CS2031 Telecommunication II

Assignment #2: OpenFlow

Kai Suzuki, 18308704
29th November, 2019

Content

1.Introduction

The task for this assignment is to create and implement protocols of the controller and routers for one end-user can interact with the other end-user through those routers. To achieve this task, this report will discuss the design in the design section, the approach for implementing this protocol in the implementation section, the way of running my program in the Demo section and my reflection in the conclusion section.

2.Design

In this section, the guideline of my design to implement protocols for this assignment will be discussed. Besides, at the end of this section, the extended version of this assignment will be introduced.

First of all, my program's design assumes one controller, three routers and two end-users for the simplicity of open flow for this assignment. For the interaction between the end-users, the message from each of them is sent to the others through those routers. From the end-user1 to the end-user2, the message goes through router1, router2 and router3 in order. Likewise, the message goes through router3, router2 and router1 in order from the end- user2 to the end-user1 (figure 1).

Figure1: the route between the end-user1 and the end-user2

Also, as it is mentioned in the description of the assignment, each router does not know the destination address to send the message from other nodes, at first, and has to send a message to the controller to know the next destination address. For example, the router1 does not know where to send the message from the end-user1 after just starting this program. Therefore, the router1 asks the controller to know the destination to send a message (figure 2). Then the router has already the best route to bring the message from the end-user1 to the end-user2 and send the next destination address to all routers (figure 3). Finally, each router knows and records the destination sent by the controller and sends a message to the correct destination (figure 4). After this process, the router 1 does not need to contact with the controller to send the subsequent message from the end-user1 to the end-user2.

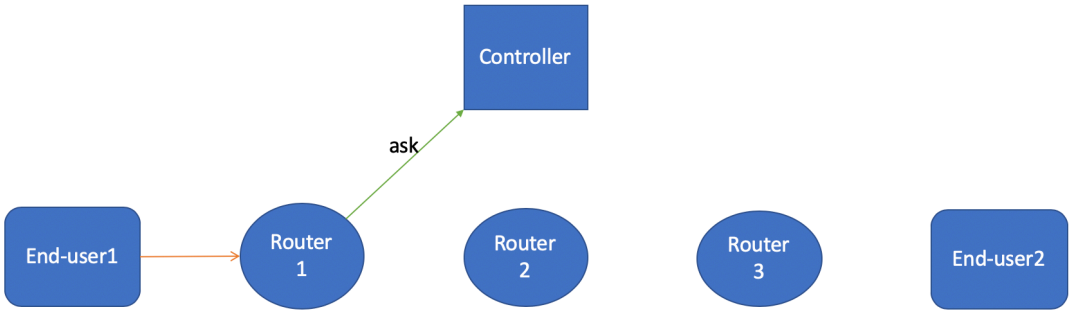

Figure 2: the router1 received the message from the end-user1 and asks the controller about the next destination address

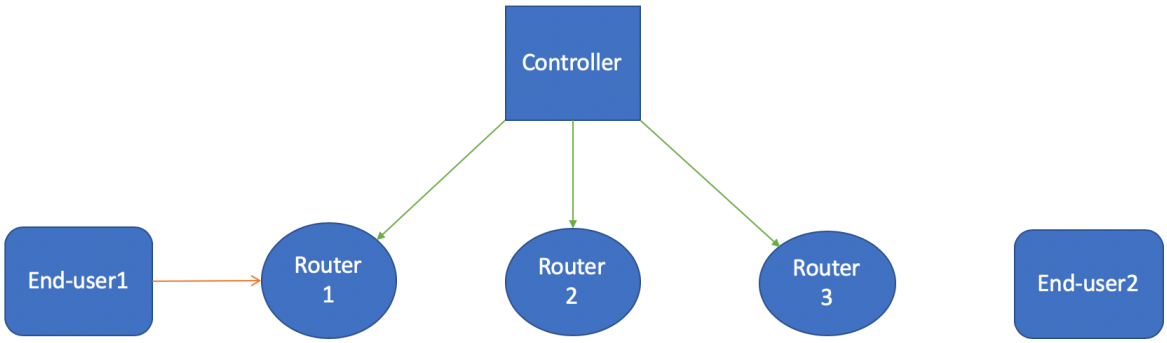

Figure 3: the controller distributes the next destination address to all of the routers, depending on the best route the controller already has

