CSCI 4211: Introduction to Computer Networks
Spring 2023
```
PROGRAMMING PROJECT 3 – SDN and Mininet
```

Due 11:59pm Sunday April 30th

Virtual networks provide a good framework for testing and experimenting with networking environments in a safe and configurable way. Additionally, SDNs (Software Defined Networks) provide an added level of easier network management for both real and simulated environments through centralized configuration and monitoring. In this assignment, you will learn how to create a virtual network on your local machine with Mininet, evaluate its performance, and build SDN controller logic to install flow rules with OpenFlow.

Your Assignment:

Setup the Testing Environment:

1. Download the required tools:
    a. Download a Virtual Machine Manager (VMM):
       You can use any virtualization system of your choice, but we recommend installing VirtualBox. It is free and runs on Windows, Linux, and macOS.
       https://www.virtualbox.org/wiki/Downloads
       OR You can also use VMWARE Fusion [Free to UMN Students]
       https://cse.umn.edu/cseit/software/vmware

    b. Download Mininet VVM:
       Mininet with lightweight LXDE GUI for xterm from [prepared by TA]:
       https://drive.google.com/file/d/1akP_TC8z2IJ38VEOZD1jgJ3x4QqNoUfV/view?usp=sharing
       OR download the original Mininet VM:
       (Based on your computer specs, download either the i386.zip or amd64.zip)
       https://github.com/mininet/mininet/releases/tag/2.3.0

2. Import the Mininet Virtual Machine Image:
   https://github.com/mininet/openflow-tutorial/wiki/Set-up-Virtual-Machine
    a. Start your VMM (Oracle VirtualBox), and in the **"File"** menu, select **"import Appliance"**.
    b. Click **"Choose"** (a folder icon may also be **"Choose"**), browse to the location containing Mininet-VM (the .ovf file you downloaded), and click **"Open"**, and click **"Continue"**.

c. Click **"Import"** and then select the imported virtual machine. This step will take a while - the unpacked image is about 3 GB.
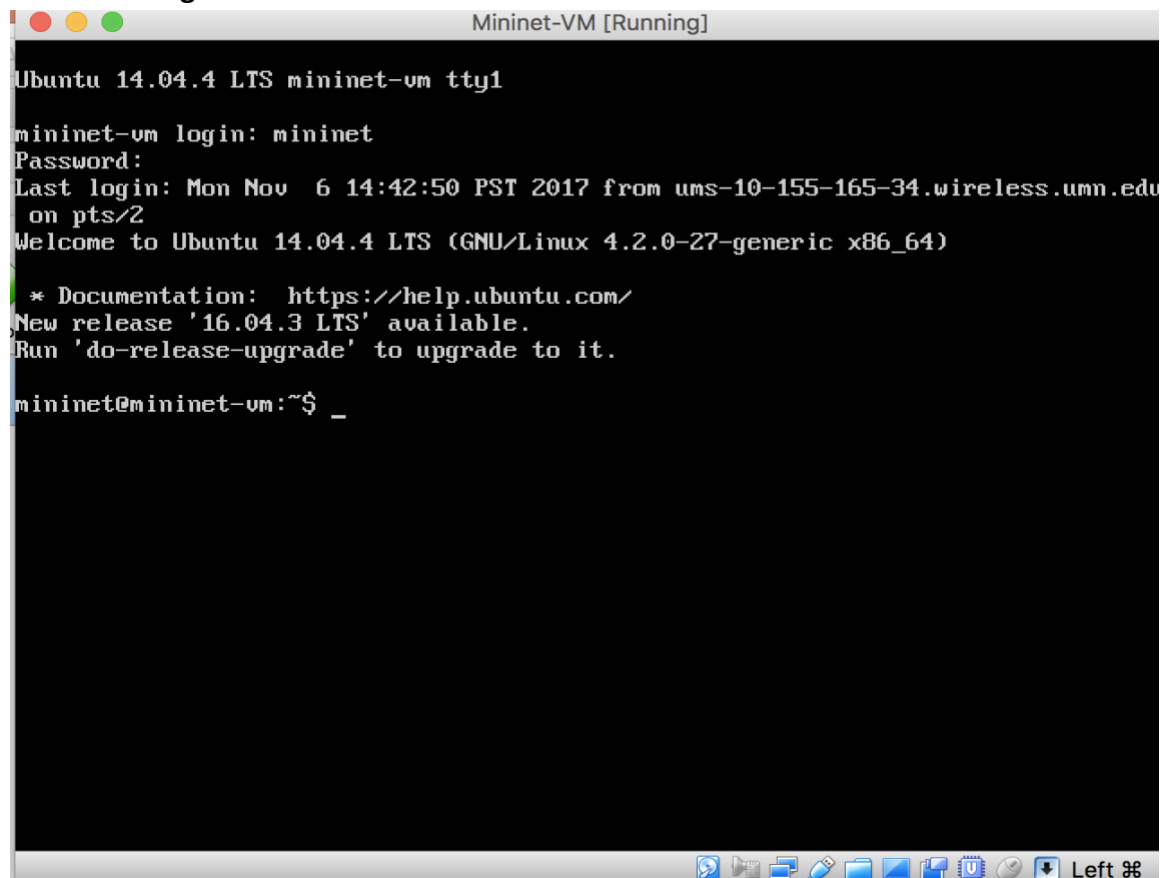
After Importing:
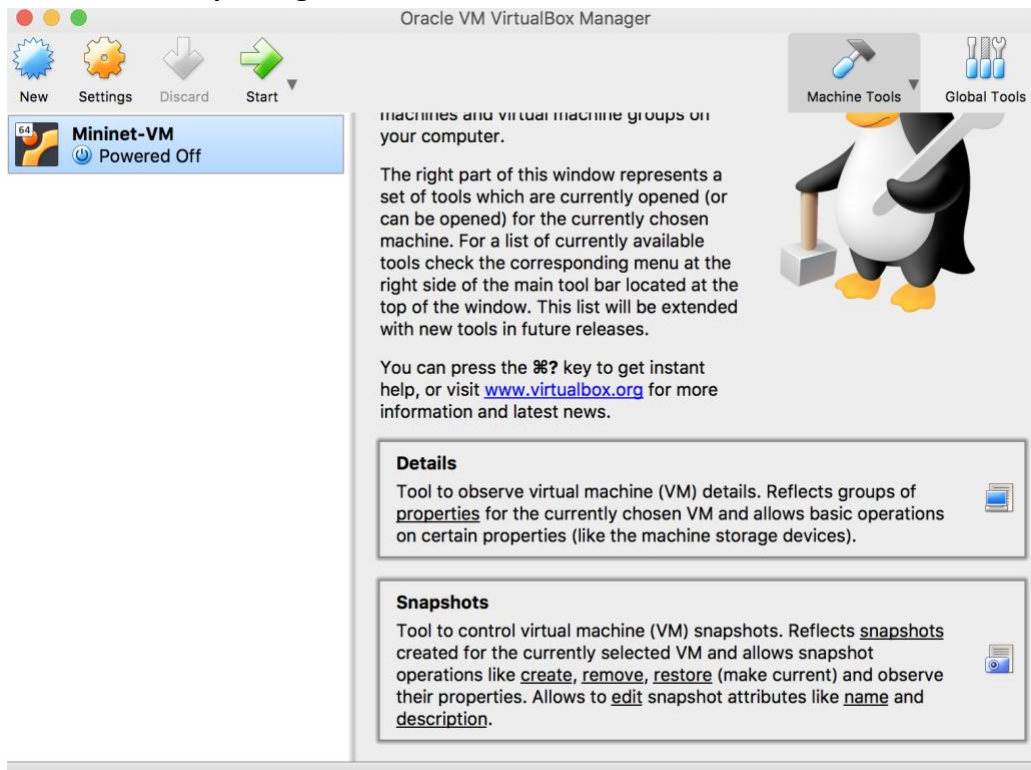
3. Finish VM Setup:
   a. Select **"Host Network Manager"** from the **"File"** menu.
   b. Click **"Create"**, you should see a new network named like **"vboxnet0"** or similar.
   c. Click **"Enable DHCP"** in the **"vboxnet0"** entry.
   d. Now go back to main menu, click on the Mininet-VM, click on **"Settings".**
      - Select **"Network"** and then set **"Attached to"** to **"Host-only Adapter"** then click **"OK".**
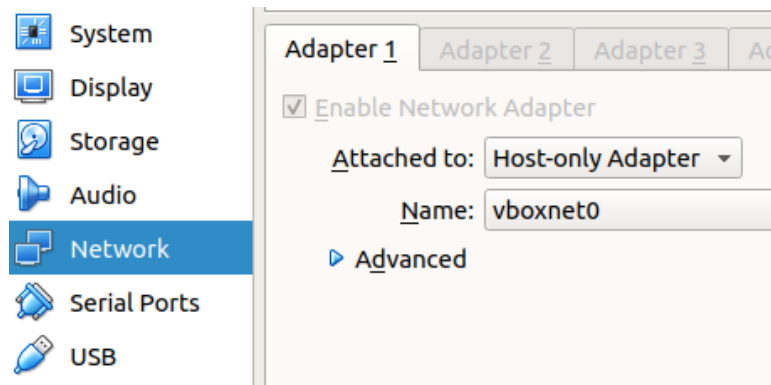
4. Start the Mininet VM:
    a. Select the Mininet-VM and click the **"Start".**
    b. If the VM is crashing on startup, increase the number of cores for the vm.
    c. In the VM console window, log in with the username and password for your VM. These should both be "mininet":
        ▪ `mininet-vm login: mininet`
        ▪ `Password: mininet`
    d. 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.

    After Running:

    ```
    Ubuntu 14.04.4 LTS mininet-vm tty1

    mininet-vm login: mininet
    Password:
    Last login: Mon Nov  6 14:42:50 PST 2017 from ums-10-155-165-34.wireless.umn.edu
     on pts/2
    Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

     * Documentation:  https://help.ubuntu.com/
    New release '16.04.3 LTS' available.
    Run 'do-release-upgrade' to upgrade to it.

    mininet@mininet-vm:~$ _
    ```

5. [SKIP this step if Downloaded VM by TA] If you downloaded the original Mininet VM, install X server based on your Operating System:
   https://github.com/mininet/openflow-tutorial/wiki/Installing-Required-Software
   Note: Linux OS has an X Server installed by default.

6. Setup Network Access:
    a. 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. For more detailed instructions, see VirtualBox Specific Instructions: https://github.com/mininet/openflow-tutorial/wiki/VirtualBox-specific-Instructions

7. Configure the VirtualBox so that you can access the VM through SSH:
    a. In this step, you'll verify that you can connect from the host PC (your laptop) to the guest VM (Mininet) 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.
    b. For Mac OS X and Linux: Open a terminal (Terminal.app in Mac, Gnome terminal in Ubuntu, etc). In that terminal, run:
        ```
        $ ssh –X mininet@VM's IP address
        ```
        (from step 6a, the IP should be 192.168.x.x which is a host-only interface)
        From this point on, you can minimize Virtualbox and interact with it purely through SSH. Don't forget -x option for graphical forwarding. If ssh does not connect, make sure that you can ping the IP address you are connecting to.
    c. Enter the password for your VM image (mininet).
    d. For Windows instructions and troubleshooting, visit: https://github.com/mininet/openflow-tutorial/wiki/Set-up-Virtual-Machine
    e. Next, try starting up an X terminal using:
        ```
        $ xterm
        ```
        A new terminal window should appear. If you have succeeded, you are done with the basic setup. Close the xterm.

8. Copying files from and to VM:
    a. use <scp> command if you can <ssh> to the VM using the IP address from step 6a, it should be 192.168.x.x which is a host-only interface):
        scp <local file> <VM user>@<VM machine IP>:<VM path>
        ```
        scp ~/Desktop/Project3/upload/part2/simple_routing.py
        mininet@VM_IP:~/pox/pox/samples/
        scp -r Project3Code mininet@192.168.56.102:/home/mininet
        ```

9. Additionally, if your VM won't boot to login, check your [BIOS/UEFI](#) settings to ensure virtualization is enabled. To do this you will need to look up how to access your bios for your machine. For example, a lenovo laptop can access its bios by restarting and hitting enter as soon as the initial boot screen appears and from there the bios configuration can be accessed.

10. Learn how to run a basic topology.
    a. First ssh to the VM
        ```
        $ ssh -X mininet@VM's IP address
        ```

    b. Start xterm for another sshed window
        ```
        $ xterm &
        ```

    c. Run a controller:
        ```
        cd pox
        ./pox.py log.level --DEBUG misc.of_tutorial
        ```
        This loads the controller in ~/pox/pox/misc/of_tutorial.py. This version acts like a dumb switch and floods all packets.

    d. In another window, set up a topology that connects to the controller:
        ```
        sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --controller remote
        ```
        This loads the topology ~/mininet/custom/ topo-2sw-2host.py. This version creates a simple topology with one switch connected to two hosts.

    e. To run the project topology:
        ```
        cd Project3Code/
        sudo python topology-a.py
        ```

    f. To run any command on any host machine within the simulator:
        The syntax is (machine name) (command), i.e., execute "ping h2" on host h1 is
        ```
        h1 ping h2
        ```

11. Some handy commands:
    a. Ping between hosts h1 and h2:   `h1 ping h2`
    b. iPerf between h1 and h2:   `h2 iperf -s -p 4000 &`
                                                  `h1 iperf -c h2 -p 4000 -t 5`
    c. Any command you want to send to host h1: `h1 cmd`
    d. Open up a new terminal for host h1:   `xtrerm h1`
    e. Analyze network traffic: `wireshark, tcpdump`
    f. Print the rules currently installed on switch s1:   `ovs-ofctl dump-flows s1`

12. iPerf is a tool to measure the bandwidth and the quality of a network link. Usage:
    a. On the server side (h2 (10.0.0.2) for example):
        ```
        $ iperf -s -p 4001
        ```
        will initiate a tcp server listening on port 4001.

    b. On the client side (h1 (10.0.0.1) for example):
        ```
        $ iperf -c 10.0.0.2 -p 4001
        ```
        will initiate a tcp client that connects to host 10.0.0.2 on port 4001 and performs measurement of the link.

    c. More examples are available on: http://openmaniak.com/iperf.php

13. Resources (you may find the following resources useful and in fact are encouraged to go through them):
    a. Introduction to Mininet video: https://www.youtube.com/watch?v=jmlgXaocwiE
    b. Mininet walkthrough: https://mininet.org/walkthrough/
    c. Introduction to Mininet: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
    d. OpenFlow Tutorial: https://github.com/mininet/openflow-tutorial/wiki
    e. Mininet Remote Controller: https://www.youtube.com/watch?v=CPasnNg9Z4I
    f. Pox: https://noxrepo.github.io/pox-doc/html/
    g. Floodlight: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview
    h. Creating a Learning Switch: https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch

Part 1: Hands on with Mininet and Measuring Performance

Consider Topology A:
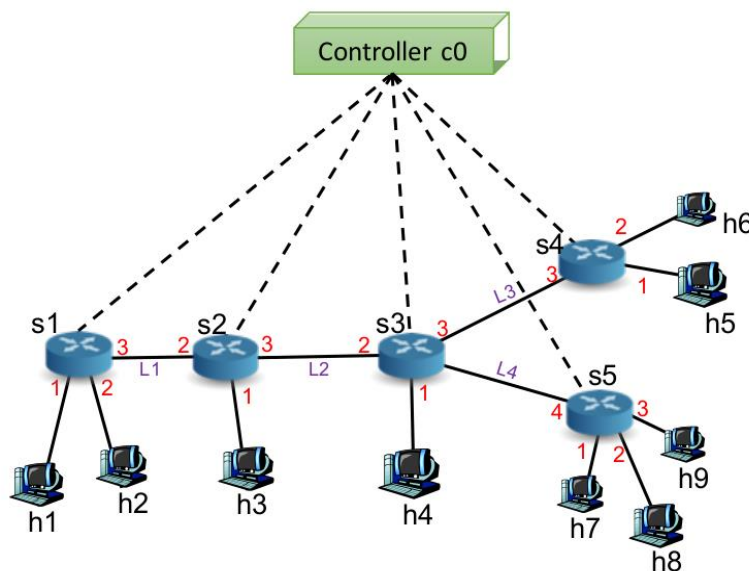
a. Provide the estimated throughput and latency for each of the links between the switches i.e., L1, L2, L3, L4, and explain the process used to estimate them in detail. Write your answers in a file named (**topology-a-report.pdf**)

- Note: To run topology A's mininet script use the command
  ```
  "sudo python topology-a.py"
  ```
- Note: The hosts (h1 - h9) are assigned IP addresses 10.0.0.1 through 10.0.0.9 (h1 is 10.0.0.1, h9 is 10.0.0.9).
- Hint: You may use iPerf and Ping to answer these questions

## Part 2: Building a Custom Virtual Network Using Mininet

Create Topology B with Mininet's Python API (using the skeleton code **topology-b.py**)
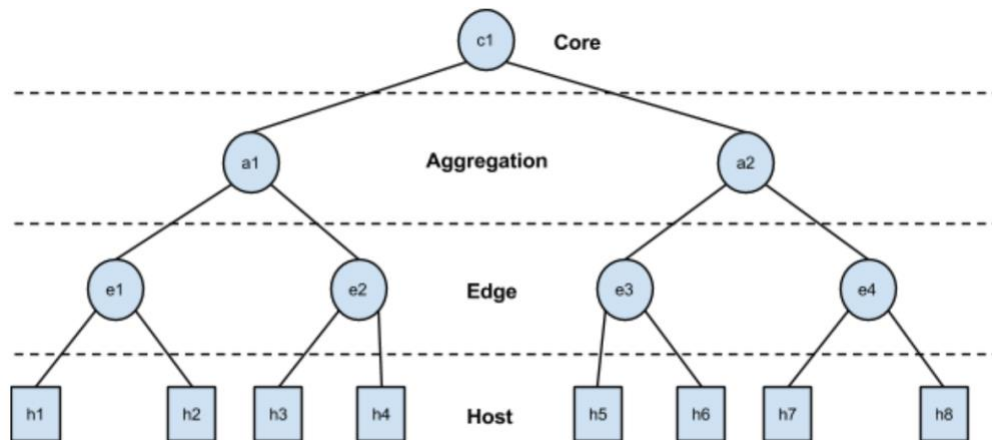
a. Note: Each level (i.e., core, aggregation, edge and host) is composed of a single layer of switches or hosts. (1: Core to Aggregation ,2: Aggregation to Edge ,3: Edge to Hosts)

b. Note: All links between hosts and edge switches will have the same predefined performance parameter. Here there are 3 types of links, the links between core and aggregation switches, the links between aggregation and edge switches, and the links between edge switches and hosts. You will see these specific parameters in the provided skeleton code (**topology-b.py**)

c. Create the topology in (**topology-b.py** or **topology-b.java**)

d. Evaluate Topology B similarly to Topology A (measurements of throughput and latency for links between switches (example: e1 to a1, a1 to c1, ... etc.). Write your answers in a file named (**topology-b-report.pdf**)

Part 3: Design and Implement Ethernet-based self-learning Algorithm

1. Consider Topology C:
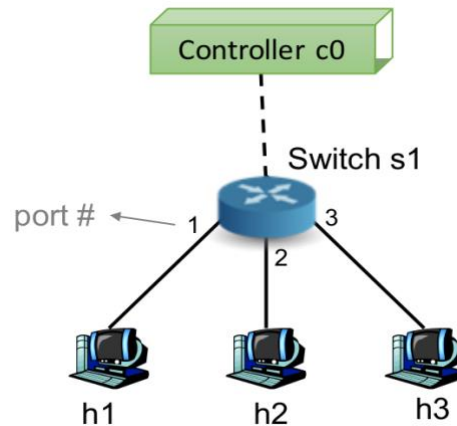
a. **Switch s1** maintains a flow table which contains match-action rules that are added/removed/updated by the controller. At the beginning, there are no rules installed in the flow table. When packets are sent by hosts to one another, the controller extracts information from these packets and builds certain internal data structures (such as a hashmap, etc.) to represent the topology. The controller will use these internal data structures to decide what rules should be installed in the switch's flow table.

b. As an example, consider **host h1** wants to send a packet to **h2**. When **switch s1** receives this packet from **h1**, it checks if there is any corresponding match-action rule (or flow entry) for this packet in its flow table. If there is no entry, the switch will encapsulate the packet and send it inside an **OFPacketIn** message to the **controller c0**. On the other hand, if there is a flow entry, **switch s1** will execute the action associated with the rule. However, as discussed earlier, the flow table is initially empty. Therefore, the switch will forward the packet to the controller. When the controller receives the **OFPacketIn** message from **switch s1**, a packet-in event is triggered. Two tasks are performed due to a packet-in event: 1) using the information from packet headers/event, the controller may update its internal state (e.g., it learns something new about the topology), 2) using the internal state, it decides on what action(s) **s1** should take to send this packet to **h2**. If the controller knows what interface **s1** should forward the packet to, it will inform **s1** and install a flow entry which could be used for future packet transmissions of similar kind. If the controller does not know what interface **s1** should forward use to forward the packet to **h2**, then it will instruct the switch to flood the packet to all its interfaces except the one the packet was received from. This flooding instruction is a one-time action without installing any rules in the switch.

2. **[BONUS]:** Try to come up with an optimized Ethernet-based self-learning algorithm at the controller that <u>does not unnecessarily flood packets while learning the topology</u>.

   a. *Hint: Try to test your algorithm under multiple scenarios such that the controller is able to learn the whole topology quickly without unnecessarily flooding packets. For example, one such scenario could be **h1** sends a packet to **h2**, then **h2** sends a packet to **h1**, then **h3** sends a packet to **h1**. Another direction to think about is if your algorithm should be source-based, destination-based, or a combination of multiple packet header fields.*

   b. Write this in your submission following the **"What to Submit"** guidelines.

3. Implement your algorithm (using the skeleton code **ethernet-learning.py** <u>or</u> **EthernetLearning.java**). You can refer to tutorial to start: https://github.com/att/pox/blob/master/pox/misc/of_tutorial.py

4. Test your algorithm using POX (Python) or Floodlight (Java):

   a. *Against Topology A:*

      - **Comment out** the line "`net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)`" and **uncomment** the line "`#net = Mininet(controller=RemoteController, topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)`". This will allow the topology to connect to your controller.

      - Make sure that each host can reach all the other hosts.

      - Use Ping and iPerf to measure the estimated latency and throughput (you should see latencies and throughput similar to what is defined for each link in the topology definition).

   b. *Against Topology B:*

      - **Comment out** the line "`net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)`" and **uncomment** the line "`#net = Mininet(controller=RemoteController, topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)`". This will allow the topology to connect to your controller.

      - Make sure that each host can reach all the other hosts.

      - Use Ping and iPerf to measure the estimated latency and throughput (you should see latencies and throughput similar to what is defined for each link in the topology definition).

c. When you start your topology (using Python) it will connect to this controller, so run your controller first, then run the target topology.

d. **To run your controller with POX:**

- Put `ethernet-learning.py` file in the directory `pox/pox/samples`. You will then run the controller and your module using the following commands:
  - `you@yourmachine$ cd pox`
  - `you@yourmachine$ ./pox.py samples.ethernet-learning`

e. **To run your controller with Floodlight:**

- **This is a more involved process:**
  - You may have to follow pre-req instructions for modifying the VM
- Steps:
  - Put `EthernetLearning.java` file in `src/main/java/net/floodlightcontroller/ethernetlearning/`
  - We need Floodlight to load the module on startup. First, we must add the module to the loader. This is done by adding the fully qualified module name on its own line in `src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule` We open that file and append this line to the file: `net.floodlightcontroller.ethernetlearning.EthernetLearning.`
  - To load the module, we modify the Floodlight module configuration file to append the `EthernetLearning` module. The default one is `src/main/resources/floodlightdefault.properties.`
    - ○ The key is `floodlight.modules` and the value is a comma separated list of fully qualified module
    - ○ `names.floodlight.modules = <leave the default list of modules in place>, net.floodlightcontroller.ethernetlearning.EthernetLearning`
  - Proceed with re-building the controller using:
    - ○ `you@yourmachine$ cd floodlight/target`
    - ○ `you@yourmachine$ ant`
  - Finally, run the controller by using the floodlight.jar file produced by ant from within the floodlight directory:
    - ○ `you@yourmachine$ java -jar target/floodlight.jar`
  - Floodlight will start running and print log and debug output to your console.

f. *Your code will be executed to show it can properly map out the topology, e.g.: all hosts can reach each other via appropriate paths.*

What to Submit:

A zip file (.tar.xz or .zip formats), named as <StudentID-FirstName_LastName>-project3.zip (or .tar.xz) (example: 1234567-Alice_Bob-project3.tar.xz), to the class canvas site containing:

1) **For Part 1:**
   a) Your evaluation report for Topology A (as **topology-a-report.pdf**)

2) **For Part 2:**
   a) Your Topology B code (as **topology-b.py** or **topology-b.java**)
   b) Your evaluation report for Topology B (as **topology-b-report.pdf**)

3) **For Part 3:**
   a) Your Ethernet-based self-learning algorithm code (as **ethernet-learning.py** or **EthernetLearning.java**)
   b) A report containing:
      i. Description section:
         - Describe at a high level your Ethernet-based self-learning algorithm.
         - Including detailed explanations of your data structures and their purposes.
      ii. A "Pseudocode" section:
         - Describe your algorithm following Pseudocode guidelines.
      iii. (Optional) A(n) comments/assumptions section.

CSci4211: Introduction to Computer Networks
Getting Started with Programming Assignment 3

Disclaimer: This document does not cover all the information but just provides a quick overview and directions on how to get started with programming assignment 3. We expect students to go through the links and references to get more information.

1. Tools Required

Mininet- Mininet is a powerful network emulation tool. It creates a realistic virtual network, running real kernel, switch and application code on a single machine. You can easily interact with your network using Command Line Interface (CLI). Notice that to test connectivity between switches, you need to run a Software Defined Network (SDN) controller.

POX- POX is a Software Defined Network (SDN) controller written in Python. POX provides a framework for communicating with SDN switches using the well-defined application programming interface (API), OpenFlow protocol. The controller exercises direct control over the state in the SDN switches via OpenFlow protocol.

Floodlight- Floodlight is Java-based Software Defined Network SDN controller and is intended to run with standard JDK tools and ant and can optionally be run in Eclipse.

2. How to Get Started with Mininet

2.1. Mininet Basics
The following commands give a brief introduction to mininet basic commands.

To start minimal topology and enter the Mininet CLI:
$ sudo mn
The default topology is the minimal topology, which includes one OpenFlow kernel switch connected to two hosts, and the switch is connected to the OpenFlow reference controller.

To display Mininet Command Line Interface (CLI) commands:
mininet> help

To display nodes:
mininet> nodes

If the first string of the CLI command is a host, switch or controller name, the command is executed on that node. For instance, to show the interface of host h1:

mininet> h1 ifconfig

Test connectivity between hosts. For example,to test the connectivity between h1 and h2:
mininet> h1 ping h2

Alternatively, you can test the connectivity between all hosts by typing:
mininet> pingall

Exit Mininet:
mininet> exit

Clean up:
$ sudo mn –c

For a more complete walkthrough of Mininet, we highly recommend everyone to go through the following link: http://mininet.org/walkthrough/

2.2. Mininet Python API

A more powerful way to construct the network topology is to use Mininet Python APIs. 'mininet.topo.Topo' is the class that defines the network topology. We can define a child class of 'mininet.topo.Topo' and initialize the network topology with the help of the class methods.

addHost(name, opts): add a host to the network with the name being 'name' parameter. opts' is a dictionary that includes the parameters for the host.
Eg: h1 = self.addHost('h1') adds a host with name 'h1'

addSwitch(name, opts): add an Openflow vSwitch to the network with the name being 'name' parameter. 'opts' is a dictionary that includes the parameters for the switch.
Eg: s1 = self.addSwitch('s1') adds a switch with name s1

addLink(node1, node2, port1, port2, opts): add a bidirectional link between port1 of node1 and port2 of nodes. 'opts' is a dictionary that includes the parameters for the link. 'bw' is one of the parameters that stands for bandwidth, and unit is Mbps.
Eg: self.addLink(h1, s1) adds a link between host h1 and switch s1.

Mininet(topo, link, controller) is the initialization function of the Mininet network emulation environment.

3. Getting Started with POX controller

After setting up the network environment in the previous step, a controller is needed to install flow entries to the switch so that packets can be forwarded among hosts.

In general a POX controller consists of three parts:
1. **Listener:** First you need to figure out the type of the event you want the controller to listen to (e.g., ConnectionUp, PacketIn, etc).
2. **Control logic:** Then using some logic you can distinguish between different flows and attach a proper action for a specific flow.
3. **Messenger:** Finally you send the message to the switch to add the new rule in the Openflow table.

More details of the above functionality is provided below.

3.1. Listener
There are two common ways for an application to register with the controller for events.

1. Register callback functions for specific events thrown by either the OpenFlow handler module or specific modules like Topology Discovery
   - From the launch or from the init of a class, perform core.openflow.addListenerByName("EVENTNAME", CALLBACK_FUNC, PRIORITY)
   - For instance you already developed a switching function named switch(). If you want to trig the function when the controller starts to work, add a listener in launch to provoke function switch when ConnectionUp handler is called: core.openflow.addListenerByName("ConnectionUp", switch)
2. Register object with the OpenFlow handler module or specific modules like Topology Discovery
   - From typically the init of a class, perform addListeners(self). Once this is added, the controller will look for a function with the name _handle_EVENTNAME(self, event). This method is automatically registered as an event handler.

3.2. Event handlers
As explained above you need to set a listener on your POX and when an event happens the relevant handler function will be activated. Your control logic should be placed in one of these handlers.
The two main handlers that you might need to modify in your program are:
- ConnectionUp, which activates when a switch turns on (or connects to the controller). The name of the handler function for this event is: _handle_ConnectionUp. Be careful about timers when implementing this handler. ConnectionUp triggers only once when the switch starts to work.

- PacketIn, activates by arriving a packet into the controller. The name of the handler is _handle_PacketIn.

3.3. Parsing Packets

POX provides several primitives to parse well-known packets. The ethernet packet received through a *packet_in* event can be extracted using the following

```
packet = event.parsed
src_mac = packet.src
dst_mac = packet.dst
if packet.type == ethernet.IP_TYPE:
   ipv4_packet = event.parsed.find("ipv4")
   # Do more processing of the IPv4 packet
   src_ip = ipv4_packet.srcip
   src_ip = ipv4_packet.dstip
```

3.4. Useful POX API

- ofp_flow_mod OpenFlow message
    This composes a message which tells a switch to install a flow entry. It has a match attribute and a list of actions. Notable fields are:
    - actions – A list of actions to perform on matching packets.
    - match – A ofp_match object. By default, none of the fields of this object is set. You may need to set some of its fields
- ofp_match class
    This class describes packet header fields and an input port to match on. Fields not specified are "wildcards" and will match on any value.
    For example, create a match which matches packets arriving on port 3:
    match = of.ofp_match()
    match.in_port = 3
- ofp_action_output class
    This is an action which specifies a switch port that you want to send the packet out of.
    There is a variety of pre-defined "port numbers" such as OFPP_FLOOD.
    For example, create an action which sends packet out of port 2.
    out_action = of.ofp_action_output(port = 2)
- connection.send(…)
    This function sends an OpenFlow message to a switch

For example, create a flow_mod that sends packets arriving on port 3 out of port 4
```
msg = of.ofp_flow_mod()
msg.match.in_port = 3
msg.actions.append(of.ofp_action_output(port = 4))
connection.send(msg)
```

Specifically, use [this link](#) to get further information for creating a module using POX.

4. Getting started with Floodlight Controller

Floodlight is Java-based SDN controller and is intended to run with standard JDK tools and ant and can optionally be run in Eclipse.

1) First, you need to figure out the type of the event you want the controller to listen to (e.g., PacketIn, etc).
2) Then using some logic you can distinguish between different flows and attach a proper action for a specific flow.
3) Finally you send the message to the switch to add the new rule in the Openflow table.

4.1. Listener:

First you need to register the event you want to listen to. Note that we are listening to PacketIn messages in the below code.

```
… implements IFloodlightModule, IOFMessageListener{
    protected IFloodlightProviderService floodlightProvider;
    …
    public void init(FloodlightModuleContext context) throws FloodlightModuleException {
        floodlightProvider = context
                .getServiceImpl(IFloodlightProviderService.class);
        floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
        …
    }
    …
}
```

Then you need to handle the packet using the receive function. Your logic goes here.

```
… implements IFloodlightModule, IOFMessageListener{

    …

    public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {

        OFPacketIn pi = (OFPacketIn) msg;
        Ethernet eth = IFloodlightProviderService.bcStore.get(cntx,
                       IFloodlightProviderService.CONTEXT_PI_PAYLOAD);

        …

    }

    …

}
```

4.2. Useful Floodlight API

Creating match

This class describes packet header fields and an input port to match on. Fields not specified are "wildcards" and will match on any value. Use the following lines to create a match.

```
OFMatch match = new OFMatch();

match.setWildcards(Wildcards.FULL.matchOn(Flag.DL_TYPE).matchOn(Flag.NW_DST).
withNwDstMask(24) );

match.setDataLayerType( Ethernet.TYPE_IPv4 );

match.setNetworkSource( IPv4.toIPv4Address("152.3.140.0") );
```

Creating an Action:

In SDN switches are dumb and action tells them what to do with a matched packet
Actions are:
  ● Send packet out to a port
  ● Modify the packet's header
You can create an action in floodlight using the following lines.

```
ArrayList<OFAction> actions = new ArrayList<OFAction>();

OFActionOutput action = new OFActionOutput().setPort((short) 3);

OFActionNetworkLayerSource ofanls = new OFActionNetworkLayerSource();

ofanls.setNetworkAddress( IPv4.toIPv4Address("8.8.8.8") );
```

Creating a Flow Mod Message

This composes a message which tells a switch to install a flow entry. It has a match attribute and a list of actions. Notable fields are:
- actions – A list of actions to perform on matching packets.
- match – A ofp_match object. By default, none of the fields of this object is set. You may need to set some of its fields

Use the following lines to create a Flow Mod Message.

```
OFFlowMod flowMod = new OFFlowMod();

flowMod.setMatch( match );

flowMod.setActions( actions );

flowMod.setLength( OFFlowMod.MINIMUM_LENGTH + OFActionOutput.MINIMUM_LENGTH +
                   OFActionNetworkLayerSource.MINIMUM_LENGTH) );

try {

        sw.write(flowMod, cntx);

        sw.flush();

} catch (IOException e) {

        log.error("Failure writing flowMod", e);

}
```

Check out this link to see how to create a PacketIn packet and this link to see how to create a PacketOut packet. Specifically, use this link for the tutorial to create a module.

References:
https://noxrepo.github.io/pox-doc/html/
https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview
http://mininet.org/
https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch