CS2031 ASSIGNMENT 2 – ROUTING

OVERVIEW

The current implementation of this program consists of four main classes; the Enduser, Router, Controller and a class called '*OpenFlowRouting*', which serves the "main" class of the implementation. To run the implementation the OpenFlowRouting.java program must be ran, as it creates concurrent threads for the other classes i.e., 6 Enduser threads, 10 Router threads and 1 Controller thread. The Enduser and Router threads take in respective unique id Integers which are used in the *init()* functions to determine which source port it has. In each of the Node-subclass programs the main function has been replaced by this *init()* function but it serves essentially the same purpose.

1.GENERAL IMPLEMENTATION

1.1. Buffer Layout

Throughout this implementation the information stored in the buffer has a specific layout with a few exceptions. The layout is as follows:

SOURCE + "-" + RULE + "-" + PACKETROUTE + "-" + PAYLOAD

Where SOURCE is the initial port the packet was sent from, RULE is a 1 or 0 depending on if the packet has a route, PACKETROUTE is the list of ports the packet must follow, and PAYLOAD is the user input message. Each element is separated by a "-" character(for String.split()), and the start and end of the packet route is defined by the LIST_START (:>) and LIST_END (<:) final strings respectively. The exceptions to this include the initial packet sent from the user and the packet sent from the router to the controller to request a route. It was ultimately unnecessary for them to follow this layout as the rule would always be zero and the route would be null.

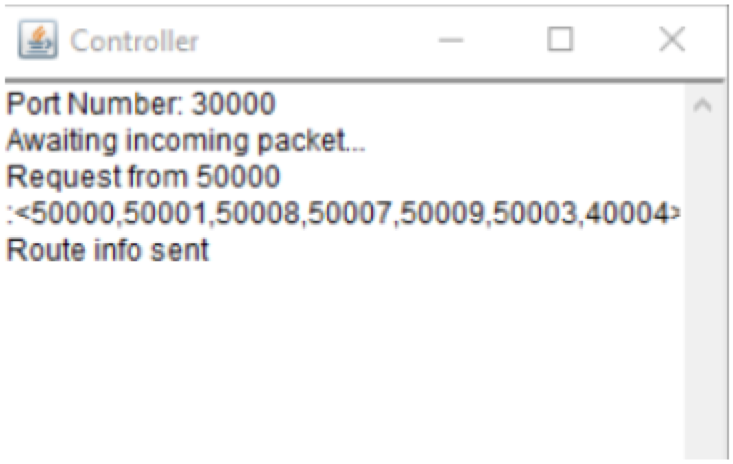
1.2. The Packet Route

- When a router receives a packet that does not have a rule, the router sends a request to the Controller containing the Id of user to which the packet is being sent.
- The Controller has a list of the user ports and the Id to which they are linked. The destination is then chosen through a switch statement and a buffer containing the route, source and rule and payload (null at this point) is created and is put in a packet.
- This packet is then sent back to the router where it is combined with the original packet which was sent from the user.
- When subsequent routers receive this packet, they process it by removing the previous router from the route and sending it on to the new first port in the route.

1.3. Generating the Route

Due to the difficulty involved in completely overhauling the system to support distance vector routing or the inclusion of Dijkstra's Algorithm, a questionable decision was made to implement a replacement for creating the route. This system would at least simulate the weight of each connection and creates a dynamic route that the packet can follow.

- The *generateRoute* method in the Controller.java class does this for us. By taking in a few ports, the program can determine which ports the packet must be sent to, and which routers the current router is connected to. The function iteratively picks a random port from an array of ports that can be travelled to in one hop.
- This array cannot contain port from the previous hop or ports belonging to an Enduser unless it is the intended destination. In the next iteration, the list is refreshed to contain the relevant ports again and this loop is broken when the destination port is found.
- Corrections have been put in place to ensure that there are no 'loops' in the generated route by cutting down the list if the same port appears twice in it.
- This method of generating the route does not fit the project specifications but it still ensures that each router is used at some stage in the execution.