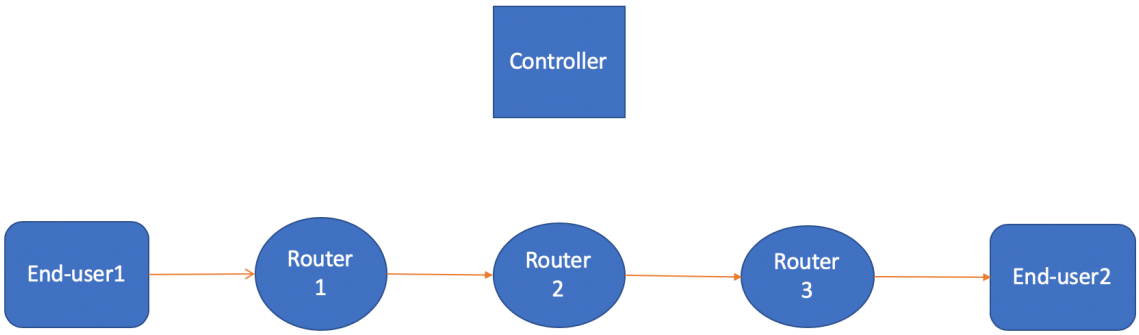

Figure 4: each router knows the next destination address to send a message from the end-user 1 to the end-user 2

For the extension version

The above example and my program assume the controller has the preconfigured and the best route to send a message from one end-user to the other. However, the controller need not have that if either Link State or Distance Vector Routing is implemented.

Link State Routing:

Each node including routers is likely to broadcast its routing table to its neighbor nodes. Also, it periodically runs an algorithm such as Dijkstra's shortest path to find the shortest path to send a message by the use of appropriate nodes.

Distance Vector Routing:

Each node periodically creates and updates its own routing table and broadcast it to its neighbor nodes. When a node receives new information on the routing table from its neighbor route, it records the new information and adds it to its own routing table.

3.Implementation

This section discusses the explanation of my actual approach to implement this protocol from controller, end-user to router after describing the common functions which are used in various classes.

My program has Controller, EndUser1, EndUser2, Router1, Router2, Router3, Node, StringContent classes and PacketContent interface. The last two classes and the interface supports the main classes: Controller, EndUser1, EndUser2, Router1, Router2 and Router3. Also, the description of this assignment mentions the "*FeatureRequest*" which is a certain feature by sending a message to the controller and receiving a reply from it. In my case, the "*FeatureRequest*" is to know whether the routers have already known the next destination address to send or not when the message arrives at certain end-user. The method of this "*FeatureRequest*" is checking if the message arrived at the destination contains "hellohello". If it has, it means the message is sent by asking the controller about the next destination address.

Common Function

`checkInteger:`

This function is seen in Controller, EndUser1, EndUser2, Router1, Router2 and Router3 classes. This can extract the integer in the String sent as a parameter if it contains an integer.

`checkDest:`
This function is seen in Controller, EndUser1, EndUser2, Router1, Router2 and Router3 classes. This can extract the String such as “E1” or “E2” in the String sent as a parameter if it contains those Strings.

`sendACK:`
This function is seen in EndUser1, EndUser2, Router1, Router2 and Router3 classes. This can send an ACK to the given packet with a String as a reply message.

`sendPacket:`
This function is seen in Controller, Router1, Router2 and Router3 classes. This can send a packet to the designated address with a source address and a content message. Also, in this function, there are small differences between the Controller class and those routers classes for my “*FeatureRequest*”.

`onReceipt:`
This function is seen in the Controller, Router1, Router2 and Router3 classes. However, the role of this in Controller class differs from that in each router class. This function in Controller receives a message from one of the routers and causes this class to give away the appropriate destination address to each router. On the other hand, the function in each router can receive a message from other routers and send it to the next destination; at the same time, if it receives new information about next destination address from the controller, it records the information and adds it to its own routing table.

Controller
`createTable:`
This function creates the table which indicates the best route from one end-user to the other. This table is used for distributing the next destination address which is suited to each router.

`foundRoute:`
This function can find the route from the routing table, matching the destination address, the source address and current router address with given information. If the route is not found the controller cannot send a message containing the route to each router. Thus, the message will not be sent.

End-User (both EndUser1 and EndUser2)

`start:`
Both EndUser1 and EndUser2 do not have the "`sendPacket`" function because they will send a packet in the "`start`" function instantly after starting this class.

