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

CSU33031 Computer Networks
Assignment #1: Flow Forwarding Protocols

Daniel Whelan, 19335045

December 3, 2021

Contents

1 Overall Design 2

1.1 Flow Forwarding 2

1.2 Routers Use of Flow Tables 2

1.3 Design of Network Elements 3

1.4 Packet Descriptions 5

2 Implementation 5

2.1 Application 5

2.2 Forwarding Service 7

2.3 Router 9

2.4 Controller 10

3 Discussion 10

4 Summary 12

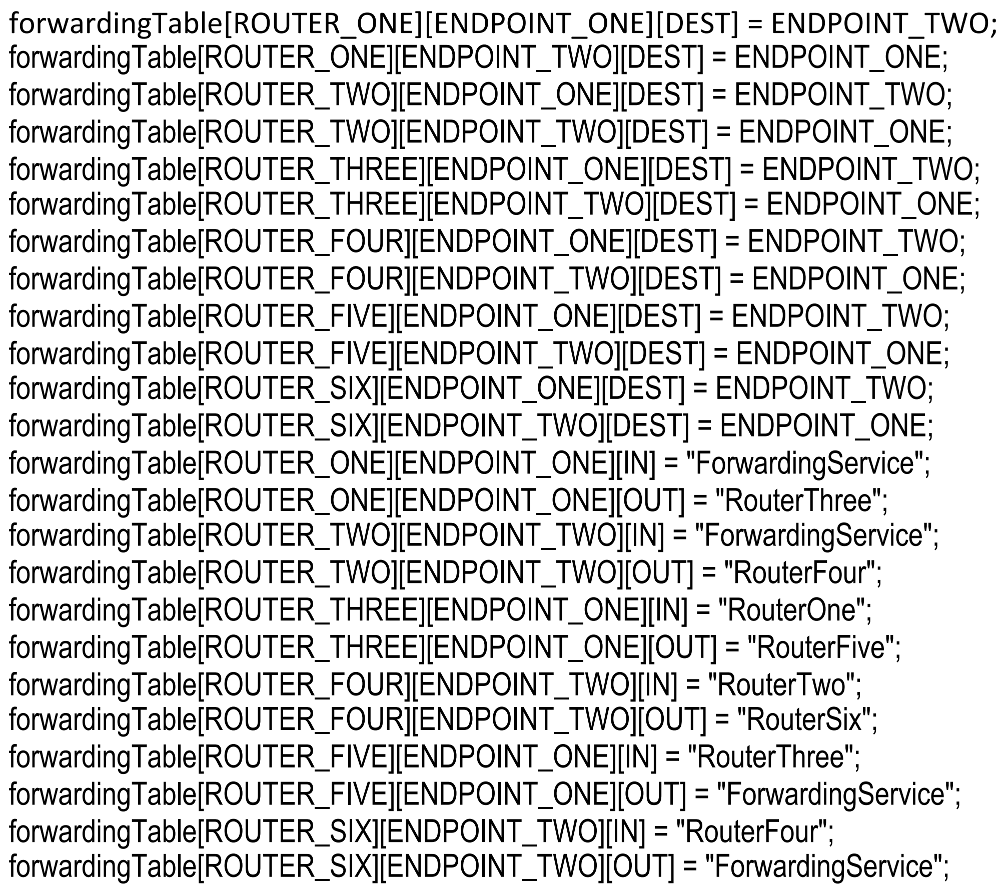
5 Reflection 12

1 Overall Design

In the first two sections I will discuss my understanding of flow forwarding and how Routers use Flow Tables to determine where packets are going to be sent throughout the defined topology. I will then discuss how I implemented these elements in my topology . Following this I will discuss the strengths and weaknesses of my design and in turn give a general summary and reflection of the assignment as a whole.

1.1 Flow Forwarding

Flow Forwarding refers to the decisions to forward a flow of packets to a designated location and the information that is contained in network nodes in order to make these forwarding decisions. The design of these protocols aims to reduce the processing that occurs when determining where packets are to be sent and thus increase the flexibility, efficiency and scalability of the network as a whole.



```
forwardingTable[ROUTER_ONE][ENDPOINT_ONE][DEST] = ENDPOINT_TWO;
forwardingTable[ROUTER_ONE][ENDPOINT_TWO][DEST] = ENDPOINT_ONE;
forwardingTable[ROUTER_TWO][ENDPOINT_ONE][DEST] = ENDPOINT_TWO;
forwardingTable[ROUTER_TWO][ENDPOINT_TWO][DEST] = ENDPOINT_ONE;
forwardingTable[ROUTER_THREE][ENDPOINT_ONE][DEST] = ENDPOINT_TWO;
forwardingTable[ROUTER_THREE][ENDPOINT_TWO][DEST] = ENDPOINT_ONE;
forwardingTable[ROUTER_FOUR][ENDPOINT_ONE][DEST] = ENDPOINT_TWO;
forwardingTable[ROUTER_FOUR][ENDPOINT_TWO][DEST] = ENDPOINT_ONE;
forwardingTable[ROUTER_FIVE][ENDPOINT_ONE][DEST] = ENDPOINT_TWO;
forwardingTable[ROUTER_FIVE][ENDPOINT_TWO][DEST] = ENDPOINT_ONE;
forwardingTable[ROUTER_SIX][ENDPOINT_ONE][DEST] = ENDPOINT_TWO;
forwardingTable[ROUTER_SIX][ENDPOINT_TWO][DEST] = ENDPOINT_ONE;
forwardingTable[ROUTER_ONE][ENDPOINT_ONE][IN] = "ForwardingService";
forwardingTable[ROUTER_ONE][ENDPOINT_ONE][OUT] = "RouterThree";
forwardingTable[ROUTER_TWO][ENDPOINT_TWO][IN] = "ForwardingService";
forwardingTable[ROUTER_TWO][ENDPOINT_TWO][OUT] = "RouterFour";
forwardingTable[ROUTER_THREE][ENDPOINT_ONE][IN] = "RouterOne";
forwardingTable[ROUTER_THREE][ENDPOINT_ONE][OUT] = "RouterFive";
forwardingTable[ROUTER_FOUR][ENDPOINT_TWO][IN] = "RouterTwo";
forwardingTable[ROUTER_FOUR][ENDPOINT_TWO][OUT] = "RouterSix";
forwardingTable[ROUTER_FIVE][ENDPOINT_ONE][IN] = "RouterThree";
forwardingTable[ROUTER_FIVE][ENDPOINT_ONE][OUT] = "ForwardingService";
forwardingTable[ROUTER_SIX][ENDPOINT_TWO][IN] = "RouterFour";
forwardingTable[ROUTER_SIX][ENDPOINT_TWO][OUT] = "ForwardingService";
```

