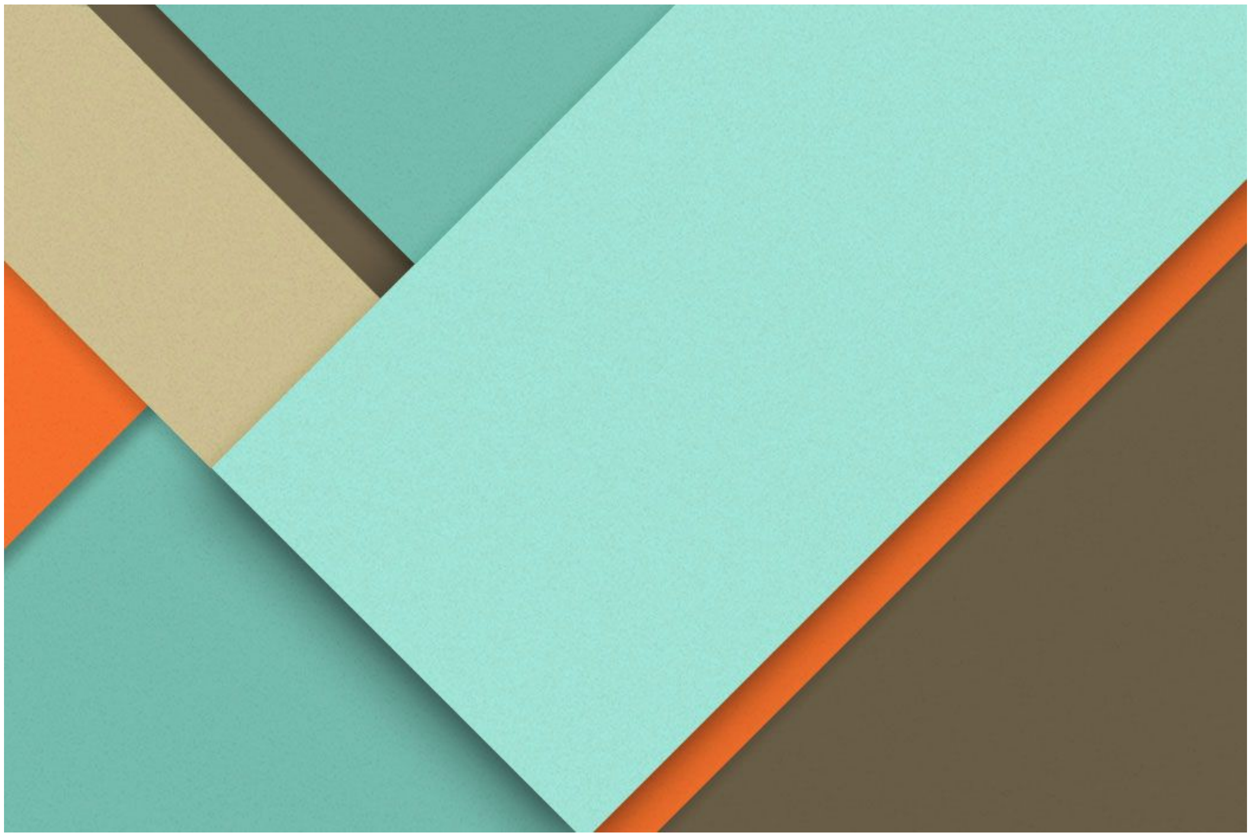


Paul Devaney | 15316723

CS2031 - Telecommunications II

Assignment II - Report

Introduction

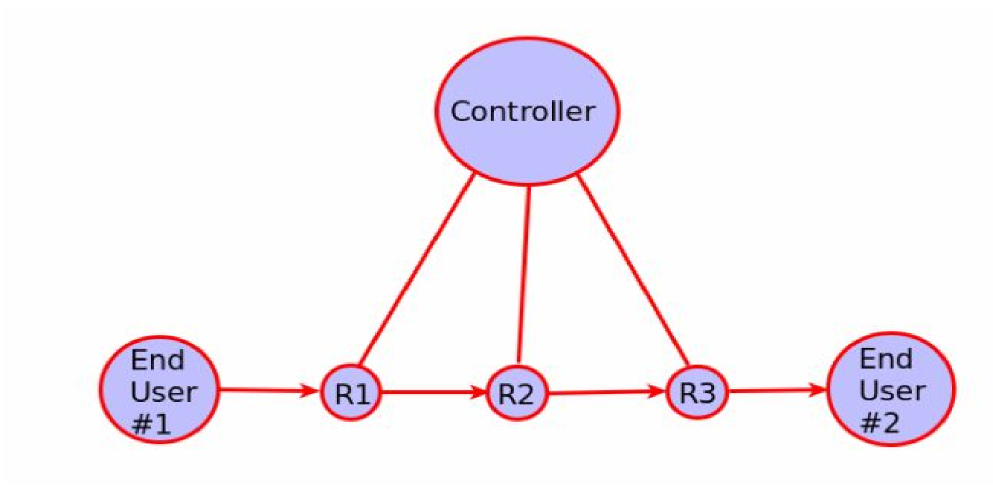
The second assignment of our CS2031-Telecommunications II module revolved around routing and protocol design. The task we were given was to develop an Openflow-like controller that configures flows in routers in a network when a packet enters the network. In the beginning, the controller was designed to have preconfigured information with regards to next hop information for each router for each possible route. Once this approach was implemented we were asked to then implement either a distance-vector or link-state routing approach to increase the efficiency of the system. Each router has a number of ports, can communicate with a subset of other routers and end-user devices and route packets based on rules that associate destinations with incoming and outgoing ports. All routers are connected to a controller and contact the controller when they receive a packet and do not have a next-hop address recorded for the destination of that packet. In order to implement a working solution to this problem, we would have to design our own protocol for each node in the communication system to implement so that they can extract and insert the correct information into each packet that is sent through the system.