In addition, both classes have "counter" variable which counts the times of sending a packet by both end-users. They can send a packet twice each. Firstly, the EndUser1 can send a packet twice, then the EndUser2 can send twice.

Router (Router1, Router2 and Router3)
The router's concept or implementation is supposed to be one of the most important in terms of this assignment. Thus, there are a number of specific features in these classes. Each router has to have the routing table depending on the information from the controller. In my program, routing table is expressed as the "`ArrayList<ArrayList<String>> destList.`" Hence the information from the controller will be added to the temporary arraylist, "`<ArrayList<String>`", containing several String messages, then the temporary arraylist will be added to another arraylist, `ArrayList<ArrayList<String>> destList.`
In addition to this, there are two functions which appear in only these router classes.

`destIsInList:`
This function can check whether the destination written in the packet sent from other routers or end-users matches the destination in its own router's routing table. If it matches, this function returns the integer of the destination address; otherwise, it returns -1 and the attempt to send a message will fail.

`foundNextDest:`
This function can change the number of the address in String as it in integer if the address number is one of the node's port number: 50001, 50002, 50010, 50011 or 50012 in my case.

4.Demo

In this section, the simple steps to run my program will be explained.
Firstly, we have to run Controller, EndUser1, EndUser2, Router1, Router2 and Router3 (Figure 5).

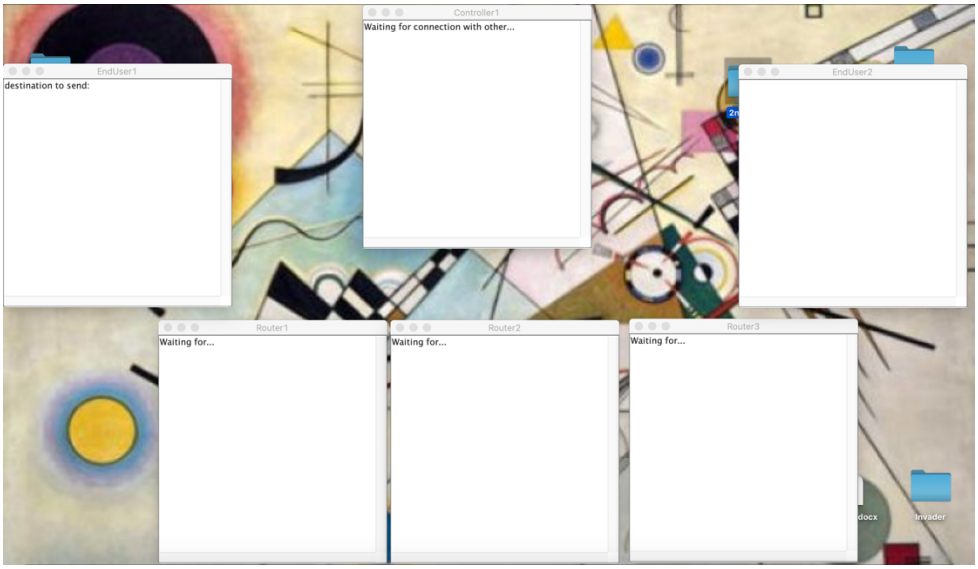

Figure 5: running Controller, EndUser1, EndUser2, Router1, Router2 and Router3

Next, we have to enter “Hello” E2” into the terminal of the Enduser1. After entering that, it automatically sends the message to the EndUser2 through all of the routers and the controller. As is mentioned before, my “*FeatureRequest*” distinguishes whether the use of the route the message went through is the first time or not (Figure 6).