Figure 1: A sample of the Master Forwarding Table contained in the controller that is information is requested from individually by routers when unsure of where to send a packet.

1.2 Routers Use of Flow Tables

Routers use Flow Tables to determine the location of where a packet is being sent. For instance, a router may receive an unknown packet and does not have the location of its destination saved in its own local forwarding table. Hence the Router must contact the Central Controller in order to access the location to which to send this packet. Once the Controller returns a location the Router updates its own address table to prevent needless packets being sent to the Controller.

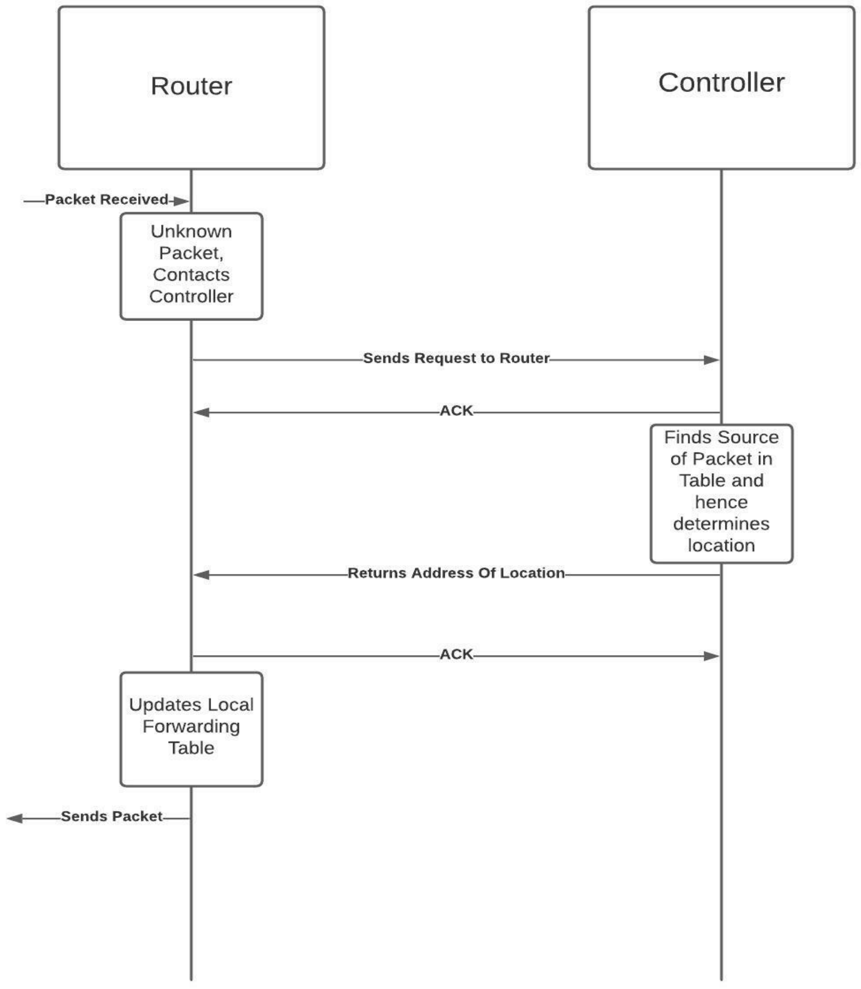

Router
Controller
Packet Received
Unknown Packet, Contacts Controller
Sends Request to Router
ACK
Finds Source of Packet in Table and hence determines location
Returns Address Of Location
ACK
Updates Local Forwarding Table
Sends Packet

Figure 2: The figure above shows a router receiving a packet whose destination it is unsure of. It contacts the Central Controller and requests the address that this packet is to be sent to. The controller determines the destination and sends the relevant address to the router after which the router updates its local address table with this value and sends the packet to the relevant location.