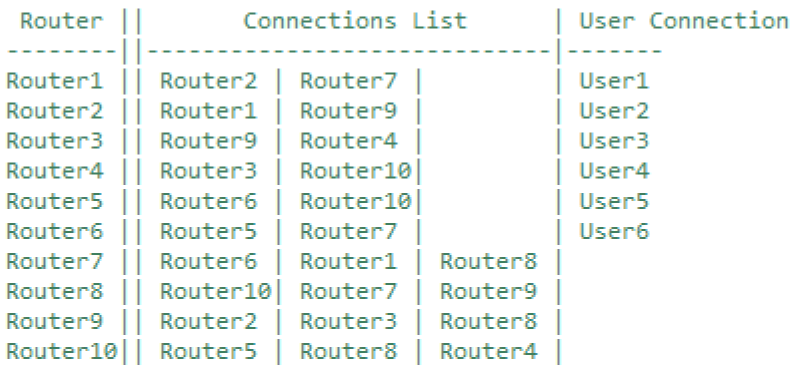


```
Port Number: 30000
Awaiting incoming packet...
Request from 50000
:<50000,50001,50008,50007,50009,50003,40004>
Route info sent
```

Figure 1: An example of a generated route from the Controller

1.4. Router Network

Since this system can support 10 routers, a table needed to be drawn up to show which routers were inter-connected. The following table is also available inside the program itself:



```
 Router ||       Connections List       | User Connection
--------||-----------------------------|-------
Router1 || Router2 | Router7 |          | User1
Router2 || Router1 | Router9 |          | User2
Router3 || Router9 | Router4 |          | User3
Router4 || Router3 | Router10|          | User4
Router5 || Router6 | Router10|          | User5
Router6 || Router5 | Router7 |          | User6
Router7 || Router6 | Router1 | Router8 |
Router8 || Router10| Router7 | Router9 |
Router9 || Router2 | Router3 | Router8 |
Router10|| Router5 | Router8 | Router4 |
```

Figure 2: List of connections for each router

Alternatively, here is an illustration of this network:

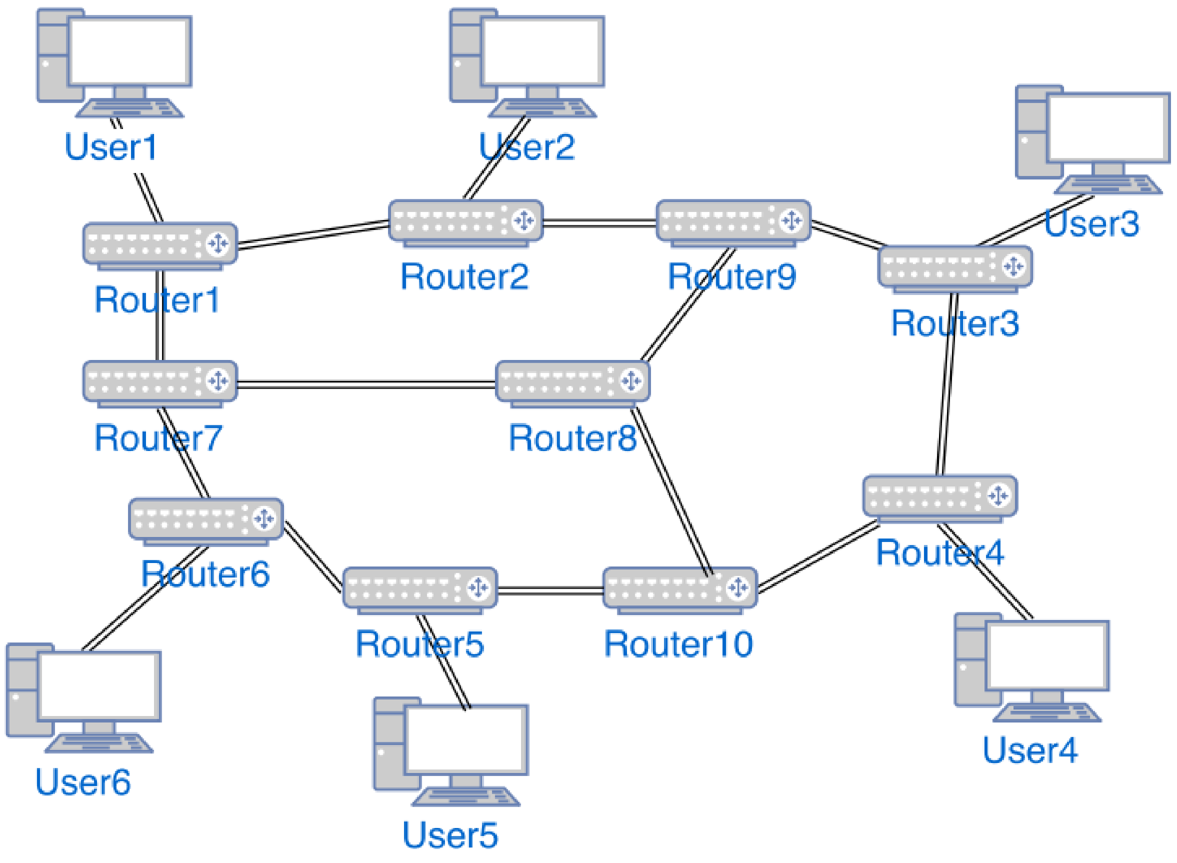


Figure 3: Visualisation of network

2.ENDUSER

2.1. Execution

- Upon execution the user is presented with 7 terminals, 6 end-users and 1 controller.
- Each end-user has its own unique Id at the top, and inside the terminal itself the user is given information such as the port number that client , and the port number of the router that it is connected to.

- This router serves as the link to the rest of the network and as such, any packet passing from the end-user must go through that router.
- Each end-user is connected to its own personal router, the list of which can be found in the *connectedRouters* array (the first user is connected to the first router, second to the second and so on).

2.2. User Input

- The user is given a prompt to type in a string which is to be sent on, upon typing this in the user is then asked which end-user they want to send it to.
- The number entered must range from 1-6, otherwise the user will receive a response with the number that they entered.
- If the correct user number is written in, a packet will be created, sent to the connected router and onwards from there.

2.3. Packet Reception

Upon receiving a packet, the Enduser will automatically convert it into a string representation and process it from there. The intended payload is the second last component of the buffer, which explains why it is split the way it is.

3. Router

3.1. Packet Reception

- When a router with receives a packet, it makes numerous checks to determine what to do with it.
- It first checks if the packet came from and router/end-user or the controller.
- If it didn't come from a controller, the router checks to see if the packet has a rule, if it has on the next port is determined and the packet sent on.
- If the packet doesn't have a rule the router will send a request to the Controller and will push the packet into a temporary list until it receives a response;
- If the packet came from the controller, the stalled packet on the temporary list will be combined with the incoming packet and sent on.

4.Functions

All of the following were added to the Node.java class:

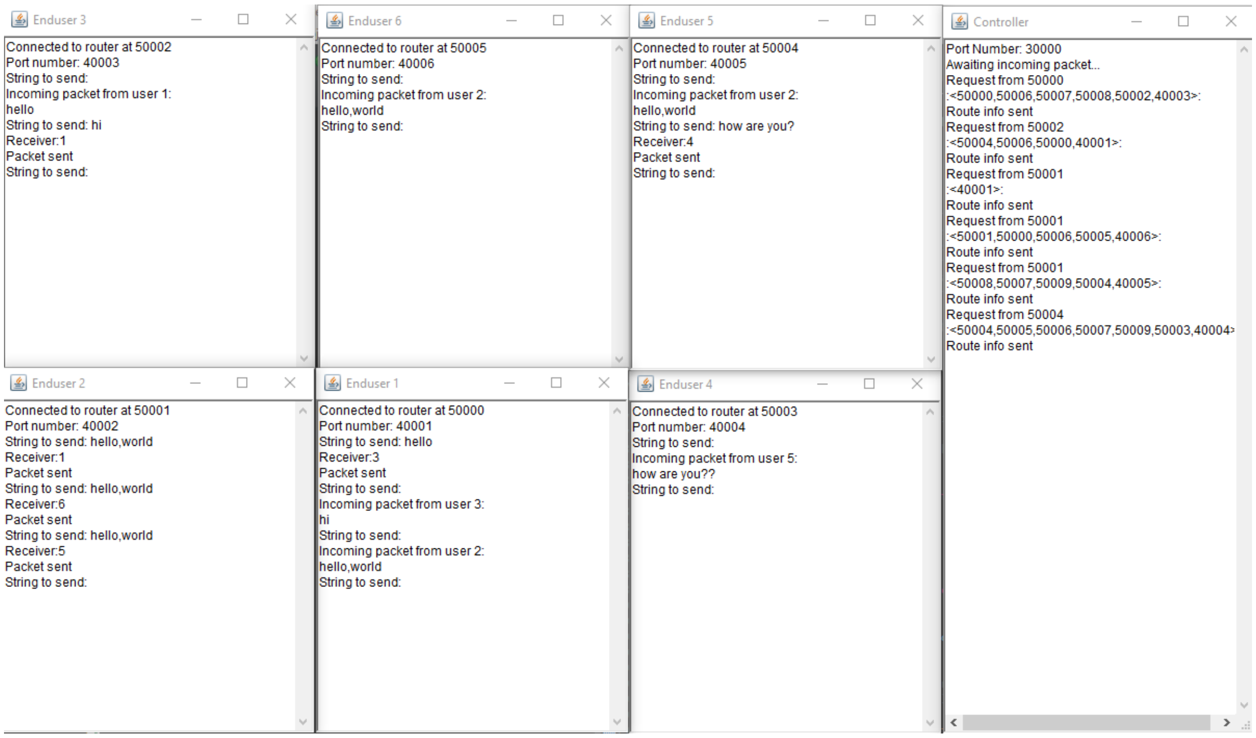
```
/*
 * Creates a buffer which is put into the packet
 *
 * @param int rule - the rule of the packet - It is
 * 1 if there exists a rule or 0 otherwise
 * @param int src - The source port
 * @param int[] path - The path which the packet will follow
 * @param String payload - The message of the packet
 *
 *  @return A string representation of the buffer of packet
 */
public static String createBuffer(int rule, int src, int[] path, String payload);

/*
 * Converts an int array to a string
 *
 * @param int[] o - the array you want to convert
 *
 * @return String representation to the array
 */
public static String toString(int[] o);

/*
 * A method which checks if a connection is valid by comparing a port with a
 * list of ports
 *
 * @param int currentPort - The port you are comparing
 *
 * @param ArrayList <Integer> portList - The list you are checking through
 *
 * @return true - if the port is an element of the list
 *
 */
public static boolean isValidConnection(int currentPort, ArrayList<Integer>
portList);

/*
 * Checks if the packet has a rule
 *
 * returns true if the value in position 6 is equal to 1
 */
public static boolean hasRule(DatagramPacket packet);
```

5. Demos / Examples



```
Connected to router at 50002
Port number: 40003
String to send:
Incoming packet from user 1:
hello
String to send: hi
Receiver:1
Packet sent
String to send:
```

```
Connected to router at 50005
Port number: 40006
String to send:
Incoming packet from user 2:
hello,world
String to send:
```

```
Connected to router at 50004
Port number: 40005
String to send:
Incoming packet from user 2:
hello,world
String to send: how are you?
Receiver:4
Packet sent
String to send:
```

```
Port Number: 30000
Awaiting incoming packet...
Request from 50000
:<50000,50006,50007,50008,50002,40003>:
Route info sent
Request from 50002
:<50004,50006,50000,40001>:
Route info sent
Request from 50001
:<40001>:
Route info sent
Request from 50001
:<50001,50000,50006,50005,40006>:
Route info sent
Request from 50001
:<50008,50007,50009,50004,40005>:
Route info sent
Request from 50004
:<50004,50005,50006,50007,50009,50003,40004>
Route info sent
```

```
Connected to router at 50001
Port number: 40002
String to send: hello,world
Receiver:1
Packet sent
String to send: hello,world
Receiver:6
Packet sent
String to send: hello,world
Receiver:5
Packet sent
String to send:
```

```
Connected to router at 50000
Port number: 40001
String to send: hello
Receiver:3
Packet sent
String to send:
Incoming packet from user 3:
hi
String to send:
Incoming packet from user 2:
hello,world
String to send:
```

```
Connected to router at 50003
Port number: 40004
String to send:
Incoming packet from user 5:
how are you??
String to send:
```

Figure 4: Demo of the program in execution.

6. Reflection

6.1. Advantages

- Design is simple and easy to understand
- Easy to use from user's point of view
- All components launch with one click

6.2. Disadvantages

- Over-reliance on string functions
- No shortest-route algorithms
- Code can be messy or badly-implemented in places e.g. The OpenFlowRouting class features the same piece of code repeatedly

6.3 What went well?

- Initial stages of development were easy to design
- Creating the buffer

6.4 What could’ve went better?

- Implementing some version of Dijkstra’s Algorithm or shortest-path