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

CSU33031 Computer Networks
Assignment #2: Flow Forwarding

Sean Gaffney, Std# 19304695

November 30, 2021

1 Introduction

This assignment tasked us with creating a flow forwarding protocol that allows many devices to communicate with each other over multiple networks. This document describes my approach to creating my own protocol based on flow tables and packet headers in a type-length-value(TLV) format. I will explain my implementation of network nodes, including a central controller, that manages flow tables, as well as routers and endpoints that receive and transmit packets. I will explain the topology of the network I created to demonstrate this solution. My implementation is built in Java and is designed to run in Docker containers in a custom environment described in Docker Compose and shell scripts so that it can be recreated anywhere. The final product is a simple messaging application that allows two or more users to send messages to each other.

2 Theory of Topic

This section discusses the major concepts that are core to understanding how this protocol works.

2.1 UDP

User Datagram Protocol is a protocol in the Transport Layer of the OSI model that builds on top of IP. A UDP header is 8 bytes long. It has a 16-bit source port number, a 16-bit destination port number, a 16-bit total length field and a 16-bit checksum field. Port numbers are used to identify a specific process to which a message is forwarded when it arrives at its destination. The checksum field is used for error control but it is optional. Because of its short length and small number of fields, UDP is a connectionless and unreliable protocol. However its lack of flow control and optional error control can be beneficial in applications that provide these themselves.

2.2 Flow Forwarding

Flow forwarding is the process of getting a network frame to a given destination by matching its pattern/destination with an forwarding table, which defines the next step in the frames route. This differs from traditional routing which uses IP addresses to decide where to send a frame. This pattern matching is the basis for how Software defined Networks(SDNs) such as OpenFlow operate. These SDNs are usually used to manage complex networks, and use a central controller to manage and update the flow tables of routers on a network. OpenFlow offers a number of advantages over traditional routing, it is not hardware specific, it also allows for a large degree of flexibility, as routing can be periodically updated and optimized.

3 Implementation

This section describes the various aspects of my implementation of a flow forwarding protocol including the packet design, routing tables, topology, and the multiple types of nodes that make up the network.

3.1 Packet Design

All packets are built on top of the UDP protocol, using the java.net.DatagramPacket library. Packets are created with the PacketContent class, which takes in all values and encodes the data using the TLV format as a byte array. The type is encoded in the first byte, with the length of the value encoded in the second. The value is a String that makes up the rest of the TLV. The value is formed with two input String, destination and message, joined with the " : " character separating them. The variables destination and message are used in different ways depending on the packet type. For example the type is MESSAGE, the destination is the final destination of the packet. Routers will use the destination to make decisions on where to send the packet, while the message will only be read by the Application when it receives the packet. However for packet type TABLEUPDATE, described further in section 3.5, the destination field serves the same function, but the message field stores the name of the next node.

Type	Value	Description
MESSAGE	Destination, message	String message
ACK	none	Placeholder: Not fully implemented
FLOWREQ	Designation, destination	Request by router to Controller for next Node in route to destination
TABLEUPDATE	Destination, nextNode	Response from Controller
APPREQ	Designation, none	Sent by Application to Router to add Designation to flowtable

Various packet types, the data stored in the value field, and the types purpose

3.2 Flow Tables

Flow tables are stored on all Service and Router Nodes in a network. They store information about all the nodes directly connected to them in the form of key value pairs in a HashMap. In this case the key is a String that stores a given destination, and the value is the InetSocketAddress of the next Node in reaching the destination. When a Node receives a packet, they simply need to extract the destination from the TLV and match it with a key in the HashMap. If the destination is not in the flow table then the node will have to contact the controller which maintains the flow table with all connections in the network. The Controller stores its own flow table, also in the form of a HashMap. However the key difference is that its keys are combinations of Nodes and and destinations. For example "Router2:Ted", and the corresponding value is the next node to reach that destination, in this example "Endpoint1".

```
masterTable = new HashMap<String, String>();
masterTable.put("Router0:Bill", "Endpoint0");
 masterTable.put("Router0:Ted", "Router2");
```

```
        masterTable.put("Router1:Bill", "Endpoint0");
        masterTable.put("Router1:Ted", "Router0");
        masterTable.put("Router2:Bill", "Router0");
        masterTable.put("Router2:Ted", "Endpoint1");
        masterTable.put("Router3:Bill", "Router0");
        masterTable.put("Router3:Ted", "Endpoint1");
```

The example flow table is stored in the Controller, with the next Node in a route stored for each combination of Router and destination.

