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

CSU33031 Computer Networks
Assignment #2:
Flow Forwarding

Saumya Bahuguna
3rd December
2021

Contents

1 Introduction

This report aims at designing a protocol similar to IPv4/IPv6 that forwards messages based on the information enclosed within their headers. To use router, containers, and flow tables to accomplish that task

2 Theory of Topic

2.1 Flow Forwarding

It is a network routing technology that increases routing efficiency by taking variations in the flow of data into account. This efficiency helps avoid excessive latency and jitter during the transmission.
With the use of adaptive routing algorithms that bases the decisions on the traffic conditions between a computer and all other computers it is connected to on subnetwork. Based on this traffic flow routing makes decisions as in where to send the packet to. A flow router evaluates traffic flows in real time .because of this evaluation the rioter prioritizes the traffic and meets the requirements.

3 Implementation

3.1 Types of Packets

To refrain from repeatedly declaring packets, I created an interface type, which has all the types of packets with their respective byte values, the types where based on the assignment description which included
HELLO: a message that is used by the router to initiate a communication with the controller
FLOW_MOD: this packet will be used by the controller to notify the router about the flow table in response to which router will update its flow table and send a FLOW_MOD_REP which has utility like its name.
FEATURES_REQUEST: this packet is sent by controller to request the features (The apps and routers nearby) from the router in response to which the router sends a FEATURES_REPLY which notifies the controller about the features of the router initially it consists of BASIC_FEATURES a packet type which tells that the sender has basic features and is not near any router or app.
PACKET_IN: if the router does not have the next Hop address in its flow table

it sends a packet of this type to the controller which decides what to do with the packet discussed.
FLOW_REMOVED: a packet type sent by the controller when it wants the router to drop the packet.
All these packets are elaborated and described further in the report.

3.2 Common Functions:

As docker containers and a default port number were used for the solution, there were few complications during the implementation to get the router name while displaying its address or to get its address. Therefore, I used three HashMaps and four functions to store the names and addresses. A function called fill values to store the byte variables used in the flow table to store with container names linked with them in a HashMap, a function add to get the InetAddress of containers and then store them with their respective byte variables in a HashMap address. Listing 1 shows these functions. I understand that I could have defined a different class to perform these operations, or used a HashMap inside HashMap, but initially, they were implemented as temporary solutions but later were deemed necessary and due to time constraints, I wasn't able to change that.

3.3 Node

The node class is defined and extended by all the classes. It defines a Listener class that extends a thread class to listen on a port defined and notify and call the class OnReciept which is then handled by respective classes when a packet is received, all the node class hasn't been changed much from the first assignment it uses setType(), getType() and some other functions that are common to all the classes. The node class is kept simple and does not handle many tasks.

```
 public static HashMap<InetSocketAddress,Byte> address;
       public static HashMap<Byte,String> values;
        public static void fillvalues() {
               values.put(R1,"Rone");
               values.put(R2,"Rtwo");
               values.put(R3,"Rthree");
               values.put(R4,"Rfour");
               values.put(R5,"Rfive");
               values.put(R6,"Rsix");
               values.put(R7,"Rseven");
               values.put(R8,"Reight");
               values.put(E1,"aone");
               values.put(E2,"atwo");
               values.put(E3,"athree");
               values.put(E4,"afour");
        }
public static void loadaddress() throws UnknownHostException {
       InetSocketAddress a=add(values.get(R1));
       address.put(a,R1);
       InetSocketAddress b=add(values.get(R2));
       address.put(b,R2);
       InetSocketAddress c=add(values.get(R3));
       address.put(c,R3);
       InetSocketAddress d=add(values.get(R4));
       address.put(d,R4);
       //continuation is not shown in the listing for size reasons
public static InetSocketAddress add(String s) throws UnknownHostException {
       InetAddress dst= InetAddress.getByName(s);
InetSocketAddress dstaddress= new InetSocketAddress(dst, PORT_NUMBER);
return dstaddress;
}
```

Listing 1: The extra common functions used to store the container names with their respective addresses and variable names, add() function gets the InetAddress.

3.4 Router

When a router starts as the first task it sends a packet with type HELLO to the controller and waits for a response, this starts a communication with the controller. After receiving a packet it checks for its type and decides how it will be handled by checking the address if the address is equal to the controller address it calls a function called handle controller packet, where it checks for the type of the packet and performs tasks accordingly, for eg after the hello message it will receive a Hello Packet which the router prints and a Features request packet replying to that the router sends a packet with BASIC_FEATURE as its type. Listing 2 shows how the router handles all the packet types.

```
private synchronized void handleControllerPacket(DatagramPacket packet) throws
UnknownHostException {
             byte[] data = packet.getData();
             switch (getType(data)) {
             case HELLO:
           System.out.println("Hello packet received from controller.");
                   break;
             case FEATURES_REQUEST:
                   byte[] reply = { FEATURES_REPLY, BASIC_FEATURES };
DatagramPacket featuresReply = new DatagramPacket(reply, reply.length);
      featuresReply.setSocketAddress(controllerAddress);
                   try {
                         socket.send(featuresReply);
      System.out.println("Sent a features reply to controller.");
                   } catch (IOException e) {
                         e.printStackTrace();
                   }
                   break;
case FLOW_MOD:
System.out.println("Flow mod packet received from controller.");
                   byte[] flatTable = Arrays.copyOfRange(data, 1, data.length);
                   updateFlowtable(flatTable);
                   for (int j = 0; j < flowTable.length; j++) {
                         for (int k = 0; k < flowTable[0].length; k++) {
                   System.out.print("" + values.get(flowTable[j][k]) + " ");
                         }
                         System.out.println("");
                   }
                   packet.setSocketAddress(controllerAddress);
                   byte[] rep =packet.getData();
                   setType(rep,FLOW_MOD_REP);
                   packet.setData(rep);
                   try {
                         socket.send(packet);
System.out.println("Flow mod confirmation sent back to controller.");
                   } catch (IOException e) {
                         e.printStackTrace();
                   }
                   setAppAddress();
                   break;
             case FLOW_REMOVED:
System.out.println("Packet from end node dropped at instruction of controller.");
             }
      }
```

Listing 2: Shows how router handles packets from the controller on receiving the FLOWMOD type packet the function calls another function called update Flowtable which is discussed later, it then prints the Flowtable to the console And calls another function called setAppAddress which checks the flowtable for the apps(Endnodes) nearby.