The objective of this assignment was to form a more comprehensive understanding of different types of routing approaches and protocol design.

Implementation with Preconfigured information

Basic Implementation

As a starting point, I chose the basic example from the image in the assignment specification. The initial implementation would consist of two end-users, three routers and the controller. Below is a visualisation of how the system would look:

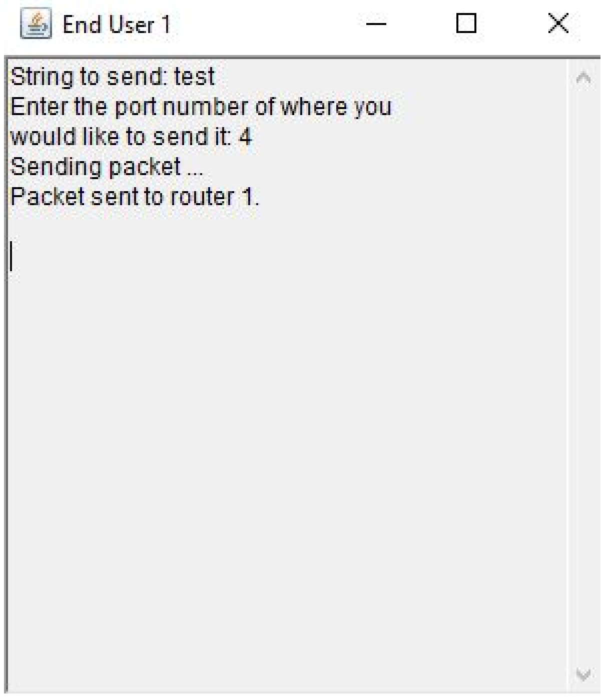


A simple explanation of this approach goes as follows: the basic implementation would simply send one single packet from ‘End User #1’ to ‘R1’. From there, ‘R1’ would check the destination of the packet, to see if it currently possessed the information required for the next hop. When ‘R1’ realised that it did not have the required information stored, it sent a request to the ‘Controller’. The controller checked the source and the destination of the packet to distinguish the route that it was hoping to take. Once it knew the route, the controller distributed the preconfigured information it had stored that route among all routers along that route. At this point each router along the route(ie. ‘R1’, ’R2’ and ‘R3’) now knew the address of the next hop for the particular destination of that packet. Subsequent packet which wanted to take this route would require no more communication between routers and the controller, as each router stored the information for future use. My first basic implementation sent one single packet in one direction. From here, the next step was to implement the ability for subsequent packets to be sent and for packets to be sent in both directions. Following that, the number of end users and the number of routers could be increased.

Moving on to the technical side of how this was implemented, we will split the explanation up into three; the operation of an end user, the operation of a router, and the operation of the controller.

1-End User

The end-user node would begin by opening a DatagramSocket on a given port to allow the sending and receiving of DatagramPackets. Whenever a packet was received by the end-user it simply had to extract the payload and convert it to a string. This string was then simply printed to the terminal. In order to send a message, the end-user would prompt a user for a message they would like to send and then the end-user they would like the message to be sent to. The end-user would then use these two pieces of information to build a DatagramPacket. To build a DatagramPacket that could be interpreted correctly by both routers and end-users it was necessary to design a protocol which they would all follow.



```
String to send: test
Enter the port number of where you
would like to send it: 4
Sending packet ...
Packet sent to router 1.
```

The protocol designed was as follows:

