Intro to OpenFlow Tutorial with Ryu Controller

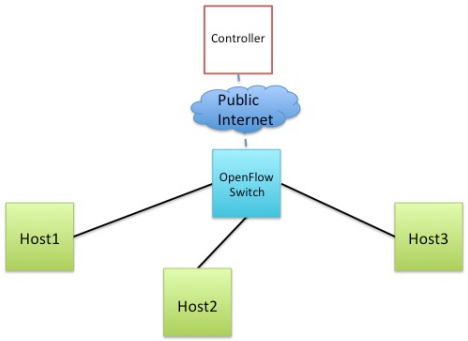
TinyURL: http://www.tinyurl.com/geni-ovs-ryu

Overview:

This is a simple OpenFlow tutorial that will guide you through the writing of simple OpenFlow controllers to showcase some of the OpenFlow capabilities. We are going to write three different controllers:

1. Write a controller that will **duplicate all the traffic** of the OpenFlow switch out a specific port
2. **TCP Port Forward** controller. Divert all traffic destined to host A on TCP port X to TCP port Y
3. **Proxy Controller**. Write a controller that will divert all traffic destined to host A, TCP port X to host B, TCP port Y

In this tutorial we are using the ***OpenFlow Software Switch***, Open vSwitch (OVS). The general topology is as pictured below. In general, the controller just needs to have a public IP address, so that it can exchange messages with the OpenFlow switch. The controller for the switch can run anywhere in the Internet. For this tutorial we are going to use a Ryu controller, which is just one example of many controller frameworks.



Prerequisites:

- A GENI account, if you don't have one sign up!
- Familiarity with how to reserve GENI resources with any of the GENI Tools (GENI Experimenter Portal, Omni, Jacks). If you don't know you can take any of the tutorials:
  - Reserving resources using Jacks tutorial
  - Reserving resources using Omni tutorial
- Familiarity with logging in to GENI compute resources.
- Basic understanding of OpenFlow. If you are doing this tutorial at home, flip through the tutorial's slides
- Familiarity with the Unix Command line
- Familiarity with the python programming language. We are going to use the Ryu controller, which is just one example of many controller frameworks, and Ryu is written in python.

Tools:

- Open vSwitch. OVS will be installed. Installation was completed as described here.
- Ryu controller. Ryu controller is installed as part of the resource reservation.

Where to get help:

- If you need help with GENI, email geni-users@googlegroups.com
- If you have questions about OpenFlow, OVS, Ryu you can subscribe to openflow-discuss or any of the other mailing lists listed.

Resources:

- Learn more about OpenFlow
- Ryu controller
- Learn more about OVS

Tutorial Instructions

- Part I: Design/Setup
  - Step 1: Reserve Resources
    - OpenFlow using Open vSwitch (OVS)
  - Step 2: Configure and Initialize Services
- Part II: Execute
  - Step 3: Execute Experiment
- Part III: Finish
  - Step 4: Teardown Experiment

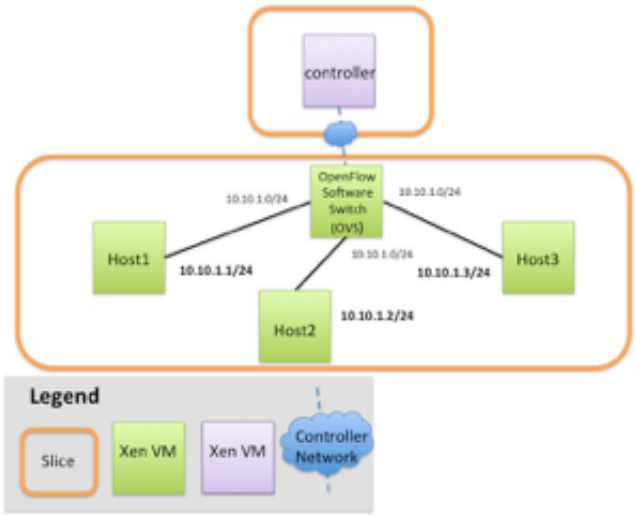
Intro to OpenFlow Tutorial (OVS) with Ryu Controller