If the address of the packet received is not controllers address the OnReceipt function calls a different function called handleApppacket which checks the

Flowtable for the nexthop and forwards the packet if the address is found, if the next hop is not found in the flowtable router send the packet to the controller to handle the packet Listing 3 shows the code for this function.

```
  private synchronized void handleAppPacket(DatagramPacket packet) throws
   UnknownHostException {
		byte[] data = packet.getData();
		if ((getType(data) == MES ||getType(data) == COM )) {
			System.out.println("Message from app received.");
			byte nextHop = checkFlowtable(data, packet.getAddress());
			if (nextHop ==0){
				System.out.println("Next hop not in flow table.");
				byte[] unrecognised = new byte[data.length];
				setType(unrecognised, PACKET_IN);
				for (int i = 1; i < unrecognised.length; i++) {
					unrecognised[i] = data[i - 1];
				}
				DatagramPacket packetIn = new
 DatagramPacket(unrecognised, unrecognised.length);
				packetIn.setSocketAddress(controllerAddress);
				try {
					socket.send(packetIn);
				} catch (IOException e) {
					e.printStackTrace();
				}
			} else {
				String Value= values.get(nextHop);
				InetAddress dst=InetAddress.getByName(Value);
 InetSocketAddress outputAddress = new InetSocketAddress(dst, PORT_NUMBER);
				packet.setSocketAddress(outputAddress);
				try {
					socket.send(packet);
					if (values.get(nextHop).contains("R")) {
						System.out.println(
								"Packet forwarded to Router "
 + nextHop + " on port " + outputAddress.getPort() + ".");
					} else {
 System.out.println("Packet forwarded to app " + nextHop + " on port "+
 outputAddress.getPort() + ".");
					}
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		}
	}
```

Listing 3: Shows how the router handles the packets from other routers or app(ENDNODES) if the nexthop is not found the router assigns a packet type PACKET_IN to the router and forward the packet to the Controller.

On receiving and updating its table another function setAppAddress is called which checks the flow table if it is near any App(ENDNODE), if the flowtable consists of any entry which has the Router and app it send the app an asynchronous packet and notifies it. Listing 4 shows the function.

```
private synchronized void setAppAddress() throws UnknownHostException {
            boolean addressSet = false;
            int i = 0;
            while (i < flowTable.length && !addressSet) {

                  if (flowTable[i][Router_INDEX]==R1 &&
(values.get(flowTable[i][INPUT_INDEX]).contains("a"))) {
                        byte portNumber = flowTable[i][INPUT_INDEX] ;
                        String value= values.get(portNumber);
                        InetAddress dst= InetAddress.getByName(value);
            this.AppAddress = new InetSocketAddress(dst, PORT_NUMBER);
                        addressSet = true;
System.out.println("This Router is connected to app " + ( value ) + ".");
                  } else if (flowTable[i][Router_INDEX]==R1
&&flowTable[i][OUTPUT_INDEX] > NUM_SWITCHES) {
                        byte portNumber = flowTable[i][OUTPUT_INDEX];
                        String value= values.get(portNumber);
                        InetAddress dst= InetAddress.getByName(value);
            this.AppAddress = new InetSocketAddress(dst, PORT_NUMBER);
                        addressSet = true;
System.out.println("This Router is connected to app " + (value) + ".");
                  }
                  i++;
            }
            if (!addressSet) {
System.out.println("This Router is not connected to an APP in the network.");
            } else {
                  byte[] data = { NODE_INITIALISE_SWITCH };
      DatagramPacket initialisation = new DatagramPacket(data, data.length);
                  initialisation.setSocketAddress(AppAddress);
                  try {
                        socket.send(initialisation);
                        System.out.println("Router req sent to the connected
App." + values.get(address.get(AppAddress)));
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
            }
      }
```

Listing 4: Shows how the router check for and sends the nearby app (ENDNODE) a packet to notify the connection.

When the router receives a packet from a router or App(EndNode) it has to check its flow table to check if it has the next hop address or not it does so by calling a defined function called checkflowtable which takes the source and destination from the packet header and the previous containers name from packet address and matches it with the entries in the flowtable to get the output container from the table Listing 5 shows the function.

```
	private synchronized byte checkFlowtable(byte[] data, InetAddress port) {
		assert (getType(data) == MES);
		byte src = getMessageSource(data);
		String src1= values.get(src);


		System.out.println("Source  = " + src1 + ".");
		byte dst = getMessageDest(data);
		String dst1= values.get(dst);

		System.out.println("Destination " + dst1 + ".");
		InetSocketAddress adde= new InetSocketAddress(port,PORT_NUMBER);

		Byte prev = address.get(adde);
		String value= values.get(prev);


		System.out.println("Previous " + value  + ".");
		int nextHop = 0;
		int i = 0;
		while (i < flowTable.length && nextHop==0) {
		/*	System.out.println("flo "+flowTable[i][SRC_INDEX]);
			System.out.println("flo2 "+flowTable[i][DST_INDEX]);
			System.out.println("fl3 "+flowTable[i][INPUT_INDEX]);*/

			if (src==flowTable[i][SRC_INDEX] &&
dst==flowTable[i][DST_INDEX] && prev==flowTable[i][INPUT_INDEX]) {
				nextHop =flowTable[i][OUTPUT_INDEX];
			}
			i++;
		}
		return (byte)nextHop;
	}
```