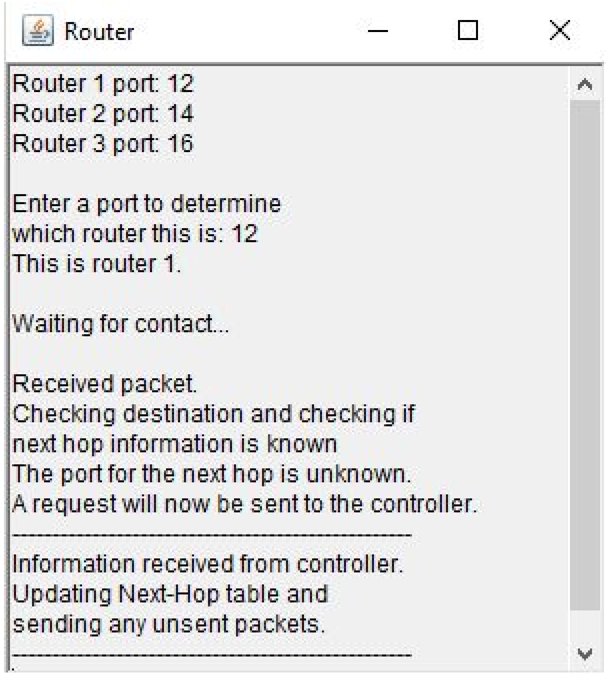
Protocol

FLAG	SOURCE	DESTINATION	NODE ID	PAYLOAD

The class would use several methods to implement this protocol. A byte array of the size of 'HEADERLENGTH' in the PacketContent class would be initialised to build the header of the packet. The message inputted by the user would be converted to a byte array and the System.arraycopy() method was used to combine the two together. The array now consisted of an empty header and the bytes of the message. The toDatagramPacket() method converted the byte array to a DatagramPacket. The setUserFlag() method took the packet and set the first index to 1, this acted as a flag to indicate to a router that a packet was sent from an end-user. The setSource() method then set the second index of the array to the source port number and the setDestination() method set the third index to the destination port number. The packet was then ready to be sent. 'R1' was the only neighbour that end-user #1 had so it sent the packet there. The end user had now completed its job, the transfer of the packet to the correct destination was now up to the routers.

2-Router

The router also began by opening up a DatagramSocket to allow the sending and receiving of messages. It also built a HashMap<Integer,Integer> to be used as a next-hop table for specific destinations. The keys of the HashMap were destination port numbers and the values were the port number of the next hop for the given destination. When a router received a packet, it first had to find out whether the packet was coming from the controller or not. The getFlag() method was used to return the first index of the byte array of tha packet. If this value was 1, the packet was coming from another router or an end user. If the value was 0, the packet was coming from the controller. When a received packet was coming from an end-user or a router, the first step it took was extracting the destination of that packet. Once it had the destination, the router referred to its next-hop table, to check if had the port of the next hop for the required destination. If it did have the correct information, it would simply set the address of the packet to port of the next hop and send it on its way. If the router did not have the necessary information, it would need to request the information from the controller. To do this the buildRequestPacket() was used. Its parameters were the source and the destination of the packet which it wanted to send on. The method then built a request packet which included the source and destination of the packet, the controller used these to get the required information and send it back. The router then received this packet and once it had checked that it was from the controller, it extracted the information and stored it in the HashMap. For any subsequent packets the controller received with the same destination the router would now know where to forward them to without any further communication with the controller.



```
Router 1 port: 12
Router 2 port: 14
Router 3 port: 16

Enter a port to determine
which router this is: 12
This is router 1.

Waiting for contact...

Received packet.
Checking destination and checking if
next hop information is known
The port for the next hop is unknown.
A request will now be sent to the controller.
--------------------------------------------------
Information received from controller.
Updating Next-Hop table and
sending any unsent packets.
--------------------------------------------------
```

A difficulty which arose here was that when a packet was received by a router and didn't know where to send it, it had to receive another packet(from the router) before forwarding on the first packet. So the DatagramPacket parameter of the onReceipt method of a router would overwrite the packet which was waiting to be forward to the next address once the onReceipt method was called for receiving an information packet from the controller. To solve this issue, a global DatagramPacket was created, called toBeSent. Any packet waiting to be sent on was stored here while the router received information from the controller. Once a router received information from the Controller the sendUnsent() method was called to send the toBeSent packet using the information the router had just received. The packet then made its way to the correct destination without any more communication with the controller as all routers along the route had been informed at the time of the first request.

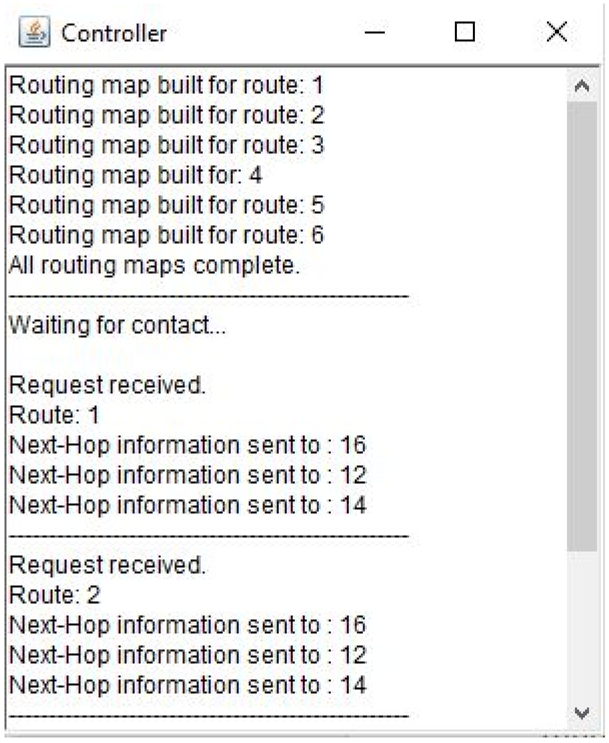
3-Controller

The controller was the source of information for all the other nodes in the system. Like the other nodes the controller first opened a DatagramSocket to allow the receiving of requests and the sending of information. The controller then built a routing table which had information on where to send a packet for each router involved in each route. This routing table was built as a nested HashMap. (ie. HashMap<Integer, HashMap<Integer, Integer>>). The keys in the outer HashMap was the route ID, each possible route had a unique ID. The values in the outer hashmap was another hashmap. For the inner HashMap, the Keys were the router for a next-hop port to be sent, and the values was the next-hop port to send. A visualisation of this structure is shown below:

KEY	VALUE	
ROUTE 1:	KEY	VALUE
	END_USER_1	ROUTER_1_PORT
	ROUTER_1_PORT	ROUTER_2_PORT
	ROUTER_2_PORT	ROUTER_3_PORT
	ROUTER_3_PORT	END_USER_2

GREEN = OUTER HASHMAP ORANGE = INNER HASHMAP

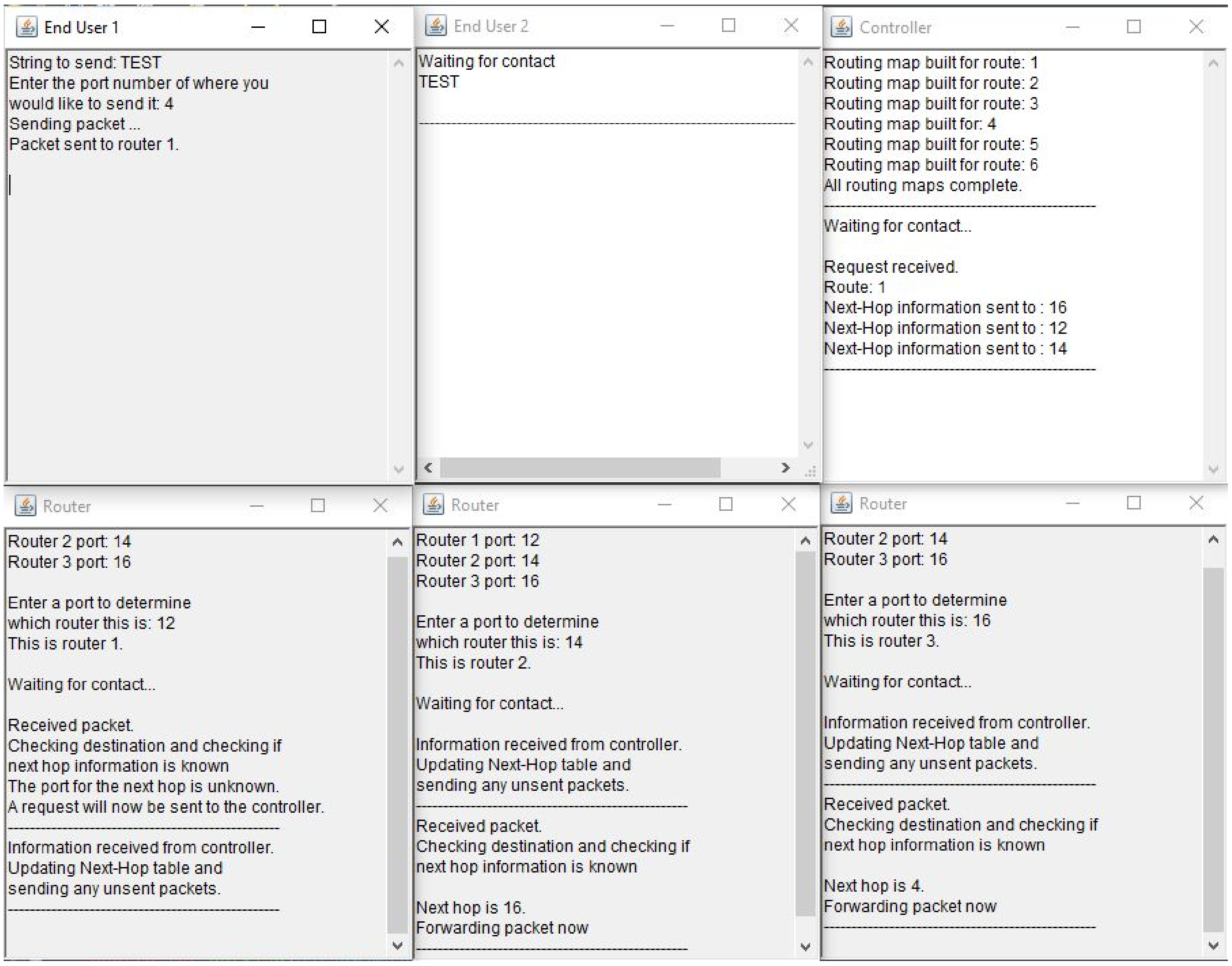
Once a request was received, the controller would distinguish what route the packet wanted to take. The getRoute() method did this by extracting the source and the destination from the header of the packet and returning the correct route ID. This route ID was used as the key for the outer HashMap to get the correct hashmap with all the next-hop information for the routers along that route. An iterator was then used to send the value integer to the key integer for each entry in this hash map. What this was doing was sending the next-hop port number to each router along the route.



```
Routing map built for route: 1
Routing map built for route: 2
Routing map built for route: 3
Routing map built for: 4
Routing map built for route: 5
Routing map built for route: 6
All routing maps complete.
------------------------------------------------
Waiting for contact...

Request received.
Route: 1
Next-Hop information sent to : 16
Next-Hop information sent to : 12
Next-Hop information sent to : 14
------------------------------------------------
Request received.
Route: 2
Next-Hop information sent to : 16
Next-Hop information sent to : 12
Next-Hop information sent to : 14
------------------------------------------------
```

This allowed packets to now be sent freely along this route as each router now knows where to forward any packets with this destination to.

