CSCI 558L FALL 2011 LAB#9

ANDREW GOODNEY - GOODNEY@USC.EDU - YOUNG CHO - YOUNGCHO@ISI.EDU

1. OpenFlow Switch on NetFPGA Tutorial

In this lab we will experiment with an OpenFlow switch. OpenFlow is a networking technology that separates the control plane and data plane of networking switching so that many fast, but dumb switches (data plane) can be controlled by a centralized and smart controller (control plane). In OpenFlow these two components are an OpenFlow switch and a software controller called NOX. We will be using the NetFPGA as the OpenFlow switch.

The essential idea of OpenFlow is as follows: when a packet arrives at a switch port that does not match an existing flow table entry, the packet is forwarded to the NOX controller. The NOX controller examines the packet and makes a decision about how to handle the packet. The NOX controller then forwards this decision back to the switch in the form of a flow table entry. Further packets matching this flow table entry are forwarded quickly without involvement of the NOX controller.

For this lab you will first get the OpenFlow switch running on the NetFPGA node, the NOX controller running on another. The NOX controller will run simple code that makes the OpenFlow NetFPGA operate as a simple switch. You will then create more sophisticated code for the NOX controller that will allow the OpenFlow NetFPGA operate as a learning switch and a firewall.

For more information please see:

`http://www.openflow.org/wk/index.php/OpenFlow_Tutorial`

Also, please note: do not copy-paste the NS file or Python file from the PDF. Doing so can cause problems, as not all of the characters are copied correctly. They are included here for your reference. Please use the files as downloaded from the Moodle.

1.1. Installing OpenFlow on the NetFPGA.

We will first download some software into your home directory on `users.isi.deterlab.net`. In the following instructions, when you see `<USER>` substitute your DETER username. Start by connecting over `ssh` to `users.isi.deterlab.net`

- First make a subdirectory in your home account and download autoconf:
  - `> mkdir OFS`
  - `> cd OFS`
  - `> wget http://ftp.gnu.org/gnu/autoconf/autoconf-2.63.tar.gz`
  - `> tar xvzf autoconf-2.63.tar.gz`
- Download OpenFlow from repository
  - `> git clone git://openflow.org/openflow.git`
  - `> cd openflow`
  - `> git checkout -b 1.0.0-netfpga origin/devel/tyabe/1.0.0-netfpga`
  - `> ./boot.sh`
  - `> cd hw-lib/nf2`
  - `> wget http://openflow.org/downloads/netfpga/openflow_switch.bit.100_3.tar.gz`
  - `> tar xfvz openflow_switch.bit.100_3.tar.gz`
- Install autoconf and OpenFlow on the NetFPGA node.
  - `ssh` to the NetFPGA host: `> ssh control.<EXPNAME>.usc558l`
  - `> sudo yum -y install git automake pkgconfig libtool gcc`

  - `> cd /tmp`
  - `> cp -r /users/<USER>/OFS/autoconf-2.63 .`
  - `> cd autoconf-2.63`
  - `> ./configure --prefix=/usr`
  - `> make`
  - `> sudo make install`
  - `> cd /tmp`
  - `> cp -r /users/<USER>/OFS/openflow .`
  - `> cd openflow`
  - `> ./boot.sh`
  - `> ./configure --enable-hw-lib=nf2`
  - `> make`
  - `> sudo make install`
- Now we will download the bitfile to the NetFPGA so it will act as an OpenFlow switch. A bitfile is the runtime configuration file that is used to program the FPGA to act in some manner, here an OpenFlow switch. Don't worry too much about this, just remember the NetFPGA is runtime configurable network hardware. These commands are performed on the NetFPGA control node (like the last section). The communication between the OpenFlow switch and the NOX controller takes place over TCP/IP. In this experiment we are using the existing DETER control network, so in the last command substitute the control network IP address of your `NOX` node for `192.168.X.X`. You can find this address in the DETER web interface, or by running `ifconfig` on the `NOX` node. The third command is to be entered on one line, but has been line-broken here for clarity. The last command will print a connection error, but remain in a loop trying to connect to the NOX controller.
  - `> sudo /usr/local/netfpga/lib/scripts/cpci_reprogram/cpci_reprogram.pl --all`
  - `> nf_download /tmp/openflow/hw-lib/nf2/openflow_switch.bit`
  - `> sudo ofdatapath punix:/var/run/test -d 000000000001`
    `-i nf2c0,nf2c1,nf2c2,nf2c3 &`
  - `> sudo ofprotocol unix:/var/run/test tcp:<192.168.X.X>:6633`
- The final command above will loop trying to connect to NOX, let it continue looping and open a new window. At this point the OpenFlow switch is running on your NetFPGA node, but it can not do anything yet. Use your new window to SSH to `users.` We will download some dependencies and then log into the `NOX` node to install the NOX controller software.
  - `> cd OFS`
  - `> git clone git://noxrepo.org/nox`
  - Now ssh to your `NOX` node: `> ssh nox.<EXPNAME>.usc558l`
  - Become root: `> sudo su -`
  - `> apt-get update`
  - `> apt-get install autoconf automake build-essential doxygen doxypy g++ git-core`
    `libboost-dev libboost-filesystem-dev libboost-test-dev libpcap-dev libssl-dev`
    `libtool make python-dev python-simplejson python-twisted swig`
  - `> /usr/testbed/bin/mkextrafs /mnt`
  - `> cd /mnt`
  - `> chmod 777 /mnt`
  - `> cp -r /users/<USER>/OFS/nox .`
  - `> cd nox`
  - `> ./boot.sh`
  - `> mkdir build`
  - `> cd build`

