

Using Geni to Create Network Components for Cloud Infrastructure

Create Geni Account to proceed

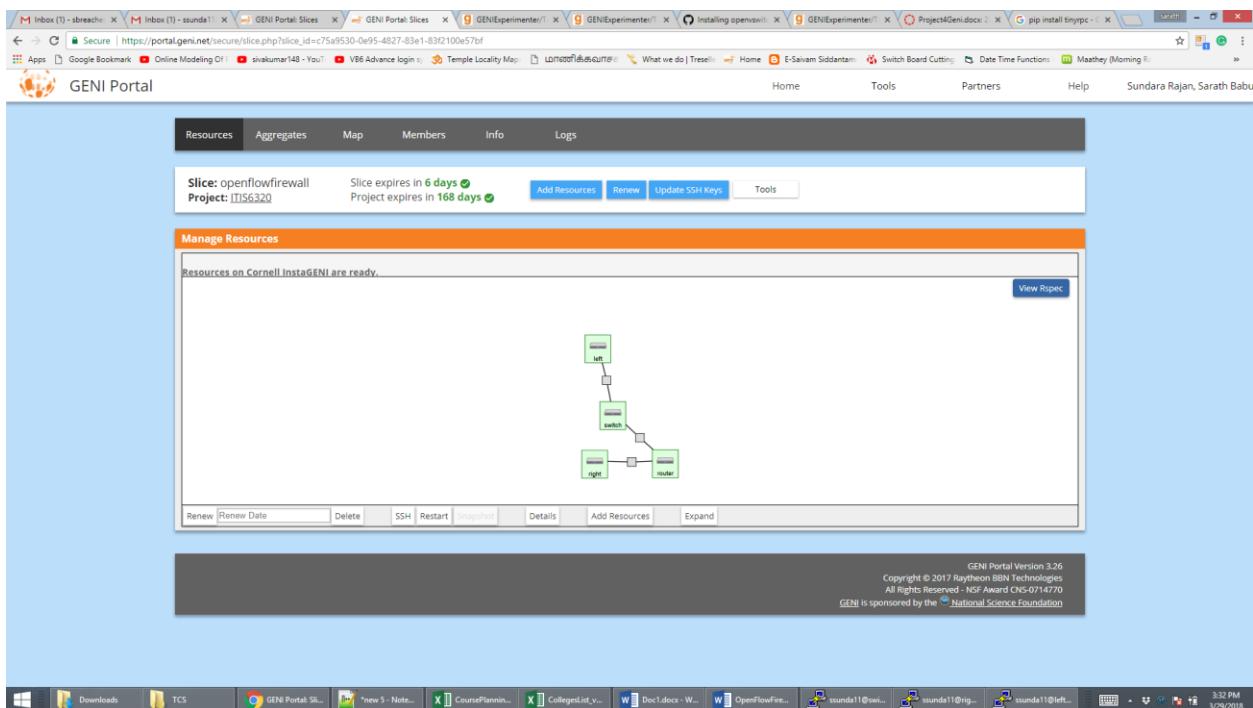
Follow the Instruction from GENIExperiments Tutorials

<http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/OpenFlowNetworkDevices>

FIREWALL Configuration

Follow Steps to from below tutorial page

<http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/OpenFlowNetworkDevices/Firewall>