A simple implementation involving just three routers and two end clients was now in place and functioning properly, with preconfigured information. Below is a snapshot of the entire system working:



```
End User 1
String to send: TEST
Enter the port number of where you
would like to send it: 4
Sending packet ...
Packet sent to router 1.
```

```
End User 2
Waiting for contact
TEST
--------------------------------------------
```

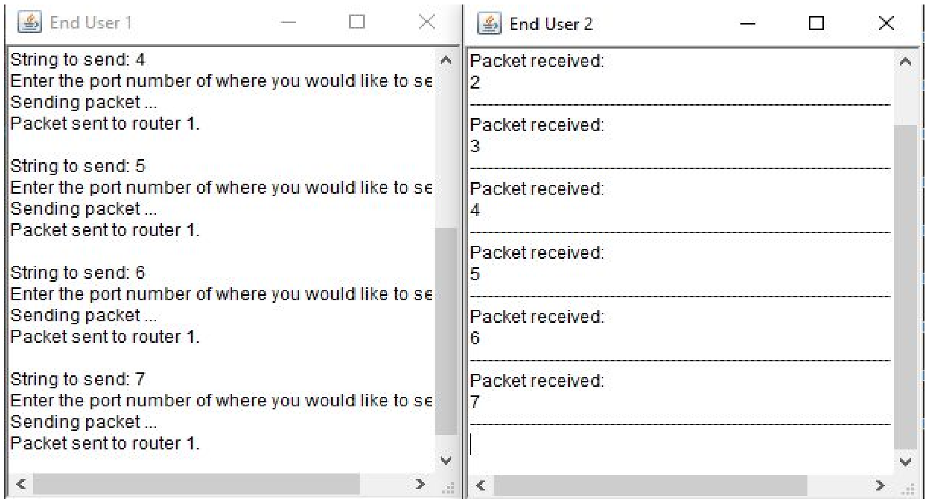
```
Controller
Routing map built for route: 1
Routing map built for route: 2
Routing map built for route: 3
Routing map built for: 4
Routing map built for route: 5
Routing map built for route: 6
All routing maps complete.
--------------------------------------------
Waiting for contact...

Request received.
Route: 1
Next-Hop information sent to : 16
Next-Hop information sent to : 12
Next-Hop information sent to : 14
--------------------------------------------
```

```
Router
Router 2 port: 14
Router 3 port: 16

Enter a port to determine
which router this is: 12
This is router 1.

Waiting for contact...

Received packet.
Checking destination and checking if
next hop information is known
The port for the next hop is unknown.
A request will now be sent to the controller.
--------------------------------------------
Information received from controller.
Updating Next-Hop table and
sending any unsent packets.
--------------------------------------------
```

```
Router
Router 1 port: 12
Router 2 port: 14
Router 3 port: 16

Enter a port to determine
which router this is: 14
This is router 2.

Waiting for contact...

Information received from controller.
Updating Next-Hop table and
sending any unsent packets.
--------------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 16.
Forwarding packet now
--------------------------------------------
```

```
Router
Router 2 port: 14
Router 3 port: 16

Enter a port to determine
which router this is: 16
This is router 3.

Waiting for contact...

Information received from controller.
Updating Next-Hop table and
sending any unsent packets.
--------------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 4.
Forwarding packet now
--------------------------------------------
```

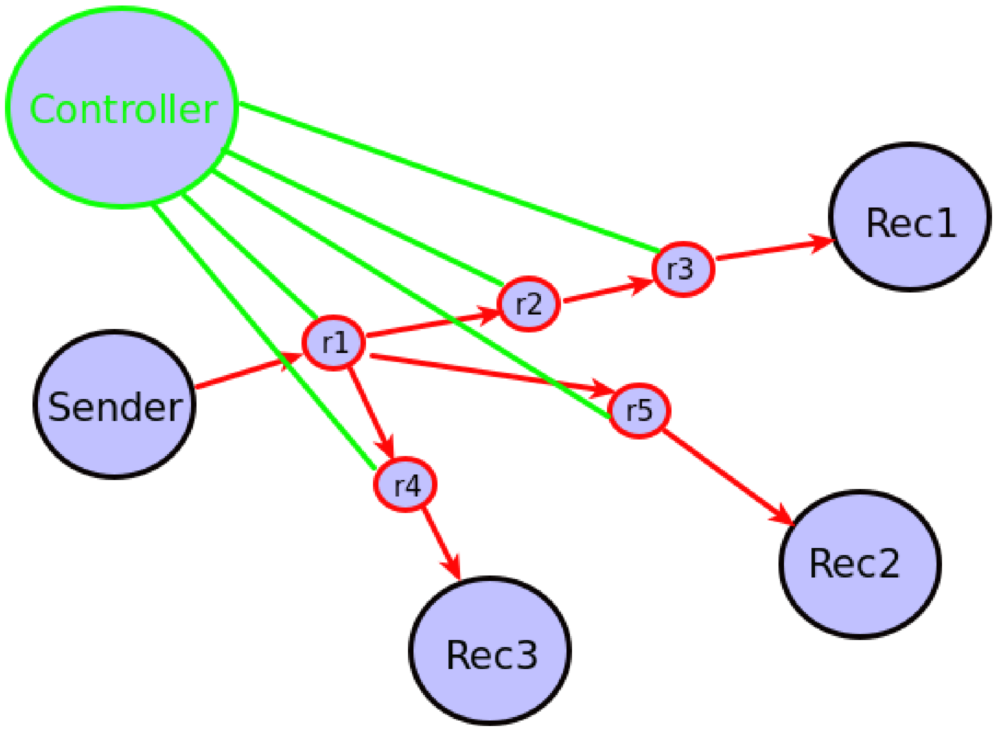
The end-user 1 node was now able to repeatedly send packets to end-user 2, without any further contact with the controller:



```
End User 1
String to send: 4
Enter the port number of where you would like to se
Sending packet ...
Packet sent to router 1.

String to send: 5
Enter the port number of where you would like to se
Sending packet ...
Packet sent to router 1.

String to send: 6
Enter the port number of where you would like to se
Sending packet ...
Packet sent to router 1.

String to send: 7
Enter the port number of where you would like to se
Sending packet ...
Packet sent to router 1.
```

