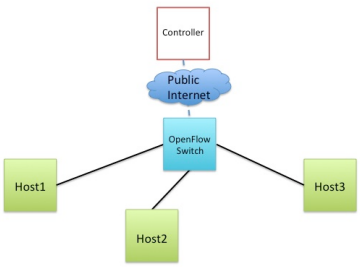
Intro to OpenFlow Tutorial

Overview:

This is a simple OpenFlow tutorial that will guide you how to use the Floodlight Controller in conjunction with Open vSwitch and OpenFlow to showcase some of the OpenFlow capabilities. We are going to perform three different taks:

1. Write a flow that will **duplicate all the traffic** of the OpenFlow switch out a specific port.
2. **TCP Port Forward** controller. Divert all traffic destined to a host on TCP port X to TCP port Y.
3. **Proxy Controller**. Write a flow that will divert all traffic destined to a host on TCP port X to host B on 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 Floodlight ; which is a JAVA based 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

Tools:

- Open vSwitch. OVS will be installed as a part of the resource reservation.
- Floodlight Controller.

Where to get help:

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

Resources:

- Learn more about OpenFlow
- POX wiki
- Learn more about OVS

Tutorial Instructions

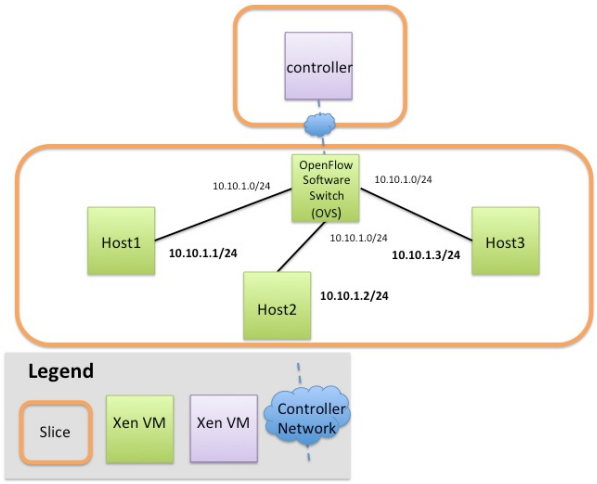
- Part I: Design Setup
  - Step 1: Reserve Resources
    - OpenFlow using Open vSwitch (OVS)
    - OpenFlow using a Hardware Switch
  - Step 2: Configure and Initialize Services
- Part II: Execute
  - Step 3: Execute Experiment
- Part III: Finish
  - Step 4: Teardown Experiment

Intro to OpenFlow Tutorial (OVS)

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 reservation you can use any aggregate.

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.

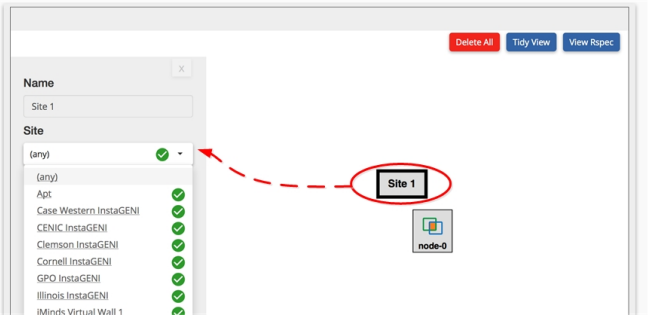
a. **Reserve a VM that runs your OpenFlow controller**.

RSpec: You can use the following url, url: http://www.gpolab.bbn.com/experiment-support/OVSFloodLight/ControllerCustom_request_rspec.xml

a. **Reserve your network**, that includes a VM with OVS installed.

RSpec: You can use the following url, url: http://www.gpolab.bbn.com/experiment-support/OVSFloodLight/OVS_TPOLOGY_request_rspec.xml

You can change the InstaGENI rack at which you reserve your resources by clicking on the node that says **UtahDDC InstaGENI** and selecting any other InstaGENI rack from the menu on the left. You can refer to the figure shown below.



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 the Open vSwitch

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. You will see three interfaces with IP's **10.10.*.***, one for each host. These are the data plane interfaces.
- 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. So now that we know the names of our three **10.10.*.*** interfaces, we can add them as ports to our bridge. These three interfaces are your data interfaces.

```
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>:6653
```

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

Step 4. 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 Floodlight Controller.