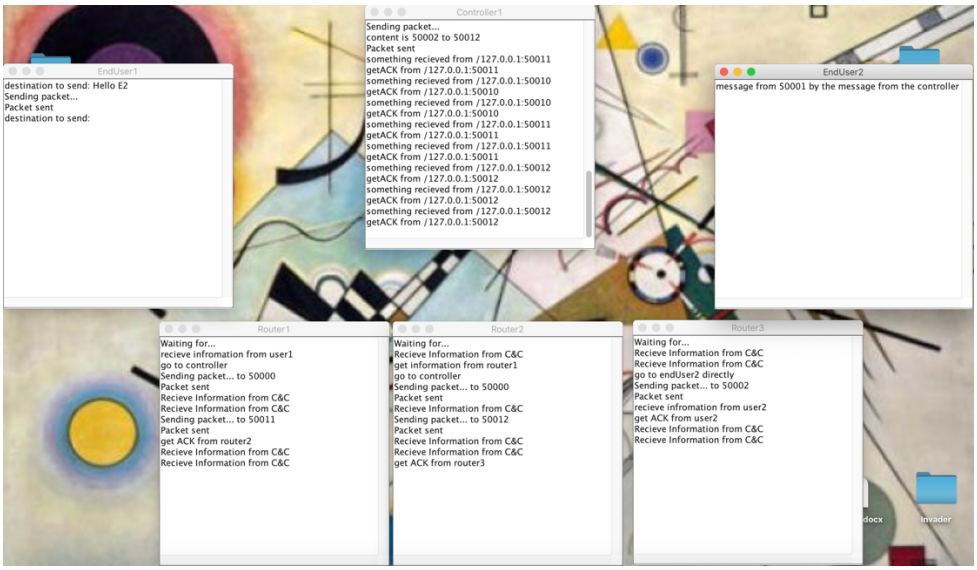

Figure 6: After entering “Hello E2” at once

Again, we have to enter the same string, “Hello E2.” The result appeared in the terminal of the EndUser2 is different from the last time, in other words, the process of this time did not ask the controller. Besides, the receiving ACK at the EndUser1 appears. Also, we can put the string into the terminal of the EndUser2 instead of the EndUser1. Enter the string “Hello E1” (figure 7).

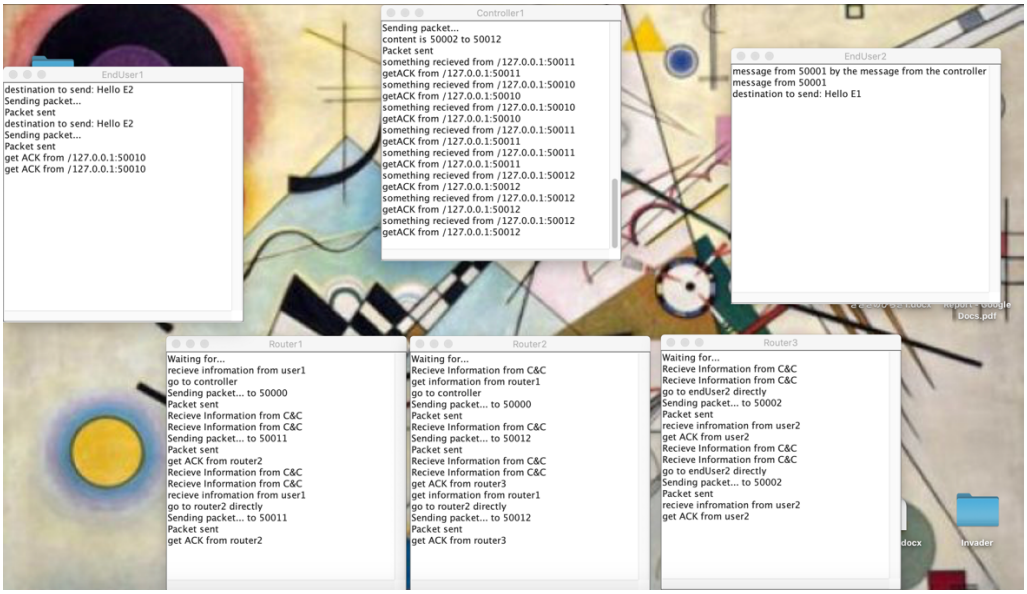

Figure 7: the result of the second time of sending a message from the EndUser1 and trying to send “Hello E1” from the EndUser2

Likewise, the result of this process is similar to the case of entering “Hello E2” (Figure 8). The result of the second time of entering the same string is also different from the previous consequence in the same way (Figure 9). Then this program finishes as it is mentioned that my program can only send a message four times in total: twice from the EndUser1 and twice from theEndUser2.