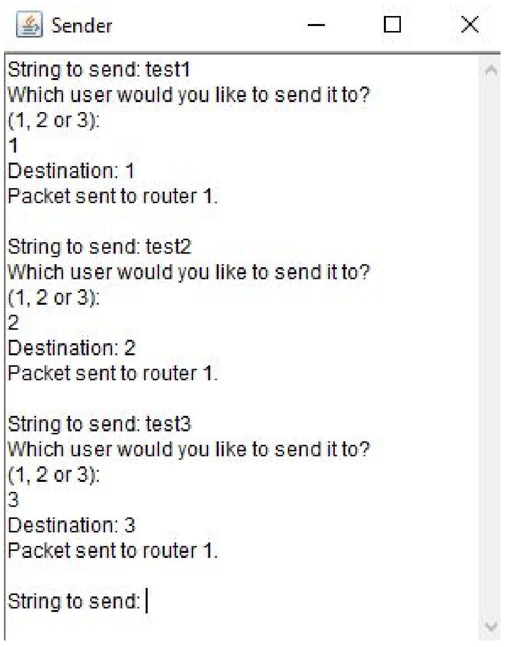
```
End User 2
Packet received:
2
--------------------------------------------
Packet received:
3
--------------------------------------------
Packet received:
4
--------------------------------------------
Packet received:
5
--------------------------------------------
Packet received:
6
--------------------------------------------
Packet received:
7
--------------------------------------------
```

To increase the complexity, I introduced two more receivers and two more routers, user-1 became the sender and the other 3 users were receivers. Below is a visualisation of my final implementation:



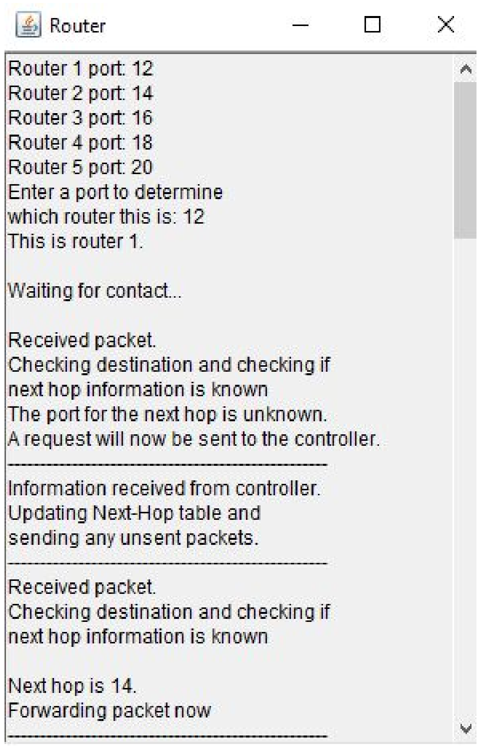
The sender could now send infinite messages to any of the the three receivers. The first time a route was to be used, a router would send a request to the controller and the controller would send out the information regarding that route. That route then became known by all the routers concerned. Below are some screenshots on on how each node functioned by the end.

Sender



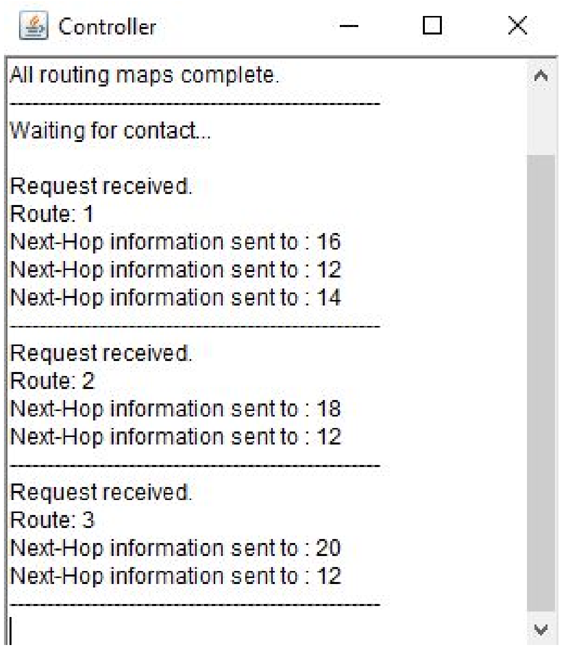
```
String to send: test1
Which user would you like to send it to?
(1, 2 or 3):
1
Destination: 1
Packet sent to router 1.

String to send: test2
Which user would you like to send it to?
(1, 2 or 3):
2
Destination: 2
Packet sent to router 1.

String to send: test3
Which user would you like to send it to?
(1, 2 or 3):
3
Destination: 3
Packet sent to router 1.

String to send:
```