4a. Login to your hosts

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

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.

4b. 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`.

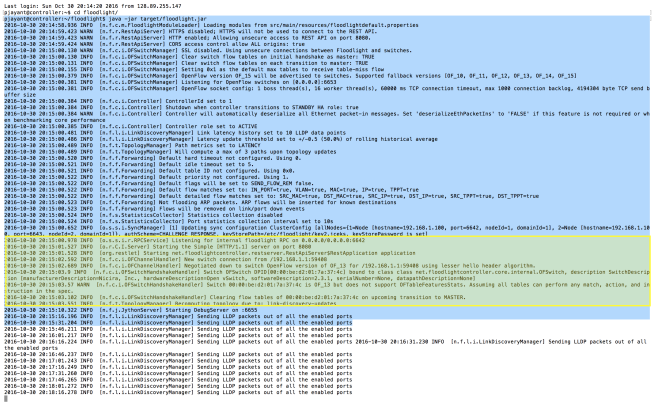
1. First start a ping from `host1` to `host2` , which should timeout, since there is no controller running.

```
ping 10.0.0.2 -c 10
```

2. Start the Floodlight Controller by running the following commands:

```
cd /local/floodlight
java -jar target/floodlight.jar
```

The output should look like this:



3. In the terminal of `host1`, ping `host2` i.e. 10.0.0.2:

```
[pjayant@host1:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=327 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=23.2 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2.08 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=1.71 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=1.77 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=1.74 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=1.69 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=1.55 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=1.60 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=1.67 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=1.73 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=1.50 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=1.82 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=1.69 ms
^C
--- 10.0.0.2 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13019ms
rtt min/avg/max/mdev = 1.501/26.489/327.007/83.532 ms
```

Now the ping should work. You can see that the time for the first ICMP packet is longer than the rest of the ICMP packets. This is because the Open vSwitch consults the controller the first time a packet-in event occurs. The controller then inserts the flow in the Open vSwitch and the switch consults this flow for further packet-in events. Similarly, ping `host 3` i.e. 10.0.0.3 from `host 1`.

4. Go to your Open vSwitch host and take a look at the flows. You should see that your controller installed flows based on the mac addresses of your packets. Enter the following command:

```
pjayant@switch:~$ sudo ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x20000005000000, duration=4.782s, table=0, n_packets=4, n_bytes=392,
 cookie=0x20000004000000, duration=4.790s, table=0, n_packets=4, n_bytes=392,
 cookie=0x0, duration=1275.644s, table=0, n_packets=6, n_bytes=512, idle_age=
```

4c. Look around your OVS switch

1. To see messages go between your switch and your controller, open a new ssh window to your controller node and run tcpdump on the `eth1` interface and on the tcp port that your controller is listening on usually 6653. (You can also run `tcpdump` on the `ovs` control interface if you desire.)

```
sudo tcpdump -i eth0 tcp port 6653
```

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 Floodlight controller by pressing `Ctrl-C`:

```
2016-10-30 21:00:47.333 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLD
2016-10-30 21:01:02.339 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLD
2016-10-30 21:01:17.344 INFO  [n.f.l.i.LinkDiscoveryManager] Sending LLD
^C
pjayant@controller:~/floodlight$
```

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
```

Since you set your switch to "secure" mode, i.e. don't forward packets if the controller fails, you will not see flow table entries. If you see flow table entries, try again after 10 seconds to give the entries time to expire.

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?

4d. 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 6653. To do that type *tcp.port eq 6653* in the filter window, assuming that 6653 is the port that the controller is listening on. And once you have lines, you can choose one of the lines and choose "Decode as ...." and choose the *OFP protocol*.

Make sure that you have a suitable **X.org** software component such as XQuartz on your local machine for Wireshark to work.

4e. Web GUI

The Floodlight Controller comes equipped with a web-based GUI. The GUI can be accessed by pointing your favorite browser to the following URL:

```
http://<controller-ip>:8080/ui/pages/index.html
```

<controller-ip> is the IP address of the control interface of the Controller node(eth0).

4f. Topology Details

Before we insert flows into the Open vSwitch, we are going to need all the details regarding the topology such as the DPID of the Open vSwitch, MAC addresses of the hosts, the port numbers on which the Hosts are connected to the Open vSwitch etc. These details can be found out by issuing the following command in a new Controller terminal:

```
curl http://localhost:8080/wm/device/ | python -m json.tool
```

In the output, we obtain a list of the devices that Floodlight Controller has learned about. Make sure you *ping* Host 2 & Host 3 from Host 1 before you issue this command. My output is pasted

below. **YOURS WILL BE DIFFERENT.**

I have highlighted all the important details you need to note down before you start inserting flows. Note down all the highlighted details for each **Host**. The Host IP's will be of the form 10.0.0.* for Host *.

4f. 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. **Insert the flow to Duplicate Traffic**
   - We are going to duplicate IPv4 traffic from Host 1 destined to Host 2 on Host 3. Open a new *Controller* terminal and type the following flow:

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-1'
```

Scroll all the way to the right to view the complete flow and get rid of the <> brackets when inserting the flow.

To see that duplication is happening, on Host 2 and Host 3 , run:

```
sudo tcpdump -i eth1
```

You should see traffic from host1 to host2 showing up in the tcpdump window for host3 as shown below:

```
pjayant@host3:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol
listening on eth1, link-type EN10MB (Ethernet), capture size 65535
02:48:38.270572 IP Host1-link-1 > Host2-link-2: ICMP echo request,
02:48:38.270639 IP Host3-link-3 > Host1-link-1: ICMP redirect Host2
02:48:38.270654 IP Host1-link-1 > Host2-link-2: ICMP echo request,
```

```
02:48:39.271350 IP Host1-link-1 > Host2-link-2: ICMP echo request,
02:48:39.271390 IP Host3-link-3 > Host1-link-1: ICMP redirect Host2
02:48:39.271403 IP Host1-link-1 > Host2-link-2: ICMP echo request,
02:48:40.272525 IP Host1-link-1 > Host2-link-2: ICMP echo request,
02:48:40.272570 IP Host3-link-3 > Host1-link-1: ICMP redirect Host2
02:48:40.272582 IP Host1-link-1 > Host2-link-2: ICMP echo request,
```

4g. 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. To test your controller we are going to use netcat. Open two terminals of host2. In one terminal run:

   ```
   nc -l 5000
   ```

   and in the other terminal run

   ```
   nc -l 6000
   ```

2. We will check the normal functionality before the flow for a Port Forwarding Controller is inserted. Go to the terminal of host1 and connect to host2 at port 5000:

   ```
   nc 10.0.0.2 5000
   ```

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

4. Now, we insert the flow for a Port Forwarding Controller:

   ```
   curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-2","pri
   ```

5. In the previous step, we inserted a flow to forward TCP traffic from Host 1 destined to Host 2 at port 5000 to port 6000. But Host 1 still thinks it is speaking to Host 2 at port 5000. So we need to insert a flow to handle traffic from Host 2 Port 6000 for a seamless transition.

   ```
   curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-3","pri
   ```

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

4h. 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. On the terminal of `host3` run a netcat server:

   ```
   nc -l 6000
   ```

3. On the controller host, we will insert a flow 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?

4. Insert the following flow in the Controller terminal to implement a Server Proxy Controller:

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-4","pri
```

5. In the previous step, we inserted a flow to forward TCP traffic from Host 1 destined to Host 2 at port 5000 to Host 3 at port 6000. But Host 1 still thinks it is speaking to Host 2 at port 5000. So we need to insert a flow to handle traffic from Host 3 Port 6000 for a seamless transition.

```
curl -X POST -d '{"switch":"<DPID OF OPEN vSWITCH>","name":"flow-5","pri
```

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

```
nc 10.0.0..2 5000
```

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

4i. 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
```

Prev: Design and Setup for OVS

Next: Finish

Attachments

- Open vSwitch.png (0.9 MB) - added by *pjayanth@bbn.com* 3 weeks ago.
- Topology_Details.png (235.4 KB) - added by *pjayanth@bbn.com* 13 days ago.

Intro to OpenFlow Tutorial

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