Intro to OpenFlow Tutorial (OVS) with Ryu Controller
Step 1. Obtain resources
Step 2. Configure and Initialize
2a. Configure the Software Switch (OVS Window)
2b. Point your switch to a controller
2c. standalone vs secure mode
Prev: Introduction
Next: Execute

Overview

In this tutorial we are going to use Open vSwitch (OVS) as an OpenFlow switch connected to three hosts. OVS is a software switch running on a compute resource. The other three hosts can only communicate through the OVS switch. The experiment will need (the rspecs for this exercise are provided later in this section):

- *1 Xen VM with a public IP to run an OpenFlow controller*
- *1 Xen VM to be the OpenFlow switch*
- *3 Xen VMs as hosts*



Step 1. Obtain resources

For the following two reservations you can use different aggregates and one slice, or same aggregate but two slices (recommended). We do this so that you can change your experiment topology (e.g. from software switches to hardware switches), but keep the same controller.

You can use compute resources from any **InstaGENI rack** and any reservation tool (Portal, jFed, Omni, etc) For a list of available InstaGENI racks see the GENI Production Resources page.

a. **Reserve a VM that runs your OpenFlow controller**.
RSpec: In the Portal: *XEN OpenFlow Controller*, ~~url: https://raw.githubusercontent.com/GENI-NSF/geni-tutorials/master/OpenFlowCtrls/Controllers_all.xml~~

a. **Reserve your network**, that includes a VM with OVS installed.
RSpec: In the Portal *OpenFlow OVS all XEN*, ~~url: https://raw.githubusercontent.com/GENI-NSF/geni-tutorials/master/OVSRyu/openflowovs-all-xen.rspec.xml~~

You will need SSH access to your nodes. If you don't know how to SSH to your reserved hosts learn how to login

Step 2. Configure and Initialize

Overview: *Although OVS is installed and initialized on the host that is meant to act as a software switch, it has not been configured yet. There are two main things that need to be configured:*

(1) configure your software switch with the interfaces as ports and
(2) point the switch to an OpenFlow controller.

2a. Configure the Software Switch (OVS Window)

i. Login to the OVS host
ii. Create an Ethernet bridge that will act as our software switch:

```
sudo ovs-vsctl add-br br0
```

iii. Prepare the interfaces to be added as ports to the OVS switch
- Your OVS bridge will be a Layer 2 switch and your ports do not need IP addresses. Before we remove them let's keep some information
  - Run `ifconfig`
  - Write down the interface names that correspond to the connections to your hosts. The correspondence is
    - Interface with IP *10.10.1.11* to host1 - ethX
    - Interface with IP *10.10.1.12* to host2 - ethY
    - Interface with IP *10.10.1.13* to host3 - ethZ
- Remove the IP from your data interfaces.
  ⚠ Be careful **not to bring down eth0**. This is the control interface, if you bring that interface down you **won't be able to login** to your host. For all interfaces other than `eth0` and `lo` (your interface names may vary) run :

```
sudo ifconfig ethX 0
sudo ifconfig ethY 0
sudo ifconfig ethZ 0
```

iv. Add all the data interfaces to your switch (bridge).
⚠ Be careful **not to add interface eth0**. This is the control interface. The other three interfaces are your data interfaces. (Use the same interfaces as you used in the previous step.)

```
sudo  ovs-vsctl add-port br0 ethX
sudo  ovs-vsctl add-port br0 ethY
sudo  ovs-vsctl add-port br0 ethZ
```

v. Trust but verify. Congratulations! You have configured your software switch. To verify the three

ports configured run:

```
sudo ovs-vsctl list-ports br0
```

2b. Point your switch to a controller

An OpenFlow switch will not forward any packet unless instructed by a controller. Basically the forwarding table is empty, until an external controller inserts forwarding rules. The OpenFlow controller communicates with the switch over the control network and it can be anywhere in the Internet as long as it is reachable by the OVS host.