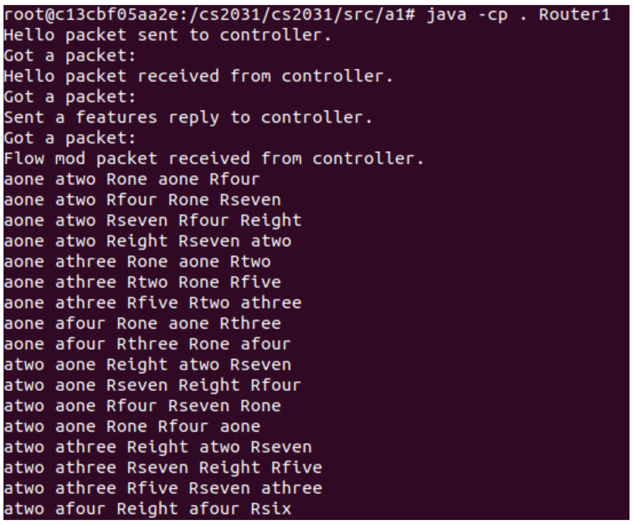
Listing 5: This function checks the flowtable for the output address by matching the flowtable with the source, previous and destination address.

When the router receives a FlowMod packet from the controller it calls a function called update flowtable to update its flow table from the flowtable received but as the datagram packet allows only a single dimension array to be stored the updateflowtable function converts the 1d array into a 2d array and stores the values in a 2d byte array. Listing 6 shows the function.

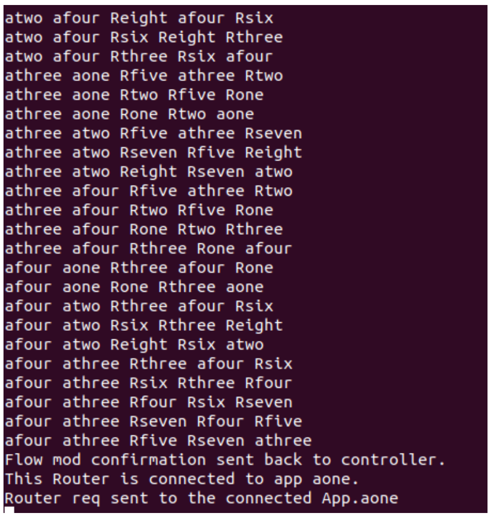
```
	private byte[][] flowTable;
	 private synchronized void updateFlowtable(byte[] flatFlowtable) {
			int rowCount;
	for (rowCount = 0; flatFlowtable[rowCount * 5] != 0; rowCount++)
				;
			flowTable = new byte[rowCount][OUTPUT_INDEX + 1];
			int i = 0;
			for (int j = 0; j < flowTable.length; j++) {
				for (int k = 0; k < flowTable[0].length; k++) {
					flowTable[j][k] = flatFlowtable[i];
					i++;
				}
			}
		}
```

Listing 6 : This function converts a 1d array to a 2d array and stores it as the routers flow table

I have 8 routers on 8 different containers to design a forwarding mechanism all the containers have their own copy of router class Figure 1 show the console of my rouuter 1 and how it behaves when initialized it also prints the flowtable received from controller.



```
root@c13cbf05aa2e:/cs2031/cs2031/src/a1# java -cp . Router1
Hello packet sent to controller.
Got a packet:
Hello packet received from controller.
Got a packet:
Sent a features reply to controller.
Got a packet:
Flow mod packet received from controller.
aone atwo Rone aone Rfour
aone atwo Rfour Rone Rseven
aone atwo Rseven Rfour Reight
aone atwo Reight Rseven atwo
aone athree Rone aone Rtwo
aone athree Rtwo Rone Rfive
aone athree Rfive Rtwo athree
aone afour Rone aone Rthree
aone afour Rthree Rone afour
atwo aone Reight atwo Rseven
atwo aone Rseven Reight Rfour
atwo aone Rfour Rseven Rone
atwo aone Rone Rfour aone
atwo athree Reight atwo Rseven
atwo athree Rseven Reight Rfive
atwo athree Rfive Rseven athree
atwo afour Reight afour Rsix
```



```
atwo afour Reight afour Rsix
atwo afour Rsix Reight Rthree
atwo afour Rthree Rsix afour
athree aone Rfive athree Rtwo
athree aone Rtwo Rfive Rone
athree aone Rone Rtwo aone
athree atwo Rfive athree Rseven
athree atwo Rseven Rfive Reight
athree atwo Reight Rseven atwo
athree afour Rfive athree Rtwo
athree afour Rtwo Rfive Rone
athree afour Rone Rtwo Rthree
athree afour Rthree Rone afour
afour aone Rthree afour Rone
afour aone Rone Rthree aone
afour atwo Rthree afour Rsix
afour atwo Rsix Rthree Reight
afour atwo Reight Rsix atwo
afour athree Rthree afour Rsix
afour athree Rsix Rthree Rfour
afour athree Rfour Rsix Rseven
afour athree Rseven Rfour Rfive
afour athree Rfive Rseven athree
Flow mod confirmation sent back to controller.
This Router is connected to app aone.
Router req sent to the connected App.aone
```

(a) (b)

Figure 1: Shows the console of Router 1 divided in to two parts (a) and (b) we can see that the router starts with sending the hello packet to the controller and then receives a hello packet and a features reply from the controller and then receives the flowtable which it prints on the console and then sends a reply back to the controller and checks for nearby nodes for the routers which are not near any app(ENDNODE) just display that on the console.