Ping to left from right node

```

ssunda11@right: ~
ping -c 4 10.10.11.1
PING 10.10.11.1 (10.10.11.1) 56(84) bytes of data.
64 bytes from left-left-switch (10.10.10.1): icmp_seq=1 ttl=63 time=1.33 ms
64 bytes from left-left-switch (10.10.10.1): icmp_seq=2 ttl=63 time=1.33 ms
64 bytes from left-left-switch (10.10.10.1): icmp_seq=3 ttl=63 time=1.33 ms
64 bytes from left-left-switch (10.10.10.1): icmp_seq=4 ttl=63 time=1.33 ms
--- left-left-switch ping statistics ---
16 packets transmitted, 16 received, +0 errors, 0% packet loss, time 151696ms
rtt min/avg/max/mdev = 1.117/1.19/1.000.740/133.488 ms, pipe 3

ssunda11@left: ~
ping -c 4 10.10.11.1
PING 10.10.11.1 (10.10.11.1) 56(84) bytes of data.
64 bytes from right-right-switch (10.10.10.1): icmp_seq=1 ttl=63 time=1.33 ms
64 bytes from right-right-switch (10.10.10.1): icmp_seq=2 ttl=63 time=1.33 ms
64 bytes from right-right-switch (10.10.10.1): icmp_seq=3 ttl=63 time=1.33 ms
64 bytes from right-right-switch (10.10.10.1): icmp_seq=4 ttl=63 time=1.33 ms
--- right-right-switch ping statistics ---
16 packets transmitted, 16 received, +0 errors, 0% packet loss, time 151696ms
rtt min/avg/max/mdev = 1.117/1.19/1.000.740/133.488 ms, pipe 3

```

OpenFlow Firewall

This exercise is based on as assignment by Sonia Farmy, Ethan Blanton and Srivarsa Gangam of Purdue University.

For this experiment we will run an OpenFlow Firewall.

- Log into **right**
- Install oslo.config
- Run a simple learning switch controller:

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

- To verify simple connectivity, log into **right** in a separate ssh terminal and ping **left**

```
ping left
```

Notice the printsout of the ryu simple switch controller.

- On the **switch** ssh session, make your switch into a firewall by downloading and running the appropriate Ryu controller:

```
sudo ovs-ofctl del-flows br0
#(optional) Notice that you can no longer ping left from right.
```

- On the **switch** ssh session, make your switch into a firewall by downloading and running the appropriate Ryu controller:

```
cd
wget https://github.com/GENI-NFV/geni-tutorials/raw/master/OpenFlowNetworkDeviceFirewall/gpo-ryu-firewall.tar.gz
tar xfz gpo-ryu-firewall.tar.gz
cd gpo-ryu-firewall/
./ryu/ryu/bin/ryu-manager simple_firewall.py
```

WARNING: If at some point your controller prints an error, kill it (ctr-c) and start it again.

- On the right ssh session run a nc server:

```
nc -l 5001
```

- Log into **left** and run a nc client:

```
nc 10.10.11.1 5001
```

```

ssunda11@right: ~
ping -c 4 10.10.11.1
PING 10.10.11.1 (10.10.11.1) 56(84) bytes of data.
64 bytes from left-left-switch (10.10.10.1): icmp_seq=1 ttl=63 time=1.00 ms
64 bytes from left-left-switch (10.10.10.1): icmp_seq=2 ttl=63 time=1.00 ms
64 bytes from left-left-switch (10.10.10.1): icmp_seq=3 ttl=63 time=1.00 ms
64 bytes from left-left-switch (10.10.10.1): icmp_seq=4 ttl=63 time=1.00 ms
--- left-left-switch ping statistics ---
16 packets transmitted, 16 received, +0 errors, 0% packet loss, time 151696ms
rtt min/avg/max/mdev = 1.117/1.19/1.000.740/133.488 ms, pipe 3

ssunda11@left: ~
ping -c 4 10.10.11.1
PING 10.10.11.1 (10.10.11.1) 56(84) bytes of data.
64 bytes from right-right-switch (10.10.10.1): icmp_seq=1 ttl=63 time=1.00 ms
64 bytes from right-right-switch (10.10.10.1): icmp_seq=2 ttl=63 time=1.00 ms
64 bytes from right-right-switch (10.10.10.1): icmp_seq=3 ttl=63 time=1.00 ms
64 bytes from right-right-switch (10.10.10.1): icmp_seq=4 ttl=63 time=1.00 ms
--- right-right-switch ping statistics ---
16 packets transmitted, 16 received, +0 errors, 0% packet loss, time 151696ms
rtt min/avg/max/mdev = 1.117/1.19/1.000.740/133.488 ms, pipe 3

```

Exercise 2: Firewall Configuration

This exercise is based on as assignment by Sonia Farmy, Ethan Blanton and Srivarsa Gangam of Purdue University.