Router



```
Router 1 port: 12
Router 2 port: 14
Router 3 port: 16
Router 4 port: 18
Router 5 port: 20
Enter a port to determine
which router this is: 12
This is router 1.

Waiting for contact...

Received packet.
Checking destination and checking if
next hop information is known
The port for the next hop is unknown.
A request will now be sent to the controller.
------------------------------------------------
Information received from controller.
Updating Next-Hop table and
sending any unsent packets.
------------------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 14.
Forwarding packet now
------------------------------------------------
```

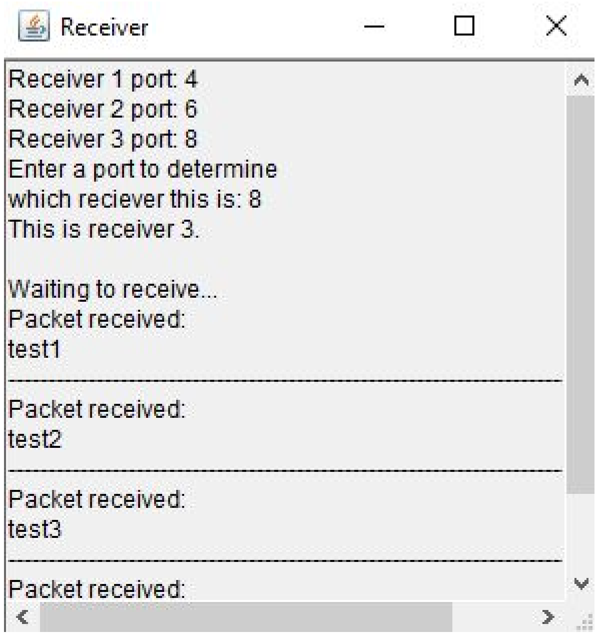
Controller



Controller

```
All routing maps complete.
---------------------------------------------
Waiting for contact...

Request received.
Route: 1
Next-Hop information sent to : 16
Next-Hop information sent to : 12
Next-Hop information sent to : 14
---------------------------------------------
Request received.
Route: 2
Next-Hop information sent to : 18
Next-Hop information sent to : 12
---------------------------------------------
Request received.
Route: 3
Next-Hop information sent to : 20
Next-Hop information sent to : 12
---------------------------------------------
```

