CSci4211: Introduction to Computer Networks
Getting Started with Programming Assignment 3

Disclaimer: This document does not cover all the information but just provides a quick overview and directions on how to get started with programming assignment 3. We expect students to go through the links and references to get more information.

1. Tools Required

Mininet- Mininet is a powerful network emulation tool. It creates a realistic virtual network, running real kernel, switch and application code on a single machine. You can easily interact with your network using Command Line Interface (CLI). Notice that to test connectivity between switches, you need to run a Software Defined Network (SDN) controller.

POX- POX is a Software Defined Network (SDN) controller written in Python. POX provides a framework for communicating with SDN switches using the well-defined application programming interface (API), OpenFlow protocol. The controller exercises direct control over the state in the SDN switches via OpenFlow protocol.

Floodlight- Floodlight is Java-based Software Defined Network SDN controller and is intended to run with standard JDK tools and ant and can optionally be run in Eclipse.

2. How to Get Started with Mininet

2.1. Mininet Basics

The following commands give a brief introduction to mininet basic commands.

To start minimal topology and enter the Mininet CLI:
```
$ sudo mn
```
The default topology is the minimal topology, which includes one OpenFlow kernel switch connected to two hosts, and the switch is connected to the OpenFlow reference controller.

To display Mininet Command Line Interface (CLI) commands:
```
mininet> help
```

To display nodes:
```
mininet> nodes
```

If the first string of the CLI command is a host, switch or controller name, the command is executed on that node. For instance, to show the interface of host h1:

mininet> h1 ifconfig

Test connectivity between hosts. For example,to test the connectivity between h1 and h2:
mininet> h1 ping h2

Alternatively, you can test the connectivity between all hosts by typing:
mininet> pingall

Exit Mininet:
mininet> exit

Clean up:
$ sudo mn –c

For a more complete walkthrough of Mininet, we highly recommend everyone to go through the following link: http://mininet.org/walkthrough/

2.2. Mininet Python API

A more powerful way to construct the network topology is to use Mininet Python APIs. 'mininet.topo.Topo' is the class that defines the network topology. We can define a child class of 'mininet.topo.Topo' and initialize the network topology with the help of the class methods.

addHost(name, opts): add a host to the network with the name being 'name' parameter. opts' is a dictionary that includes the parameters for the host.
Eg: h1 = self.addHost('h1') adds a host with name 'h1'

addSwitch(name, opts): add an Openflow vSwitch to the network with the name being 'name' parameter. 'opts' is a dictionary that includes the parameters for the switch.
Eg: s1 = self.addSwitch('s1') adds a switch with name s1

addLink(node1, node2, port1, port2, opts): add a bidirectional link between port1 of node1 and port2 of nodes. 'opts' is a dictionary that includes the parameters for the link. 'bw' is one of the parameters that stands for bandwidth, and unit is Mbps.
Eg: self.addLink(h1, s1) adds a link between host h1 and switch s1.

Mininet(topo, link, controller) is the initialization function of the Mininet network emulation environment.

3. Getting Started with POX controller

After setting up the network environment in the previous step, a controller is needed to install flow entries to the switch so that packets can be forwarded among hosts.

In general a POX controller consists of three parts:

1. **Listener:** First you need to figure out the type of the event you want the controller to listen to (e.g., ConnectionUp, PacketIn, etc).
2. **Control logic:** Then using some logic you can distinguish between different flows and attach a proper action for a specific flow.
3. **Messenger:** Finally you send the message to the switch to add the new rule in the Openflow table.

More details of the above functionality is provided below.

3.1. Listener

There are two common ways for an application to register with the controller for events.

1. Register callback functions for specific events thrown by either the OpenFlow handler module or specific modules like Topology Discovery
    - From the launch or from the init of a class, perform core.openflow.addListenerByName("EVENTNAME", CALLBACK_FUNC, PRIORITY)
    - For instance you already developed a switching function named switch(). If you want to trig the function when the controller starts to work, add a listener in launch to provoke function switch when ConnectionUp handler is called: core.openflow.addListenerByName("ConnectionUp", switch)
2. Register object with the OpenFlow handler module or specific modules like Topology Discovery
    - From typically the init of a class, perform addListeners(self). Once this is added, the controller will look for a function with the name _handle_EVENTNAME(self, event). This method is automatically registered as an event handler.

3.2. Event handlers

As explained above you need to set a listener on your POX and when an event happens the relevant handler function will be activated. Your control logic should be placed in one of these handlers.
The two main handlers that you might need to modify in your program are:

- ConnectionUp, which activates when a switch turns on (or connects to the controller). The name of the handler function for this event is: _handle_ConnectionUp. Be careful about timers when implementing this handler. ConnectionUp triggers only once when the switch starts to work.

- PacketIn, activates by arriving a packet into the controller. The name of the handler is _handle_PacketIn.