For this experiment we will run an OpenFlow Firewall.

- Log into **right**
- Install oslo.config
- Run a simple learning switch controller:

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

- To verify simple connectivity, log into **right** in a separate ssh terminal and ping **left**

```
ping left
```

Notice the printsout of the ryu simple switch controller.

- On the **switch** ssh session, make your switch into a firewall by downloading and running the appropriate Ryu controller:

```
sudo ovs-ofctl del-flows br0
#(optional) Notice that you can no longer ping left from right.
```

- On the **switch** ssh session, make your switch into a firewall by downloading and running the appropriate Ryu controller:

```
cd
wget https://github.com/GENI-NFV/geni-tutorials/raw/master/OpenFlowNetworkDeviceFirewall/gpo-ryu-firewall.tar.gz
tar xfz gpo-ryu-firewall.tar.gz
cd gpo-ryu-firewall/
./ryu/ryu/bin/ryu-manager simple_firewall.py
```

WARNING: If at some point your controller prints an error, kill it (ctr-c) and start it again.

- On the right ssh session run a nc server:

```
nc -l 5001
```

- Log into **left** and run a nc client:

```
nc 10.10.11.1 5001
```

- Type some text in **left** and it should appear in **right** and vice versa.
- In the terminal you should see messages similar to those below about the flow being passed or not:

```
Extracted rule: {'sport': '57430', 'dport': '5001', 'sip': '10.10.10.1', 'dip': '10.10.11.1'}
Allow Connection rule: {'sport': '5001', 'dport': '57430', 'sip': '10.10.11.1', 'dip': '10.10.10.1', 'sport': 'any'}
```

- Type **ctr-c** in **left** to stop to kill nc.
- Run a nc server on port 5002, then 5003.
- Compare the observed behavior to the contents of `~/.gpo-ryu-firewall/ru.conf`. Does the behavior match the configuration file?
- Stop the Firewall controller and run a simple switch controller. Is there any traffic being blocked now? Don't forget to delete the flows after you stop the controller
- Feel free to modify the configuration file to allow more traffic.

Return to the main page



working commands

in switch node

```
sudo pip install tiny rpc  
sudo pip install oslo.config  
cd /tmp/ryu  
.bin/ryu-manager --verbose ryu/app/simple_switch.py
```

