Tutorial

Setup

Reference
http://www.openflowswitch.org/foswiki/bin/view/OpenFlow/MininetGettingStarted for more thorough Mininet walkthrough if desired

Necessary Downloads

1. Download VM at http://www.cs.princeton.edu/courses/archive/fall10/cos561/assignments/COS561Tutorial.zip
2. Download VirtualBox at http://www.virtualbox.org/wiki/Downloads (VMWare will work too but VirtualBox is free)

Installing VM image (In VirtualBox)
The image we provide already has Mininet and Nox installed on it so we need to load it into VirtualBox.

1. **Open** VirtualBox.
2. Click **New** then **Next**
3. Enter desired name for VM (Ex: COS561Tutorial). OS: Linux. Version: Ubuntu. Click **Next**
4. Choose desired memory (Default 512 MB is fine). Click **Next**
5. Check **Use existing hard disk**. The folder button will open a pop-up. Click **add** and locate OpenFlowTutorial.vmdk. Click **Open**. Click **Select**.
6. Click **Continue**
7. You have installed your VM image!

Configuring VirtualBox for SSH
http://www.linuxjournal.com/content/tech-tip-port-forwarding-virtualbox-vboxmanage
The VM image we provided is only command line. We will need to SSH and use X Forwarding in order to load certain graphic application. There are subtle differences in this step between Mac/Linux and Windows, please follow the specific instructions for your machine.

Mac/Linux Instructions
Enable VM for SSH (In VirtualBox)

1. Select your VM and click **Settings**.
2. Go to the **Network** tab and click on **Advanced**
3. Check **Enable Network Adapter**. Attached to: **NAT**. Adapter type: PCnet-FAST III (Am79C973). Check **Cable Connected**.

Configure VM for SSH (through Terminal)

1. Open up Terminal and find the Virtualbox Application directory where VirtualBox is installed (the VM image should not be running)
2. Enter the following commands substituting *"VM Name Here"* with your VM name from above (Ex: COS561Tutorial)

```
$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/Protocol" TCP

$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/GuestPort" 22

$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/HostPort" 2222
```

Starting SSH Session

1. Open VirtualBox
2. **Start** the VM that we created (Ex: COS561Tutorial) and let it proceed until it asks for username. We will be using SSH to login to the VM. Login with usersname/pw: mininet/mininet
3. In VM image, *sudo dhclient eth1* (This is to "seed" the ip address. It's a strange issue that we found with Macs)
4. Open Terminal and SSH to VM image
   ssh -Y -l mininet -p 2222 localhost

If everything has proceeded correctly, you should see a couple folders (Ex: mininet, openflow, noxcore etc.) indicating that you have logged into the VM session. We can now proceed with developing inside the VM.

Windows Instructions

1. Select your VM and click **Settings**.
2. Go to the **Network** tab and click on **Advanced**
3. Check **Enable Network Adapter**. Attached to: **NAT**. Adapter type: PCnet-FAST III (Am79C973). Check **Cable Connected**.

Configuring VM for SSH (through CMD)

Open up command prompt and find your Virtualbox application directory where it VirtualBox is installed. (Be sure that the VM image is not running)

3. Enter the following commands substituting *"VM Name Here"* with your VM name from above (Ex: COS561Tutorial)

```
$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/Protocol" TCP

$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/GuestPort" 22

$ VBoxManage setextradata "VM Name Here" \
      "VBoxInternal/Devices/pcnet/0/LUN#0/Config/guestssh/HostPort" 2222
```

Enable X-Forwarding (In SSH application)

We will be using SSH to connect to the VM image when it is running

1. Choose your favorite SSH client and make sure that X Forwarding is **enabled**

2. We also need an application for X Forwarding so **download** an X Forwarding client (Ex: XMing, X-Win11, etc)

Starting Session

1. **Open** VirtualBox
2. **Start** the VM that we created (Ex: COS561Tutorial) and let it proceed until it asks for username. We will be using SSH to login to the VM. Login with usersname/pw: mininet/mininet
3. **IMPORTANT:** Once logged in, create a dummy file. Take a snapshot of the image (Machine → Take Snapshot). Shut down the VM image and then start it up again. **Check** that the dummy file still exists. This will ensure that your modifications are remembered.
4. SSH to VM Image. **Open** some SSH client (Putty, Secure Shell etc.) Login with *hostname: localhost, username: mininet, password: mininet, port: 2222*

If everything has proceeded correctly, you should see a couple folders (Ex: mininet, openflow, noxcore etc.) indicating that you have logged into the VM session. We can now proceed with developing inside the VM.