- `> ../configure`
- `> make -j 5`
- Now we verify that the NOX build was successful - the help page should be displayed
- `> cd src/`
- `> ./nox_core -h`
- If the above command shows you the help properly, that means you are ready. Now run your NOX controller.
- `> ./nox_core -v -i ptcp:  pyswitch`

Once NOX and the switch are connected, you will see this line: `DBG: Registering switch with DPID = 1`. The NOX controller can control several switches. There is a secure data path between each switch and NOX. `DPID = 1` refers to this data path ID. You should also see output in your OpenFlow switch (the NetFPGA control node) window indicating that the switch has connected to the NOX controller.

At this point you should be able to ping between the nodes of your experiment. Try this out while observing the two windows. For every ping packet you should see an event in each window. What is happening is that the OpenFlow switch does not have a flow table entry for the ping packet, so it forwards it to the NOX controller. The pyswitch module acting as the "brains" makes a simple decision and tells the switch to forward the packet out all of the ports on the switch. This is the so-called "hub" behavior and results in the packet being delivered to the correct host, however at the cost of it being duplicated several times.

2. Creating a Learning Switch and Firewall

The above command starts the NOX controller with a python script as the "brains." This file is located in `/mnt/nox/build/src/nox/coreapps/examples/pyswitch.py`. To complete the rest of the lab you will be modifying this file. Start by overwriting this file with the one provided. The file provided acts as a hub, where every incoming packet is forwarded to all other switch ports. This starter file also does not install a flow on the OpenFlow switch, so every packet is forwarded to the NOX controller. You must make modifications to make it act as a learning switch. A learning switch is one which learns the MAC addresses of the computers connected to each port. That way when a packet comes into the switch, the packet is only forwarded to one port. Of course the proper way to do this is by adding a flow to the flow table on the NetFPGA OpenFlow switch. You may want to read up on how an Ethernet switch works, but the essential idea is that based on the source MAC address seen when a packet enters the switch, a switch can learn which end host MAC addresses are connected to which ports. This information can be used to forward layer-2 packets out the proper port. Packets which contain an unknown destination address are forwarded out all switch ports.

The file `/mnt/nox/build/src/nox/coreapps/examples/pyswitch.py` is actually a symlink to the original source file. A few steps are necessary to ensure you are updating the proper file, and thus activating your changes:

- `> cd /mnt/nox/build/src/nox/coreapps/examples/`
- `> rm pyswitch.py`

If you go back to `/mnt/nox/build/src/` and try running NOX again, you will get an error stating that the pyswitch module can not be found. This verifies that you are ready to make modifications to the pyswitch.py file. Return to `/mnt/nox/build/src/nox/coreapps/examples/` and make your modifications starting with the pyswitch.py provided. You must always run `nox_core` from `/mnt/nox/build/src/`.