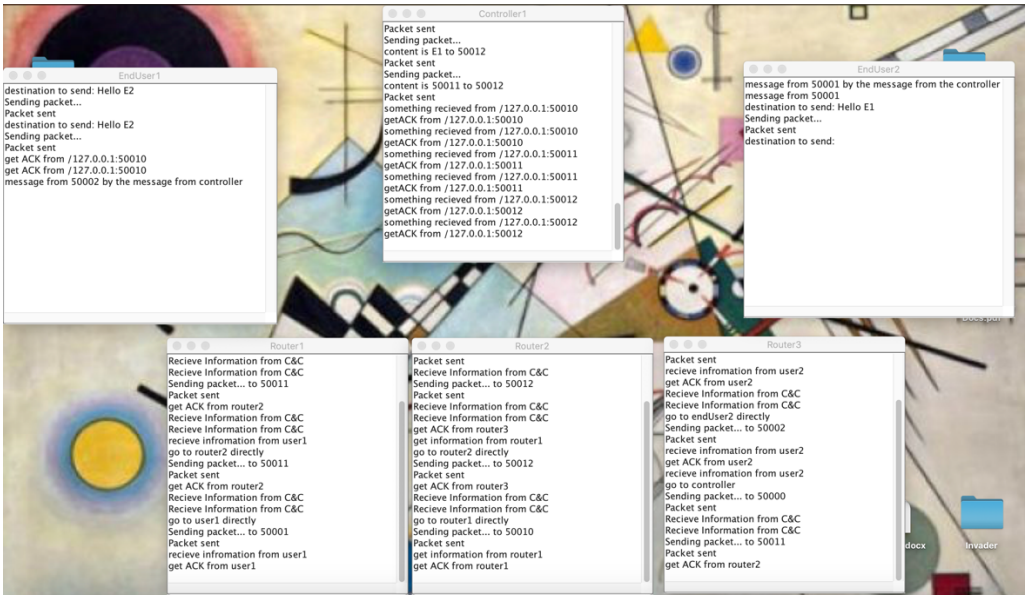

Figure 8: after entering “Hello E1” at once

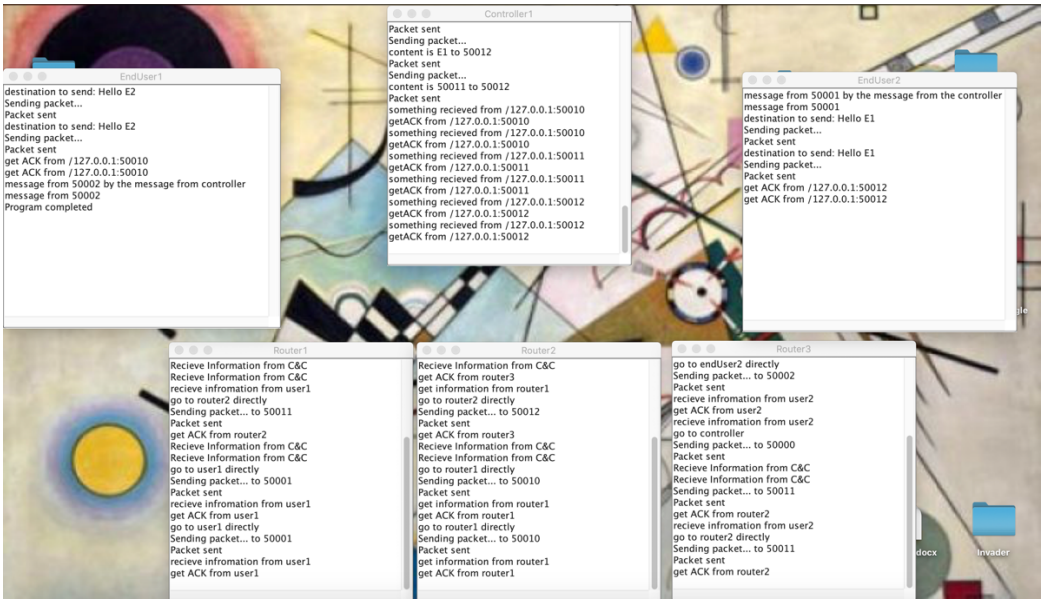

Figure 9: after entering after entering “Hello E1” at the second time

5.Choices

The most advantageous choice I made is creating functions which are commonly used in various classes such as `sendACK` and `sendPacket`. It makes more straightforward to implement the same role in each class and should be readable and understandable for other readers. In contrast, the `onReceipt` functions in router classes are so complicated that this is the disadvantage choice I made through creating this program.

6.Conclusion

My program can work along with the description of this assignment and helped me to understand the mechanism of the open flow of the network. The extension version should implement Link State or Distance Vector Routing but they are not implemented in my program. Furthermore, my program decides the order of sending: sending a message from the EndUser1 twice and then from the EndUser2 twice. It should decide freely which end-User to send a message. Also, while my program can send a message only four times, it should send any time the user wants.
This assignment is one of the hardest works in this semester, but I hope this is useful to understand the topic of this course.