i. Login to your controller
ii. Find the control interface IP of your controller, use *ifconfig* and note down the IP address of `eth0`.
iii. In order to point our software OpenFlow switch to the controller, in the *ovs* terminal window, run:

```
sudo ovs-vsctl set-controller br0 tcp:<controller_ip>:6633
```

iv. Set your switch to `fail-safe-mode`. For more info read the standalone vs secure mode section. Run:

```
sudo ovs-vsctl set-fail-mode br0 secure
```

v. Trust but verify. You can verify your OVS settings by issuing the following:

```
sudo ovs-vsctl show
```

2c. `standalone` vs `secure` mode

The OpenFlow controller is responsible for setting up all flows on the switch, which means that when the controller is not running there should be no packet switching at all. Depending on the setup of your network, such a behavior might not be desired. It might be best that when the controller is down, the switch should default back to being a learning layer 2 switch. In other circumstances however this might be undesirable. In OVS this is a tunable parameter, called `fail-safe-mode` which can be set to the following parameters:

- `standalone` *[default]: in this case OVS will take responsibility for forwarding the packets if the controller fails*
- `secure`: *in this case only the controller is responsible for forwarding packets, and if the controller is down all packets are dropped.*

In OVS when the parameter is not set it falls back to the `standalone` mode. For the purpose of this tutorial we will set the `fail-safe-mode` to `secure`, since we want to be the ones controlling the forwarding.

Prev: Introduction

Next: Execute

Intro to OpenFlow Tutorial (OVS) with Ryu Controller

Intro to OpenFlow Tutorial (OVS) with Ryu Controller
Step 3. Execute Experiment
3a. Login to your hosts
3b. Use a Learning Switch Controller
3c. Look around your OVS switch
Soft vs Hard Timeouts
3d. Download the Ryu apps
Useful Tips for writing your controller
3e. Debugging your Controller
i. Print messages
ii. Check the status in the switch
iii. Use Wireshark to see the OpenFlow messages
3f. Run a traffic duplication controller
3g. Run a port forward Controller
3h. Run a Server Proxy Controller
3i. Delete your bridge

Step 3. Execute Experiment

Now that the switch is up and running we are ready to start working on the controller. For this tutorial we are going to use the Ryu controller. The software is already installed in the controller host for running the Ryu controller.

3a. Login to your hosts

To start our experiment we need to ssh into all of our hosts.

To get ready for the tutorial you will need to have the following windows open:

- one window with ssh into the controller
- four windows with ssh into OVS
- one window with ssh into host1
- two windows with ssh into host2
- one window with ssh into host3

Depending on which tool and OS you are using there is a slightly different process for logging in. If you don't know how to SSH to your reserved hosts learn how to login. Once you have logged in follow the rest of the instructions.

3b. Use a Learning Switch Controller

In this example we are going to run a very simple learning switch controller to forward traffic between `host1` and `host2`.

1. First start a ping from `host1` to `host2`, which should timeout, since there is no controller running.

```
ping host2 -c 10
```

2. We have installed the Ryu controller under `/tmp/ryu` on the controller host. Ryu comes with a set of example modules that you can use out of the box. One of the modules is a learning switch. Start the learning switch controller which is already available by running the following two commands:

```
cd /tmp/ryu
./bin/ryu-manager ryu/app/simple_switch.py
```

The output should look like this:

```
loading app ryu/app/simple_switch.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/simple_switch.py of SimpleSwitch
```

3. In the terminal of `host1`, ping `host2`:

```
[experimenter@host1 ~]$ ping host2
PING host2-lan1 (10.10.1.2) 56(84) bytes of data.
From host1-lan0 (10.10.1.1) icmp_seq=2 Destination Host Unreachable
From host1-lan0 (10.10.1.1) icmp_seq=3 Destination Host Unreachable
From host1-lan0 (10.10.1.1) icmp_seq=4 Destination Host Unreachable
64 bytes from host2-lan1 (10.10.1.2): icmp_req=5 ttl=64 time=23.9 ms
64 bytes from host2-lan1 (10.10.1.2): icmp_req=6 ttl=64 time=0.717 ms
64 bytes from host2-lan1 (10.10.1.2): icmp_req=7 ttl=64 time=0.654 ms
64 bytes from host2-lan1 (10.10.1.2): icmp_req=8 ttl=64 time=0.723 ms
64 bytes from host2-lan1 (10.10.1.2): icmp_req=9 ttl=64 time=0.596 ms
```

Now the ping should work.