3.5 Controller

The controller is a class defined which handles all the requests from the routers it stores the Flow Table of all the routers which consist of the routing route for all the routers from a source to a destination, all the details are stored in the flow table which in my case is a 2d array. Which as column 1 as source address column 2 as destination address column 3 as routers address, column 4 as input address and column 5 as output address
Controller when started waits for a message constantly. The OnReceipt function handles the packets by checking their packet type and running a switch case for it Listing 7 shows the OnReceipt function of the controller.

```
public synchronized void onReceipt(DatagramPacket packet) {
            byte[] data = packet.getData();
            InetAddress dst= packet.getAddress();
            InetSocketAddress dstAddress= new InetSocketAddress(dst,PORT_NUMBER);
            byte port = address.get(dstAddress);
            String value=values.get(port);
            byte type = getType(data);
            switch (type) {
            case HELLO:
System.out.println("Hello packet received from Router number " + value + ".");
                   sendHello((InetSocketAddress) packet.getSocketAddress());
      sendFeaturesRequest((InetSocketAddress) packet.getSocketAddress());
                   break;
            case FEATURES_REPLY:
System.out.println("Features reply received from Router number " + value + ".");
                   if (data[1] == BASIC_FEATURES) {
      System.out.println("Router " + value + " has initial features");
                   } else {
      System.out.println("Features of Router " + value + " unknown.");
                   }
                   sendTable(dstAddress);
                   break;
      case PACKET_IN:
                   setType(data, FLOW_REMOVED);
                   packet.setData(data);
                   packet.setSocketAddress(packet.getSocketAddress());
                   try {
                          socket.send(packet);
      System.out.println("Told Router " + value + " to drop packet.");
                   } catch (IOException e) {
                          e.printStackTrace();
                   }
                   break;
            case FLOW_MOD_REP:
                   System.out.println("A flow Mod reply recieved from " + value);
                   break;
                   }
      }
```

Listing 7: This functions handles all the packets received by controller.

As shown in listing 7 when the packet of type PACKET_IN is received the controller without any check send a FLOW_REMOVED type packet to the router to drop the package which is not good but the feature was actually defined to be used when the entry to the FlowTable was supposed to be dynamic and would change whenever a new router sent hello and updated the table that was also supposed to lead to a routing algorithm being implemented to find the shortest path but as it was not implemented the checking the table again after receiving a PACKET_IN would just be redundant therefore it directly replies with a instruction to drop and also leaves the option open to implement that feature later.

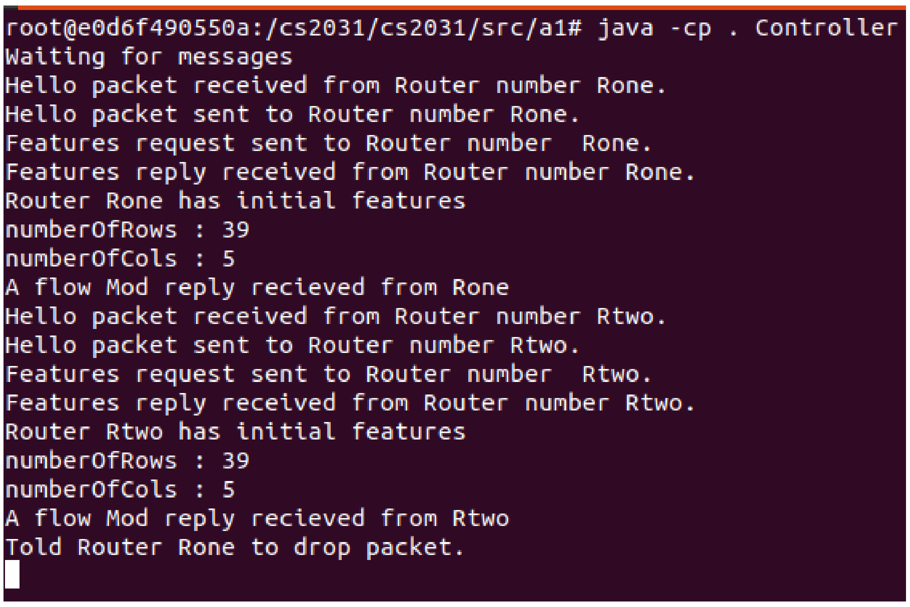
All the other function i.e sendHello(), sendFeatureRequest(), are basic function which set the packet type as HELLO or FEATURE_REQUEST and send the packet, the function that is useful and is called when the Controller sends a FLOW_MOD request to the router to direct it to update it's flowtable is the sendTable function shown in Listing 8 which converts the flowtable to a 1d array and then sends it after setting is type as FLOW_MOD. I understand that sending the whole flow table rather then just forwarding the out address to the router is much more time consuming and I would have done that if my flow table was not a 2d array.

```
private synchronized void sendTable(InetSocketAddress dstAddress) {
            Byte RouterNumber= address.get(dstAddress);
            String value=values.get(RouterNumber);

                        int  numberOfCols;
                         int numberOfRows = PRECONF_INFO.length;
                           if (numberOfRows > 0) {
                              numberOfCols = PRECONF_INFO[0].length;
                           } else {
                              numberOfCols = 0;
                           }
                          byte[] table = new byte[numberOfRows * numberOfCols];
                    System.out.println("numberOfRows : "+numberOfRows);
                    System.out.println("numberOfCols : "+numberOfCols);
            for (int row = 0, count = 0; row < numberOfRows; row++) {
                  for (int col = 0; col < numberOfCols; col++) {
                              table[count] =PRECONF_INFO [row][col];
                              count++;
                           }  }
            byte[] flowTable = new byte[table.length + 1];
       for (int i = 0; i < table.length; i++) {
                 flowTable[i + 1] = table[i];
            }
 flowTable[0] = FLOW_MOD;
 DatagramPacket packet = new DatagramPacket(flowTable,  flowTable.length);
            packet.setSocketAddress(dstAddress);
            try {
                 this.socket.send(packet);
            } catch (IOException e) {
                 e.printStackTrace();      }
       }
```