in right node

```
ping left
```

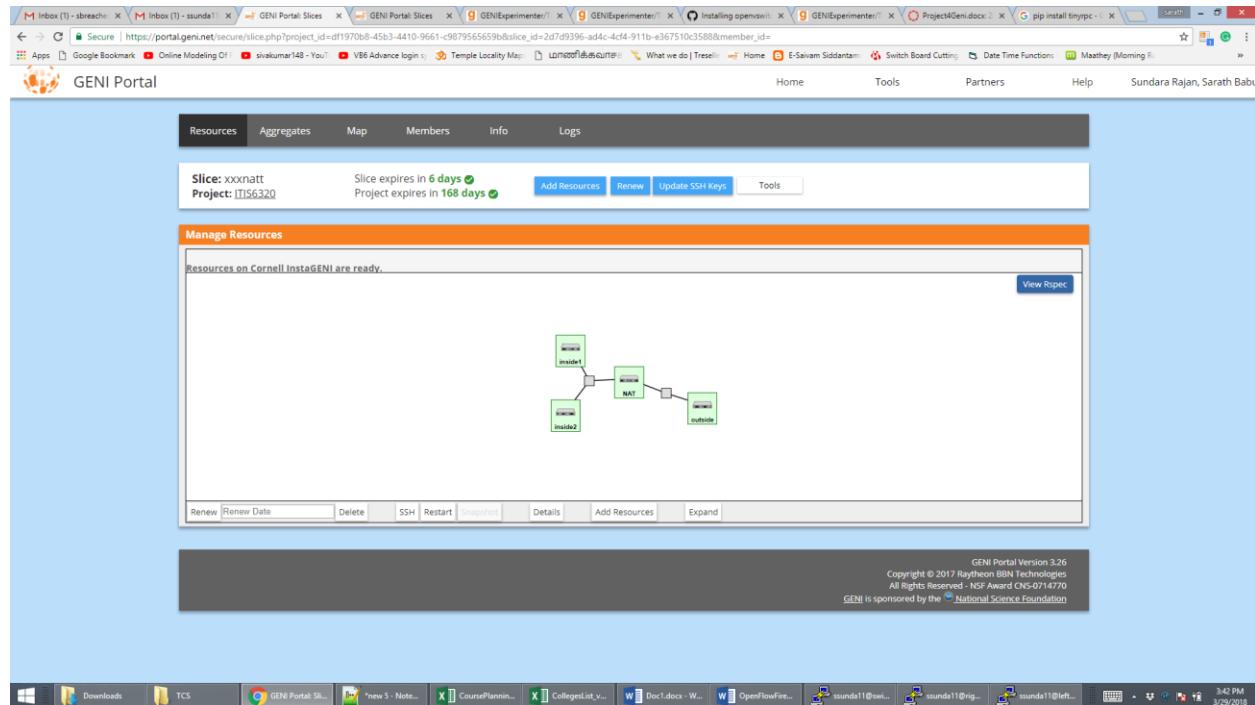
in switch node

```
sudo ovs-ofctl del-flows br0  
cd  
wget https://github.com/GENI-NSF/geni-tutorials/raw/master/OpenFlowNetworkDeviceFirewall/gpo-ryu-firewall.tar.gz  
tar xvfz gpo-ryu-firewall.tar.gz  
cd gpo-ryu-firewall/  
/tmp/ryu/bin/ryu-manager simple_firewall.py
```

NAT Configuration

Follow Steps in below tutorial

<http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/OpenFlowNetworkDevices/NAT>



Pre-run steps before executing the commands

IN NAT NODE

```
sudo apt-get update && sudo apt-get install --only-upgrade openssl  
sudo apt-get update && sudo apt-get install --only-upgrade libssl-dev  
wget https://bootstrap.pypa.io/ez_setup.py -O - | sudo python  
sudo -s easy_install pip==7.1.2  
sudo pip install oslo.config  
cd /tmp/ryu/  
ls  
sudo python setup.py install  
sudo pip install tinyrpc  
.bin/ryu-manager nat.py
```

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 take a look in this page. Once you have logged in, follow the rest of the instructions.

1.1a Install some software

On the NAT node run:

```
 wget https://bootstrap.pysc.io/ez_setup.py -O - | sudo python
 sudo pip install oslo.config
```

1.2 Test reachability

- First we start a ping from inside1 to inside2, which should work since they are both inside the same LAN.

```
inside1:$ ping 192.168.0.3 -c 10
```

- Then we start a ping from outside to inside1, which should timeout as the interface is not reachable.

```
outside:$ ping 192.168.0.2 -c 10
```

- Similarly, we cannot ping from inside2 to outside (128.128.128.2).

- You can also use Netcat (nc) to test reachability of TCP and UDP, as in the following examples.

2 Start controller to enable NAT

2.1 Access a server from behind the NAT

You can try to write your own controller to implement NAT. However, we've provided one for you.

- Start the controller on NAT host:

```
nat$ cd /tmp/ryu/
nat$ ./bin/ryu-manager nat.py
```

You should see output similar to following log after the switch is connected to the controller.

```
loading app nat.py
loading app ryu.controller.dpset
loading app ryu.services.alarms.alarm_handler
instantiating app ryu.controller.dpset of DPSet
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app net.py of Net
switch connected ryu.controller.controller.Datapath object at 0x21B5210>
```

- On outside, we start a nc server:

```
outside:$ nc -l 6666
```

and we start a nc client on inside1 to connect to it:

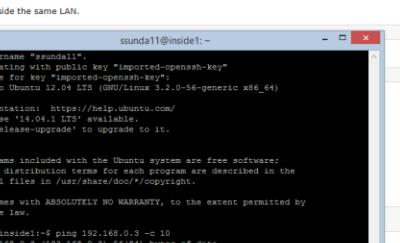
```
inside1:$ nc 128.128.128.2 6666
```

c. You should now be able to type a message in either nc session and see it appear in the other session. Now, try the same thing between outside and inside2.