4. Go to your controller host and take a look at the print outs. You should see that your controller installed flows based on the mac addresses of your packets.

3c. Look around your OVS switch

1. If you are using OVS, to see the flow table entries on your OVS switch:

```
sudo ovs-ofctl dump-flows br0
```

You should see at least two table entries: One for ICMP Echo (icmp_type=8) messages from host1 to host2 and one for ICMP Echo Reply (icmp_type=0) messages from host2 to host1. You may also see flow entries for arp packets.

2. To see messages go between your switch and your controller, open a new ssh window to your controller node and run tcpdump on the `eth0` interface and on the tcp port that your controller is listening on usually 6633. (You can also run `tcpdump` on the `OVS` control interface if you desire.)

```
sudo tcpdump -i eth0 tcp port 6633
```

You will see (1) periodic keepalive messages being exchanged by the switch and the controller, (2) messages from the switch to the controller (e.g. when there is a table miss) and an ICMP Echo message in, and (3) messages from the controller to the switch (e.g. to install new flow entries).

3. Kill your Ryu controller by pressing `Ctrl-C`.

4. Notice what happens to your ping on host1.

5. If you are using OVS, check the flow table entries on your switch:

```
sudo ovs-ofctl dump-flows br0
```

You will see flow table entries in the switch. The entries time to expire value is set to infinity.

6. Delete the entries on the OVS:

```
sudo ovs-ofctl del-flows br0
```

   Notice what happens to your ping on host1.

Soft vs Hard Timeouts

All rules on the switch have two different timeouts:

- **Soft Timeout**: This determines for how long the flow will remain in the forwarding table of the switch if there are no packets received that match the specific flow. As long as packets from that flow are received the flow remains on the flow table.
- **Hard Timeout**: This determines the total time that a flow will remain at the forwarding table, independent of whether packets that match the flow are received; i.e. the flow will be removed after the hard timeout expires.

Can you tell now why there were packets flowing even after you killed your controller?

3d. Download the Ryu apps

To help you get started with your controller writing, we will provide:

- skeleton files for the controllers where you only need to complete some missing functionality
- the solution: fully implemented controllers
- a utility library that makes some of the Ryu messages easier to write

In the controller terminal execute:

```
mkdir /tmp/ryu/ryu/ext
cd /tmp/ryu/ryu/ext/
sudo wget https://github.com/GENI-NSF/geni-tutorials/raw/master/OVSRyu/ryu-intr
sudo tar xvfz ryu-intro-ctrlapps.tar.gz
```

Useful Tips for writing your controller

In order to make this first experience of writing a controller easier, we wrote some helpful functions that will abstract some of the particularities of Ryu away. These functions are located in `/tmp/ryu/ryu/ext/utils.py`, so while you write your controller consult this file for details.

Functions that are implemented include:

- packetIsIP : Test if the packet is IP
- packetIsARP : Test if the packet is ARP
- packetIsRequestARP : Test if this is an ARP Request packet
- packetIsReplyARP : Test if this is an ARP Reply packet
- packetArpDstIp : Test what is the destination IP in an ARP packet
- packetArpSrcIp : Test what is the source IP in an ARP packet
- packetIsTCP : Test if a packet is TCP
- packetDstIp : Test the destination IP of a packet
- packetSrcIp : Test the source IP of a packet
- packetDstTCPPort : Test the destination TCP port of a packet
- packetSrcTCPPort : Test the source TCP port of a packet
- createOFAction : Create one OpenFlow action

