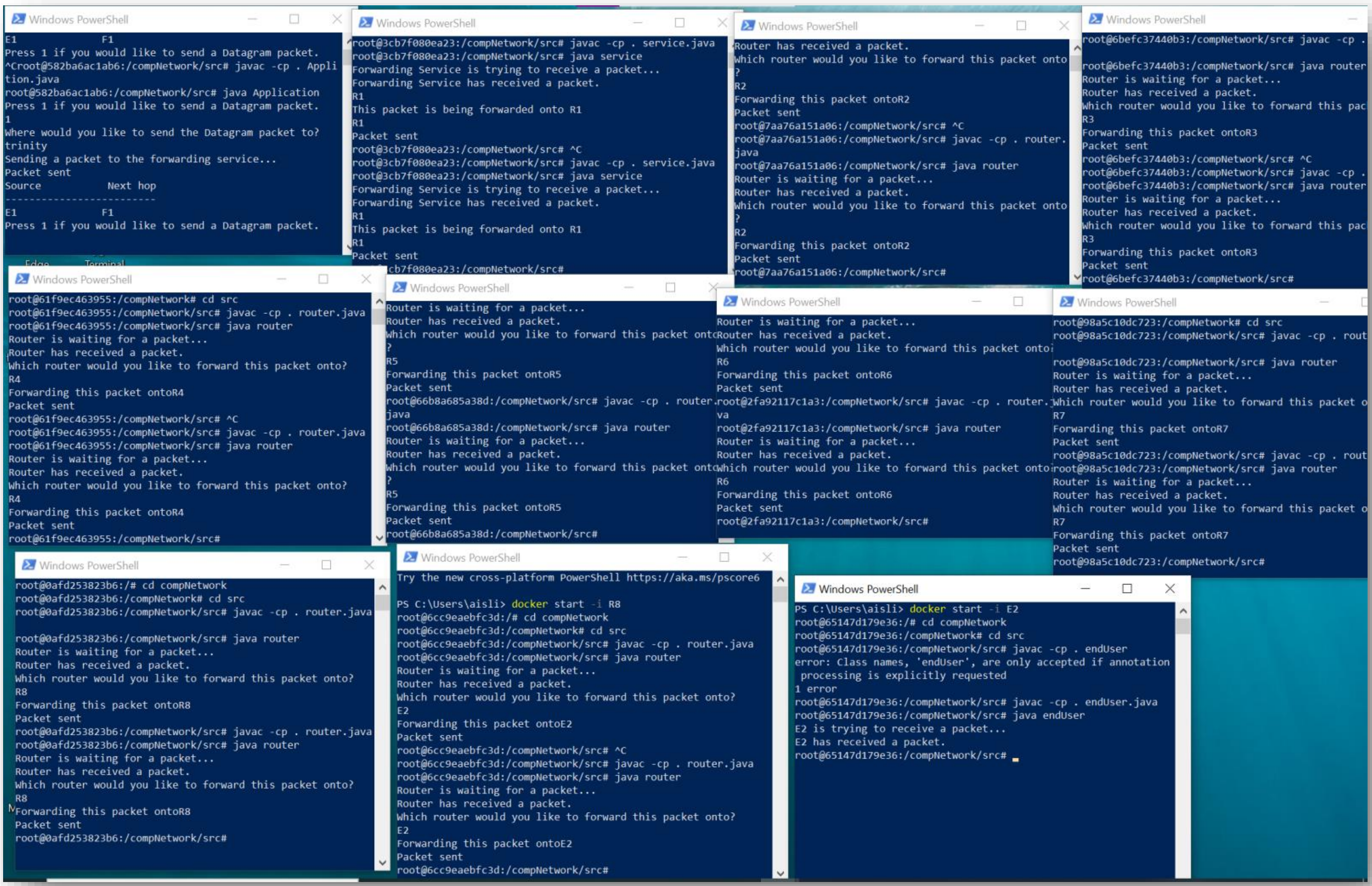
Figure 3.6: Receive function within the endUser class.

4 Discussion

The strength of this program is that all network components are connected to each other in some way and there is an ability to send UDP datagram packets to every network element. I believe the functionality of this code is implemented to a very high standard, with error handling also present within the application. If a 1 is not pressed when prompted the code will give an error message and restart, prompting you again to press a 1. The application class also loops back once a datagram packet is sent making it fast and efficient to send packets, even though my protocol can only have one datagram packet sent at any given time, it is still ideal to not have to restart my code.

Although the flow control in my program, in my opinion, was implemented to a high standard and was fast, I believe it could have been made a lot more efficient. Although nothing in my code is broken, once the routers, forwarding service or end user end, the class will terminate and to send a packet again you must recompile. This is tedious and takes time, especially when there are eight routers to recompile, although my code still works, it would be a lot more efficient if I could've implemented loopbacks.

Our assignment specifically asked us to use an external host to run our code instead of our own local hosts, so I decided to run my code via docker. I will attach my running program in docker below, in figure 4.1. You can see the application asking for a 1 to send a datagram packet, from there it asks for a destination, in this case, I used trinity which is linked to router 1. From there it goes to the forwarding service which forwards this packet onto R1. I then send the packet from R1 to end user via user input.



```
E1              F1
Press 1 if you would like to send a Datagram packet.
^Croot@582ba6ac1ab6:/compNetwork/src# javac -cp . Appli
tion.java
root@582ba6ac1ab6:/compNetwork/src# java Application
Press 1 if you would like to send a Datagram packet.
1
Where would you like to send the Datagram packet to?
trinity
Sending a packet to the forwarding service...
Packet sent
Source             Next hop
-------------------------
E1              F1
Press 1 if you would like to send a Datagram packet.
```

```
root@3cb7f080ea23:/compNetwork/src# javac -cp . service.java
root@3cb7f080ea23:/compNetwork/src# java service
Forwarding Service is trying to receive a packet...
Forwarding Service has received a packet.
R1
This packet is being forwarded onto R1
R1
Packet sent
root@3cb7f080ea23:/compNetwork/src# ^C
root@3cb7f080ea23:/compNetwork/src# javac -cp . service.java
root@3cb7f080ea23:/compNetwork/src# java service
Forwarding Service is trying to receive a packet...
Forwarding Service has received a packet.
R1
This packet is being forwarded onto R1
R1
Packet sent
root@3cb7f080ea23:/compNetwork/src#
```

```
Router has received a packet.
Which router would you like to forward this packet onto
?
R2
Forwarding this packet ontoR2
Packet sent
root@7aa76a151a06:/compNetwork/src# ^C
root@7aa76a151a06:/compNetwork/src# javac -cp . router.
java
root@7aa76a151a06:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto
?
R2
Forwarding this packet ontoR2
Packet sent
root@7aa76a151a06:/compNetwork/src#
```

```
root@6befc37440b3:/compNetwork/src# javac -cp .

root@6befc37440b3:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this pac
R3
Forwarding this packet ontoR3
Packet sent
root@6befc37440b3:/compNetwork/src# ^C
root@6befc37440b3:/compNetwork/src# javac -cp .
root@6befc37440b3:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this pac
R3
Forwarding this packet ontoR3
Packet sent
root@6befc37440b3:/compNetwork/src#
```

```
root@61f9ec463955:/compNetwork# cd src
root@61f9ec463955:/compNetwork/src# javac -cp . router.java
root@61f9ec463955:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto?
R4
Forwarding this packet ontoR4
Packet sent
root@61f9ec463955:/compNetwork/src# ^C
root@61f9ec463955:/compNetwork/src# javac -cp . router.java
root@61f9ec463955:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto?
R4
Forwarding this packet ontoR4
Packet sent
root@61f9ec463955:/compNetwork/src#
```

```
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto
?
R5
Forwarding this packet ontoR5
Packet sent
root@66b8a685a38d:/compNetwork/src# javac -cp . router.
java
root@66b8a685a38d:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto
?
R5
Forwarding this packet ontoR5
Packet sent
root@66b8a685a38d:/compNetwork/src#
```

```
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto
R6
Forwarding this packet ontoR6
Packet sent
root@2fa92117c1a3:/compNetwork/src# javac -cp . router.j
va
root@2fa92117c1a3:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto
R6
Forwarding this packet ontoR6
Packet sent
root@2fa92117c1a3:/compNetwork/src#
```

```
root@98a5c10dc723:/compNetwork# cd src
root@98a5c10dc723:/compNetwork/src# javac -cp . rout

root@98a5c10dc723:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet o
R7
Forwarding this packet ontoR7
Packet sent
root@98a5c10dc723:/compNetwork/src# javac -cp . rout
root@98a5c10dc723:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet o
R7
Forwarding this packet ontoR7
Packet sent
root@98a5c10dc723:/compNetwork/src#
```