Listing 8: This function converts the flowtable to a 1d array and forward it to the router

The Controller run on the container named “con“ and is connected to the address 172.30.0.0/16 on docker Figure 2 shows the console of the controller.



```
root@e0d6f490550a:/cs2031/cs2031/src/a1# java -cp . Controller
Waiting for messages
Hello packet received from Router number Rone.
Hello packet sent to Router number Rone.
Features request sent to Router number  Rone.
Features reply received from Router number Rone.
Router Rone has initial features
numberOfRows : 39
numberOfCols : 5
A flow Mod reply recieved from Rone
Hello packet received from Router number Rtwo.
Hello packet sent to Router number Rtwo.
Features request sent to Router number  Rtwo.
Features reply received from Router number Rtwo.
Router Rtwo has initial features
numberOfRows : 39
numberOfCols : 5
A flow Mod reply recieved from Rtwo
Told Router Rone to drop packet.
```

Figure 2: shows the console of the controller it only shows the initialization of two routers , we can see that when it receives a hello packet it sends one back and sends a feature request and the whole table we can also see that in the end as the router didn’t have address of destination node controller asked to drop the packet.

3.6 APP

The EndNode in my implementation is called App. I used the same class for both the tasks and the same class runs on 4 containers resulting in 4 Apps' the user can enter if they want to send the packet or receive one by entering send or Receive, if Send is entered a function sendmessage is called which assigns the Source and destination to the packet then asks the user to enter a message and calls setmessage. Listing 9 shows the code for sendmessage function.

The function setmessage check if the message that was entered has a “#” symbol If yes it assigns the type COM(Combination) to the packet and sets the header of the packet in TLV format, splitting the message into number of “#” Listing 10 shows the code for setmessage function. If there were no “#” in the entered string, the type of packet is set to MES. Then the message is forwarded to the router.

```
private synchronized void sendMessage() {
                    String dest;
                    byte finalDst ;
                    boolean validInput = false;
                    byte[] data = new byte[PACKETSIZE];
                    setType(data, MES);
                    setSrc(data, E1);
                    do {System.out.println("Enter the reciever number ");
                           dest= sc.next();
                           switch(dest) {
                           case "2":
                                  validInput = true;
                                  finalDst = E2;
                                  setDst(data, finalDst);
                                  break;
                           case "3":
                                  validInput = true;
                                   finalDst = E3;
                                  setDst(data, finalDst);
                                  break;
                           case "4":
                                  validInput = true;
                                   finalDst = E4;
                                  setDst(data, finalDst);
                                  break;
                           case "5":
                                  validInput = true;
                                   finalDst = E5;
                                  setDst(data, finalDst);
                                  break;
                                  default:
                    System.out.println("Invalid input.");
                           }
                    } while (!validInput);
                    System.out.println("Please enter a message to send: ");
                    String stringMessage = sc.next();
                     setMessage(data, stringMessage);
            DatagramPacket message = new DatagramPacket(data,
            data.length,dstAddress,PORT_NUMBER);
                    try {
                           socket.send(message);
                           System.out.println("Message sent.");
                    } catch (IOException e) {
                           e.printStackTrace();
                    }
             }
```

Listing 9: This function takes the user input on the destination of the message and calls the function setmessage to set the header file and data in the packet and then forwards the message

```
private synchronized void setMessage(byte[] data, String message) {
                  if(message.contains("#"))
                  {
                         data[0]=COM;

 String[] parts = message.split("#", -2);
 for(int i=0;i<parts.length;i++)
 {
    System.out.println("The Combination message is "+parts[i]+" of
   length"+parts[i].length());
                  }
  data[1]= (byte)parts.length;
  int i=0;
  int c=4;
  do{

     byte[] message1=(byte[]) parts[i].getBytes();


            String str= parts[i];
            System.out.println(i+1+"="+str);
            data[c]=(byte)str.length();
            int len =data[c];
            c++;
            byte[] content = (byte[]) str.getBytes();
            for(int j = 0;j<len;j++)
            {
                  data[c+j] = content[j];
                  }

            c=c+len;
     i++;


     }while(i<data[1]);
  }
                  else {
                  data[1]=(byte)message.length();
                  byte[] content = (byte[]) message.getBytes();
                  for (int i = 0; i < content.length; i++) {
                         data[i + 4] = content[i];
                  }
                  }
                  }
```

Listing 10: This function check if the message has a “#” which divides the messages into parts, based on the message the function assigns the header of the packet in TLV format.