Receiver

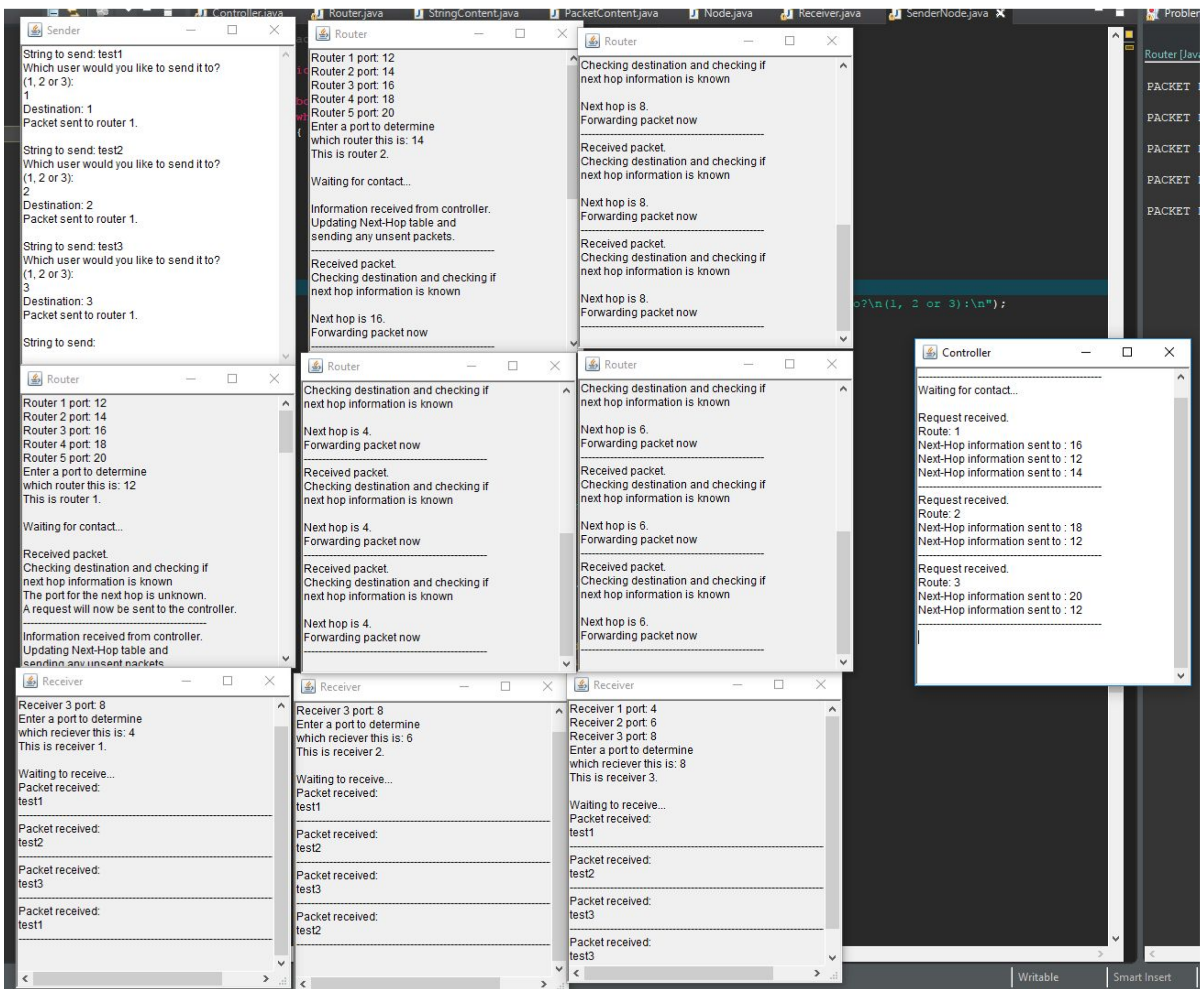


Receiver

```
Receiver 1 port: 4
Receiver 2 port: 6
Receiver 3 port: 8
Enter a port to determine
which reciever this is: 8
This is receiver 3.

Waiting to receive...
Packet received:
test1
------------------------------------------------------------
Packet received:
test2
------------------------------------------------------------
Packet received:
test3
------------------------------------------------------------
Packet received:
```

Conclusion

In conclusion, the implementation I designed worked flawlessly with preconfigured information. Any amount of routers and receivers could now be added, as long as the controller has the information for the route, the routers would be able to get a packet to the right destination. On the next page there is a screenshot of the entire program in operation. In total, between research, implementation and the report I spent approximately 18 hours doing this assignment. I learned a huge amount about routing and protocol design in the process. This is also the largest program I have written to date, so I gained some valuable experience in writing larger programs and finding and fixing problems in large chunks of code. I also formed a huge interest in this field of computer science from completing this assignment.



Sender
```
String to send: test1
Which user would you like to send it to?
(1, 2 or 3):
1
Destination: 1
Packet sent to router 1.

String to send: test2
Which user would you like to send it to?
(1, 2 or 3):
2
Destination: 2
Packet sent to router 1.

String to send: test3
Which user would you like to send it to?
(1, 2 or 3):
3
Destination: 3
Packet sent to router 1.

String to send:
```

Router
```
Router 1 port: 12
Router 2 port: 14
Router 3 port: 16
Router 4 port: 18
Router 5 port: 20
Enter a port to determine
which router this is: 14
This is router 2.

Waiting for contact...

Information received from controller.
Updating Next-Hop table and
sending any unsent packets.
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 16.
Forwarding packet now
----------------------------------------
```

Router
```
Checking destination and checking if
next hop information is known

Next hop is 8.
Forwarding packet now
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 8.
Forwarding packet now
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 8.
Forwarding packet now
----------------------------------------
```

Router
```
Router 1 port: 12
Router 2 port: 14
Router 3 port: 16
Router 4 port: 18
Router 5 port: 20
Enter a port to determine
which router this is: 12
This is router 1.

Waiting for contact...

Received packet.
Checking destination and checking if
next hop information is known
The port for the next hop is unknown.
A request will now be sent to the controller.
----------------------------------------
Information received from controller.
Updating Next-Hop table and
sending any unsent packets
```

Router
```
Checking destination and checking if
next hop information is known

Next hop is 4.
Forwarding packet now
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 4.
Forwarding packet now
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 4.
Forwarding packet now
----------------------------------------
```

Router
```
Checking destination and checking if
next hop information is known

Next hop is 6.
Forwarding packet now
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 6.
Forwarding packet now
----------------------------------------
Received packet.
Checking destination and checking if
next hop information is known

Next hop is 6.
Forwarding packet now
----------------------------------------
```

Controller
```
----------------------------------------
Waiting for contact...

Request received.
Route: 1
Next-Hop information sent to : 16
Next-Hop information sent to : 12
Next-Hop information sent to : 14
----------------------------------------
Request received.
Route: 2
Next-Hop information sent to : 18
Next-Hop information sent to : 12
----------------------------------------
Request received.
Route: 3
Next-Hop information sent to : 20
Next-Hop information sent to : 12
----------------------------------------
```

Receiver
```
Receiver 3 port: 8
Enter a port to determine
which reciever this is: 4
This is receiver 1.

Waiting to receive...
Packet received:
test1
----------------------------------------
Packet received:
test2
----------------------------------------
Packet received:
test3
----------------------------------------
Packet received:
test1
----------------------------------------
```

Receiver
```
Receiver 3 port: 8
Enter a port to determine
which reciever this is: 6
This is receiver 2.

Waiting to receive...
Packet received:
test1
----------------------------------------
Packet received:
test2
----------------------------------------
Packet received:
test3
----------------------------------------
Packet received:
test2
----------------------------------------
```

Receiver
```
Receiver 1 port: 4
Receiver 2 port: 6
Receiver 3 port: 8
Enter a port to determine
which reciever this is: 8
This is receiver 3.

Waiting to receive...
Packet received:
test1
----------------------------------------
Packet received:
test2
----------------------------------------
Packet received:
test3
----------------------------------------
Packet received:
test3
```