1.3 Design of Network Elements

My network consists of a multitude of nodes that all work off of four individual pieces of code. There is the Application class which takes in where the user wishes to send data alongside the data it wants to send, and also displays data to the user that has been sent to their specific endpoint. The Forwarding Service that receives packets from the Application Endpoint and Forwards these into the network of nodes that connect the number of application endpoints. The Routers are simple elements that there are six of in my topology that simply receive a packet and forward it to the next node based on its local address table that can be updated by the Controller which acts as a centralised forwarding table for all of the packets, containing all of the information regarding where packets are coming from and going to.

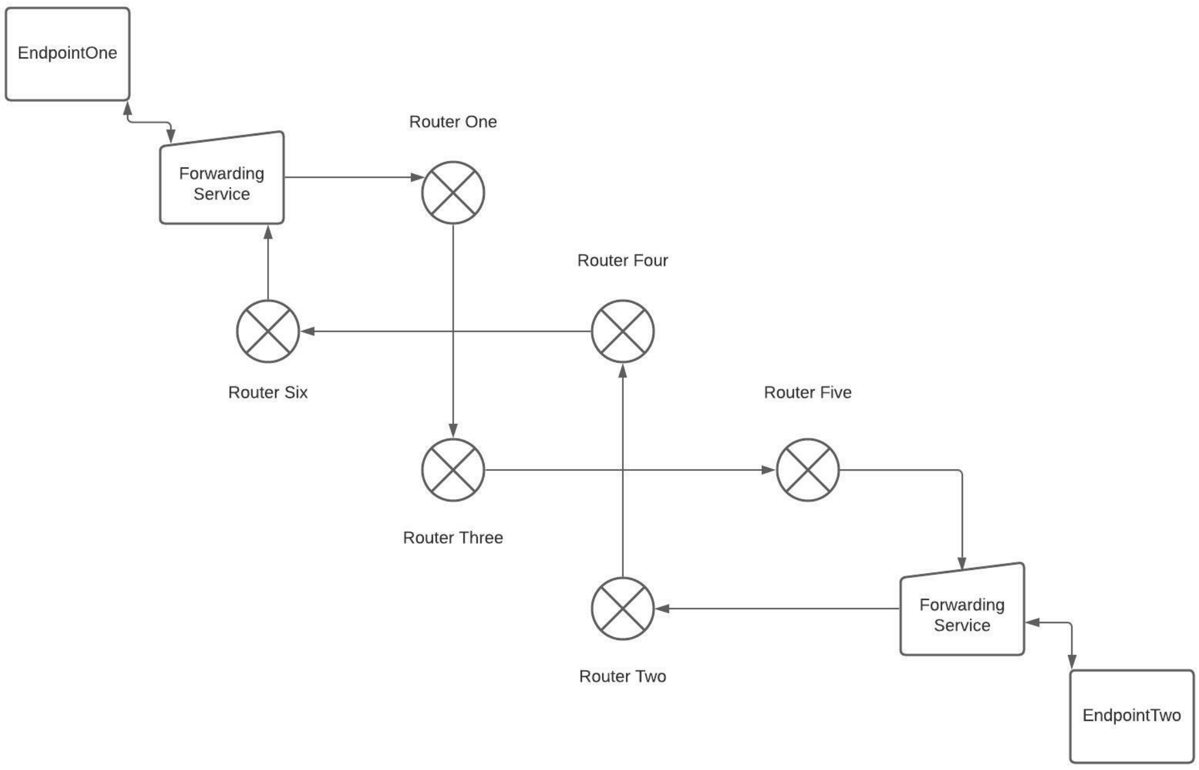


Figure 3: The network topology of this assignment, The Endpoint Applications are each connected to the Forwarding Service which in turn are linked to a series of Routers that lead a packet to its destination, the opposite Application endpoint, which each can contact a controller (not included in diagram above) that if the destination of a packet is unknown, can be contacted and the destination will be sent to the router.