- On nat, where you started the controller, you should see a log similar to:

```
Created mapping 192.168.0.3 31996 to 128.128.129.1 59997
```

Note that there should be only one log per connection, because the rest of the communication will re-use the mapping.



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

1.1 Install some software

On the NAT node run:

```
 wget https://bootstrap.pypa.io/ez_setup.py -O - | sudo python -
```

1.2 Test reachability

- First we start a ping from inside1 to inside2, which
- outside1:~\$ ping 192.168.0.3 -c 10
- Then we start a ping from outside to inside1, which
- outside2:~\$ ping 192.168.0.2 -c 10
- Similarly, we cannot ping from inside2 to outside1
- You can also use Netcat (-nc) to test reachability of

2 Start controller to enable NAT

2.1 Access a server from behind the NAT

You can try to write your own controller to implement

a. Start the controller on NAT host:

```
 natd:~$ cd /tmp/ryu
 natd:~$ ./bin/rpc_manager nat
```

You should see output similar to fc

```
 loading app nat ...
 loading app ryu.controller ...
 loading app ryu.controller.controller ...
 instantiating app ryu.controller.controller ...
 instantiating app ryu.controller.ofswitch ...
 instantiating app ryu.controller.ofswitch ...
 switch connected ryu.controller.ofswitch
```

D. On outside, we start a nc serve

```
 outside2:~$ nc -l 6666
```

and we start a nc client on inside1:

```
 inside1:~$ nc 128.128.128.2 6666
```

c. You should be able to type

d. On nat, where you started the controller

```
 Create mapping 192.168.0.3
```

Note that there should be only one log per connection, because the rest of the communication will re-use the mapping.

ssundal1@inside1:~

```
Collecting metadata>0.7.18 (from oslo.config)
  Downloading metadata-0.7.18-py3.py3-none-any.whl (1.00kB)
    100% |████████████████████████████████| 1.00kB 1.25kB/s
Collecting pbr>1.1.0,<2.0.0 (from debcollector>1.2.0->oslo.config)
  Downloading pbr-1.4.0-py3.py3-none-any.whl (100kB)
    100% |████████████████████████████████| 102KB 3.2MB/s
Collecting wrapt>1.7.0 (from debcollector>1.2.0->oslo.config)
  Downloading wrapt-1.11.1.tar.gz
Collecting funcsigs>1.0.0 (from debcollector>1.2.0->oslo.config)
  Downloading funcsigs-1.5.3-py3.py3-none-any.whl (6.89kB)
Collecting babel>2.6.1,<3.4 (from debcollector>1.2.0->oslo.config)
  Downloading Babel-2.6.3-py3.py3-none-any.whl (6.89kB)
  100% |████████████████████████████████| 6.89kB 38KB/s
Collecting pytz>2018.3 (from Babel>2.6.1->oslo.config)
  Downloading pytz-2018.3-py3.py3-none-any.whl (509kB)
  100% |████████████████████████████████| 512KB 674KB/s
Install for collected packages: pygments, six, pbr, wrapt, funcsigs, debcollector, stevedore, rfc3986, enum34, pytz, Babel, oslo.i18n, netaddr, stevedore
Running setup.py install for PYTAN
  Running setup.py install for wrapt
    successfully installed wrapt-1.12.2 debcollector-1.1.0.0 netaddr-1.1.6 funcsigs-1.0.2 netaddr-0.7.19 oslo.config-6.0.0 oslo.i18n-3.2.0 pytz-2018.3 pbr-2018.3 rfc3986-1.1.0.i18n-1.1.0 stevedore-1.2.0 wrapt-1.10.1
    /usr/local/lib/python2.7/dist-packages/pip/_internal/_internal.py:74: InsecurePlatformWarning: A t
    SSLContext object is not available. This prevents urllib3 from configuring SSL appropriately and may cause certain SSL connections to fail
    more details and help: https://urllib3.readthedocs.org/en/latest/security.html#insecureplatformwarning
    InsecurePlatformWarning
You are using pip version 7.1.2, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
ssundal1@inside1:~$ rm -rf /tmp/ryu/
ssundal1@inside1:~$
```