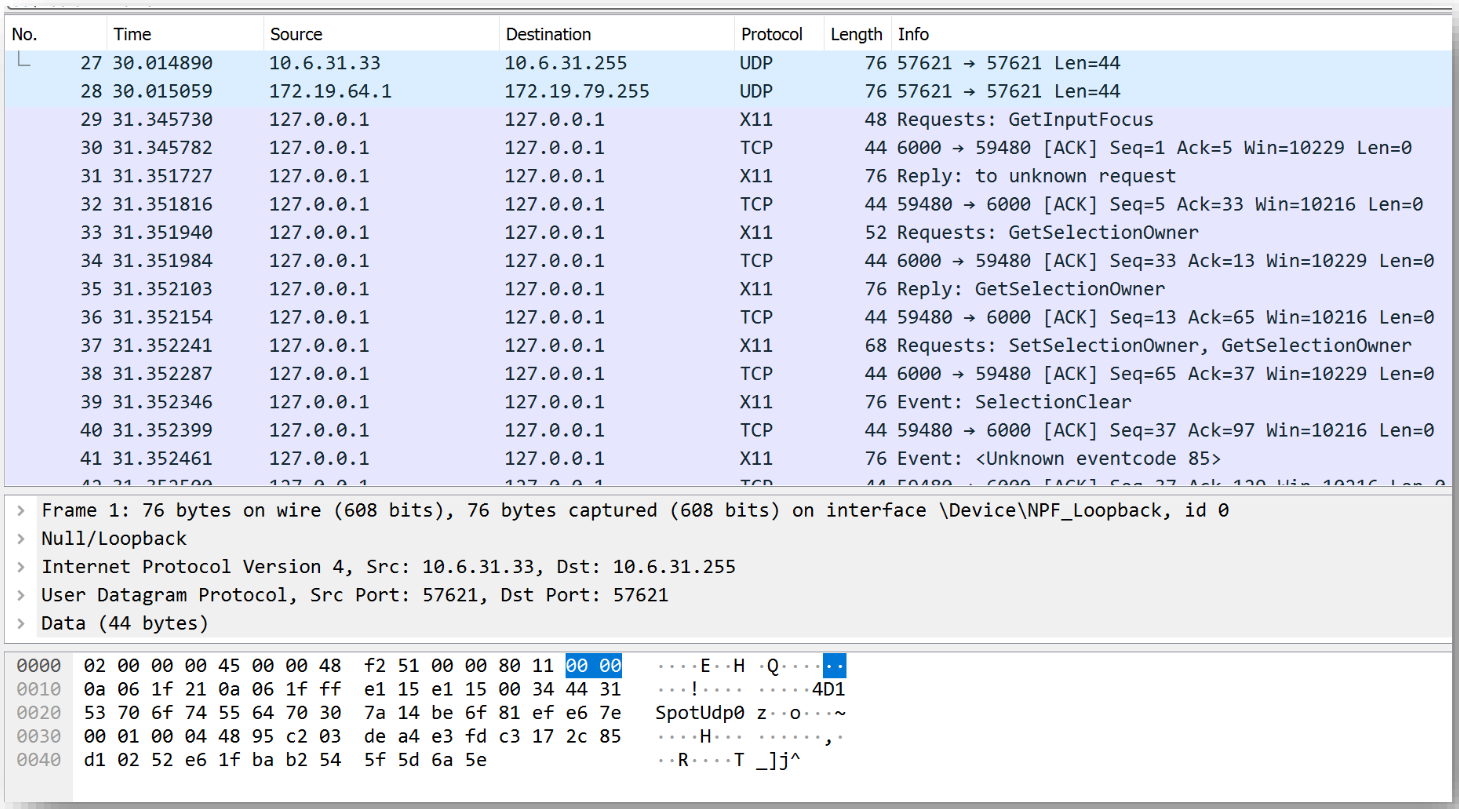
```
root@0afd253823b6:/# cd compNetwork
root@0afd253823b6:/compNetwork# cd src
root@0afd253823b6:/compNetwork/src# javac -cp . router.java

root@0afd253823b6:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto?
R8
Forwarding this packet ontoR8
Packet sent
root@0afd253823b6:/compNetwork/src# javac -cp . router.java
root@0afd253823b6:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto?
R8
Forwarding this packet ontoR8
Packet sent
root@0afd253823b6:/compNetwork/src#
```

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

PS C:\Users\aisli> docker start -i R8
root@6cc9eaebfc3d:/# cd compNetwork
root@6cc9eaebfc3d:/compNetwork# cd src
root@6cc9eaebfc3d:/compNetwork/src# javac -cp . router.java
root@6cc9eaebfc3d:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto?
E2
Forwarding this packet ontoE2
Packet sent
root@6cc9eaebfc3d:/compNetwork/src# ^C
root@6cc9eaebfc3d:/compNetwork/src# javac -cp . router.java
root@6cc9eaebfc3d:/compNetwork/src# java router
Router is waiting for a packet...
Router has received a packet.
Which router would you like to forward this packet onto?
E2
Forwarding this packet ontoE2
Packet sent
root@6cc9eaebfc3d:/compNetwork/src#
```

```
PS C:\Users\aisli> docker start -i E2
root@65147d179e36:/# cd compNetwork
root@65147d179e36:/compNetwork# cd src
root@65147d179e36:/compNetwork/src# javac -cp . endUser
error: Class names, 'endUser', are only accepted if annotation
 processing is explicitly requested
1 error
root@65147d179e36:/compNetwork/src# javac -cp . endUser.java
root@65147d179e36:/compNetwork/src# java endUser
E2 is trying to receive a packet...
E2 has received a packet.
root@65147d179e36:/compNetwork/src#
```

Figure 4.1: Docker running through my code, from top left being the application to bottom right being end User: E1,F1,R1,R2,R3,R4,R5,R6,R7,R8,E2.

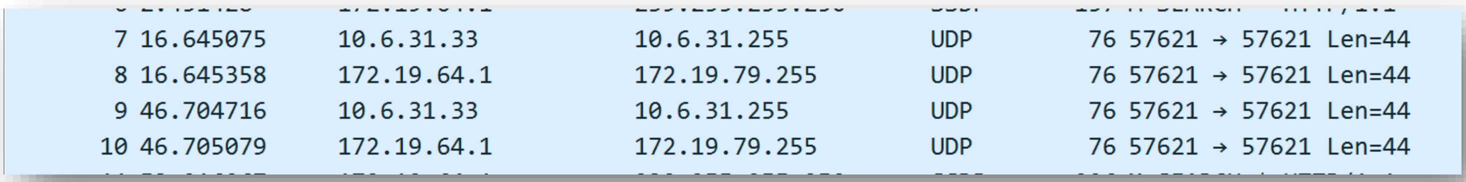
We are also asked to capture the traffic between network elements in docker, to do so I downloaded a software called Wireshark. Wireshark captures the communication between network elements, i.e., when there are UDP datagram packets being sent between containers, it will be displayed in Wireshark. See figure 4.2 and 4.3. You can see a blue banner with the label UDP, this is signalling



```
No.  Time        Source        Destination     Protocol Length Info
  27 30.014890   10.6.31.33    10.6.31.255     UDP       76 57621 → 57621 Len=44
  28 30.015059   172.19.64.1   172.19.79.255   UDP       76 57621 → 57621 Len=44
  29 31.345730   127.0.0.1     127.0.0.1       X11       48 Requests: GetInputFocus
  30 31.345782   127.0.0.1     127.0.0.1       TCP       44 6000 → 59480 [ACK] Seq=1 Ack=5 Win=10229 Len=0
  31 31.351727   127.0.0.1     127.0.0.1       X11       76 Reply: to unknown request
  32 31.351816   127.0.0.1     127.0.0.1       TCP       44 59480 → 6000 [ACK] Seq=5 Ack=33 Win=10216 Len=0
  33 31.351940   127.0.0.1     127.0.0.1       X11       52 Requests: GetSelectionOwner
  34 31.351984   127.0.0.1     127.0.0.1       TCP       44 6000 → 59480 [ACK] Seq=33 Ack=13 Win=10229 Len=0
  35 31.352103   127.0.0.1     127.0.0.1       X11       76 Reply: GetSelectionOwner
  36 31.352154   127.0.0.1     127.0.0.1       TCP       44 59480 → 6000 [ACK] Seq=13 Ack=65 Win=10216 Len=0
  37 31.352241   127.0.0.1     127.0.0.1       X11       68 Requests: SetSelectionOwner, GetSelectionOwner
  38 31.352287   127.0.0.1     127.0.0.1       TCP       44 6000 → 59480 [ACK] Seq=65 Ack=37 Win=10229 Len=0
  39 31.352346   127.0.0.1     127.0.0.1       X11       76 Event: SelectionClear
  40 31.352399   127.0.0.1     127.0.0.1       TCP       44 59480 → 6000 [ACK] Seq=37 Ack=97 Win=10216 Len=0
  41 31.352461   127.0.0.1     127.0.0.1       X11       76 Event: <Unknown eventcode 85>

> Frame 1: 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: 10.6.31.33, Dst: 10.6.31.255
> User Datagram Protocol, Src Port: 57621, Dst Port: 57621
> Data (44 bytes)

0000  02 00 00 00 45 00 00 48  f2 51 00 00 80 11 00 00   ····E··H ·Q······
0010  0a 06 1f 21 0a 06 1f ff  e1 15 e1 15 00 34 44 31   ···!···· ·····4D1
0020  53 70 6f 74 55 64 70 30  7a 14 be 6f 81 ef e6 7e   SpotUdp0 z··o···~
0030  00 01 00 04 48 95 c2 03  de a4 e3 fd c3 17 2c 85   ····H··· ······,·
0040  d1 02 52 e6 1f ba b2 54  5f 5d 6a 5e               ··R····T _]j^
```

that a UDP datagram packets are being sent across the channel.

Figure 4.2: UDP Datagram packets captured using Wireshark.



```
 7 16.645075   10.6.31.33    10.6.31.255     UDP   76 57621 → 57621 Len=44
 8 16.645358   172.19.64.1   172.19.79.255   UDP   76 57621 → 57621 Len=44
 9 46.704716   10.6.31.33    10.6.31.255     UDP   76 57621 → 57621 Len=44
10 46.705079   172.19.64.1   172.19.79.255   UDP   76 57621 → 57621 Len=44
```

Figure 4.3: UDP datagram packets being captured on Wireshark.

5 Summary

This report has described my attempt at a solution in understanding and designing a protocol that forwards payloads based on a collection of strings. A lot of the code I used for this assignment was a build on of the code from my first assignment. It is evident that you do not need the most complex code to implement code to a high standard. The description of my implementation throughout this report highlights all the essential components which were a key part of my solution.

6 Reflection

Upon reflection at the last six weeks of attempting this assignment and comparing my current Flow Forwarding assignment to my previous IoT Subscribe/Protocol assignment, I am extremely happy with the progress I have made. In my first assignment I was unable to get docker working correctly, however with this assignment, not only did I understand to a better degree how to implement my components, but I successfully was able to get docker working and proper pcapp files from Wireshark.

Apart from this report, this code took approximately 20 hours to complete.