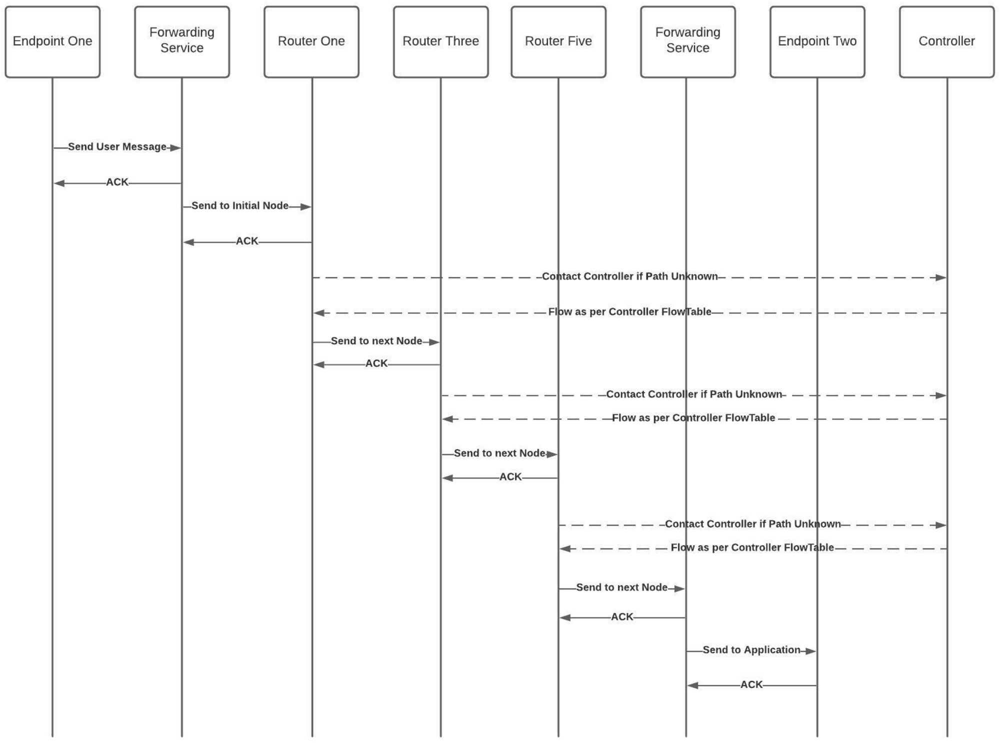


Figure 4: The standard order of operation for sending a message from Endpoint One to Endpoint Two. The Endpoint One sends its user message to a forwarding service which hen forwards to an initial node, each router then checks if the destination address is present in its local address table, if not it contacts the controller and updates its address table with the response, sending the packet across the path until it reaches the Endpoint Two.

1.4 Packet Descriptions

The Packets sent across this topology are created using the Type-Length-Value (TLV) method of creating the header for a packet. The Type is the element that determines where in the network the packet is being sent and can either be Endpoint One or Endpoint Two for this topology, the Length is the length of the value that is being sent in this packet and the value is the content that will be decoded once the packet has reached its final destination. An interesting aspect of this method is that the packets can be nested inside larger versions of a header and allows for more specific values to be created from the strings that could implement different routes based off of the combination of values. However, it was felt unnecessary to use for this topology and hence is not utilised, however, in later iterations of this topology it could be used to generate new routes for the packets to take in the network of routers

2 Implementation

This section presents the implementation details of the individual network elements, The Application, The Forwarding Service, The Routers and the Controller.

2.1 Application

The Client is the element of the topology that the user interacts with in order to send packets across the network to the other endpoint in use. Here the user is asked to input the Destination of the packet that is being sent and the content of the packet that they would like to send. The Application then uses this destination data to determine the Type that will be associated with the header of the packet using a simple if/else statement based on a specified user input.

```
System.out.println("Please enter destination of packet (EndpointOne or EndpointTwo) >");
destination = input.next();
System.out.println("Please enter message to send >");
message = input.next();
data = new byte[HEADER_LENGTH + message.length()];
if(destination.equalsIgnoreCase("endpoint1") || destination.equalsIgnoreCase("endpointone")) {
    data[TYPE] = ENDPOINT_TWO;
}
else if(destination.equalsIgnoreCase("endpoint2") || destination.equalsIgnoreCase("endpointtwo")) {
    data[TYPE] = ENDPOINT_ONE;
}
else {
    data[TYPE] = ERROR;
}
data[LENGTH] = (byte) message.length();
```

Listing 1: At the Application the user is asked to enter a destination, which in turn determines the Type value that will be used in the header, following this, the user is then asked what message they would like to send, which is the value element of the header of which the length element is determined using the length() function included in the String class.

After the class constructor initialises the socket and listener, the start() function is called which asks for the user input specified above, and initialises the header as also shown above. Following this the Application sends the packet to a Forwarding Service which will be discussed later in this report. Upon completing this the Application waits for a packet to be received from another endpoint, in the case of this topology Endpoint Two. Upon receiving this packet the application then judges what to do next using a switch statement that is based off of the type element of the header.

```
switch(data[TYPE]) {
    case ACK:
        System.out.println("Packet Received by Forwarding Service");
        break;
    case ENDPOINT_ONE:
        message = sendAck(packet, data);
        System.out.println("Endpoint One Says: " + message);
        start();
        break;
    case ENDPOINT_TWO:
        message = sendAck(packet, data);
        System.out.println("Endpoint Two Says: " + message);
        start();
        break;
    default:
       message = sendAck(packet, data);
       System.err.println("ERROR: Unexpected Packet Received");
       start();
       break;
  }
```

Listing 2: The switch statement that is used to determine what the Application program will do after receiving a packet. It reads the Type element of the header and in turn determines what to do if the packet has come from Endpoint One or Endpoint Two. If it is an unknown packet it is dropped.

Once the packet has been received and dealt with the program then runs the start() function again so that it can send messages back to other endpoints as it has received a response to its message. This process continue indefinitely until the program is exited and no longer is needed.

2.2 Forwarding Service

The Forwarding Service acts like a bridge between the Application and the network of routers that connect endpoints in this topology. As a result of this the program itself revolves around the receipt of packets from an endpoint and then sending them to the relevant initial router so that they are sent on the correct path to the destination endpoint.

```
switch(data[TYPE]) {
    case ACK:
        System.out.println("Packet Received");
        break;
    case ENDPOINT_ONE:
        sendAck(packet, data);
        router = new InetSocketAddress("RouterOne", DEFAULT_PORT);
        application = new InetSocketAddress("EndpointTwo", DEFAULT_PORT);
        System.out.println("Source is EndpointOne");
        packet.setSocketAddress(router);
        socket.send(packet);
        break;
     case ENDPOINT_TWO:
        content = sendAck(packet, data);
        router = new InetSocketAddress("RouterTwo", DEFAULT_PORT);
        application = new InetSocketAddress("EndpointOne",DEFAULT_PORT);
        System.out.println("Source is EndpointTwo");
        packet.setSocketAddress(router);
        socket.send(packet);
        break;
```

Listing 3: The first half of the switch statement, differentiated by the type in the header that is used in the Forwarding Service Program. Here it determines whether the packet has been sent from Endpoint One or Two and in turn sends the packet to the initial router in the path that is relevant to that endpoint.

The other function of the Forwarding Service is to send packets to the endpoints once they have come out of the network of routers. It does this in another section of the switch statement above where it determines which final router it has come from and in turn it then sends the packet to the relevant endpoint.

```
case ROUTER_FIVE:
    content = sendAck(packet, data);
    sendPacket(ENDPOINT_ONE, content, application);
    break;
case ROUTER_SIX:
    content = sendAck(packet, data);
    sendPacket(ENDPOINT_TWO, content, application);
    break;
case ERROR:
    content = sendAck(packet, data);
    System.out.println("Unknown destination in topology, packet dropped");
    break;
default:
    System.err.println("ERROR: Unexpected packet received");
    break;
```

Listing 4: The second half of the above switch statement that determines which endpoint a packet from the network of routers is to be sent to. Based off of the type header that is changed in the Router program when the router entered is marked as a final router in the path taken, this will be discussed later in the report.

The packet is then sent to the relevant endpoint, implementing the application program, and the message is displayed on the terminal.

Above it can be seen that two methods, sendAck and sendPacket, are called, these are called in every node within this topology and server as the two base methods for sending and receiving information in the topology.

```
private String sendAck(DatagramPacket packet, byte[] data) {
    try{
        String content;
        DatagramPacket response;
        byte[] buffer = new byte[data[LENGTH]];
        System.arraycopy(data, HEADER_LENGTH, buffer, 0, data[LENGTH]);
        content = new String(buffer);
        data = new byte[HEADER_LENGTH];
        data[TYPE] = ACK;
        data[ACKCODE] = ACKPACKET;
        response = new DatagramPacket(data, data.length);
        response.setSocketAddress(packet.getSocketAddress());
        socket.send(response);
        return content;
    } catch(Exception e) {
        e.printStackTrace();
        return "";
    }
}
```

Listing 5: The sendAck method. It takes in the packet that was sent and the data contained within the packet. Within this method the content of the received packet is extracted and stored in a String variable until returned at the end of the method. An Ack is sent from the method to the sender of the packet by creating a new packet containing the ACK type in the header.

```
private void sendPacket(byte type, String content, InetSocketAddress dstAddress){
    try {
        byte[] data;
        DatagramPacket packet;
        data = new byte[HEADER_LENGTH + content.length()];
        data[TYPE] = type;
        data[LENGTH] = (byte) content.length();
        System.arraycopy(content.getBytes(), 0, data, HEADER_LENGTH, content.length());
        packet = new DatagramPacket(data, data.length);
        packet.setSocketAddress(dstAddress);
        socket.send(packet);
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

Listing 6: The sendPacket method. It takes in the type of the packet, the content to be sent in the packet and the destination address of the packet. It constructs the packet in a standard way, formulating the header followed by content in a byte array and in turn storing this in a packet to be sent to the relevant address.

The Forwarding Service then continues this mode of operation until it is shut down at the end of the networks use.

2.3 Router

The Router acts as a middleman in the sending of packets between endpoints. Upon its construction the Router prompts the user for its router designation number. As the topology uses six routers the router numbers range from 1-6 with 1 and 2 are initial routers the forwarding service sends the packets to and 5 and 6 being final routers that send specialised headers to the forwarding service to know when to send packets back to the application endpoints. These values are stored in a variable called routerNumber that is individual to each router, i.e. non-static.

Packets being sent from endpoint one take a different path between the routers than taken by those sent from endpoint two. These paths are controlled by the flow tables that are stored locally in each of the individual routers. However, if the packet received is of a type the Router does not know, it then consults the Controller in order to determine the destination this packet needs to be forwarded to. It does this upon the receipt of an unknown packet

```
content = sendAck(packet, data);
if(addressTable[ENDPOINT_ONE].equals("")) {
    System.out.println("Unknown address calling on the Controller");
    tempContent = content;
    temp = data;
    String message = "0";
    sendPacket((byte)(routerNumber-1), message, controllerAddress);
 }
else {
    if(routerNumber == ROUTER_FIVE) {
        sendPacket((byte) ROUTER_FIVE, content, (new
        InetSocketAddress(addressTable[ENDPOINT_ONE], DEFAULT_PORT)));
    }
    else {
        sendPacket(ENDPOINT_ONE, content, (new
        InetSocketAddress(addressTable[ENDPOINT_ONE], DEFAULT_PORT)));
    }
 }
```

Listing 7: The Router determines whether the destination for a packet type is stored in its forwarding table, if it is not it stores this packets data in a temporary state and sends a packet to the controller requesting the address to be sent to the router. If it knows the address it determines whether the router is a final router and sends the packet accordingly

When a packet has been sent to the controller by a router it waits for a response and once received the data stored temporarily in variables is sent to the relevant destination and the forwarding table is updated in order to prevent controller access being requested again.