ssundal1@outside1:~

```
Using username "ssundal1".
Authenticating with public key "imported-openid-key"
Passphrase for key "imported-openid-key":
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-56-generic x86_64)

 * Documentation: https://help.ubuntu.com/
 New release '14.04.1 LTS' is available.
 Run 'sudo apt-get update' to upgrade to it.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ssundal1@outside1:~$ ping 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=3 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=4 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=5 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=6 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=7 ttl=64 time=1.06 ms
64 bytes from 192.168.0.3: icmp_seq=8 ttl=64 time=1.09 ms
64 bytes from 192.168.0.3: icmp_seq=9 ttl=64 time=1.05 ms
64 bytes from 192.168.0.3: icmp_seq=10 ttl=64 time=1.05 ms

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

ssundal1@outside1:~$ ping 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
--- 192.168.0.2 ping statistics ---
10 packets transmitted, 0 received, 0% packet loss, time 900ms
ssundal1@outside1:~$
```

```
nat># cd /tmp/rwu/  
nat># ./bin/rwu-manger net  
  
You should see output similar to following:  
  
loading app nat...  
loading app controller...  
loading app ryu.controller...  
instantiating app ryu.controller...  
instantiating app net-port...  
instantiating app switch...  
instantiating app ryu.controller.ofswitch...  
  
D. On outside, we start a nc service  
  
outside>$ nc -l 6666  
  
and we start a nc client on inside.  
  
inside>$ nc 128.128.128.3 6666  
  
c. You should now be able to type  
d. On nat, where you started the nc  
  
Created mapping 192.168.0.3  
  
Note that there should be only one log per connection, because the rest of the communication will re-use the mapping.
```

The screenshot shows a Linux desktop environment with several open windows:

- Terminal 1 (top left):** Shows the command `netcat -l -p 1234` being run on the host machine.
- Terminal 2 (top center):** Shows the command `curl http://192.168.0.2:1234` being run from the guest machine.
- Terminal 3 (top right):** Shows the command `netcat -l -p 1234` being run on the guest machine.
- Terminal 4 (bottom left):** Shows the command `netcat -l -p 1234` being run on the host machine.
- Terminal 5 (bottom center):** Shows the command `curl http://192.168.0.2:1234` being run from the guest machine.
- File Browser (bottom right):** Shows a file named `httpd.conf` with the content: `Listen 1234`.

The desktop environment includes icons for the Dash, Home, Downloads, TCS, and GIMP. The taskbar at the bottom shows the names of the open windows.

Note that there should be only one log per connection, because the rest of the communication will re-use the mapping.



You should see output similar to following log after t

```
loading app nat.py
loading app ryu.controller.dpset
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.dpset ofp
instantiating app ryu.controller.ofp_handler
instantiating app nat.py of NAT
switch connected cryo.controller.controller.

b. On outside, we start a nc server:
outside1$ nc -l 6666
and we start a nc client on inside1 to connect it:
inside1$ nc 128.128.128.2 6666
c. You should be able to type a message in eth
d. On nat, where you started the controller, you sho
    [Created mapping 192.168.0.3 31596 to 128.128
Note that there should be only one log per connector
```

