This file briefly introduces the design and implementation of the application of part 3 and part 4 and acknowledge the reference. For a detailed documentation of the implementation, please see the project report. You can also refer to the code. I provide detailed annotation to increase the readability of the code.

**Part 3:**

**Layer-3 “Shortest-Path Switching” Routing Application Implementation**

I declare a variable HashMap<IOFSwitch, HashMap<IOFSwitch, IOFSwitch>> **paths** to store the shortest path from switches to other switches.

The format of paths is like this

{switch1 : {switch1 : null, switch2 : switch4, ...},  
switch2 : {switch1: switch3, switch2 : null, ...},  
...,

Switch:{ switch1: switchx, switch2 : switchy, ...}}

paths.get(switch x) will return another hashmap that contains the predecessor of each switch along the shortest path.

It is calculated with another method called **bfShortestPath()**. It is a Bellman-Ford algorithm used to calculate the shortest path. I get all the switches with build-in method of the class getSwitches().values(), and get all the links with build-in method getLinks(). I calculate the shortest path for each switch. I won’t introduce the basic principle of Bellman-Ford algorithm here, and I’ll put it in the report file.

There is also a method called **logData()** and **getShortestPathsAsString()**. It’s used to print the **paths** variable. I didn’t write it freshly. I modify it from others to print my **paths**. The original code is from <https://github.com/Randalthor95>. I use it to test and debug.

There are four methods I wrote to update the flow tables. I’ll introduce them together.

public void **setFlow**(Host host)

public void **buildFlowTables**(){

for(Host host : getHosts()) {

setFlow(host);

}

}

public void **deleteFlow**(Host host)

public void **deleteFlowTables**(){

for(Host host : getHosts()) {

deleteFlow(host);

}

}

The **deleteFlowTables** and **buildFlowTables** are easy to understand. They just apply the **deleteFlow** and **setFlow** to each host.

**deleteFlow** and **setFlow** are used to delete the flow table and set the flow table of a switch, respectively.

They work similarly. I use OFMatch and SwitchCommand.installRules to install specific rules.

For **setFlow,** I want the packet with Ethernet type of IPv4, and destination MAC address of host. Then I update the output port for each switch. I refer to the paths to get the next hop of switch and I traverse the link set to find the link whose source switch is the current host, and the destination switch is the next hop and add the link’s output port to the flow table.

For **deleteFlow,** cases are simpler. I just delete all the rules that match: 1. the Ethernet type is IPv4; 2. The destination MAC address is the host; 3. The source MAC address is the host.

With these methods I can build the trigger function easily.

For **deviceAdded** I call **buildFlowTables()** to install the rules in both ways (from others to the new device and the other way around).

For **deviceRemoved** I call **deleteFlow(host)** on the host to remove the rules of that device.

For **deviceMoved** I call **deleteFlow(host)** **buildFlowTables()** to remove the old rules of the host and build new rules.

For **switchAdded** I call **deleteFlowTables(), bfShortestPath(),** and **buildFlowTables()** on the host to install new rules of the new shortest path.

For **switchRemoved** I call **deleteFlowTables(), bfShortestPath(),** and **buildFlowTables()** because the removal of a switch is basically the same as adding a switch in my implementation.

For **linkDiscoveryUpdate()** I call **bfShortestPath(),** and **buildFlowTables()** to update the flow tables of each switch.

**Part 4:**

**Distributed Load Balance Routing Application Implementation**

I implemented couple of methods to realize the functions including:

public void **addSwitchRules**(IOFSwitch sw, String flag): It adds ARP and IP switch rules according to the value of flag for switch sw. I add switch rules for every virtual IP in this function. For IP rules, I deal with IPv4 packets with destination IP of a virtual host. For ARP rules, I deal with ARP packets with target IP of a virtual host. I set the output port to OFPort.OFPP\_CONTROLLER. Also I set the rules added here with priority of 1.

public void **addOther**(IOFSwitch sw): It adds all the switches rules, and it uses the tables from **ShortestPathSwitching** class. Just set the table variable to static in **ShortestPathSwitching.**

public void **arpReply**(Ethernet ethPkt, OFPacketIn pktIn, IOFSwitch sw): I create the ARP reply packet and the encapsulation Ethernet packet from the old one. Just make modification on their content. First, I check the packet, because I only deal with ARP request sent to the virtual IP. What I did is that set the opcode to ARP.OP\_REPLY, and set the target IP and MAC of the host which sent the request and set the sender IP and MAC of the host who sent the request.

public void **rewrite**(Ethernet ethPkt, OFPacketIn pktIn, IOFSwitch sw): This method has two responsibilities: one is to deal with the TCP SYNs and rewrite the IP and MAC. Another is to deal with TCP reset requests. For TCP reset, it’s simple, just do three encapsulations, create TCP, IP, and Ethernet packet sequentially. For TCP SYNs, I use **getNextHostIP**() and **getHostMACAddress**(int IP) method to get the IP and MAC of the host which the virtual IP should redirect packets to. Then I do the install rules two times, each for one direction (host to virtual IP, virtual IP to host). These connection-specified rules have the highest priority (set as 2) and idle timeout of 20 seconds.

Then I implement these methods to achieve the goals of the needed functions.

For **switchAdded(long switchId):** call **addSwitchRules(sw, “ip”), addSwitchRules(sw, “arp”) ,** and **addOther(sw)**

For **net.floodlightcontroller.core.IListener.Command** **receive,** I apply different methods in different cases. If the packet is an ARP packet, I call **arpReply** method, and for IPv4 packet, I call **rewrite** method.