```
if(routerNumber == ROUTER_FIVE) {
        content = sendAck(packet, data);
        updateAddressTable(content, ENDPOINT_ONE);
        sendPacket((byte) ROUTER_FIVE, tempContent, (new
        InetSocketAddress(addressTable[ENDPOINT_ONE], DEFAULT_PORT)));
}
else {
        content = sendAck(packet, data);
        updateAddressTable(content, ENDPOINT_ONE);
        sendPacket(ENDPOINT_ONE, tempContent, (new
        InetSocketAddress(addressTable[ENDPOINT_ONE], DEFAULT_PORT)));
}
```

Listing 8: Upon receiving an update from the controller the router determines whether the router is a final router in the network and if it is updates the flow table and sends the specialised packet using the Router number as a new type in the header

The Router should not need to contact the controller again once it has been set a value for its address table as it is updated immediately after receiving a value from the controller. The routers continue this operation until they are exited

2.4 Controller

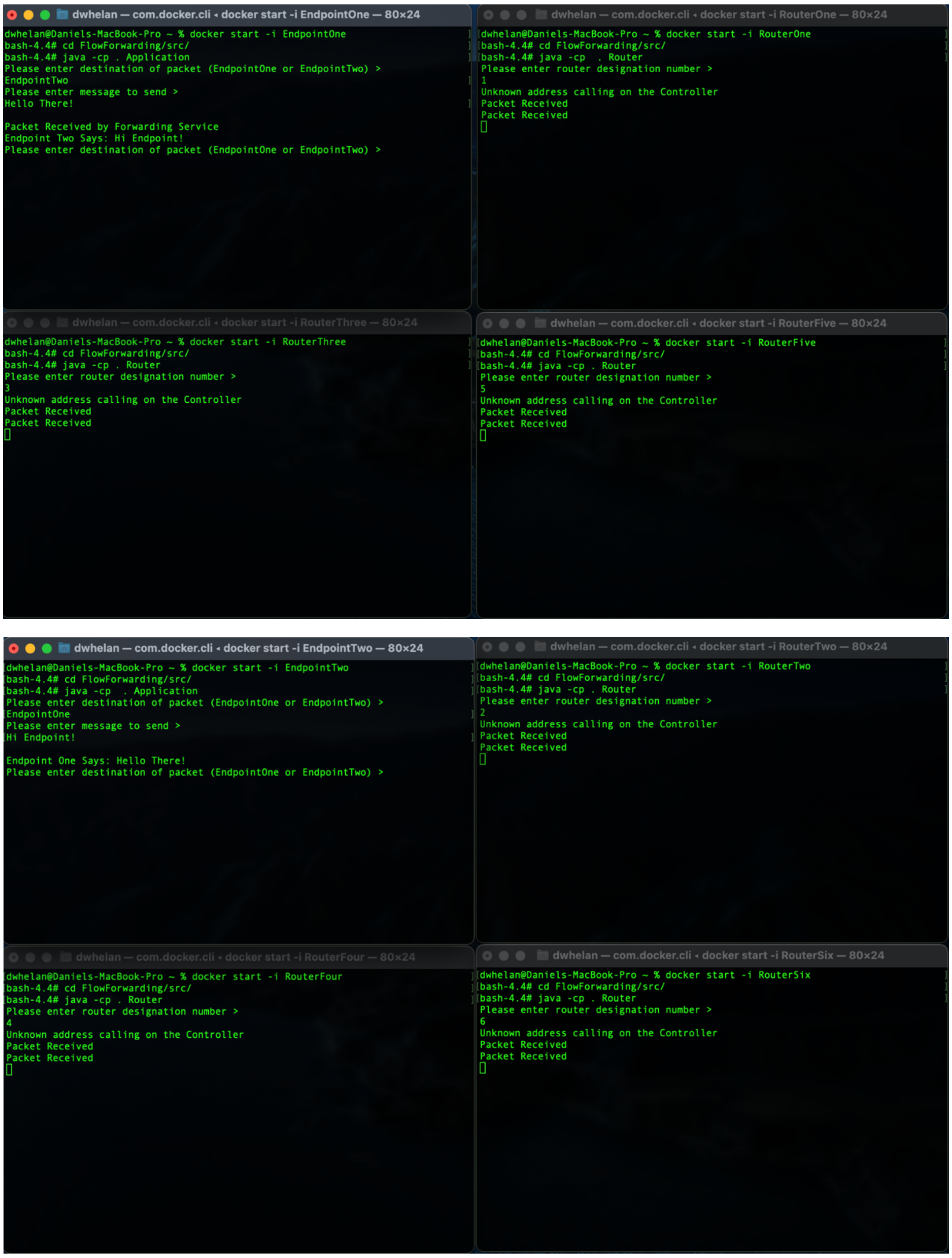
The Controller in this topology acts as a centralised "master" forwarding table for all of the routers that are contained in the topology. If a router does not know where to send a packet, they contact the Controller and the Controller returns a value based off of the Router that the request was sent from and the source of the packet. Upon construction of the Controller it initialises the forwarding table, seen at the beginning of the report, and initialises the address table for all of the routers currently in the topology. It then waits for a router to send a request for an address. It continues this process until it is shut down.