3 Handle ARP and ICMP

One common mistake experimenters make, when writing OpenFlow controllers, is forgetting to handle ARP and ICMP messages causing their controllers to not work as expected.

```
ssunda11@inside1:~ - x ssunda11@outside1:~ - x
ssunda11@inside2:~ - x ssunda11@inside2:~ - x

bytes from 128.128.128.2 icmp req51 ct1=63 time=0.55 ms
bytes from 128.128.128.2 icmp req55 ct1=63 time=5.59 ms
bytes from 128.128.128.2 icmp req56 ct1=63 time=5.60 ms
bytes from 128.128.128.2 icmp req57 ct1=63 time=5.61 ms
bytes from 128.128.128.2 icmp req58 ct1=63 time=5.63 ms
bytes from 128.128.128.2 icmp req59 ct1=63 time=7.00 ms
bytes from 128.128.128.2 icmp req60 ct1=63 time=6.19 ms
bytes from 128.128.128.2 icmp req61 ct1=63 time=5.60 ms
bytes from 128.128.128.2 icmp req62 ct1=63 time=5.48 ms
bytes from 128.128.128.2 icmp req63 ct1=63 time=5.47 ms
bytes from 128.128.128.2 icmp req64 ct1=63 time=4.30 ms
bytes from 128.128.128.2 icmp req65 ct1=63 time=5.46 ms
bytes from 128.128.128.2 icmp req66 ct1=63 time=5.13 ms
bytes from 128.128.128.2 icmp req67 ct1=63 time=5.45 ms
bytes from 128.128.128.2 icmp req68 ct1=63 time=5.45 ms
bytes from 128.128.128.2 icmp req69 ct1=63 time=5.59 ms
bytes from 128.128.128.2 icmp req70 ct1=63 time=5.16 ms
bytes from 128.128.128.2 icmp req71 ct1=63 time=5.41 ms
bytes from 128.128.128.2 icmp req72 ct1=63 time=5.23 ms

--- 128.128.128.2 ping statistics ---
72 packets transmitted, 72 received, 0% packet loss, time 7114ms
min/avg/max/mdev = 0.593/5.456/7.004/0.545 ms
ssunda11@inside1:~
```

```
ssunda11@inside2:~ - x
bytes from 128.128.128.2 icmp req51 ct1=63 time=5.31 ms
bytes from 128.128.128.2 icmp req55 ct1=63 time=5.91 ms
bytes from 128.128.128.2 icmp req56 ct1=63 time=5.30 ms
bytes from 128.128.128.2 icmp req57 ct1=63 time=5.13 ms
bytes from 128.128.128.2 icmp req58 ct1=63 time=5.20 ms
bytes from 128.128.128.2 icmp req59 ct1=63 time=5.30 ms
bytes from 128.128.128.2 icmp req60 ct1=63 time=5.29 ms
bytes from 128.128.128.2 icmp req61 ct1=63 time=5.22 ms
bytes from 128.128.128.2 icmp req62 ct1=63 time=5.28 ms
bytes from 128.128.128.2 icmp req63 ct1=63 time=5.20 ms
bytes from 128.128.128.2 icmp req64 ct1=63 time=5.28 ms
bytes from 128.128.128.2 icmp req65 ct1=63 time=4.14 ms
bytes from 128.128.128.2 icmp req66 ct1=63 time=5.24 ms
bytes from 128.128.128.2 icmp req67 ct1=63 time=5.24 ms
bytes from 128.128.128.2 icmp req68 ct1=63 time=5.58 ms
bytes from 128.128.128.2 icmp req69 ct1=63 time=5.64 ms
bytes from 128.128.128.2 icmp req70 ct1=63 time=5.64 ms
bytes from 128.128.128.2 icmp req71 ct1=63 time=5.65 ms
bytes from 128.128.128.2 icmp req72 ct1=63 time=5.44 ms

128.128.128.2 ping statistics ---
72 packets transmitted, 47 received, 0% packet loss, time 46075ms
min/avg/max/mdev = 4.148/5.605/7.870/0.887 ms
ssunda11@inside2:~
```

References: -

<http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/OpenFlowNetworkDevices>

<https://portal.geni.net>