Demonstrating the use of custom OpenFlow controllers (using POX)

Run the mininet VM image.

Note: If the VM is being run freshly for the first time: run `ifconfig`, if eth1 doesn't appear, run `sudo dhclient eth1`. Note the adaptor address for use in the next step.

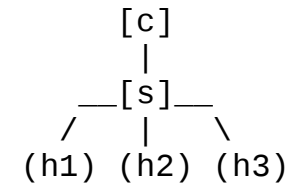
SSH in to mininet (user and password is *mininet* by default):

```
$ ssh -X mininet@<local vm adapter addr>
```

Set up basic switch and 3 hosts topology with interface for remote controller:

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

Note: `--mac` tells mininet to use simple MAC naming; 0...:01 for host 1, 0...:02 for host 2 etc. Throw in a `-x` option to spawn xterms for each node, and skip the `mininet> xterm h2 h3` step later.



Try `mininet>h1 ping -c1 h2` without setting up a controller. Nothing. The switch has an empty flow table. When no flow rules match, the packet is dropped by default.

Use POX hub example to tell the switch a simple * => flood rule:

```
$ ./pox/pox.py --verbose forwarding.hub
```

Switch now has a flow rule to operate on (a very dumb one)
To see the rules, we can ask the switch to dump its flow table:

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

To verify this, we can monitor ping behaviour.
Open terminals for host 2 and 3:

```
mininet> xterm h2 h3
```

in each of them, observe traffic (on h2-eth0 and h3-eth0 interfaces respectively):

```
$ tcpdump -n -i h2-eth0
$ tcpdump -n -i h3-eth0
```

Note: -n option tells tcpdump not to translate host addresses.

Now ping from h1 to h2 and notice that h3 gets it as well:

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

The switch is behaving like a hub. Notice that you can kill the controller at any time and the flow will still function. This is because the hub controller transmits the single flood rule to the switch when first connects; the switch will remember this rule in the absence of the controller. (This wont be the case for more sophisticated controllers with flow rule timeouts)

Use the POX l2_learning switch example to see some intelligent behaviour:

```
$ ./pox/pox.py –verbose forwarding.l2_learning
```

Ping from h1 and h2 like before and notice that h3 doesn't get it any more; the switch is actually switching now.

Study the source for l2_learning.py (it's well commented) and try adding your own functionality to it. e.g. Drop packets that have an odd or even sequence number. Verify this with iperf -u (UDP mode) and Wireshark.

More advanced: make a better guess at which ports to flood ARP messages on.

Even more challenging: make ARPing safe for topologies with cycles. See spanning tree example.

TODO: Try out and write hints for these challenges. Maybe include other POX modules for fancy colours and custom topologies?