3.3 Endpoint: Application and Service

The Endpoint is made up of two nodes, Application and Service, that reside on the same host. Application is how a user interacts with the network, it offers a command line interface where a user can send a message to other Endpoints. It takes just two inputs, a message as a string, as well as the destination, also as a string. When a user sends a message, it is converted to a datagram packet and sent to port 51510 on the localhost where the Service is listening. The Service acts as the Applications gateway to the wider network. It sends messages received from an Application to the first Router in its flow table. It also can receive packets from routers and will pass the packet onto the Application if it matches the designation given store in its flow table

```
public void start(){
     try {
        InetSocketAddress service = new InetSocketAddress(localAddress,
DEFAULT_PORT);
        DatagramPacket appREQ = new
PacketContent(PacketContent.APPREQ,designation,"null").toDatagramPacket();
        appREQ.setSocketAddress(service);
                socket.send(appREQ);
        System.out.println("Designation sent to Service");
        Scanner input = new Scanner(System.in);

        while(true) {
           System.out.println("Enter destination: ");
           String dest = input.nextLine();
           System.out.println("Enter message: ");
           String mes = input.nextLine();
           if(dest.equals("exit") || mes.equals("exit")) {
              break;
           }
           DatagramPacket packet = new PacketContent(1,dest,mes).toDatagramPacket();
           packet.setSocketAddress(service);
           socket.send(packet);
           System.out.println("Sent");

        }
        System.out.println("Ended user input, will still receive packets until quit");
        this.wait();
```

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

The start function of the Application class. Sends its designation to the Service to be stored in the flowTable and then takes User input until the user chooses to exit.

3.4 Router

Routers act as intermediaries between Endpoints and act very similarly to Services, however they do not need to manage local connections. They also contain a flow table that maintains all other currently connected Routers and Endpoints. They wait for connections and on receipt of a packet, employ a switch statement to decide what to do based on the packet type, if a route is found it will immediately forward the packet, if not it will contact the Controller for the route. Only if no route can be found by the controller will it drop the packet.

```
public synchronized void onReceipt(DatagramPacket packet) {
        try {
            PacketContent content = new PacketContent(packet);
            int packetType = content.getType();
            String destination = content.getDestination();
            switch(packetType) {
                case PacketContent.ACK:
                    System.out.println("Recieved ACK");
                    break;
                case PacketContent.MESSAGE:
                    System.out.println("Recieved Message");
                    if(flowTable.containsKey(destination)) {
                        packet.setSocketAddress(flowTable.get(destination));
                        socket.send(packet);
                        System.out.println("Forwarded Message:" + flowTable.get(destination));
                    }
                    else {
                        DatagramPacket requestPac = new
PacketContent(PacketContent.FLOWREQ,designation,destination).toDatagramPacket();
                        requestPac.setSocketAddress(controllerAddress);
                        socket.send(requestPac);
                        temp = packet;
                        System.out.println("Route unknown, contacting Controller");
                    }
                    break;
                case PacketContent.TABLEUPDATE:
                    System.out.println(""+content.getMessage());
                    if(content.getMessage().equals("none")) {
                        System.out.println("No route found: Dropped Message");
                        break;
                    }
                    else {
                        InetSocketAddress next = new InetSocketAddress(content.getMessage(),
DEFAULT_PORT);
```

```
                    flowTable.put(content.getDestination(),next);
                    temp.setSocketAddress(next);
                    socket.send(temp);
                    System.out.println("Forwarded Message:" + content.getMessage());
                }
                break;
            default:
                System.out.println("Invalid message type");
                break;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }


}
```

The onReceipt() function of the Router, which employs a switch statement to read a packet based on its type and ignore any messages sent incorrectly. If a route is not found, the packet will be stored in the temp variable, and called again when an update is received from the Controller.

