**Homework 2: Floodlight-based OpenFlow controller development (Due Dec 11 Monday)**

Overview

In this homework, you will work with Floodlight, a java-based open source SDN controller, and mininet, a network emulator. You are asked to write a Floodlight module that writes flows into a switch’s forwarding table.

Setup

**Floodlight SDN controller**:

1. Download the VM hard disk file here (1.5G)

OneDrive link:

<https://liveumb-my.sharepoint.com/:u:/g/personal/bo_sheng_umb_edu/ERzS34THqttHof4J8rxL24QBAFQ8mU2KMqo26Bk_uDn41Q?e=hfOmaM>

1. Unzip the virtual hard disk file and import it in VirtualBox:
   1. Create a new VM using Linux Ubuntu
   2. In the dialog for “Hard Disk”, choose “*use an existing virtual hard disk file*”
2. Start the VM, and login with username/password as floodlight/floodlight
3. Open Eclipse, and it should automatically load ‘floodlight’ project.
4. Click ‘run’ button if available, or right-click project “floodlight” and choose run as “Java application” -> “Launch Floodlight”, and Floodlight should be started
5. (optional) Confirm it by checking its web UI. Open the web browser and visit <http://127.0.0.1:8080/ui/>index.html

**Mininet network:**

1. Open a terminal, and confirm the working directory is ~
2. Execute “sudo ./hw2” to create the network.
   1. This script creates a network topology with 4 switches and 2 hosts. The topology file is cs446-topo.py.
   2. The script hw2 is

**sudo mn --custom cs446-topo.py --topo=mytopo --controller=remote,ip=127.0.0.1,port=6653**

**--switch=ovsk,protocols=OpenFlow10**

It links the switches to the Floodlight controller running at 127.0.0.1, port 6653.

s1

s4

s2

s3

h1

h4

* 1. (Optional) Verify the topology on Floodlight’s web UI <http://127.0.0.1:8080/ui/>index.html

**Homework Task**

The goal of this homework is to add flows to switches with the following requirements

1. s1: When network traffic from h1 to h4 is generated, there are two possible paths, s1->s2->s4, and s1->s3->s4. Our objective is to periodically change the path that carries the flow from s1 to s4. For example, when we first let h1 ping h4, we can add a rule to the forwarding table that takes path s1->s2->s4. If we pause the traffic, wait for the rule to expire, and start the ping again. The controller should add another rule that directs the traffic through s1->s3->s4.
2. s4: If a ping request comes from s1->s2->s4, then the response from h4 should be routed to s4->s3->s1. Similarly, if a ping request comes from s1->s3->s4, then the response from h4 should be routed to s4->s2->s1.

In this homework, we are adding a new module named net.floodlightcontroller.cs446.MyController. It has to register as a system module and implement the interface to listen to the OpenFlow messages. The VM has got this part configured. But you are encouraged to read the document here <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343513/How+to+Write+a+Module>

You can find the main java file, MyController.java, as well as the package net.floodlightcontroller.cs446 under src/main/java. The core method for this homework is

**public net.floodlightcontroller.core.IListener.Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx)**

It handles the packet-in messages that are sent when the switch can’t find a matching flow in its forwarding table. The sample code gives some examples of extracting header information from the message, and the method **addMyFlow** is an example of adding a flow to a switch. The default routing module net.floodlightcontroller.forwarding.Forwarding has been disabled. And the current codes add flows to s1, s2, and s4 to enable the communication between h1 and h4. You can test with “h1 ping h4”, and then use “dpctl dump-flows” to check the flow tables.

The method uses factory and builder to set up the flow rule. You should keep the main structure, and work on the required inputs for each rule:

1. A match object, that defines the matching conditions. The sample code lists some fields.
2. An action list, that defines the actions if the matching conditions are satisfied. There’s only one action in the sample code (forwarding the packet to an output port). You should define the action list in the same way, but change the port number according to this homework’s requirements.

Note that besides the “ping” packets, there are other management traffic involved including packets that do not have IP headers, and your codes have to allow those traffic to go through. The “ping” packets can be identified by the following condition given an Ethernet object “eth”,

(eth.getEtherType()==EthType.IPv4)&&(((IPv4)eth.getPayload()).getProtocol()==IpProtocol.ICMP)

Java docs/APIs are be found in these two links

http://floodlight.github.io/floodlight/javadoc/floodlight/index.html

http://floodlight.github.io/floodlight/javadoc/openflowj-loxi/index.html

Floodlight REST APIs

https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API

**Testing**

1. Execute “sudo ./hw2” and enter mininet interface.
2. Execute “dpctl dump-flows” to check the forwarding tables on all the switches
3. Execute “h1 ping h4” to generate ping traffic
4. Or execute “xterm h1” to open another terminal, and then execute “ping 10.0.0.2” in the new terminal.
5. Check this link if you want to learn more about mininet commands, <http://mininet.org/walkthrough/>

**Submission**

Submit the following items on Blackboard:

1. The java file ***MyController.java***
2. A description of your test cases that validate the rules with screen snapshots.