```
case ROUTER_ONE:
        content = sendAck(packet, data);
        address = (String)
        forwardingTable[ROUTER_ONE][Integer.parseInt(content)][OUT];
        sendPacket(CONTROLLER, address, addressTable[ROUTER_ONE]);
        break;
```

Listing 8: The Switch statement that reads the Router that has sent in a request for an address based on the type of the header in the request. The Controller then accesses the forwarding table using the content sent in the request and sends a packet that contains this address back to the router that sent the packet based off of the address table stored in the Controller containing all Router addresses. The above is standard case for all routers with the example being Router One.

3 Discussion

The strengths of this program are its efficiency and scalability. The Packets are sent through a number of nodes in the network and take a very small amount of time to get from endpoint one to endpoint two and vice versa. Furthermore, the network topology is very scalable. Additional routers can be added to the path of routers with very few modifications required in the code itself, adding a section in the controller forwarding table for the router(s) and updating address paths for routers in the controller to incorporate the new router(s) into the paths. To add multiple endpoints, modifications would only be required in the Forwarding Service program to incorporate where they're sent in the switch statement defined above, as well as incorporating the endpoint name as a viable option for user input in the Application terminal. However it's disadvantage is that it is a very hardcoded implementation of the Flow Forwarding Protocol and should an endpoint be implemented that is not included in the initial code the topology could not handle packages being sent from this completely unknown source.



dwhelan — com.docker.cli ◂ docker start -i EndpointOne — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i EndpointOne
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp . Application
Please enter destination of packet (EndpointOne or EndpointTwo) >
EndpointTwo
Please enter message to send >
Hello There!

Packet Received by Forwarding Service
Endpoint Two Says: Hi Endpoint!
Please enter destination of packet (EndpointOne or EndpointTwo) >
```

dwhelan — com.docker.cli ◂ docker start -i RouterOne — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i RouterOne
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp  . Router
Please enter router designation number >
1
Unknown address calling on the Controller
Packet Received
Packet Received
```

dwhelan — com.docker.cli ◂ docker start -i RouterThree — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i RouterThree
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp . Router
Please enter router designation number >
3
Unknown address calling on the Controller
Packet Received
Packet Received
```

dwhelan — com.docker.cli ◂ docker start -i RouterFive — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i RouterFive
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp . Router
Please enter router designation number >
5
Unknown address calling on the Controller
Packet Received
Packet Received
```

dwhelan — com.docker.cli ◂ docker start -i EndpointTwo — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i EndpointTwo
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp  . Application
Please enter destination of packet (EndpointOne or EndpointTwo) >
EndpointOne
Please enter message to send >
Hi Endpoint!

Endpoint One Says: Hello There!
Please enter destination of packet (EndpointOne or EndpointTwo) >
```

dwhelan — com.docker.cli ◂ docker start -i RouterTwo — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i RouterTwo
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp . Router
Please enter router designation number >
2
Unknown address calling on the Controller
Packet Received
Packet Received
```

dwhelan — com.docker.cli ◂ docker start -i RouterFour — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i RouterFour
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp . Router
Please enter router designation number >
4
Unknown address calling on the Controller
Packet Received
Packet Received
```

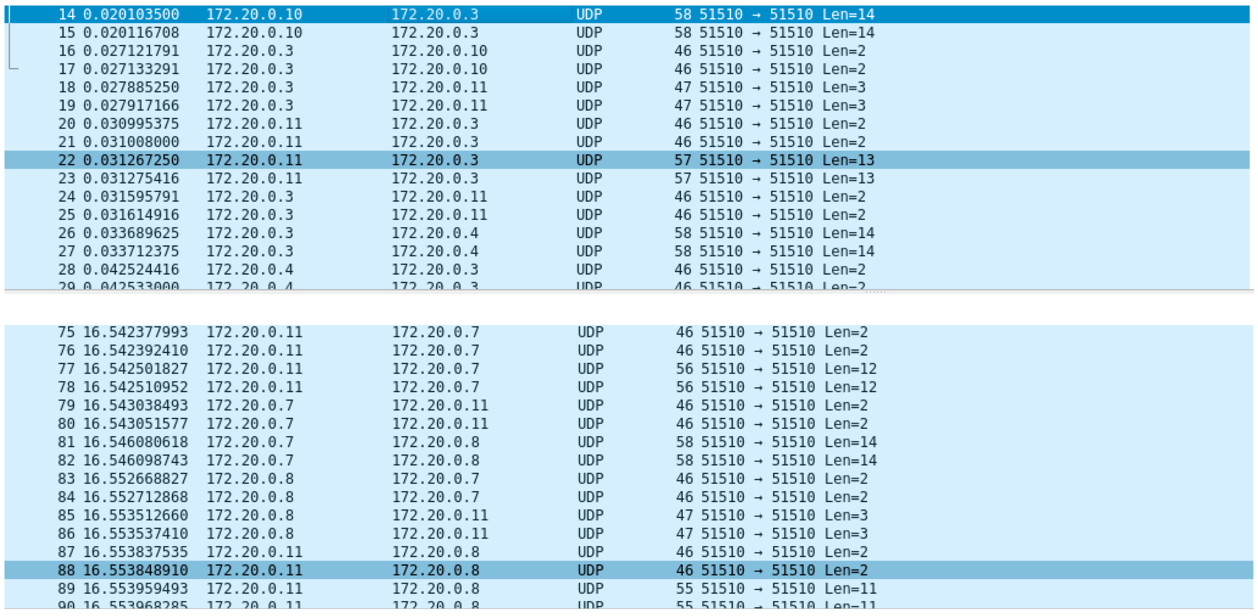
dwhelan — com.docker.cli ◂ docker start -i RouterSix — 80×24

```
dwhelan@Daniels-MacBook-Pro ~ % docker start -i RouterSix
bash-4.4# cd FlowForwarding/src/
bash-4.4# java -cp . Router
Please enter router designation number >
6
Unknown address calling on the Controller
Packet Received
Packet Received
```

Figure 5: Demonstration of messages from Endpoint One and Two being sent through Routers in defined paths.