Developing a Network Topology

All development takes places through a **SSH session** to the VM. Thus, you will need your X Forwarding Client running in order to have any graphical interaction through this session. (For Windows, check that your X Forwarding application is running. For OS/Linux ensure that you have the –Y option when you ssh to the VM image)

Reference

http://yuba.stanford.edu/cs244wiki/index.php/Overview

Wireshark Analyzer

Wireshark is a great tool to help you analyze traffic flowing through nodes in the network

1. We will use wireshark to analyze network behavior with the command:
   sudo wireshark &
   This should open a graphical pop-up window (If there is an error saying a window cannot be created, your X Forwarding client may not be running)
2. In the wireshark filter box, enter *of* and then click **Apply**
3. Click **Capture**, then **Interfaces**, then **Start** on the loopback interface (lo). All packets flowing through the controller will show up here including Flow Modifications. If you wish to capture specific interfaces such as specifics hosts, simply click **Capture** and then **Interfaces** and then select the desired host.
4. We will now return to the SSH session looking at this wireshark application to view traffic

Exploring the Default Topology (2 hosts, 1 switch, 1 controller)

1. We will first start mininet; which, when no specific topology is given, will create the default topology of 2 hosts, 1 switch, and 1 controller

sudo mn

2. We can check and verify that the topology is correct by trying out the commands: *help, nodes, net, dump*
3. If we want to issue a specific command for a host, switch, or controller, we simply place the node's name in front of the command. (Ex: h2 ifconfig –a) *ifconfig, ps, etc.*
4. Pinging combined with wireshark is very useful in diagnosing controller behavior. *h2 ping –c 1 h3*
   Mininet will replace h3 with its IP address. If you view the wireshark output, you should be able to view the traffic in the network as a result of this ping.
   A convenient built-in command is *pingall*
5. *exit* will end the mininet session

Custom Topologies

There are many custom examples that can be found in ~/mininet/examples/
They use a lot of functions that can be found in ~/mininet/mininet/net.py
Specifically, the most relevant for this assignment will be emptynet.py and scratchnet.py.
Note: Many of the custom topologies use the default NOX controller. If we want to run our custom NOX controller, then we need to change one line:

net = Mininet(controller=Controller)

TO

net = Mininet(controller=lambda name: NOX(name, 'COS561Test'))

Once this is done, one can run the topology *sudo python [xyz].py* in mininet and then we can interact with it just like we did with the default topology above.

Developing a Custom Controller

Running NOX Controller

We have set up a NOX Controller for you to build.
Many of the **sample controller code** can be found in ~/noxcore/src/nox/ directory.
The NOX Controller we have set up can be **found** at
~/noxcore/src/nox/COS561Tutorial/COS561Test/COS561Test.py
Once you are done developing your python controller and wish to test it, simply save the file and then enter the following command in the shell:

sudo mn --controller=nox_cos561

nox_cos561 is a pointer to our NOX Controller, COS561Test, that we have installed to help mininet specify which NOX controller to install. **(!* If you get warning regarding Dissector bug, you should issue command sudo mn –c to clean up *!)**

Example Controllers

At this point, we need to program our controller in Python. It would simply take too long to thoroughly learn Python so we will simply learn "on the go" through examples. Basic

controllers will only really require basic data structures for which the API can easily be found online here http://docs.python.org/tutorial/datastructures.html.

Basic NOX Controller
~/noxcore/src/nox/coreapps/examples/pyloop.py
This is a very basic shell of a controller that does not perform any forwarding at this point. What it does is illustrate how one would be able to install desired event handlers. If we wish to have the controller react to packet_in events, then we would install an event handle in the install function.

Hub NOX Controller
~/noxcore/src/nox/coreapps/tutorial/pytutorial.py
This controller implements a simple hub. The first item to notice is the install function. This controller has installed an event handler for packet_in. Whenever a packet arrives at a switch that does not match any rules, it is sent to the controller. Currently, packets that arrive at the controller are forwarded to everyone except the port from which it came from. (Note: This is also the same controller that is currently in COS561Test.py.)