3.5 Controller

There can be only one Controller per network. The controller is connected to the default network, along with all routers. The controller contains a complete flow table with all connections between various Nodes, as described in section 3.2. This table must be manually generated for a given network, and has no way of being updated. Routers can send FLOWREQ packets to request the next step in their route, the Controller will compare the given request against its master table, and if a route is found it will respond with a TABLEUPDATE packet.

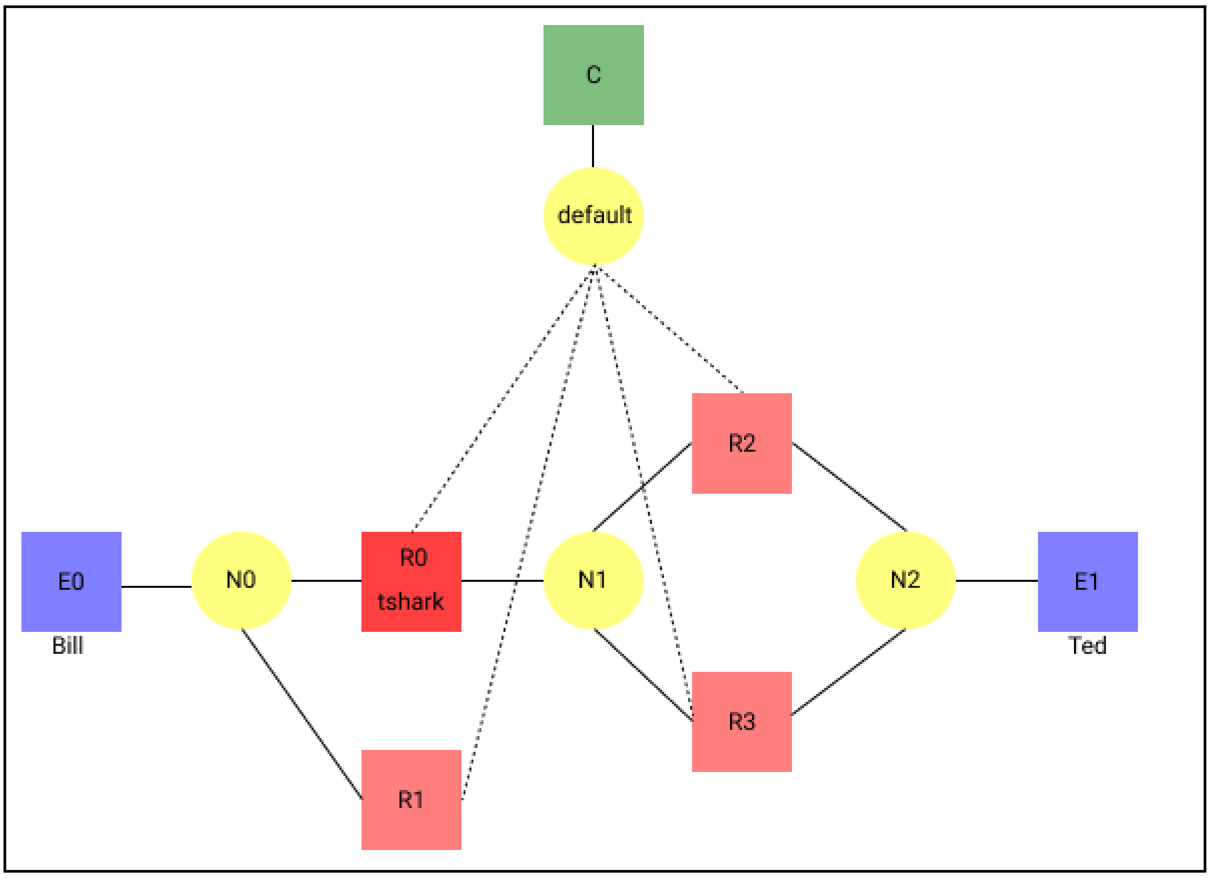
```
  PacketContent content = new PacketContent(packet);
        int packetType = content.getType();
        switch(packetType) {
            case PacketContent.ACK:
                System.out.println("Recieved ACK");
                break;
            case PacketContent.FLOWREQ:
                System.out.println("Recieved Flow Request");
                String request = content.getValue();
                System.out.println("" + request);
                String response = "none";
                if(masterTable.containsKey(request)) {
                    response = masterTable.get(request);
                    System.out.println("Route Found:" + response);
                }
                else {
                    System.out.println("Route unknown, sending Error");
                }
                DatagramPacket tableUpdate = new
```

```
PacketContent(PacketContent.TABLEUPDATE,content.getDestination(),response).toData
gramPacket();
                    tableUpdate.setSocketAddress(packet.getSocketAddress());
                    socket.send(tableUpdate);
                    break;
                default:
                    System.out.println("Invalid message type");
                    break;
            }
```