Above is an image of messages being sent through the network of routers from one endpoint to another and vice versa. One can see the prompt for the user to enter the destination of the packet and the message in both Application Endpoint screens. This is then sent to the Forwarding Service, seen through the ACK sent to Endpoint One, the messages are then sent through the Router network. It can be seen when a destination of a packet is unknown the

Controller is contacted and then the packet is forwarded onto the relevant addresses until it reaches its final destination of an endpoint. Below see some of the packet captures made by Wireshark during this transfer.



```
14 0.020103500   172.20.0.10    172.20.0.3     UDP    58 51510 → 51510 Len=14
15 0.020116708   172.20.0.10    172.20.0.3     UDP    58 51510 → 51510 Len=14
16 0.027121791   172.20.0.3     172.20.0.10    UDP    46 51510 → 51510 Len=2
17 0.027133291   172.20.0.3     172.20.0.10    UDP    46 51510 → 51510 Len=2
18 0.027885250   172.20.0.3     172.20.0.11    UDP    47 51510 → 51510 Len=3
19 0.027917166   172.20.0.3     172.20.0.11    UDP    47 51510 → 51510 Len=3
20 0.030995375   172.20.0.11    172.20.0.3     UDP    46 51510 → 51510 Len=2
21 0.031008000   172.20.0.11    172.20.0.3     UDP    46 51510 → 51510 Len=2
22 0.031267250   172.20.0.11    172.20.0.3     UDP    57 51510 → 51510 Len=13
23 0.031275416   172.20.0.11    172.20.0.3     UDP    57 51510 → 51510 Len=13
24 0.031595791   172.20.0.3     172.20.0.11    UDP    46 51510 → 51510 Len=2
25 0.031614916   172.20.0.3     172.20.0.11    UDP    46 51510 → 51510 Len=2
26 0.033689625   172.20.0.3     172.20.0.4     UDP    58 51510 → 51510 Len=14
27 0.033712375   172.20.0.3     172.20.0.4     UDP    58 51510 → 51510 Len=14
28 0.042524416   172.20.0.4     172.20.0.3     UDP    46 51510 → 51510 Len=2
29 0.042533000   172.20.0.4     172.20.0.3     UDP    46 51510 → 51510 Len=2
```

```
75 16.542377993  172.20.0.11    172.20.0.7     UDP    46 51510 → 51510 Len=2
76 16.542392410  172.20.0.11    172.20.0.7     UDP    46 51510 → 51510 Len=2
77 16.542501827  172.20.0.11    172.20.0.7     UDP    56 51510 → 51510 Len=12
78 16.542510952  172.20.0.11    172.20.0.7     UDP    56 51510 → 51510 Len=12
79 16.543038493  172.20.0.7     172.20.0.11    UDP    46 51510 → 51510 Len=2
80 16.543051577  172.20.0.7     172.20.0.11    UDP    46 51510 → 51510 Len=2
81 16.546080618  172.20.0.7     172.20.0.8     UDP    58 51510 → 51510 Len=14
82 16.546098743  172.20.0.7     172.20.0.8     UDP    58 51510 → 51510 Len=14
83 16.552668827  172.20.0.8     172.20.0.7     UDP    46 51510 → 51510 Len=2
84 16.552712868  172.20.0.8     172.20.0.7     UDP    46 51510 → 51510 Len=2
85 16.553512660  172.20.0.8     172.20.0.11    UDP    47 51510 → 51510 Len=3
86 16.553537410  172.20.0.8     172.20.0.11    UDP    47 51510 → 51510 Len=3
87 16.553837535  172.20.0.11    172.20.0.8     UDP    46 51510 → 51510 Len=2
88 16.553848910  172.20.0.11    172.20.0.8     UDP    46 51510 → 51510 Len=2
89 16.553959493  172.20.0.11    172.20.0.8     UDP    55 51510 → 51510 Len=11
90 16.553968285  172.20.0.11    172.20.0.8     UDP    55 51510 → 51510 Len=11
```

Figure 6: This figure shows some of the packets that make up the message exchange in Wireshark. As seen the port numbers of all of the nodes are identical in 51510. The lengths also highlight the different packets, the larger lengths over ten highlight the messages sent by users, the lengths of two highlight an ACK and the lengths of three show a request to the Controller by a Router.

4 Summary

This report has described my attempt at implementing a Flow Forwarding Protocol for a topology consisting of six Routers, two Endpoints and a Forwarding Service, with Flow being managed by a centralised controller. It has highlighted the overall design of the topology that was created in order to complete the assignment and the methods that were used to emulate the network that was outlined in this assignment. The description of how the packets were designed was outlined and how the header was organised, using Type/Length/Value (TLV) as the basis. Furthermore, the classes that were used in each individual node were detailed and discussed, highlighting the programming choices that were made during the development of the topology.

5 Reflection

I found that the structure of these assignments was very helpful. The bi-weekly submission of a video demonstration and pcap files allowed for me to manage and maintain my time for the assignment and ensured that I was not only reaching the set deadlines but my own personal deadlines for the assignment. Overall I feel that I have learned a vast amount throughout this assignment. I have furthered my skills with docker to emulate the creation of a network and the communication that occurs between nodes within a topology and enhanced my ability working with containers. Furthermore I have learned advanced methods required for the creation of the header and content of a packet and how to send it using a program that I have written. As a final word, this assignment has definitely aided me in growing as a Software Developer.