Simple Learning NOX Controller
We now leave it for you to implement the learning switch by building off of the hub from above. The algorithm is very simple:

1. We look at the incoming port number and MAC source of the incoming packet and store it in a data structure (We currently use the Python map). If a packet arrives with destination to this MAC, then we know what port to forward this packet to.
2. If we have an entry in our data structure for the MAC destination of the packet, then we know exactly which port to forward it.
3. Otherwise, we will flood this packet on all ports except for the port from which the packet came in.

Hints

There is not a great nox API around so there are several files that we will need to look into in order to find the appropriate function declarations that we will need to use. Included in this list of hints are other Python functions that you may find useful. This list should cover all the basic functionality that will be necessary to complete the assignment.

1. **References**

There are several useful files to take a look at since NOX is not very well documented
~/noxcore/src/nox/lib/core.py
Contains most of the higher level functions (sending packets, installing flows, etc)

~/noxcore/src/nox/packet/packet_utils.py
~/noxcore/src/nox/packet/ethernet.py
~/noxcore/src/nox/packet/ipv4.py
~/noxcore/src/nox/packet/packet_base.py
Contains some convenient functions for packet header analysis/parsing

~/noxcore/src/include/openflow/openflow/openflow.h

Contains many of the openflow variables that you may need to use

~/mininet/examples/
~/mininet/mininet/

Examples for custom topologies and their function definitions

2. **Event Registration**

There are several event handlers that are important:
register_for_packet_in(handler) – no matching flow rules come to controller
register_for_datapath_leave(handler) – switch down
register_for_datapath_join(handler) – switch join
post_callback(handler) – custom event handler

3. **Python Dictionary**

The built-in Dictionary structure may be useful for keeping track of learned MACs.

To init:	self.myDictionary = {}
To insert 3 item tuple with key i:	self.myDictionary[i] = (x, y, z)
Exists entry with key i:	self.myDictionary.has_key(i)
Get 2nd item in tuple with key i:	self.myDictionary[i][2]

There is much more you can accomplish with Python by looking at simple examples online.

4. **Python Functions:**

Python functions are pretty straightforward (Notice how there is no type casting and function declaration ends with a colon):

def myFunction(arg1, arg2, arg3):

5. **Packet Analysis**

You will notice that in your learn_and_forward function, we pass arguments ***packet*** and ***packet.arr***. ***packet*** contains parsed header information that is easy to grab using functions found in the packet library. ***packet.arr*** is the actual buffer data that needs to be modified if you wish to send this buffer.

Hint: Take a look at the packet library posted above to find useful address functions such as mac_to_int, tostring(), ipstr_to_int etc.

6. **Printing Controller Information**

In contrast to using the typical print statements, it is much more useful to print to a log. You will find it useful to declare the following at the top of your file:

```
logger = logging.getLogger('nox.COS561Tutorial.COS561Test.COS561Test')
logger.info('XYZ')
```

To view the output, you simply take a look at the controller log file that can be found at **/tmp/c0.log**

7. **Installing Dataflow Rules**

Installing dataflow rules is pretty straightforward if you understand how openflow works. First, we specify a flow rule then the action associated with it.

```
flow = extract_flow(packet)
flow[core.IN_PORT] = import
actions = [[openflow.OFPAT_OUTPUT, [0,prt[0]]]]
self.install_datapath_flow(dpid, flow, CACHE_TIMEOUT,
        openflow.OFP_FLOWPERMANT, actions, bufid,
        openflow.OFP_DEFAULT_PRIORITY, import, buf)
```

One can perform more actions by simply adding to the list (Ex: [[openflow.XYZ, args], [openflow.WXY, args]]). Consult with core.py for more details.

8. **Sending Packet**

Sending a single packet is a good starting point before attempting to install dataflow rules. Try to see if you can forward on a specific port once you have learned the MAC.

```
self.send_openflow(dpid, bufid, buf, ?, inport)
```

9. **Switch Identifier**

The switch ID (dpid) will be its MAC address if specified in the Mininet topology.
(Ex: If mac = ’00:01:02:03:0401’, then its dpid will be 0x000102030401)