Advanced Cyberinfrastructure Research & Education Facilitators Virtual Residency 2016

Lab 1: Openflow

Purpose:

OpenFlow is an open interface for remotely controlling the forwarding tables in network switches, routers, and access points. Upon this low-level primitive, researchers can build networks with new high-level properties. For example, OpenFlow enables more secure default-off networks, wireless networks with smooth handoffs, scalable data center networks, host mobility, more energy-efficient networks and new wide-area networks – to name a few.

This tutorial is your opportunity to gain hands-on experience with the platforms and debugging tools most useful for developing network control applications on OpenFlow.

NOTE: This tutorial borrows heavily from the Openflow Tutorial available at http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial#Download_Files

Pre-requisites:

The user should have a basic knowledge of:

1. A computer with at least 2GB of RAM and 10GB of free hard disk space
2. Administrative access to your computer
3. Operating System (OS) Graphical User Interface (GUI) navigation.
4. Comfort with typing text into a command or shell window.
5. Basic grasp of computer networking terms such as “host”, “switch”, and “IP Address”.

Lab Requirements:

- A functioning Internet connection.
- A laptop, mobile device, or access to a remote device that meets the basic hardware requirements for running Oracle’s Virtual Box.
- A functioning installation of the latest version of Oracle’s Virtual Box, available via: https://www.virtualbox.org/wiki/Downloads. Depending on your host device, you should also load the VirtualBox Extension Pack, located at the same URL.
- A copy of the “Virtual Machine Image for Mininet 2.0” 64bit virtual machine image, provided on a USB device, and also available within the download section here https://github.com/downloads/mininet/mininet/mininet-2.0.0-113012-amd64-ovf.zip

Important: For this VM image, the user name is 'mininet' with password 'mininet'.

- An X server, and an ssh-capable terminal emulator. Details based on operating system are provided in the following spreadsheet:

OS Type	OS Version	Virtualization Software	X Server	Terminal
Windows	7+	VirtualBox	Xming	PuTTY
Windows	XP	VirtualBox	Xming	PuTTY
Mac	OS X 10.7-10.8 Lion/Mountain Lion	VirtualBox	download and install XQuartz	Terminal.app (built in)
Mac	OS X 10.5-10.6 Leopard/Snow Leopard	VirtualBox	X11 (install from OS X main system DVD, preferred), or download XQuartz	Terminal.app (built in)
Linux	Ubuntu 10.04+	VirtualBox	X server already installed	gnome terminal + SSH built in

Install and Verify

After you have downloaded the appropriate software and VM images, make sure that each column item (X server, Virtualization software, and SSH terminal) is installed and working for your platform, and that the VM image loads and runs correctly for your configuration.

Import Virtual Machine Image

If you downloaded the .ovf image,

Start up VirtualBox, then select File>Import Appliance and select the .ovf image that you downloaded.

You may also be able to simply double-click the .ovf file to open it up in your installed virtualization program.

Next, press the "Import" button.

This step will take a while - the unpacked image is about 3 GB.

Name your VM OpenFlowTutorial, Operating System Linux, Version Ubuntu. Click Continue.

Your VM installation is complete. Press Done.

Finish VM Setup

You will need to complete **one more step before you are done with the VM setup**.

Select your VM and go to the Settings Tab. Go to Network->Adapter 2. Select the "Enable adapter" box, and attach it to "host-only network".(**Sidenote**: on a new VirtualBox installation you may not have any "host-only network" configured yet. To have one select File menu/Preferences/Network and "Add host-only network" button with default settings. Then you can try the attach.) **This will allow you to easily access your VM through your host machine.**

At that point you should be ready to start your VM. Press the "Start" arrow icon or double-click your VM within the VirtualBox window.

In the VM console window, log in with the user name and password for your VM.

Note that this user is a sudoer, so you can execute commands with root permissions by typing `sudo` *`command`*, where *`command`* is the command you wish to execute with root permission.

Command Prompt Notes

In this tutorial, commands are shown along with a command prompt to indicate what subsystem they are intended for . For example,

```
$ ls
```

indicates that the `ls` command should be typed at a Unix (e.g. Linux or OS X) command prompt (which generally ends in `$` if you are a regular user or `#` if you are root.

Other prompts used in this tutorial include

```
mininet>
```

for commands entered in the Mininet console and

```
C:>
```

for code entered into a Windows command window.