The onReceipt() function of the Controller, which employs a switch statement to read a packet based on its type and ignore any messages sent incorrectly. If a request is made by a router, the route will be checked against the master flow table, and if possible a response will be issued.

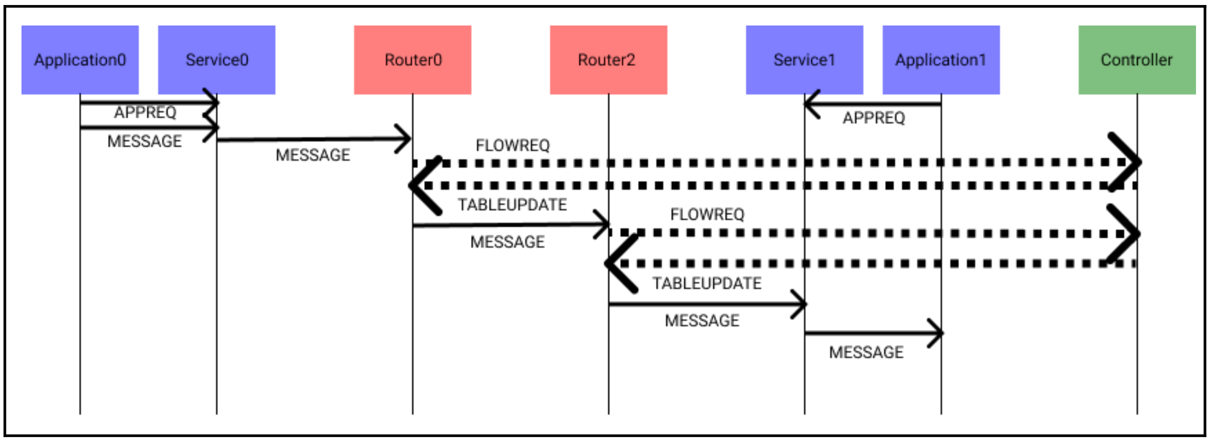
3.6 Topology

The topology of my network is defined below. However using this protocol you could create a much larger and more complex network. This topology was chosen to demonstrate every routing decision and network interaction with the fewest number of nodes, as to be as straightforward to follow and easy to run on any hardware.

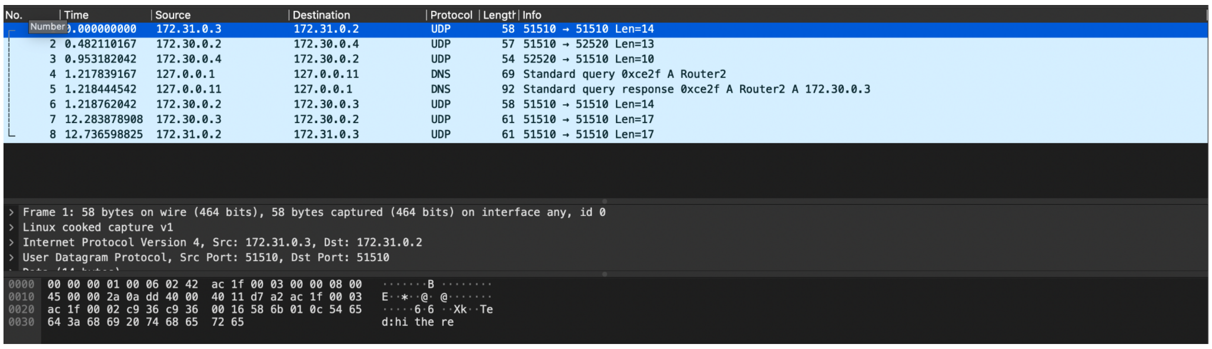


Networks are defined by yellow circles, with lines drawn to show connections.