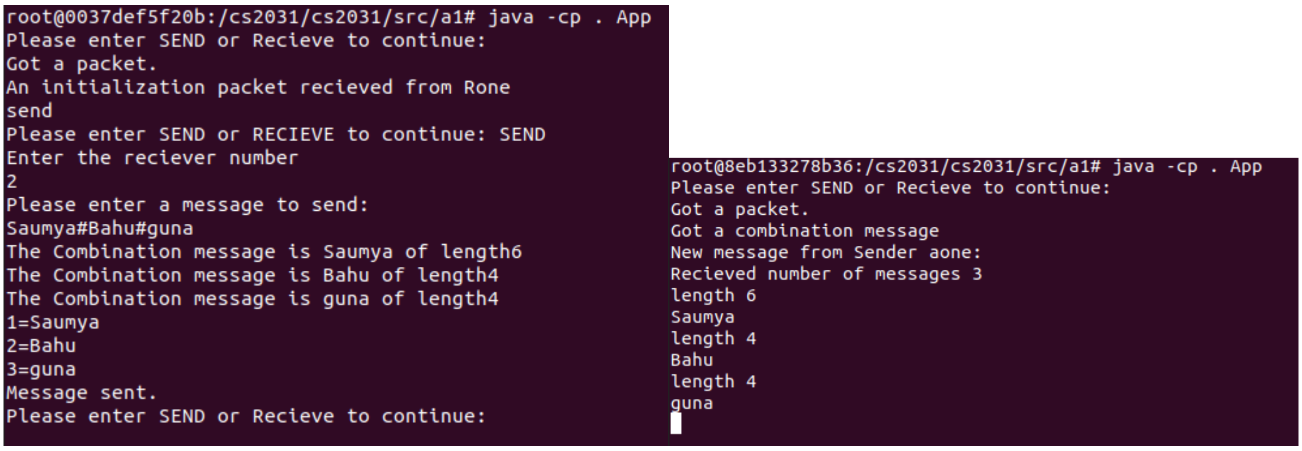
If the user enters receive the App waits for a message once a message is received the OnReceipt function notifies and checks if the message is a combination type or a simple message depending on the type it prints the message using the same technique they were stored with if the message is of type NODE_INITIALIZE_SWITCH, which is a message from the router to notify the app that it is nearby and will be sending or receiving the messages. Lisitng 11 shows the onReciept Function of the App class.

```
 public void onReceipt(DatagramPacket packet) {
                   System.out.println("Got a packet.");
                   if (getType(packet.getData()) ==MES) {
                          byte[] data = packet.getData();
                          byte src = getMessageSource(data);
                          String src1= values.get(String.valueOf(src));
                          int len=(int)data[1];
                          String message = getMessageContent(data);
      System.out.println("The length of the message is " +len);
      System.out.println("New message from Sender " + src1+ ": " + message);
                   }
                   else if(getType(packet.getData()) == COM ){
                     System.out.println("Got a combination message");

                     byte[] data = packet.getData();
                     int leng= data[1];
                          byte src = getMessageSource(data);
                                 String src1= values.get(src);
      System.out.println("New message from Sender " + src1+ ": ");
                          System.out.println("Recieved number of messages" +leng);
                          int c=4;
                          int j=0;
                          int temp=0;
                          do{ int len=data[c];
      String[] str= new String[len];
      System.out.println("length "+len);
                          for(int i=0;i<len;i++)
                          {
                          str[i]= new String(new byte[]{ (data[c+i+1])});
                                 temp+=len;}
                           c++;
                                 c=c+len;
                          for(int i=0;i<str.length;i++)
                          {
                                 System.out.print(str[i]);
                          }
                          System.out.println("");
                          j++;
                   }while(j<leng);
                   }
                   else if(getType(packet.getData())== NODE_INITIALISE_SWITCH) {
                          System.out.println("An initialization packet recieved
from "+values.get(address.get(packet.getSocketAddress())));
                   }
                   this.start();}
```

Listing11: This is Onreceipt function for App class it checks for the message and takes action based on that

As mentioned earlier I have four containers running as apps and flowtable has routes for all containers. Figure 3 shows the console of aone, a container for app class.



```
root@0037def5f20b:/cs2031/cs2031/src/a1# java -cp . App
Please enter SEND or Recieve to continue:
Got a packet.
An initialization packet recieved from Rone
send
Please enter SEND or RECIEVE to continue: SEND
Enter the reciever number
2
Please enter a message to send:
Saumya#Bahu#guna
The Combination message is Saumya of length6
The Combination message is Bahu of length4
The Combination message is guna of length4
1=Saumya
2=Bahu
3=guna
Message sent.
Please enter SEND or Recieve to continue:
```

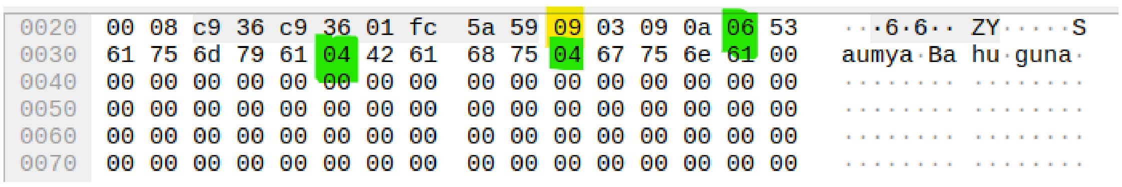
```
root@8eb133278b36:/cs2031/cs2031/src/a1# java -cp . App
Please enter SEND or Recieve to continue:
Got a packet.
Got a combination message
New message from Sender aone:
Recieved number of messages 3
length 6
Saumya
length 4
Bahu
length 4
guna
```

(a) (b)

Figure 3: shows the app classes running in two containers used for (a) sending and (b) receiving we can see that how the class divides the packets depending on the number of ‘ #’.