Set Up Network Access

The tutorial VM is shipped without a desktop environment, to reduce its size. All the exercises will be done through X forwarding, where programs display graphics through an X server running on the host OS.

To start up the X forwarding, you'll first need to find the guest IP address.

If you are running VirtualBox, you should make sure your VM has **two network interfaces**. One should be a **NAT interface** that it can use to access the Internet, and the other should be a **host-only interface** to enable it to communicate with the host machine. For example, your NAT interface could be eth0 and have a 10.x IP address, and your host-only interface could be eth1 and have a 192.168.x IP address. You should ssh into the **host-only interface** at its associated IP address. Both interfaces should be configured using DHCP. If they are not already configured, you may have to run dhclient on each of them, as described below.

Access VM via SSH

In this step, you'll verify that you can connect from the host PC (your laptop) to the guest VM (OpenFlowTutorial) via SSH.

From the virtual machine console, log in to the VM, then enter:

```
$ ifconfig -a
```

You should see three interfaces(eth0, eth1, lo), Both eth0 and eth1 should have IP address assigned. If this is not the case, type

```
$ sudo dhclient ethX
```

Replacing ethX with the name of a downed interfaces; sometimes the eth ports appear as eth2 or eth3, you can fix this by editing /etc/udev/rules.d/70-persistent-net.rules and removing the existing configuration lines.

Note the IP address (probably the 192.168 one) for the **host-only** network; you'll need it later. Next, log in, which will depend on your OS.

Mac OS X and Linux

Open a terminal (Terminal.app in Mac, Gnome terminal in Ubuntu, etc). In that terminal, run:

```
$ ssh -X [user]@[Guest IP Here]
```

Replace [user] with the `mininet`

Replace [Guest IP Here] with the IP you just noted. If ssh does not connect, make sure that you can ping the IP address you are connecting to.

Enter the password for your VM image. Next, try starting up an X terminal using

```
$ xterm
```

and a new terminal window should appear. If you have succeeded, you are done with the basic setup. Close the xterm. If you get a 'xterm: DISPLAY is not set error', verify your X server installation from above.

Windows

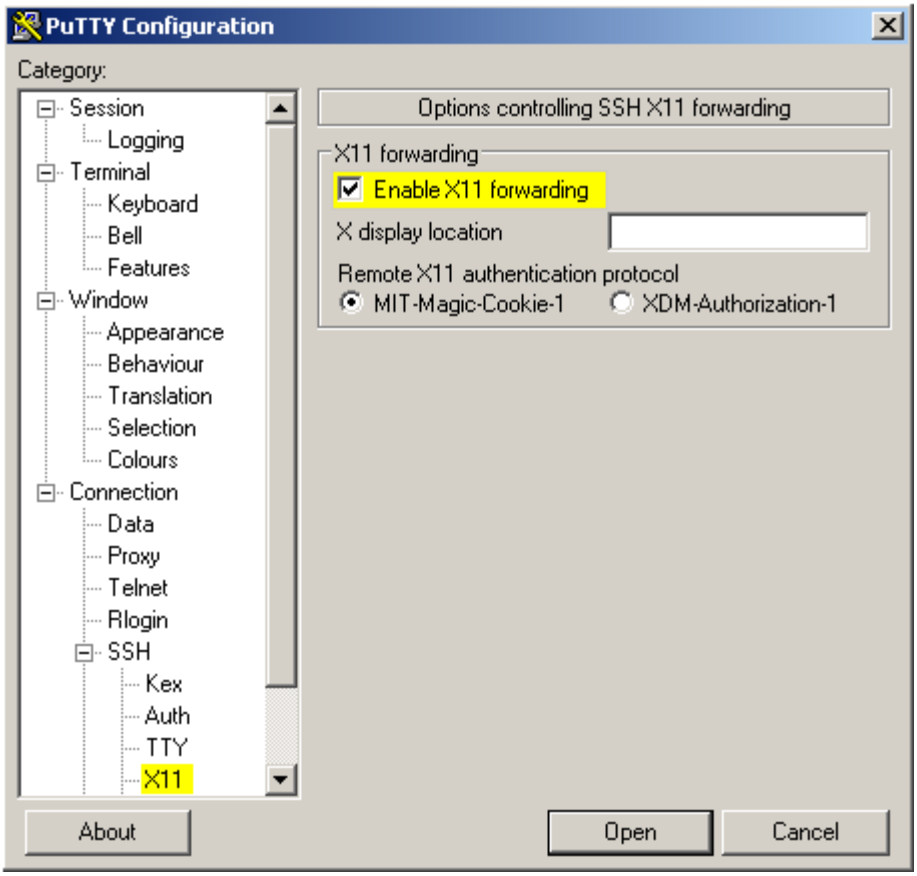
In order to use X11 applications such as xterm and wireshark, the Xming server must be running, and you must make an ssh connection with X11 forwarding enabled.

First, start Xming (e.g. by double-clicking its icon.) No window will appear, but if you wish you can verify that it is running by looking for its process in Windows' task manger.

Second, make an ssh connection with X11 forwarding enabled.

If you start up puTTY as a GUI application, you can connect by entering your VM's IP address and enabling X11 forwarding.

To enable X11 forwarding from puTTY's GUI, click puTTY->Connection->SSH->X11, then click on Forwarding->"Enable X11 Forwarding", as shown below:



You can also run putty (with the -X option for X11 forwarding) from the Windows command line:

Open a terminal: click the Windows 'Start' button, 'run', then enter 'cmd'.

Change to the directory where you saved putty.

```
C:> cd <dir>
```

Run:

```
C:> putty.exe -X openflow@[Guest IP Here]
```

Replace [Guest IP Here] with the IP you just noted.

If putty cannot connect, try pinging the VM's IP address to make sure you are connecting to the correct interface.

```
C:> ping [Guest IP Here]
```

Once the ssh connection succeeds or a terminal window for the VM pops up, log in to the VM. Now, type:

```
$ xterm -sb 500
```

to start an X terminal (the -sb 500 is optional but gives 500 lines of scrollback.)

A white terminal window should appear. If you have succeeded, you are done with the basic setup. Close the xterm.

If the xterm window does not appear, or if you get an error like "xterm: DISPLAY is not set," make sure that Xming is running in Windows and that you have correctly enabled X11 forwarding.

Alternative: Run X11 in the VM console window

As an alternative to running X11 on your host machine, you may find it useful or convenient to install X11 into the VM itself. To install X11 and a simple window manager, log in to the **VM console window** - *not* via an ssh session! - using the correct user name and password for your VM, and type:

```
$ sudo apt-get update && sudo apt-get install xinit flwm
```

At this point, you should be able to start an X11 session in the VM console window by typing:

```
$ startx
```

If you are familiar with Linux, you may wish to install another desktop manager (e.g. `lxde` or even the full `ubuntu-desktop`) or any other GUI packages that you prefer.

Tools

In this section, you'll bring up the development environment. In the process, you'll be introduced to tools that will later prove useful for turning the provided hub into a learning switch. You'll cover both general and OpenFlow-specific debugging tools.

Let's define some terminology, starting with terminal types:

VirtualBox console terminal: connects to OpenFlowTutorial. This is the one created when you started up the VM. You can't copy and paste from this tutorial page to the console terminal, so it's a bit of a pain. **Minimize this NOW, if you haven't already done so**. Once you've used it to set up networking, it won't be needed.

SSH terminal: connects to OpenFlowTutorial. Created by using putty on Windows or SSH on OS X / Linux, as described in the previous section. Copy and paste should work on this terminal.

xterm terminal: connects to a host in the virtual network. Created in the next section when you start up the virtual network. Will be labeled at the top with the name of the host.

The OpenFlowTutorial VM includes a number of OpenFlow-specific and general networking utilities pre-installed. Please read the short descriptions:

OpenFlow Controller: sits above the OpenFlow interface. The OpenFlow reference distribution includes a controller that acts as an Ethernet learning switch in combination with an OpenFlow switch. You'll run it and look at messages being sent. Then, in the next section, you'll write our own controller on top of NOX or Beacon (platforms for writing controller applications).

OpenFlow Switch: sits below the OpenFlow interface. The OpenFlow reference distribution includes a user-space software switch. Open vSwitch is another software but kernel-based switch, while there is a number of hardware switches available from Broadcom (Stanford Indigo release), HP, NEC, and others.

dpctl: command-line utility that sends quick OpenFlow messages, useful for viewing switch port and flow stats, plus manually inserting flow entries.

Wireshark: general (non-OF-specific) graphical utility for viewing packets. The OpenFlow reference distribution includes a Wireshark dissector, which parses OpenFlow messages sent to the OpenFlow default port (6633) in a conveniently readable way.

iperf: general command-line utility for testing the speed of a single TCP connection.

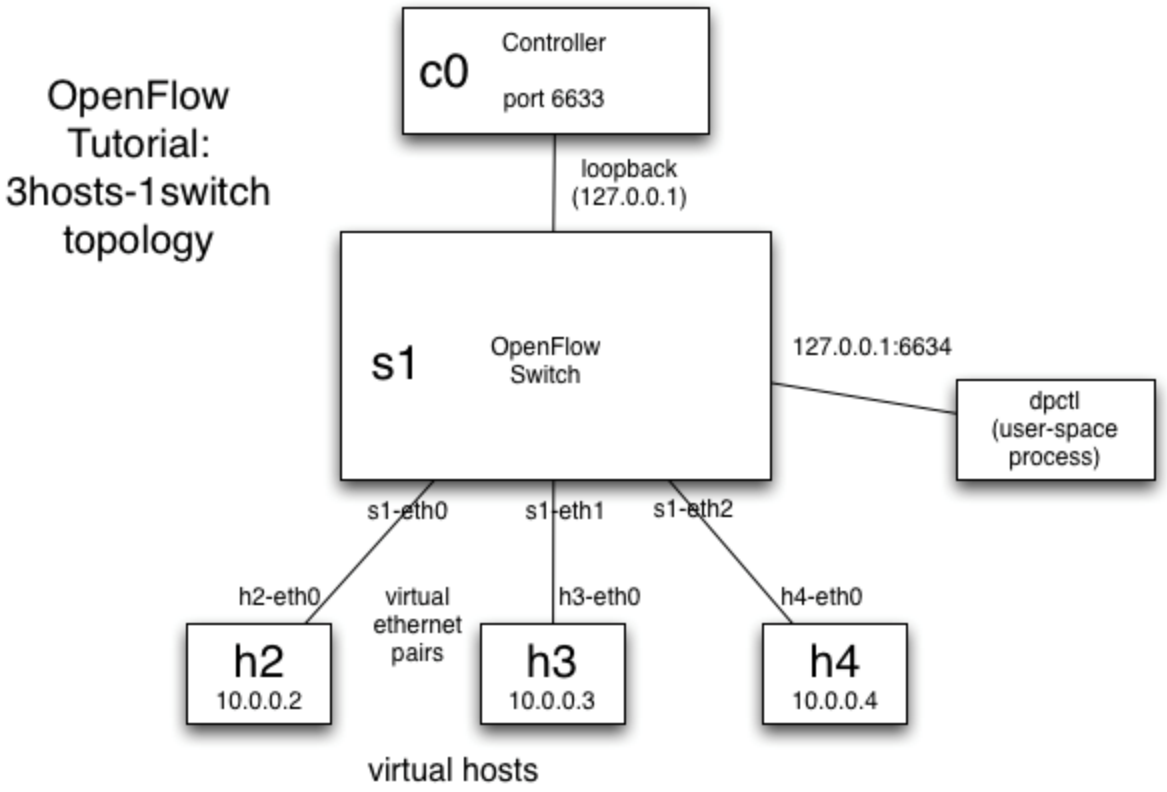
Mininet: network emulation platform. Mininet creates a virtual OpenFlow network - controller, switches, hosts, and links - on a single real or virtual machine. More Mininet details can be found at the Mininet web page.

From here on out, make sure to copy and paste as much as possible! For example, manually typing in ‘sudo dpctl show n1:0’ may look correct, but will cause a confusing error; the 'nl' is short for NetLink, not n-one.

Let's get started...

Start Network

The network you'll use for the first exercise includes 3 hosts and a switch:



To create this network in the VM, in an SSH terminal, enter:

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

This tells Mininet to start up a 3-host, single-(openvSwitch-based)switch topology, set the MAC address of each host equal to its IP, and point to a remote controller which defaults to the localhost.

Here's what Mininet just did:

Created 3 virtual hosts, each with a separate IP address.

Created a single OpenFlow software switch in the kernel with 3 ports.

Connected each virtual host to the switch with a virtual ethernet cable.

Set the MAC address of each host equal to its IP.

Configure the OpenFlow switch to connect to a remote controller.

Mininet Brief Intro

Since you'll be working in Mininet for the whole tutorial, it's worth learning a few Mininet-specific commands:

To see the list of nodes available, in the Mininet console, run:

```
mininet> nodes
```

To see a list of available commands, in the Mininet console, run:

```
mininet> help
```

To run a single command on a node, prepend the command with the name of the node. For example, to check the IP of a virtual host, in the Mininet console, run:

```
mininet> h1 ifconfig
```

The alternative - better for running interactive commands and watching debug output - is to spawn an xterm for one or more virtual hosts. In the Mininet console, run:

```
mininet> xterm h1 h2
```

You can close these windows now, as we'll run through most commands in the Mininet console.

If Mininet is not working correctly (or has crashed and needs to be restarted), first quit Mininet if necessary (using the `exit` command, or control-D), and then try clearing any residual state or processes using:

```
$ sudo mn -c
```

and running Mininet again.

NB: The prompt `mininet>` is for Mininet console, `$` is for SSH terminal (normal user) and `#` is for SSH terminal (root user) (See Command Prompt Notes). Hereafter we follow with this rule.

Mininet has loads of other commands and startup options to help with debugging, and this brief starter should be sufficient for the tutorial. If you're curious about other options, follow the Mininet Walkthrough after the main tutorial.

dpctl Example Usage

dpctl is a utility that comes with the OpenFlow reference distribution and enables visibility and control over a single switch's flow table. It is especially useful for debugging, by viewing flow state and flow counters. Most OpenFlow switches can start up with a passive listening port (in your current setup this is 6634), from which you can poll the switch, without having to add debugging code to the controller.

Create a second SSH window if you don't already have one, and run:

```
$ dpctl show tcp:127.0.0.1:6634
```

The 'show' command connects to the switch and dumps out its port state and capabilities.

Here's a more useful command:

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```

Since we haven't started any controller yet, the flow-table should be empty.

Ping Test

Now, go back to the mininet console and try to ping h2 from h1. In the Mininet console:

```
mininet> h1 ping -c3 h2
```

Note that the name of host h2 is automatically replaced when running commands in the Mininet console with its IP address (10.0.0.2).

Do you get any replies? Why? Why not?

As you saw before, switch flow table is empty. Besides that, there is no controller connected to the switch and therefore the switch doesn't know what to do with incoming traffic, leading to ping failure.

You'll use dpctl to manually install the necessary flows. In your SSH terminal:

```
$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
$ dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
```

This will forward packets coming at port 1 to port 2 and vice-verca. Verify by checking the flow-table

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```

Run the ping command again. In your mininet console:

```
mininet> h1 ping -c3 h2
```

Do you get replies now? Check the flow-table again and look the statistics for each flow entry. Is this what you expected to see based on the ping traffic? NOTE: if you didn't see any ping replies coming through, it might be the case that the flow-entries expired before you start your ping test. When you do a "dpctl dump-flows" you can see an "idle_timeout" option for each entry, which defaults to 60s. This means that the flow will expire after 60secs if there is no incoming traffic. Run again respecting this limit, or install a flow-entry with longer timeout.

```
$ dpctl add-flow tcp:127.0.0.1:6634
in_port=1,idle_timeout=120,actions=output:2
```

Start Wireshark

The VM image includes the OpenFlow Wireshark dissector pre-installed. Wireshark is extremely useful for watching OpenFlow protocol messages, as well as general debugging.

Start a new SSH terminal and connect to the VM with X11 forwarding.

(Reminder: here are the command-line commands to do so:

If you're using MAC OS X or Linux, enter:

```
$ ssh -X openflow@[guest ip address]
```

If you're using putty.exe from the Windows command-lineshell, enter:

```
C:> putty.exe -X openflow@[guest ip address]
```

If you're using putty's GUI, make sure X11 forwarding is enabled.)

Now open Wireshark:

```
$ sudo wireshark &
```

You'll probably get a warning message for using wireshark with root access. Press OK.

Click on Capture->Interfaces in the menu bar. Click on the Start button next to 'lo', the loopback interface. You may see some packets going by.

Now, set up a filter (a View filter, not a Capture filter)for OpenFlow control traffic, by typing 'of' in Filter box near the top:

```
of
```

Press the apply button to apply the view filter to all recorded traffic.