Routers are shown in red, Endpoints in blue, and the Controller in green.
Router0(R0) is where traffic is captured



An alternate view of a subsection of the topology, that shows all the necessary steps to get a single message from one User to another.



```
No.      |Time          |Source        |Destination    |Protocol |Length|Info
Number 0.000000000   172.31.0.3    172.31.0.2     UDP       58  51510 → 51510 Len=14
     2 0.482110167   172.30.0.2    172.30.0.4     UDP       57  51510 → 52520 Len=13
     3 0.953182042   172.30.0.4    172.30.0.2     UDP       54  52520 → 51510 Len=10
     4 1.217839167   127.0.0.1     127.0.0.11     DNS       69  Standard query 0xce2f A Router2
     5 1.218444542   127.0.0.11    127.0.0.1      DNS       92  Standard query response 0xce2f A Router2 A 172.30.0.3
     6 1.218762042   172.30.0.2    172.30.0.3     UDP       58  51510 → 51510 Len=14
     7 12.283878908  172.30.0.3    172.30.0.2     UDP       61  51510 → 51510 Len=17
     8 12.736598825  172.31.0.2    172.31.0.3     UDP       61  51510 → 51510 Len=17

> Frame 1: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 172.31.0.3, Dst: 172.31.0.2
> User Datagram Protocol, Src Port: 51510, Dst Port: 51510

0000  00 00 00 01 00 06 02 42  ac 1f 00 03 00 00 08 00   ·······B ········
0010  45 00 00 2a 0a dd 40 00  40 11 d7 a2 ac 1f 00 03   E··*··@· @·······
0020  ac 1f 00 02 c9 36 c9 36  00 16 58 6b 01 0c 54 65   ·····6·6 ··Xk··Te
0030  64 3a 68 69 20 74 68 65  72 65                     d:hi the re
```

Traffic capture at Router0, showing MESSAGE received from Endpoint0, a FLOWREQ to the Controller, a TABLEUPDATE response, a separate message from Router2, that was forwarded from Endpoint1

4 Summary

This implementation of a flow forwarding protocol is extremely simple and lightweights, with only two bytes dedicated to a header and a TLV format that can be used in various different ways. Once the network and Nodes are initialized, communication between nodes is essentially instantaneous. HashTables are used to simply and efficiently create and store flow tables and a Controller is able to update Routers on the state of the network. Users can enter a name and message other users on the network. Anyone with Docker can easily create a network with the included shell scripts and docker-compose.yml file. However with minor modifications, such as replacing container names with IP addresses, this protocol could work on a real network, albeit with some limitations.

4.1 Limitations

There are some drawbacks to this implementation of flow forwarding. The first is that it would need further modification to support acknowledgements, as I deemed these unnecessary or a simulated Docker network, all Nodes can receive acknowledgements but none send them. as it is essentially impossible for packets to get dropped, and so Nodes can safely assume their packet has reached its destination. However if this protocol was further developed, add acknowledgements would be trivial. The bigger limitation is the aspects of the Network that currently must be hardcoded. Currently this includes Services, which have set behaviour that only works with the given example Application designations, Bill and Ted. The other is the Controller, whose master table is hardcoded and has no way of being updated without modifying source code. Fixing either of these issues would be larger undertakings.

6 Reflection

Despite the fact that I seriously mismanaged my time and submitted this assignment late, I am much happier with my progress than I was when writing the report for assignment one. I am satisfied with my final design and feel that I have gained significant experience with many aspects of Java, including threading and networking, as well as the nuance of header design and optimization. My skills supporting my protol implementation have also greatly improved, leveraging dockerfiles, docker-compose,shell scripting, and VSCode tools, which took a lot of the headache out of running and debugging my code. Overall, this assignment has seriously advanced my skills as a software engineer.