3.7 Captures

The whole flow forwarding protocol makes different types of transmissions from one container to another all the captures are recorded using Wireshark Figure 4 shows the header files of the packets whose transmission was recorded by Wireshark we can see that the 0th position for the header of the packet is the type (int this case 9 as in combination) the 1st position is the number of strings enclosed 4th position is the length of the 1st string and the other one is for 2nd string. All the other captures and transmission(s) PCAP files are uploaded with the Assignment.



```
0020  00 08 c9 36 c9 36 01 fc  5a 59 09 03 09 0a 06 53   ···6·6·· ZY····S
0030  61 75 6d 79 61 04 42 61  68 75 04 67 75 6e 61 00   aumya·Ba hu·guna·
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

Figure 4: - shows the Wireshark capture for the transmission done in figure 3 shows the header file of the packet yellow(0th) is type combination, 03 after yellow 1st position is the number of strings enclosed, green are the lengths of the strings in parts as we can see in the right hand side.

3.8 Protocol

All the classes and functions implemented to make a flow forwarding protocol that uses routers and a controller to send a message from on app to another. Figure 5 shows the sequence diagram for communication between the three classes.

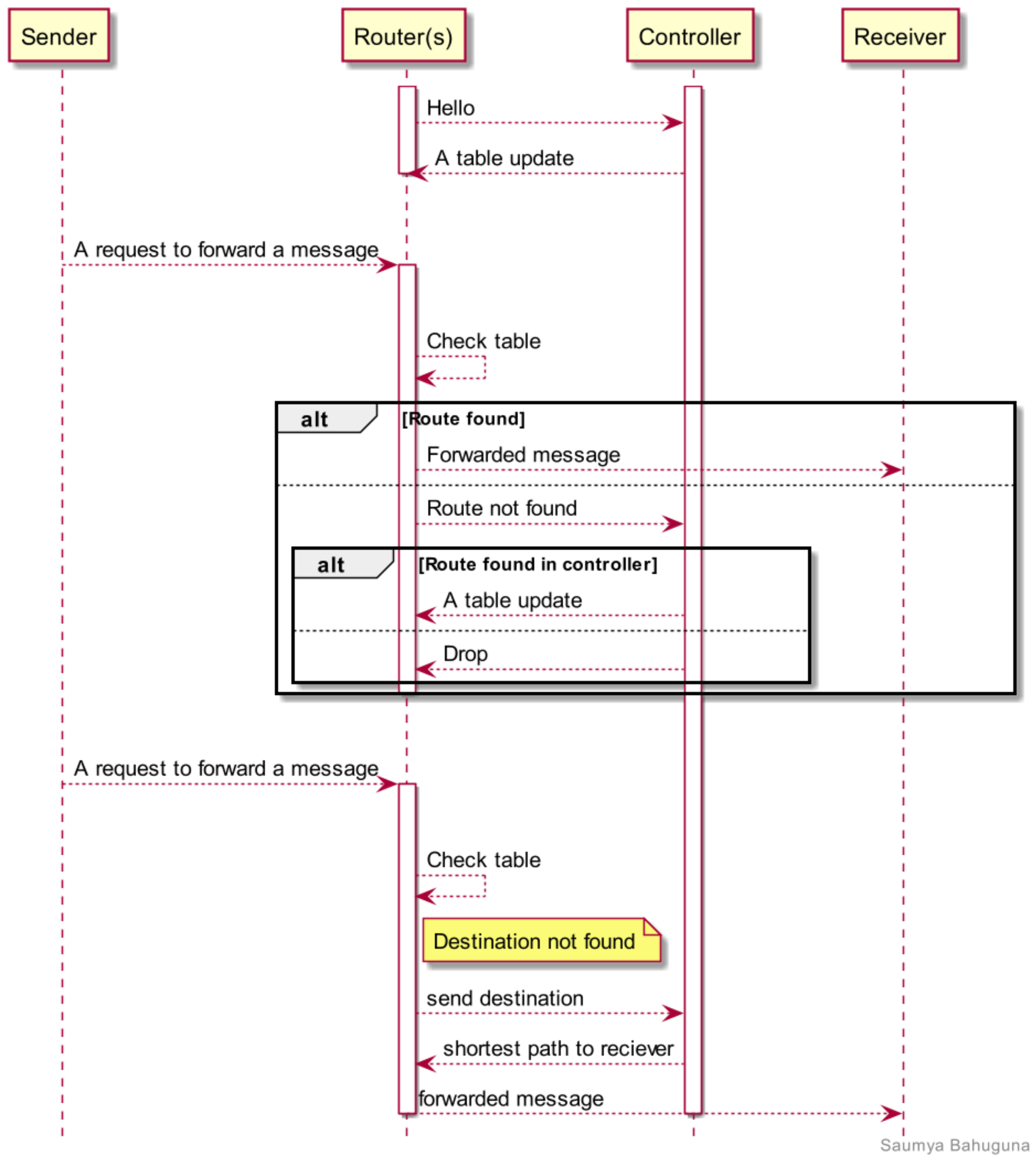


Figure 5: Shows the communication between the three classes (sender and receiver are functionality of app)

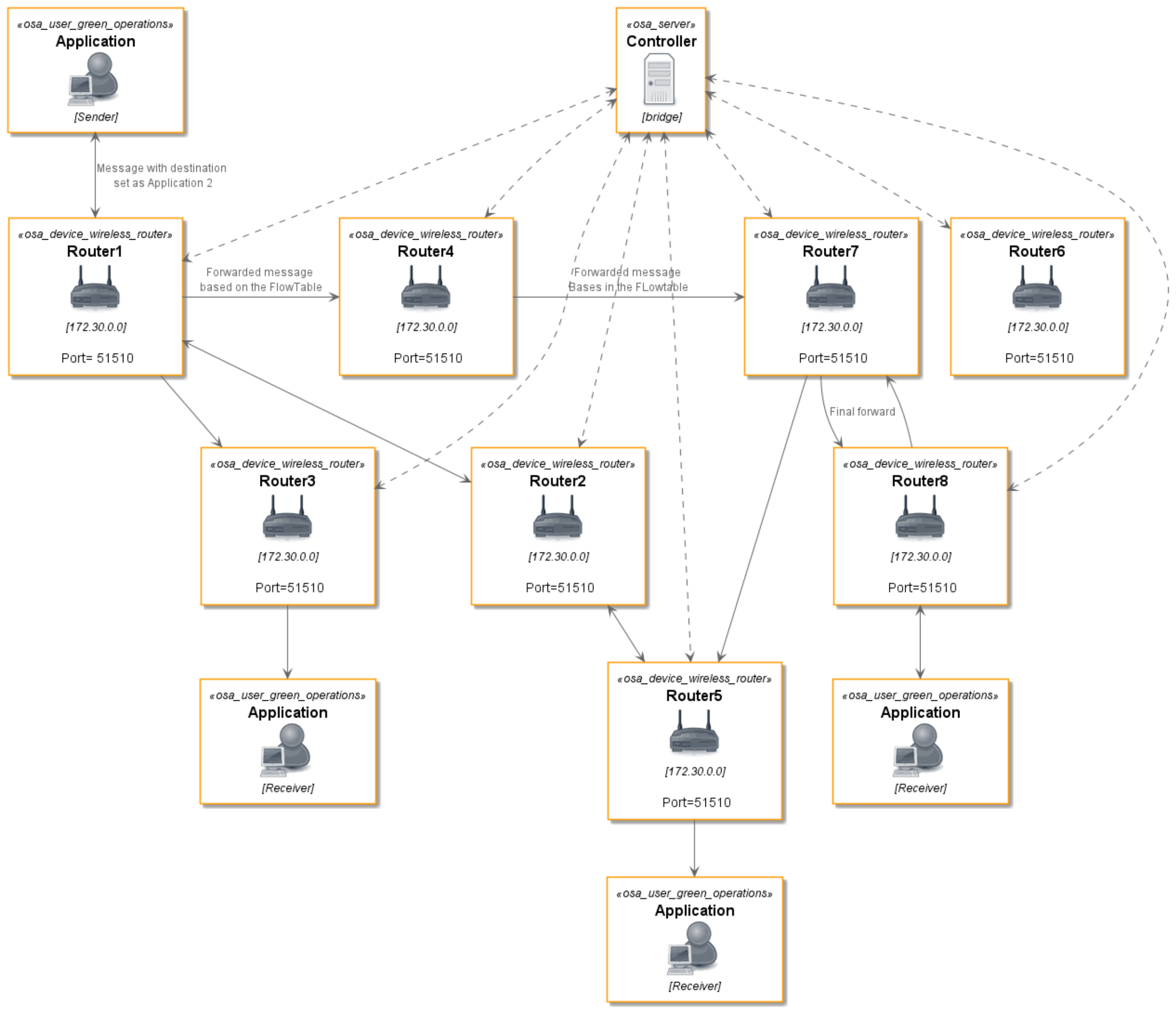

«osa_user_green_operations» Application [Sender]
«osa_server» Controller [bridge]
Message with destination set as Application 2
«osa_device_wireless_router» Router1 [172.30.0.0] Port= 51510
Forwarded message based on the FlowTable
«osa_device_wireless_router» Router4 [172.30.0.0] Port=51510
Forwarded message Bases in the FLowtable
«osa_device_wireless_router» Router7 [172.30.0.0] Port=51510
«osa_device_wireless_router» Router6 [172.30.0.0] Port=51510
Final forward
«osa_device_wireless_router» Router3 [172.30.0.0] Port=51510
«osa_device_wireless_router» Router2 [172.30.0.0] Port=51510
«osa_device_wireless_router» Router8 [172.30.0.0] Port=51510
«osa_user_green_operations» Application [Receiver]
«osa_device_wireless_router» Router5 [172.30.0.0] Port=51510
«osa_user_green_operations» Application [Receiver]
«osa_user_green_operations» Application [Receiver]

Figure 6: shows the connection of my routers(8) with the app(Endnode)(4) and the controller. Note: This is not the full routing table.

4 **Summary**

This report has described the basic solution for the assignment 2, which was my implementation of Flow forwarding protocol, which consists of an abstract Node class which extends an interface type, which is then extended by a Controller a Router and an App class. The features of the overall system can be summarized as the following:

Abstract Node Class: has common function and listener class which listens to a port

Controller: Stores all the information about the route, and communicated with the router and send them flowtable as in where to send a received packet

App: This function accepts an input from the user to send or receive a packet and sets the type of the packet based on the message.

Router: receives the packet from the app class and from other router and forwards the package based on the flowtable gets the flow table from controller.

6 Reflection:-

Upon reflection, I found Assignment 2 to be very informative as it helped in my understanding of IPv4/IPv6 and flow forwarding.
I learned a lot about using different enclosed java classes.
It also helped in my problem-solving abilities as to find the solution of minor complications faced during the implementation.
Overall, I liked how I managed to implement the protocol and was happy with my performance only thing is there are minor flaws which could easily have been handled if I managed my time.

References

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