Start Controller and view Startup messages in Wireshark

Now, with the Wireshark dissector listening, start the OpenFlow reference controller. In your SSH terminal:

```
$ controller ptcp:
```

This starts a simple controller that acts as a learning switch without installing any flow-entries.

You should see a bunch of messages displayed in Wireshark, from the Hello exchange onwards. As an example, click on the Features Reply message. Click on the triangle by the 'OpenFlow Protocol' line in the center section to expand the message fields. Click the triangle by Switch Features to display datapath capabilities - feel free to explore.

These messages include:

Message	Type	Description
Hello	Controller->Switch	following the TCP handshake, the controller sends its version number to the switch.
Hello	Switch->Controller	the switch replies with its supported version number.
Features Request	Controller->Switch	the controller asks to see which ports are available.
Set Config	Controller->Switch	in this case, the controller asks the switch to send flow expirations.
Features Reply	Switch->Controller	the switch replies with a list of ports, port speeds, and supported tables and actions.

Port Status	Switch->Controller	enables the switch to inform that controller of changes to port speeds or connectivity. Ignore this one, it appears to be a bug.

Since all messages are sent over localhost when using Mininet, determining the sender of a message can get confusing when there are lots of emulated switches. However, this won't be an issue, since we only have one switch. The controller is at the standard OpenFlow port (6633), while the switch is at some other user-level port.

View OpenFlow Messages for Ping

Now, we'll view messages generated in response to packets.

Before that update your wireshark filter to ignore the echo-request/reply messages (these are used to keep the connection between the switch and controller alive): Type the following in your wireshark filter, then press apply:

```
of && (of.type != 3) && (of.type != 2)
```

Run a ping to view the OpenFlow messages being used. In the Mininet console:

```
mininet> h1 ping -c1 h2
```

In the Wireshark window, you should see a number of new message types:

Message	Type	Description
Packet-In	Switch->Controller	a packet was received and it didn't match any entry in the switch's flow table, causing the packet to be sent to the controller.
Packet-Out	Controller->Switch	controller send a packet out one or more switch ports.
Flow-Mod	Controller->Switch	instructs a switch to add a particular flow to its flow table.

Flow-Expired	Switch->Controller	a flow timed out after a period of inactivity.

First, you see an ARP request miss the flow table, which generates a broadcast Packet-Out message. Next, the ARP response comes back; with both MAC addresses now known to the controller, it can push down a flow to the switch with a Flow-Mod message. The switch does then pushes flows for the ICMP packets. Subsequent ping requests go straight through the datapath, and should incur no extra messages; with the flows connecting h1 and h2 already pushed to the switch, there was no controller involvement.

Re-run the ping, again from the Mininet console (hitting up is sufficient - the Mininet console has a history buffer):

```
mininet> h1 ping -c1 h2
```

If the ping takes the same amount of time, run the ping once more; the flow entries may have timed out while reading the above text.

This is an example of using OpenFlow in a *reactive* mode, when flows are pushed down in response to individual packets.

Alternately, flows can be pushed down before packets, in a *proactive* mode, to avoid the round-trip times and flow insertion delays.

Benchmark Controller w/iperf

iperf is a command-line tool for checking speeds between two computers.

Here, you'll benchmark the reference controller; later, you'll compare this with the provided hub controller, and your flow-based switch (when you've implemented it).

In the mininet console run :

```
mininet> iperf
```

This Mininet command runs an iperf TCP server on one virtual host, then runs an iperf client on a second virtual host. Once connected, they blast packets between each other and report the results.

Now compare with the user-space switch. In the mininet console:

```
mininet> exit
```

Stop the ptcp controller in the shell window it was started in using a Control C on the keyboard

Start the same Mininet with the user-space switch and run another iperf test:

```
$ sudo mn --topo single,3 --switch user –test iperf
```

See a difference? With the user-space switch, packets must cross from user-space to kernel-space and back on every hop, rather than staying in the kernel as they go through the switch. The user-space switch is easier to modify (no kernel oops'es to deal with), but slower for simulation.

Exit Mininet:

```
mininet> exit
```

Close Wireshark and shut down your minimnet virtual machine

```
$ sudo telinit 0
```

This concludes Lab 1 Openflow Tutorial