For the second task, you'll need to modify the learning switch to include fire-walling capabilities. For the lab, make the switch block flows that match the following case: it is a UDP flow and it is directed to port 9999. This way you can use iperf to test your firewall. TCP iperfs would never be blocked, while a UDP iperf to port 9999 would be blocked. All other UDP iperfs would be allowed.

There are two commands you can run on the NetFPGA control node to see details about the operation of your OpenFlow switch. If you are connected with SSH to the control node:

- To see details about the switch: `$ dpctl show tcp:127.0.0.1:6634`
- To see the flows: `$ dpctl dump-flows tcp:127.0.0.1:6634`

Listing 1. Lab #9 NS file

```
# USC558L Lab 9 NS File. Creates a network with 4 nodes and a NetFPGA host.
source tb_compat.tcl
set ns [new Simulator]

set nfrouter [$ns node]
tb-set-hardware $nfrouter netfpga2

set control [$ns node]
tb-bind-parent $nfrouter $control

# Create end nodes
set nox [$ns node]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

set lan0 [$ns make-lan "$nfrouter $n0" 1000Mb 0ms]
tb-set-ip-lan $nfrouter $lan0 10.1.1.1
tb-set-ip-lan $n0 $lan0 10.1.1.5
tb-fix-interface $n0 $lan0 nf2c0

set lan1 [$ns make-lan "$nfrouter $n1" 1000Mb 0ms]
tb-set-ip-lan $nfrouter $lan1 10.1.1.2
tb-set-ip-lan $n1 $lan1 10.1.1.6
tb-fix-interface $n1 $lan1 nf2c1

set lan2 [$ns make-lan "$nfrouter $n2" 1000Mb 0ms]
tb-set-ip-lan $nfrouter $lan2 10.1.1.3
tb-set-ip-lan $n2 $lan2 10.1.1.7
tb-fix-interface $n2 $lan2 nf2c2

set lan3 [$ns make-lan "$nfrouter $n3" 1000Mb 0ms]
tb-set-ip-lan $nfrouter $lan3 10.1.1.4
tb-set-ip-lan $n3 $lan3 10.1.1.8
tb-fix-interface $n3 $lan3 nf2c3

$ns rtproto Static

$ns run
```

Listing 2. Lab #9 pyswitch.py

```
# Tutorial Controller
# Starts as a hub, and your job is to turn this into a learning switch.

import logging

from nox.lib.core import *
import nox.lib.openflow as openflow
from nox.lib.packet.ethernet import ethernet
from nox.lib.packet.packet_utils import mac_to_str, mac_to_int

log = logging.getLogger('nox.coreapps.tutorial.pytutorial')


class pytutorial(Component):

    def __init__(self, ctxt):
        Component.__init__(self, ctxt)
        # Use this table to store MAC addresses in the format of your choice;
        # Functions already imported, including mac_to_str, and mac_to_int,
        # should prove useful for converting the byte array provided by NOX
        # for packet MAC destination fields.
        # This table is initialized to empty when your module starts up.
        self.mac_to_port = {} # key: MAC addr; value: port
    def learn_and_forward(self, dpid, inport, packet, buf, bufid):
        """Learn MAC src port mapping, then flood or send unicast."""

        # Initial hub behavior: flood packet out everything but input port.
        # Comment out the line below when starting the exercise.
        self.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD, inport)

        # Starter psuedocode for learning switch exercise below: you'll need to
        # replace each pseudocode line with more specific Python code.

        # Learn the port for the source MAC
        #self.mac_to_port = <fill in>
        #if (destination MAC of the packet is known):
            #Send unicast packet to known output port
            #self.send_openflow( <fill in params> )
            # Later, only afte learning controller works:
            # push down flow entry and remove the send_openflow command above.
            #self.install_datapath_flow (<fill in params> )
        #else:
            #flood packet out everything but the input port
            #self.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD, inport)

    def packet_in_callback(self, dpid, inport, reason, len, bufid, packet):
        """Packet-in handler"""
        if not packet.parsed:
            log.debug('Ignoring incomplete packet')
        else:
            self.learn_and_forward(dpid, inport, packet, packet.arr, bufid)

        return CONTINUE

    def install(self):
        self.register_for_packet_in(self.packet_in_callback)

    def getInterface(self):
        return str(pytutorial)

def getFactory():
```

```
class Factory:
    def instance(self, ctxt):
        return pytutorial(ctxt)

return Factory()
```