Lab 10: SDN Basics

50.012 Networks

Hand-out: November 29
eDimension hand-in: December 9 (longer than normal)

1 Overview

- In this execise, we will use mininet to write a simple SDN controller for switches.
- Software-Defined Networking (SDN) is a promising new approach to network management
- We have linked a nice video intro to SDN in eDimension
- We start by using OpenFlow and the POX controller to make a SDN switch learn mac addresses (similar to the *cam* table in a normal non-SDN switch)
- We then extend the controller to install specific *flows* in the switch
- This exercise is built on the official mininet tutorial at `https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch`

2 Setup

- During the setup, you should be connected to `SUTD_Student` to have Internet access.
- This exercise requires a working mininet setup. All lab machines have this running already.

2.1 POX

- Install the POX SDN controller into a directory of your choice (e.g., ~/lab10/pox)

```
mkdir ~/lab10
cd ~/lab10
git clone http://github.com/noxrepo/pox
cd pox
```

3 Create a Learning Switch

3.1 Simple SDN Hub (controllerA)

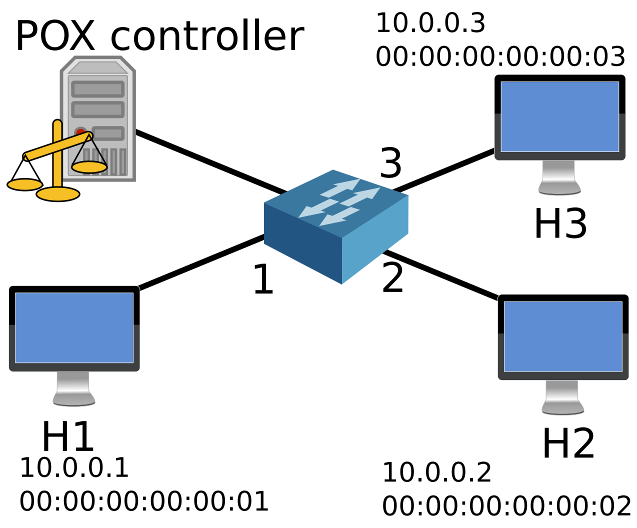
- Start by executing the following to terminate potentially running SDN controllers, and close remaining mininet sessions:

```
sudo mn -c
```

- Now start a mininet session with a simple topology, but without an SDN controller:

```
sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

- At this point, nodes should not be able to ping each other, because the switch cannot reach its SDN controller.



- To start the SDN controller, cd into the folder where you installed POX (e.g., ~/lab10/pox), and start your pox controller to listen to the virtual switch.

```
cd ~/lab10/pox
./pox.py log.level --DEBUG misc.of_tutorial
```

- This tells POX to enable verbose logging and to start the of_tutorial component which you'll be using (which currently acts like a hub).
- The switches may take a little bit of time to connect. When an OpenFlow switch loses its connection to a controller, it will generally increase the period between which it attempts to contact the controller, up to a maximum of 15 seconds. Since the OpenFlow switch has not connected yet, this delay may be anything between 0 and 15 seconds. If this is too long to wait, the switch can be configured to wait no more than N seconds using the –max-backoff parameter. Alternately, you exit Mininet to remove the switch(es), start the controller, and then start Mininet to immediately connect.
- Wait until the application indicates that the OpenFlow switch has connected. When the switch connects, POX will print something like this:

```
 INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
 DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
```

- The first line is from the portion of POX that handles OpenFlow connections. The second is from the tutorial component itself.

- Currently, the SDN controller is actually telling the switch to behave like a hub, i.e. it will broadcast all received traffic to all ports except the original port.
- To verify this, start wireshark on h3, and then send a ping from h1 to h2. You should see the ping packets at h3 as well!

3.2 SDN controller with MAC learning (controllerB)

- We will now change the SDN controller to remember MAC/port mappings. For each packet, the controller can now tell the switch on which port to send it.
- Stop the tutorial hub controller in the second terminal using Ctrl-C. The file you'll modify is pox/misc/of_tutorial.py. Open this file in your favorite editor.
- The current code calls act_like_hub() from the handler for packet_in messages to implement switch behavior. You'll want to switch to using the act_like_switch() function, which contains a sketch of what your final learning switch code should look like.
- To quickly understand the objects in the pox code, you can use