3.3. Parsing Packets

POX provides several primitives to parse well-known packets. The ethernet packet received through a *packet_in* event can be extracted using the following

```
packet = event.parsed
src_mac = packet.src
dst_mac = packet.dst
if packet.type == ethernet.IP_TYPE:
   ipv4_packet = event.parsed.find("ipv4")
   # Do more processing of the IPv4 packet
   src_ip = ipv4_packet.srcip
   src_ip = ipv4_packet.dstip
```

3.4. Useful POX API

- ofp_flow_mod OpenFlow message

  This composes a message which tells a switch to install a flow entry. It has a match attribute and a list of actions. Notable fields are:
  - actions – A list of actions to perform on matching packets.
  - match – A ofp_match object. By default, none of the fields of this object is set. You may need to set some of its fields
- ofp_match class

  This class describes packet header fields and an input port to match on. Fields not specified are “wildcards” and will match on any value.
  For example, create a match which matches packets arriving on port 3:
  ```
  match = of.ofp_match()
  match.in_port = 3
  ```
- ofp_action_output class

  This is an action which specifies a switch port that you want to send the packet out of.
  There is a variety of pre-defined “port numbers” such as OFPP_FLOOD.
  For example, create an action which sends packet out of port 2.
  ```
  out_action = of.ofp_action_output(port = 2)
  ```
- connection.send(…)

  This function sends an OpenFlow message to a switch

For example, create a flow_mod that sends packets arriving on port 3 out of port 4

```
msg = of.ofp_flow_mod()
msg.match.in_port = 3
msg.actions.append(of.ofp_action_output(port = 4))
connection.send(msg)
```

Specifically, use this link to get further information for creating a module using POX.

4. Getting started with Floodlight Controller

Floodlight is Java-based SDN controller and is intended to run with standard JDK tools and ant and can optionally be run in Eclipse.

1) First, you need to figure out the type of the event you want the controller to listen to (e.g., PacketIn, etc).
2) Then using some logic you can distinguish between different flows and attach a proper action for a specific flow.
3) Finally you send the message to the switch to add the new rule in the Openflow table.

4.1. Listener:

First you need to register the event you want to listen to. Note that we are listening to PacketIn messages in the below code.

```
… implements IFloodlightModule, IOFMessageListener{
     protected IFloodlightProviderService floodlightProvider;
     …
     public void init(FloodlightModuleContext context) throws FloodlightModuleException {
          floodlightProvider = context
               .getServiceImpl(IFloodlightProviderService.class);
          floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
          …
     }
     …
}
```

Then you need to handle the packet using the receive function. Your logic goes here.

```
… implements IFloodlightModule, IOFMessageListener{
     …
     public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
          OFPacketIn pi = (OFPacketIn) msg;
          Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
                         IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
          …
     }
     …
}
```

4.2. Useful Floodlight API

Creating match

This class describes packet header fields and an input port to match on. Fields not specified are "wildcards" and will match on any value. Use the following lines to create a match.

```
OFMatch match = new OFMatch();
match.setWildcards(Wildcards.FULL.matchOn(Flag.DL_TYPE).matchOn(Flag.NW_DST).
withNwDstMask(24) );
match.setDataLayerType( Ethernet.TYPE_IPv4 );
match.setNetworkSource( IPv4.toIPv4Address("152.3.140.0") );
```

Creating an Action:

In SDN switches are dumb and action tells them what to do with a matched packet
Actions are:

- Send packet out to a port
- Modify the packet's header

You can create an action in floodlight using the following lines.

```
ArrayList<OFAction> actions = new ArrayList<OFAction>();
OFActionOutput action = new OFActionOutput().setPort((short) 3);
OFActionNetworkLayerSource ofanls = new OFActionNetworkLayerSource();
ofanls.setNetworkAddress( IPv4.toIPv4Address("8.8.8.8") );
```

Creating a Flow Mod Message

This composes a message which tells a switch to install a flow entry. It has a match attribute and a list of actions. Notable fields are:

- actions – A list of actions to perform on matching packets.
- match – A ofp_match object. By default, none of the fields of this object is set. You may need to set some of its fields

Use the following lines to create a Flow Mod Message.

```
OFFlowMod flowMod = new OFFlowMod();
flowMod.setMatch( match );
flowMod.setActions( actions );
flowMod.setLength( OFFlowMod.MINIMUM_LENGTH + OFActionOutput.MINIMUM_LENGTH +
                    OFActionNetworkLayerSource.MINIMUM_LENGTH) );
try {
          sw.write(flowMod, cntx);
          sw.flush();
} catch (IOException e) {
          log.error("Failure writing flowMod", e);
}
```

Check out this link to see how to create a PacketIn packet and this link to see how to create a PacketOut packet. Specifically, use this link for the tutorial to create a module.

References:

http://sdnhub.org/tutorials/sdn-tutorial-vm/
https://floodlight.atlassian.net/wiki
http://mininet.org/
https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch