Lab 3 - Build a Learning Switch on Ryu

In this exercise, you will learn how to build a learning switch on Ryu , using wireshark to capture the openflow and data packets and then you can identify if your switch is learning. Besides, you could also verify behavior defined in your code by dumping the flow entries on SDN switch to verify your code.

Through the practice, you'll be more familiar with the Openflow protocol and SDN architecture and how to write basic application on controller.

[Important Resource] RYU is a Python based SDN controller platform geared towards research and education. For more details on RYU, see the OFFICAL WEBSITE OF RYU

RYU Overview

Getting Started

What's Ryu

Ryu is a component-based software defined networking framework.

Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4 and Nicira Extensions.

All of the code is freely available under the Apache 2.0 license. Ryu is fully written in Python.

How to install RYU(Controller)

Get the latest official version source build and install it.

(The Easiest Way)

An automatic installation script for build and install Ryu source code on Ubuntu 12.04+.

To make installation easiest. This helper script which should get all dependencies and download, build, and install Ryu.

```
$ wget https://raw.githubusercontent.com/sdnds-tw/ryuInstallHelper/master/ryuInstallHelper.sh
$ bash ryuInstallHelper.sh
```

Note: This script has only been tested on the most recent stable version of Ubuntu.

If you want to write your Ryu application, have a look at **Writing ryu application** document. After writing your application, just type:

```
% ryu-manager yourapp(name of your code).py
```

Please finish the following page first:

http://ryu.readthedocs.org/en/latest/writing_ryu_app.html

The First Application

Whetting Your Appetite

If you want to manage the network gears (switches, routers, etc) at your way, you need to write your Ryu application. Your application tells Ryu how you want to manage the gears. Then Ryu configures the gears by using OpenFlow protocol, etc.

Writing Ryu application is easy. It's just Python scripts.

Start Writing

We show a Ryu application that make OpenFlow switches work as a dumb layer 2 switch.

Open a text editor creating a new file with the following content:

```
from ryu.base import app_manager


class L2Switch(app_manager.RyuApp):

    def __init__(self, *args, **kwargs):

        super(L2Switch, self).__init__(*args, **kwargs)
```

Ryu application is just a Python script so you can save the file with any name, extensions, and any place you want. Let's name the file 'l2.py' at your home directory.

This application does nothing useful yet, however it's a complete Ryu application. In fact, you can run this Ryu application:

```
% ryu-manager ~/l2.py
loading app /Users/fujita/l2.py
instantiating app /Users/fujita/l2.py
```

All you have to do is defining needs a new subclass of RyuApp to run your Python script as a Ryu application.

Next let's add the functionality of sending a received packet to all the ports.

Dumb L2 switch (as below)

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls


class L2Switch(app_manager.RyuApp):
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
```

```
        ofp_parser = dp.ofproto_parser
        actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        out = ofp_parser.OFPPacketOut(
            datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
            actions=actions)
        dp.send_msg(out)
```

A new method ‘packet_in_handler’ is added to L2Switch class. This is called when Ryu receives an OpenFlow packet_in message. The trick is ‘set_ev_cls’ decorator. This decorator tells Ryu when the decorated function should be called.

The first argument of the decorator indicates an event that makes function called. As you expect easily, every time Ryu gets a packet_in message, this function is called.

The second argument indicates the state of the switch. Probably, you want to ignore packet_in messages before the negotiation between Ryu and the switch finishes. Using ‘MAIN_DISPATCHER’ as the second argument means this function is called only after the negotiation completes.

Next let’s look at the first half of the ‘packet_in_handler’ function.

- ev.msg is an object that represents a packet_in data structure.
- msg.dp is an object that represents a datapath (switch).
- dp.ofproto and dp.ofproto_parser are objects that represent the OpenFlow protocol that Ryu and the switch negotiated.

Ready for the second half.

- OFPActionOutput class is used with a packet_out message to specify a switch port that you want to send the packet out of. This application need a switch to send out of all the ports so OFPP_FLOOD constant is used.
- OFPPacketOut class is used to build a packet_out message.
- If you call Datapath class’s send_msg method with a OpenFlow message class object, Ryu builds and send the on-wire data format to the switch.

Here, you finished implementing your first Ryu application. You are ready to run this Ryu application that does something useful.y.

Assignment

Any plagiarism is prohibited (including codes and report)

Modify the code above and make your switch can memorize the route. So, your code need to memorize the relationship among input port, the source mac, and the destination mac and add this rule in switch's flow table.

Please modify the skeleton code with the functionality as a learning switch and then observe the difference between the original one (dumb switch) and your learning switch through wireshark and the flow entries dumped on your switch.

Learning Switch

Learning Switchs have a variety of functions. Here, we take a look at a Learning Switch having the following simple functions.

- Learns the MAC address of the host connected to a port and retains it in the MAC address table.
- When receiving packets addressed to a host already learned, transfers them to the port connected to the host.
- When receiving packets addressed to an unknown host, performs flooding.

Learning Switch by OpenFlow

OpenFlow switches can perform the following by receiving instructions from OpenFlow controllers such as Ryu.

- Rewrites the address of received packets or transfers the packets from the specified port.
- Transfers the received packets to the controller ***(Packet-In).***
- Transfers the packets forwarded by the controller from the specified port ***(Packet-Out).***

It is possible to achieve a Learning Switch having those functions combined.

First of all, you need to use the Packet-In function to learn MAC addresses. The controller can use the Packet-In function to receive packets from the switch. The switch analyzes the received packets to learn the MAC address of the host and information about the connected port.

After learning, the switch transfers the received packets. The switch investigates whether the destination MAC address of the packets belong to the learned host. Depending on the investigation results, the switch performs the following processing.

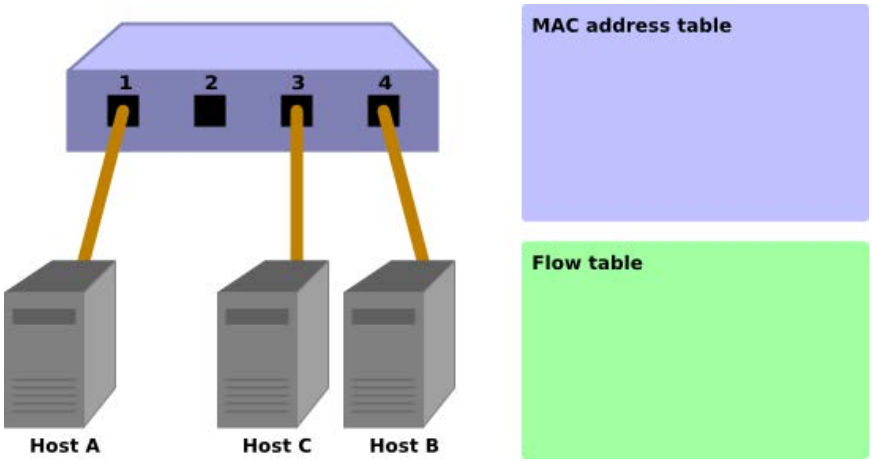
- If the host is already a learned host ... Uses the Packet-Out function to transfer the packets from the connected port.
- If the host is unknown host ... Use the Packet-Out function to perform flooding.

The following explains the above operation in a step-by-step way using figures.

1. Initial status

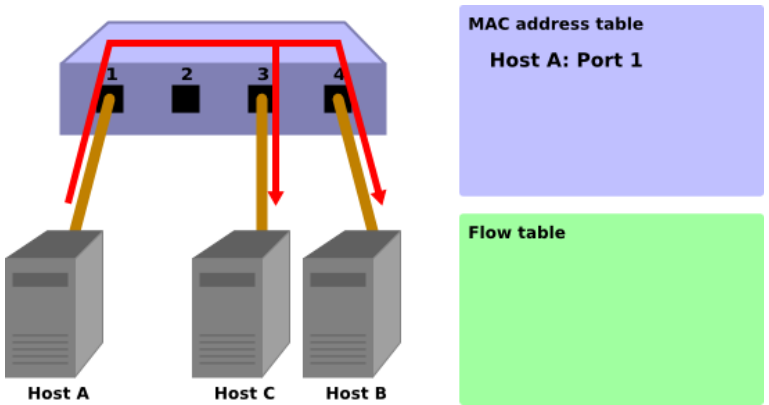
This is the initial status where the flow table is empty.

Assuming host A is connected to port 1, host B to part 4, and host C to port 3.



2. Host A -> Host B

When packets are sent from host A to host B, a Packet-In message is sent and the MAC address of host A is learned by port 1. Because the port for host B has not been found, the packets are flooded and are received by host B and host C.



Packet-In:

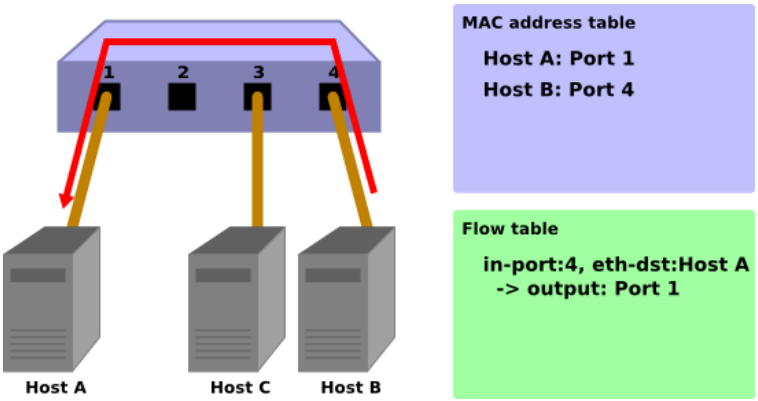
```
in-port: 1
eth-dst: Host B
eth-src: Host A
```

Packet-Out:

```
action: OUTPUT:Flooding
```

3. Host B -> Host A

When the packets are returned from host B to host A, an entry is added to the flow table and also the packets are transferred to port 1. For that reason, the packets are not received by host C.



Packet-In:

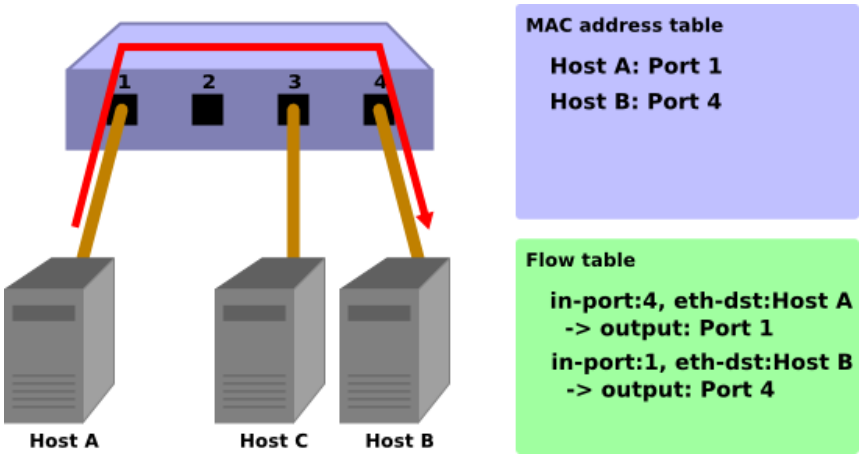
```
in-port: 4
eth-dst: Host A
eth-src: Host B
```

Packet-Out:

```
action: OUTPUT:Port 1
```

4. Host A -> Host B

Again, when packets are sent from host A to host B, an entry is added to the flow table and also the packets are transferred to port 4.



Packet-In:

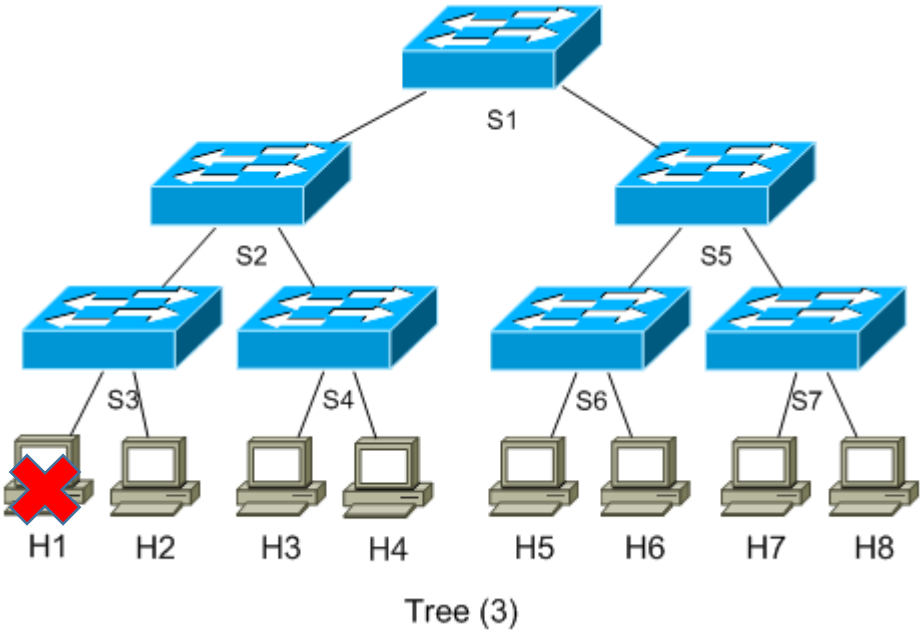
```
in-port: 1
eth-dst: Host B
eth-src: Host A
```

Packet-Out:

```
action: OUTPUT:Port 4
```

Extra Bonus

After the above exercise, you will be asked to create and submit a network application that implements ***Layer 2 Firewall*** that disables inbound and outbound traffic between two systems based on their ***MAC address***. More details on creating and submitting the code will be provided later on in the instructions. So, make sure that you follow each step carefully.



Tree (3)

Build the topology like above, and every switch not only could learning the mac address but also have the functionality of simple firewall.
sudo mn --topo tree,3 –mac

Firewall-policies

- Build firewall to block all the traffic coming from H1 (means all the hosts won't receive the packets whose source mac address is H1)
- H8 hates H4 and H8 would like to block all the traffic coming from H4