```
print dir(object)
```

- Overall, you should only have to change few lines. Test the new behaviour by again pinging h2 from h1, and using wireshark on h3.

1. OpenFlow messages with POX
    - The subsections below give details about POX APIs that should prove useful in the exercise. There is also other documentation available in the appropriate section of POX's website at `https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-POXAPIs`
        - You will need to accept the outdated certificate to visit that page...

    ```
    connection.send( ... ) # send an OpenFlow message to a switch
    ```

    - When a connection to a switch starts, a ConnectionUp event is fired. The example code creates a new Tutorial object that holds a reference to the associated Connection object. This can later be used to send commands (OpenFlow messages) to the switch.

    (a) Parsing Packets with the POX packet libraries.
    The POX packet library is used to parse packets and make each protocol field available to Python. This library can also be used to construct packets for sending. The parsing libraries are in pox/lib/packet/. Each protocol has a corresponding parsing file.
    For the first exercise, you'll only need to access the Ethernet source and destination fields. For example use the dot notation, use `packet.src` to extract the source of a packet.
    The Ethernet src and dst fields are stored as pox.lib.addresses.EthAddr objects. These can easily be converted to their common string representation (str(addr) will return something like "01:ea:be:02:05:01"), or created from their common string representation (EthAddr("01:ea:be:02:05:01")).

(b) ofp_packet_out OpenFlow message. The ofp_packet_out message instructs a switch to send a packet. The packet might be one constructed at the controller, or it might be one that the switch received, buffered, and forwarded to the controller (and is now referenced by a buffer_id).

    Notable fields are:

    - buffer_id - The buffer_id of a buffer you wish to send. Do not set if you are sending a constructed packet.
    - data - Raw bytes you wish the switch to send. Do not set if you are sending a buffered packet.
    - actions - A list of actions to apply (for this tutorial, this is just a single ofp_action_output action).
    - in_port - The port number this packet initially arrived on if you are sending by buffer_id, otherwise OFPP_NONE.

    Example. of_tutorial's send_packet() method:

```
action = of.ofp_action_output(port = out_port)
msg.actions.append(action)

#Send message to switch
self.connection.send(msg)
```

(c) ofp_action_output class.

    - This is an action for use with ofp_packet_out and ofp_flow_mod. It specifies a switch port that you wish to send the packet out of. It can also take various "special" port numbers. An example of this would be OFPP_FLOOD which sends the packet out all ports except the one the packet originally arrived on.
    - Example. Create an output action that would send packets to all ports:

```
out_action = of.ofp_action_output(port = of.OFPP_FLOOD)
```

3.3 Installing flow table entries in the switch (controllerC)

- So far, the switch will contact the controller for each received packet. We will now change the controller to *install flows* in the switch. Those flows are essentially rule sets that are evaluated for each incoming packet.

- Modify the controller to create suitable flows for the network for all mac addresses that are learned. The following information might be helpful:

1. ofp_match class. Objects of this class describe packet header fields and an input port to match on. All fields are optional – items that are not specified are "wildcards" and will match on anything.

   Some notable fields of ofp_match objects are:

   - dl_src - The data link layer (MAC) source address
   - dl_dst - The data link layer (MAC) destination address
   - in_port - The packet input switch port

Example. Create a match that matches packets arriving on port 3:

```
match = of.ofp_match()
match.in_port = 3
```

2. ofp_flow_mod OpenFlow message. This instructs a switch to install a flow table entry. Flow table entries match some fields of incoming packets, and executes some list of actions on matching packets. The actions are the same as for ofp_packet_out, mentioned above (and, again, for the tutorial all you need is the simple ofp_action_output action). The match is described by an ofp_match object.

   Notable fields are:

   - idle_timeout - Number of idle seconds before the flow entry is removed. Defaults to no idle timeout.
   - hard_timeout - Number of seconds before the flow entry is removed. Defaults to no timeout.
   - actions - A list of actions to perform on matching packets (e.g., ofp_action_output)
   - priority - When using non-exact (wildcarded) matches, this specifies the priority for overlapping matches. Higher values are higher priority. Not important for exact or non-overlapping entries.
   - buffer_id - The buffer_id of a buffer to apply the actions to immediately. Leave unspecified for none.
   - in_port - If using a buffer_id, this is the associated input port.
   - match - An ofp_match object. By default, this matches everything, so you should probably set some of its fields!

   Example. Create a flow_mod that sends packets from port 3 out of port 4.

```
fm = of.ofp_flow_mod()
fm.match.in_port = 3
fm.actions.append(of.ofp_action_output(port = 4))
```

4 What to Hand in

4.1 eDimension submission:

- Please hand in your modified `of_tutorial.py` file.
- Make sure to include you name in the header of the file
- If you could not fulfill all requirements (i.e. install working flows), please mention that, and list what works in your header.

4.2 Checkoff:

- No checkoff required if you submitted your reply sheet