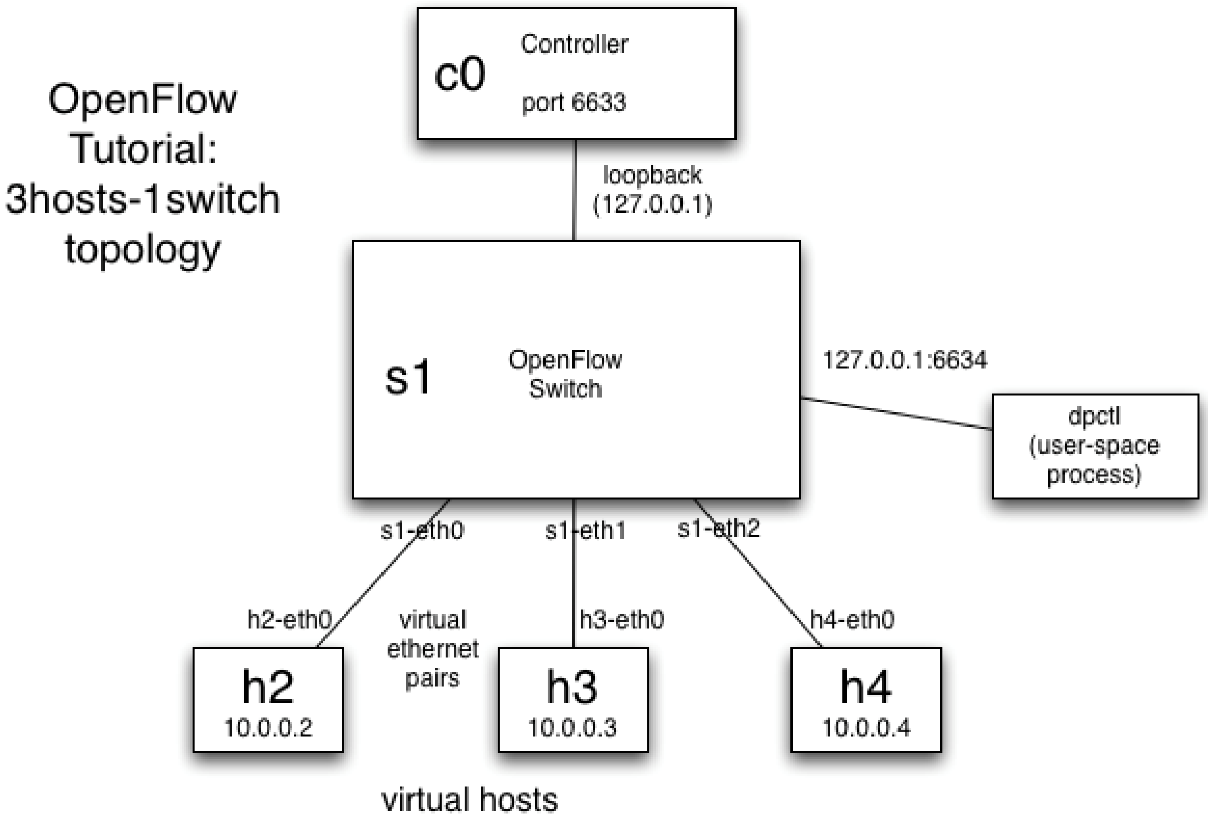
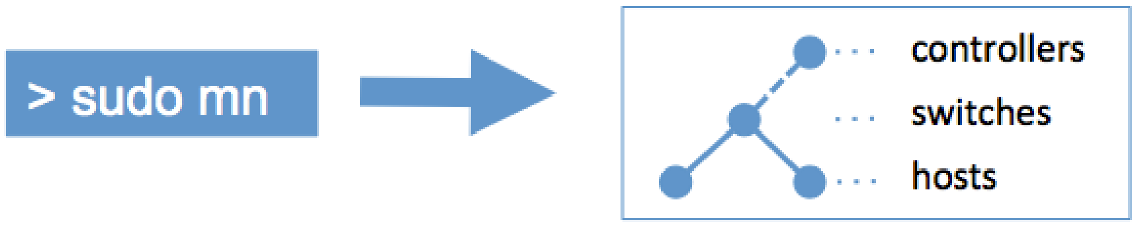
Laboratory 8: Software Defined Networking

Objective:

- To learn the basics of Mininet and OpenFlow
- To learn how to create a SDN topology and install flows in mininet
- To know various SDN controller supported by mininet



Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command:



You can easily interact with your network using the Mininet CLI (and API), customize it, share it with others, or deploy it on real hardware, Mininet is useful for development, teaching, and research.
Mininet is also a great way to develop, share, and experiment with OpenFlow and Software-Defined Networking systems.
Mininet can be run on the leab machines based on the following instruction. Also, you can download the virtual Linux machine, including the installed Mininet and related tools, in your laptop/pc and run it via VirtualBox.

Development environment:

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.

The reference distribution Mininet 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. Mininet reference distribution comes with built-in Controller() classes to support several controllers, including the OpenFlow reference controller (controller), Open vSwitch's ovs-controller, and the now-deprecated NOX Classic.

- **OpenFlow Switch**: sits below the OpenFlow interface.
- **ovs-ofctl**: command-line utility that sends quick OpenFlow messages, useful for viewing switch port and flow stats or 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.
- **cbench**: utility for testing the flow setup rate of OpenFlow controllers.

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.

Run mininet on CSE machines:

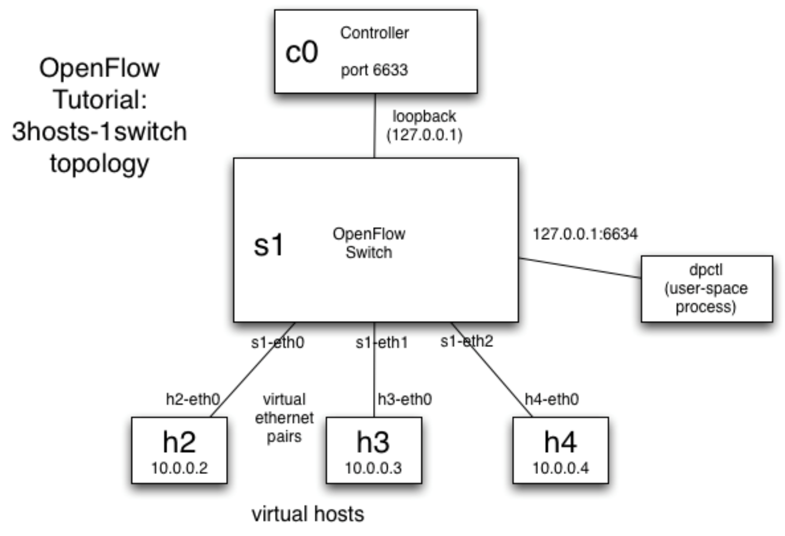
In lab computer, we call it host, open terminal and run `vm`, then choose mininet. The Linux VM which support mininet will boot up. The static IP address of this machine is 192.168.0.140. We will not use this Linux directly. Instead we use ssh to access this machine from the host. Press ctrl+shift+alt to release your mouse pointer. Minimise the mininet VM and open a terminal in host (lab computer).

```
# ssh -X mininet@192.168.0.140
```

Enter "mininet" as password (same as username) and from now you are connected to mininet VM. All commands that we will enter for lab8 and lab9 should be done in SSH terminal from host. You may need to open more terminals and do SSH login.

Start Network

The network you'll use for the first exercise includes 3 hosts and a switch (and, eventually, an OpenFlow controller, but we'll get to that later):



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

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

```
mininet> nodes
```

To see the links:

```
mininet> net
```

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
```

Note that The prompt `mininet>` is for Mininet console, `$` is for SSH terminal (normal user) and `#` for host terminal before using SSH.

ovs-ofctl Utility

(Just to remind you, we are assuming that you have already started up Mininet in another window using `sudo mn --topo single,3 --mac --switch ovsk --controller remote`, as directed above.)

`ovs-ofctl` is a utility that comes with Open vSwitch 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.

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

```
$ sudo ovs-ofctl show s1
```

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

Here's a more useful command:

```
$ sudo ovs-ofctl dump-flows s1
```

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

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 `ovs-ofctl` to manually install the necessary flows. In your SSH terminal:

```
 $ ovs-ofctl add-flow s1 in_port=1,actions=output:2

 $ ovs-ofctl add-flow s1 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

```
 $ ovs-ofctl dump-flows s1
```

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?

Start Wireshark

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.

```
# ssh -X mininet@192.168.0.140
```

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 for OpenFlow control traffic, by typing 'of' in Filter box near the top:

```
of
```

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

Start Controller

First stop mininet and clean

```
 mininet> exit

 $sudo mn -c
```

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

```
$sudo controller ptcp:6633
```

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

Restart now the topology

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

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.

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 and not (of10.echo_request.type or of10.echo_reply.type)
```

Note that there are a variety of other ways that you could express that filter.

Run a ping to view the OpenFlow messages being used. You will need to run this without having any flows between h1 and h2 already installed, e.g. from the "add-flows" command above:

```
sudo ovs-ofctl del-flows s1
```

It's also recommended to clean up ARP cache on both hosts, otherwise you might not see some ARP requests/replies as the cache will be used instead:

```
mininet> h1 ip -s -s neigh flush all

mininet> h2 ip -s -s neigh flush all
```

Do the ping 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.