CSCI4211: Introduction to Computer Networks
Spring 2018
Programming Assignment 3

Ethernet-based Self-Learning using SDN Controller

Due Friday April 20, 11:55 PM

In this project you will learn how to use Mininet to create virtual networks and run simple experiments. You will also learn about the SDN Controllers and OpenFlow protocol. There are 4 sections in this programming assignment. Please read the description and requirements carefully.

1 Design

The topology in Figure 1 has three hosts `h1`, `h2`, and `h3` connected to switch `s1`. The switch is connected to the controller `c0`. Switch `s1` maintains a flow table which contains match-action rules that are 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.

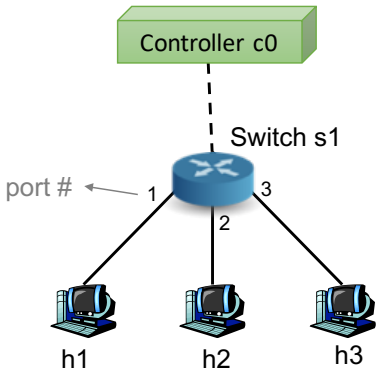

Figure 1: Figure for Section 1

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 as 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 makes a decision 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 also 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 to, then it will instruct the switch to flood the packet to all the interfaces except the one. This flooding instruction is a one-time action without installing any rules in the switch.

Your task is to come up with an Ethernet-based self-learning algorithm at the controller that does not unnecessarily flood packets. More specifically, you need to decide what kind of internal data structures should be maintained by the controller to represent the topology and later make decisions on how to forward packets. These internal data structures are populated by the controller by extracting information from the packet-in events (e.g. packet headers).

Name your submission file as `q1_pseudo_code.txt` – it should include:

1. Detailed information of the data structures and its purpose.
2. Pseudo code which specifies how your algorithm works, how the internal data structures are used and updated, which fields from the packets/events are used by your logic. Please be clear, concise and explicit!

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.*

2 Implementation

In this section, you should implement a controller module that uses the same algorithm/pseudo code that you proposed in the earlier section. However, we will test it on a new topology (see Figure 2 for topology). Note, if your proposed algorithm in the previous section is correct, it should work for this new topology as well without any modification. A Mininet-based Python script (`proj3_topo.py`) is provided which constructs the required topology. For your implementation, you can either use Python-based POX controller or Java-based Floodlight controller.

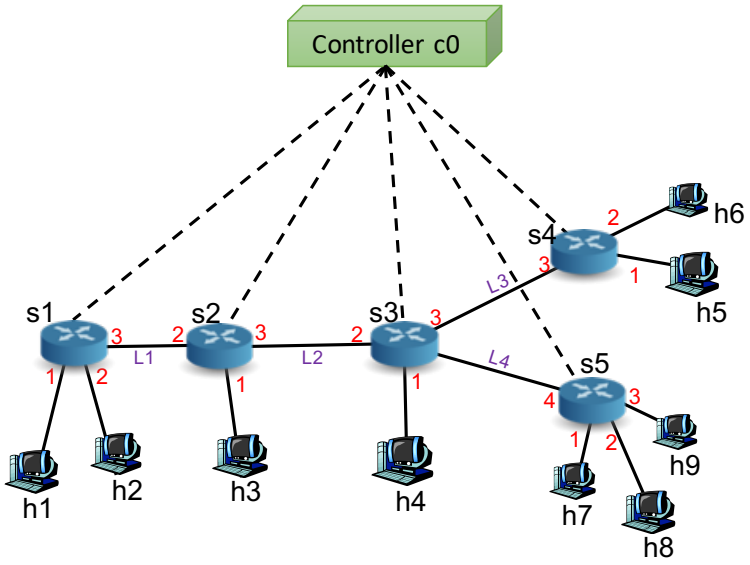

Figure 2: Figure for Section 2

- **POX Controller:** To run your module in POX, you should 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
  ```

- **Floodlight Controller:** To run your own module in Floodlight, put `EthernetLearning.java` file in `src/main/java/net/floodlightcontroller/ethernetlearning/` .

  We need to tell Floodlight to load the module on startup. First we have to tell the loader that the module exists. This is done by adding the fully qualified module name on it's 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`

  Then we tell the module to be loaded. 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.

3 Link Latency and Throughput

After completing Section 2, consider the same topology in Figure 2. The names of hosts, switches and the controller in Mininet match as depicted in the figure. The hosts are assigned IP addresses `10.0.0.1` through `10.0.0.9`; the last number in the IP address matches the host ID. Link labels are depicted in Figure 2. To run Mininet with the provided topology (written in a Python script `proj3_topo.py`) using `sudo` privileges:

```
> sudo python proj3_topo.py
```

To answer this section, you are allowed to use tools such as `ping` and `iperf`.

Is it possible for you to estimate the throughput and latency of individual links between switches, viz, `L1`, `L2`, `L3` and `L4`? If your answer is yes, please provide the estimated throughput and latency for each of the four links and describe the process in estimating them. If your answer is no, then justify your response. Feel free to state any reasonable assumptions you want to make in answering this section. Write your response in a file named `q3_topo_response.txt` and if there are any other supplementary files you would like to submit, please name such files with `q3_topo_` as prefix along with a descriptive file name (e.g. `q3_topo_<DescriptiveFileName>.<extension>`).

4 Building your own Topology

As shown in Figure 3, data center networks typically have a tree-like topology. End-hosts connect to top-of-rack switches, which form the leaves (edges) of the tree; one or more core switches form the root; and one or more layers of aggregation switches form the middle of the tree. In a basic tree topology, each switch (except the core switch) has a single parent switch. Additional switches and links may be added to construct more complex tree topologies (e.g., fat tree) in an effort to improve fault tolerance or increase inter-rack bandwidth. In this section, your task is to create a simple tree topology by updating the skeleton code provided in the `q4_tree_topo.py` file. You will assume each level i.e., core, aggregation, edge and host to be composed of a single layer of switches/hosts.

For creating this topology, you should consider that links at the same levels have some specified performance parameter. For example, 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, links between edge switches and host. Your logic should support whatever bandwidth and

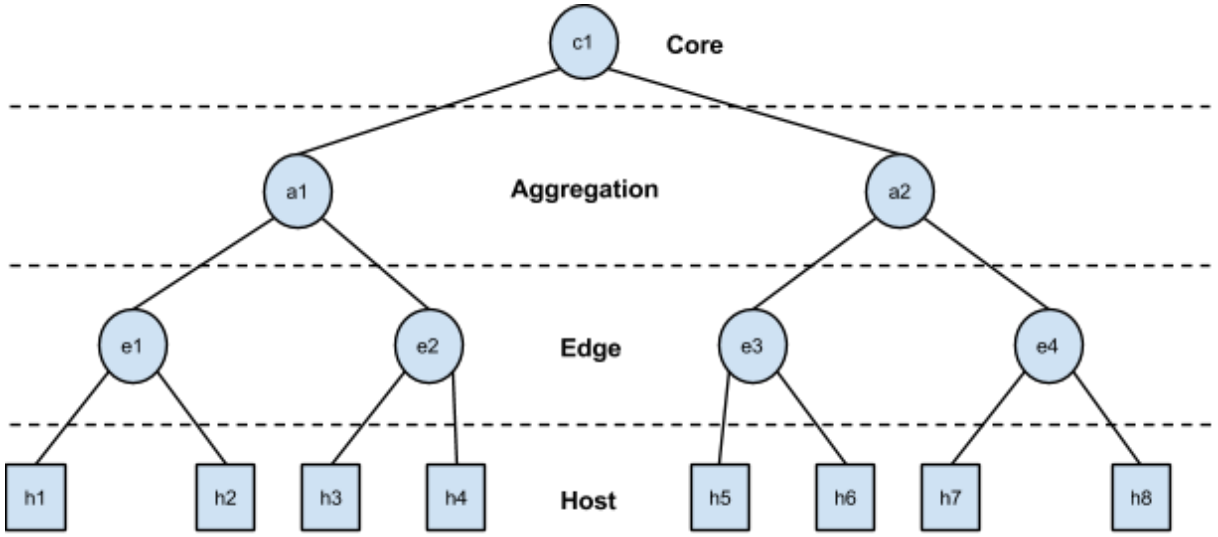

Figure 3: Figure for Section 4

delay parameter you set for each link. The parameters are given in the `q4_tree_topo.py` file. After building your topology, try pinging all hosts to see if the connections are working. Store the output to `q4_tree_topo_output.txt` file. Submit both the files for this section – `q4_tree_topo.py` and `q4_tree_topo_output.txt`.

Methods you need in creating and testing a topology:

- `Topo`: the base class for Mininet topologies
- `addSwitch()`: adds a switch to a topology and returns the switch name
- `addHost()`: adds a host to a topology and returns the hostname
- `addLink()`: adds a bidirectional link to a topology (and returns a link key, but this is not important). Links in Mininet are bidirectional unless noted otherwise.
- `Mininet`: main class to create and manage a network
- `start()`: starts your network
- `pingAll()`: tests connectivity by trying to have all nodes ping each other
- `stop()`: stops your network
- `net.hosts`: all the hosts in a network
- `dumpNodeConnections()`: dumps connections to/from a set of nodes.

Setting up the Environment

We recommend two ways to install the experiment testbed.

1. **Setting up a Virtual Machine (VM) on your local/personal machine**
   This is the easiest and fastest way to get started as this VM will have all the necessary software/tools installed.

   - You can download the VM using the following link:
     Link: `https://z.umn.edu/csci4211s18proj3vm`
   - You will find `Mininet-VM.ova` file downloaded.
   - You can use any virtualization system of your choice, but we recommend installing VirtualBox. It is free and runs on Windows, Linux and macOS. VirtualBox Download Link: `https://www.virtualbox.org/wiki/Downloads` *(we have tried and tested on the latest version of Virtual Box 5.2.8)*
   - Install Oracle VM VirtualBox and start it.
   - In the `File` menu, select `Import Appliance`.
   - The Appliance Import wizard is displayed in a new window.
   - Click `Choose`, browse to the location containing `Mininet-VM.ova` file, and click `Open`, and click `Continue`.
   - The Appliance Import Settings step is displayed.
   - Click `Import` and then select the imported virtual machine.
   - Click the `Start` button.
   - If you get a prompt to change the network settings, click `Change Network Settings`. It should open up a window with the network adapter settings. We recommend you to set "`Attached to`" parameter to *Bridged Adapter* and set "`Name`" to the network interface that you use to connect to the Internet. Then click `OK`.
   - The VM may take sometime to boot up. Login credentials to the VM:
     username: mininet
     password: mininet
   - You are all set, enjoy!

2. **Connect to a VM installed in CSE Lab machines**
   You can use virtual machines that are spread across a cluster of network-accessible machines run by CSE Labs. Send an email to the `varya001@umn.edu` with your group number requesting access to a remote VM. You'll get a reply with unique passwords for your group to access the remote VM along with the instructions of how to connect to it.

What to Submit?

You should upload the project files to the Moodle site. Your submission should be one zip file using the format `groupID_project3.zip` (e.g. `group3_project3.zip` for Group 3). This file should contain the following:

1. `q1_pseudo_code.txt`
2. For POX based code - `ethernet_learning.py`
   For Floodlight based code - `EthernetLearning.java`
3. `q3_topo_response.txt` and other supplementary files for Section 3.
4. `q4_tree_topo.py`
   `q4_tree_topo_response.txt`
5. `README` or comments you would like the TA to know.