- getFullMatch : get the full match out of a packet
- createFlowMod : create a flow mod
- createArpRequest : Create an Arp Request for a different destination IP
- createArpReply : Create an Arp Reply for a different source IP

3e. Debugging your Controller

While you are developing your controller, some useful debugging tools are:

i. Print messages

Run your controller in verbose mode (add --verbose) and add print messages at various places to see what your controller is seeing.

ii. Check the status in the switch

If you are using an OVS switch, you can dump information from your switch. For example, to dump the flows:

```
sudo ovs-ofctl dump-flows br0
```

Two other useful commands show you the status of your switch:

```
sudo ovs-vsctl show
sudo ovs-ofctl show br0
```

iii. Use Wireshark to see the OpenFlow messages

Many times it is useful to see the OpenFlow messages being exchanged between your controller and the switch. This will tell you whether the messages that are created by your controller are correct and will allow you to see the details of any errors you might be seeing from the switch. You can use wireshark on both ends of the connection, in hardware switches you have to rely only on the controller view.

The controller host and OVS has wireshark installed, including the openflow dissector. For more information on wireshark you can take a look at the wireshark wiki.

Here we have a simple case of how to use the OpenFlow dissector for wireshark.

If you are on a Linux friendly machine (this includes MACs) open a terminal and ssh to your controller machine using the -Y command line argument, i.e.

```
ssh -Y <username>@<controller>
```

Assuming that the public IP address on the controller is eth0, run wireshark by typing:

```
sudo wireshark -i eth0&
```

When the wireshark window pops up, you might still have to choose eth0 for a live capture. And you will want to use a filter to cut down on the chatter in the wireshark window. One such filter might be just seeing what shows up on port 6633. To do that type *tcp.port eq 6633* in the filter window, assuming that 6633 is the port that the controller is listening on.

3f. Run a traffic duplication controller

In the above example we ran a very simple learning switch controller.

The power of OpenFlow comes from the fact that you can decide to forward the packet anyway you want based on the supported OpenFlow actions. A very simple but powerful modification you can do, is to duplicate all the traffic of the switch out a specific port. This is very useful for application and network analysis. You can imagine that at the port where you duplicate traffic you connect a device that does analysis. For this tutorial we are going to verify the duplication by doing `tcpdump` on two ports on the OVS switch.

1. **Use the interfaces that are connected to `host2` and `host3`.**
   - Software Switch (OVS): You should have noted down the interfaces for host2 and host3 in section 2a. If you have not noted the interfaces for host2 and host3 down (in section 2a), you can run tcpdump on OVS interfaces to figure out the interfaces for host2 and host3. This will allow you to see all traffic going out the interfaces.

   To see that duplication is happening, on the OVS host, run:

   ```
   sudo tcpdump -i <data_interface_name_to_host2>
   sudo tcpdump -i <data_interface_name_to_host3>
   ```

   You should see traffic from host1 to host2 showing up in the tcpdump window for host3. As a comparison, you will notice that no traffic shows up in that window when the controller is running the learning switch (at /tmp/ryu, type ./bin/ryu-manager ryu/ext/simple_switch.py).

2. **In the controller host directory `/tmp/ryu/ryu/ext` you should see two files:**

i. **myDuplicateTraffic.py** : This is the file that has instructions about how to complete the missing information. Go ahead and try to implement your first controller.
ii. **DuplicateTraffic.py** : This has the actual solution. You can just run this if you don't want to bother with writing a controller.
iii. **duplicate.config** : in this file, you need to specify which port you want to duplicate traffic to. We will be duplicating host2 traffic and send it to host3, so put host3 port number here. We noted down port numbers in section 2a. To figure out which port maps to which interface, use can also use "sudo ovs-ofctl show br0" on ovs node. (You can use any editor to edit the file, e.g. use nano by typing "sudo nano duplicate.config")

3. Run your newly written controller to duplicate the traffic. (update **duplicate.config** file with `host3` port number. We noted it down in section 2a). You can also use the solution file **DuplicateTraffic.py** to duplicate traffic:

   ```
   cd /tmp/ryu
   ./bin/ryu-manager ryu/ext/myDuplicateTraffic.py
   ```

4. To test it go to the terminal of host1 and try to ping host2:

   ```
   ping 10.10.1.2
   ```

   If your controller is working, your packets will register in both terminals running tcpdump.

5. Stop the Ryu controller using `Ctrl-C`.

3g. Run a port forward Controller

Now let's do a slightly more complicated controller. OpenFlow gives you the power to overwrite fields of your packets at the switch, for example the TCP source or destination port and do port forwarding. You can have clients trying to contact a server at port 5000, and the OpenFlow switch can redirect your traffic to a service listening on port 6000.

1. Under the `/tmp/ryu/ryu/ext` directory there are two files: **PortForwarding.py** and **myPortForwarding.py** that are similar to the previous exercise. Both of these controllers are configured by a configuration file at `ext/port_forward.config`. Use myPortForwarding.py to write your own port forwarding controller.

2. To test your controller we are going to use netcat. Go to the two terminals of host2. In one terminal run:

   ```
   nc -l 5000
   ```

   and in the other terminal run

   ```
   nc -l 6000
   ```

3. Now, start the simple layer 2 forwarding controller. We are doing this to see what happens with a simple controller.

   ```
   cd /tmp/ryu
   ./bin/ryu-manager ryu/ext/simple_switch.py
   ```

4. Go to the terminal of host1 and connect to host2 at port 5000:

   ```
   nc 10.10.1.2 5000
   ```

5. Type something and you should see it at the terminal of host2 at port 5000.

6. Now, stop the simple layer 2 forwarding controller by `Ctrl-C`.

7. And start your port forwarding controller (if you have written your controller then use myPortForwarding in the following command):

   ```
   ./bin/ryu-manager ryu/ext/PortForwarding.py
   ```

8. Repeat the netcat scenario described above. Now, your text should appear on the other terminal of host2 which is listening to port 6000.

9. Stop your port forwarding controller using `Ctrl-C`.

3h. Run a Server Proxy Controller

As our last exercise, instead of diverting the traffic to a different server running on the same host, we will divert the traffic to a server running on a different host and on a different port.

1. Under the `/tmp/ryu/ryu/ext/` directory there are two files: **Proxy.py** and **myProxy.py** that are similar to the previous exercise. Both of these controllers are configured by the configuration file `proxy.config`. Use myProxy.py to write your own proxy controller.

2. On the terminal of `host3` run a netcat server:

```
nc -l 7000
```

3. On your controller host, open the /tmp/ryu/ryu/ext/myProxy.py file, and edit it to implement a controller that will divert traffic destined for `host2` to `host3`. Before you start implementing think about what are the side effects of diverting traffic to a different host.
   - Is it enough to just change the IP address?
   - Is it enough to just modify the TCP packets?

   If you want to see the solution, it's available in file /tmp/ryu/ryu/ext/Proxy.py file.

4. To test your proxy controller run (if you have written your controller then use myProxy in the following command):

```
cd /tmp/ryu
./bin/ryu-manager ryu/ext/Proxy.py
```

5. Go back to the terminal of `host1` and try to connect netcat to `host2` port 5000

```
nc 10.10.1.2 5000
```

6. If your controller works correctly, you should see your text showing up on the terminal of `host3`.

3i. Delete your bridge

Before moving to the next step make sure you delete the bridge you have created, especially if you are using the same reservation for a different exercise:

```
sudo ovs-vsctl del-br br0
```

Previous: Design and Setup

Next: Finish

Intro to OpenFlow Tutorial (OVS) with Ryu Controller

Step 4. Teardown Experiment

After you are done with this experiment release your resources. In the GENI Portal select the slice click on the "Delete" button. If you have used other tools to run this experiment than release resources as described in the Prerequisites for Tutorials on reservation tools pages.

Now you can start designing and running your own experiments!

Prev: Execute

Introduction