ECE 361 Lab #5: Tapping, Blocking and Point-to-Point circuits with SDN

Overview

In this lab, you will learn about Software-Defined Networking (SDN) and gain some basic hands-on experience with making on-the-fly changes to real network flows with OpenFlow. There are 2 Parts to this lab: Tapping and Blocking and Point-to-Point circuits.

In the first part, you will create a virtual network and dynamically alter its default behaviour by implementing a network tap, and by blocking traffic. To do this, you will write short snippets of Python code, using a custom library that allows you to modify the flow tables in the switches.

In the second part, you will implement end-to-end bi-directional circuits within a virtual OpenFlow-enabled network. You will complete a script (which contains skeleton code) that allows you to install end-to-end circuits between any two given hosts. You will leverage the custom library to query information about the network topology, use Dijkstra's algorithm to compute the shortest paths, and install network flows into the switches along the resulting path.

NOTE: This lab will require you to run a virtual machine (VM), using the VirtualBox virtualization software on your own laptop computer. Use the instructions (posted in the portal) to setup your environment. If neither you nor your partner has a laptop computer, we can provide a limited number of remotely accessible VMs that you can SSH into (with VNC enabled). You should have past experience using VirtualBox from first and second year courses.

Part 1: Tapping and Blocking using SDN

As you perform the lab, record your answers to the questions (noted as **Q#**) in a text file name **explanations_lab5.txt**. You will need to submit this file along with other files after completing the lab.

1.1 Manual Operation of Network Data Plane

We first illustrate how in OpenFlow a freshly initiated network data plane, absent any controllers providing instructions, is unable to process any packets. For simulating the OpenFlow-based SDN network, we utilize a simulator called Mininet[1] to set up a simple star topology, where all hosts are connected via a central switch (see Figure 1). The switch in use is a production-grade software OpenFlow switch called Open vSwitch[2] (OVS).

To create this topology in Mininet, run the following command all in one line:

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

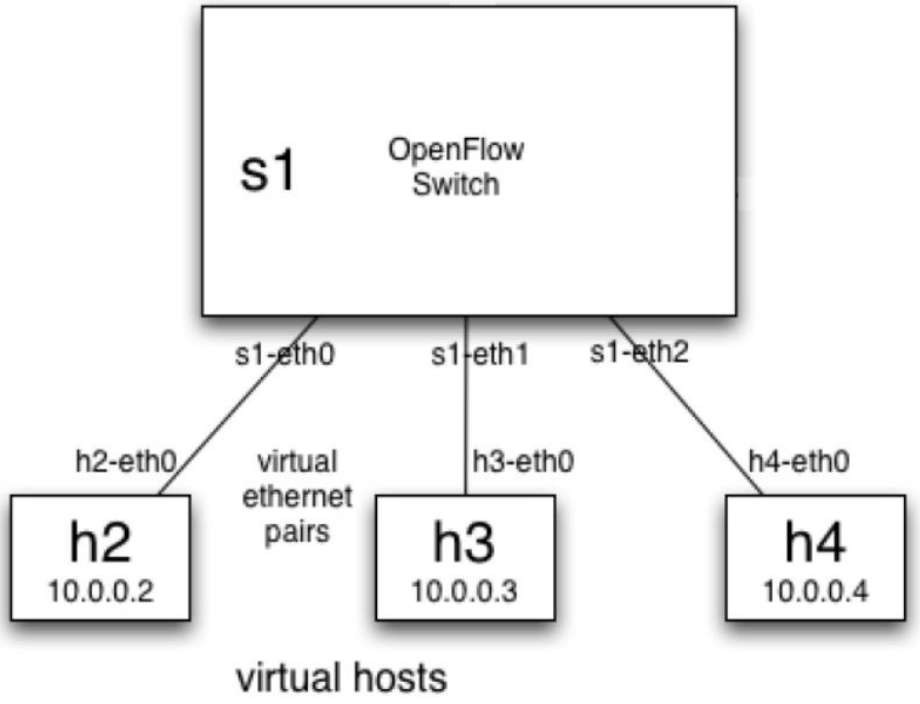

Figure 1: Star Network Topology

[1] Mininet: http://mininet.org/

[2] Open vSwitch: http://www.openvswitch.org/

The command above instructs Mininet to create a topology that consists of a single switch, connected to three hosts (h1, h2, and h3). Once in Mininet, it creates a console-like environment for you where each line starts with the prompt **`mininet>`**. You can run commands in each of the virtual hosts by typing the host name followed by the command (**`<host> <command>`**) in the Mininet console, for example: **`mininet> h1 ifconfig`**
Try to ping from host h1 to host h2 by typing: **`mininet> h1 ping –c4 h2`**

Q.1: Do you observe any successful pings?

While keeping the terminal with Mininet active, open up a second terminal window (and SSH into the VM).

Q2. Dump the current state of the packet forwarding table in the switch by typing: `sudo ovs-ofctl dump-flows s1`

Now insert a flow to allow h1 to send packets to h2, and vice-versa, by issuing the following two commands:

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

Q3. Dump the state of the packet forwarding table again. Note the differences and interpret the flow table.

Q4. Now in the Mininet terminal, try pinging from h1 to h2 again. Are the pings successful? Why?

1.2 Starting Network Controller

Previously, you manually installed flows into the switch in order to define the network's behaviour. The next step is to start a network controller daemon process, which the switch will automatically connect to. The controller will be able to provide instructions to the switch on how it should handle its packets by installing the necessary flows on-the-fly.

The controller used in this lab is called **`Ryu`**, which comes pre-installed in the VM you are using. The default instructions it provides to the data plane emulates the logic of a simple layer-2 learning switch[3]. Start the network controller (**`Ryu`**) by issuing the following command from the second (non-Mininet) terminal: **`ece361-start-controller switch`**

[3] Basic operation of learning switches: https://telconotes.wordpress.com/2013/03/09/how-a-switch-works/

If at any time you want to stop it, you can issue the command:
`ece361-stop-controller`

In order to pull the latest version from github, run: **`ece361-update`**

Q5. Dump the state of the packet forwarding table again. Note the differences and see if you can interpret the flow table.

Q6. With the controller started, dump of the state of the packet forwarding table, then ping from any host to any other host, and dump the packet forwarding table again to note the differences. Record these differences and interpret them.

1.3 Tapping Traffic

Through a programmatic interface provided to you, you are to override the default behaviour within the network. This abstraction is provided in the form of a Python library, **`ryu_ofctl`**, that allows you to call functions that can install and remove flows into the switch, hence, allowing you define the behaviour of the network via software. Note that the functionality of this library leverages the controller in the previous section, so ensure it is running first.

In this part of the lab, you will install flows in order to tap the traffic flowing between h1 and h3. This involves duplicating each packet being sent between h1 and h3, and sending the duplicate copy to h2, allowing h2 to monitor the inter-communications of h1 and h3.

TASK

From the non-Mininet terminal, use the Python **`ryu_ofctl`** library (a few sample workflows can be found in the Github repository) to write a short Python script that will install the necessary flows to implement the network tap.

Refer to the following README for examples:
https://github.com/t-lin/ryu_ofctl/blob/master/README.md

To check if your code is working, it may help to open up a separate terminal session for just h2. To do this in the Mininet console, run: **`mininet> xterm h2`**
In the resulting terminal that pops up, which resides within h2, you can listen on its network interface for all traffic (incoming or outgoing) by issuing the command: **`tcpdump -n -i h2-eth0`**

To test your implementation, from the Mininet console, ping between h1 and h3. If the tapping is working, h2 should be able to see all the traffic between the two other hosts.
After completing the script, save it as **tapping_traffic.py**

1.4 Blocking Traffic

In this part, you will install flows to block the traffic flowing between h1 and h3. Blocking involves inserting a flow that matches on certain header fields, while leaving the actions empty. Note that the blocking should be specific to the communications between these two hosts (i.e. both h1 and h3 should still be able to ping h2).

TASK

From the non-Mininet terminal, use the Python `ryu_ofctl` library to write a short Python script that first flushes the flow table of the switch (to clear any flows you installed as part of the previous tapping exercises), then installs the necessary flows to implement the blocking functionality.

To test your implementation, simply try to ping between the two hosts within the Mininet console and observe if the blocking operation succeeds.

After completing the script, save it as **blocking_traffic.py**

PART 2: Point-to-Point circuits

In this part of the lab, you will implement bi-directional circuits in a virtual OpenFlow-enabled network. While traditional link-state and distance-vector routing protocols can accomplish the same task, they were designed to function in a distributed fashion, and thus takes a longer time to converge when changes occur in the network (e.g. link failure, bandwidth fluctuations, etc.). One of the key advantages of Software-Defined Networking (SDN) is notion of a logically centralized controller that has knowledge of the entire network state, allowing it to react quickly to network changes and choose the best end-to-end paths with up-to-date state information.

The topology APIs you will be using in this lab are similar in function to real APIs used in live production cloud data centres to manage the network traffic; the only difference here is the scale of the network you are operating.

2.1 Starting Network Controller

If you had the network controller running before, you need to first stop the controller: `ece361-stop-controller`

And then starting it again (without any arguments): **`ece361-start-controller`**

2.2 Topology Setup

Please make sure that you exit the Mininet consoles used in previous sections. For this part, we will use Mininet to create the following topology (shown in figure 2). In order to create this, you need to get the topology file using this command:
`wget` https://raw.githubusercontent.com/t-lin/ece361-vm/master/custom-topo.py

Then, run the following command:

```
sudo mn --custom ~/custom-topo.py --topo mytopo --mac --arp
--switch ovsk --controller remote
```

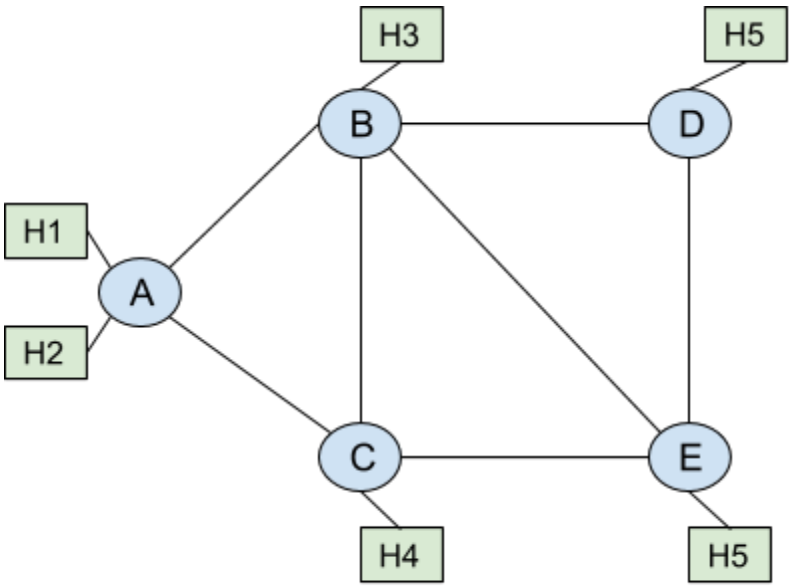

Figure 2: Topology for Part 2

At this stage, if you try to ping from any host to any other host, you should **not** see any successful replies. Try pinging from h1 to h2 from the Mininet console:
`mininet> h1 ping –c4 h2`

While keeping the terminal with Mininet active, open up a second terminal window. Dump the current state of the packet forwarding table for s3 (the switch that interconnects h1 and h2) by typing: **`sudo ovs-ofctl dump-flows s3`**

When you dump the table, you should observe only one flow entry. This flow rule is automatically installed by the controller, and specifically matches on Link Layer Discovery Protocol (LLDP) packets, with an action to forward these packets to the controller. This rule is used to help the controller learn the

underlying network topology. A similar rule was automatically installed into every switch in the network by the controller.

2.2 Querying Topology Information

In a fresh terminal window, type **`Python`** to start a Python console. First, import the **`ryu_ofctl`** library, then try out the four topology-related APIs:

- **`listSwitches()`**
  - Returns dictionary with a list of switch IDs (a.k.a. datapath IDs)
- **`listLinks()`**
  - Returns dictionary with a list of all links in the network topology
- **`listSwitchLinks(dpid)`**
  - Returns dictionary with a list of all links connected to a given datapath ID (dpid)
- **`getMacIngressPort(mac)`**
  - Returns the datapath ID (dpid) and physical port number where the given MAC address was first seen coming into the network

Refer to the following README for examples:
https://github.com/t-lin/ryu_ofctl/blob/master/README.md

Two important things to note about the links returned by **`listLinks()`** and **`listSwitchLinks()`**:

1. Links only exists between two switches (i.e. it doesn't show the links from switches to hosts)
2. Links are uni-directional. For this lab, we allow full bi-directional communication between any two neighbouring switches, thus you should observe two entries that are opposites of one another.

2.3 Installing Shortest-Path Circuits

As part of the update process for the VM, a script named **`install_path.py`** should have been created in your home directory. This script contains skeleton code for installing end-to-end bi-directional circuits between any two hosts specified by their MAC addresses. Currently, the script does not do anything other than validate the input parameters and print out a message. You can execute the script as follows:

```
python install_path.py 00:00:00:00:00:01 00:00:00:00:00:02
```

Since the **`ryu_ofctl`** library allows you to query the entire topology of the network, we can apply a shortest-path algorithm to find paths from any host to any other host within the network. For this lab, you will implement Dijkstra's

Algorithm to find the path between any two hosts, and use the flow installation APIs you learned from the previous lab to install bi-directional flow rules in each switch along the path.

Your task is to complete the script, filling out the sections labelled for you with the comments "**##### YOUR CODE HERE #####**". It is suggested you read the comments as they may provide helpful information and suggestions. You are free to modify the script however you like, so long as you do not hardcode the paths (i.e. we may re-run the script with a different topology and it should still work).

To test your implementation, from the Mininet console, ping between h1 and h2 (single-hop case). Then, run your script with the MACs of h1 and h2 and observe whether or not the pings start going through. If you ever need to troubleshoot your flows, you can always dump the flow tables of the switch. Repeat the test with h1 and h6 (multi-hop case).

Hints: Python List and Dictionary

Two Python data structures that you will have to use as part of this lab are the List and Dictionary structures, essentially equivalent to Vector and Map structures in C++. You can familiarize yourself with their built-in methods here:

- https://docs.python.org/2.7/tutorial/datastructures.html#more-on-lists
- https://docs.python.org/2/library/stdtypes.html#mapping-types-dict

Hints: Securely Copy Files within VM to EECG machines

In order to submit the python files, you need to copy the files from the VM to EECG machines. You can do that using the following command from within the VM: **`scp <file_name> <utorid>@ug221.eecg.utoronto.ca:~`**
For more information, refer to the following:
https://haydenjames.io/linux-securely-copy-files-using-scp/

Demonstration

Once you completed the exercises, **signal to the Lab TA that you are ready to be marked**. Be prepared to show your code to the TAs, discuss the results, and answer any questions from them. All code should be properly commented.

Submission Instructions

You must submit the following files:

- Your Tapping Traffic python script (must be named **tapping_traffic.py**)
- Your Blocking Traffic python script (must be named **blocking_traffic.py**)
- Your Install Path python script (must be named **install_path.py**)
- Your answers to the questions from **1.1** and **1.2**. It must be named **explanations_lab5.txt** and should be concise as possible. This file must also contain the names and student numbers of the group members (at the beginning of the file) prefixed by **#**. Please do not prefix other lines by **#** as this would confuse the automated scripts.

```
#first1 last1, studentnum1
#first2 last2, studentnum2
```

You must combine all these files into a tarball (tar.gz format) using the following command:

```
tar –czvf lab5.tar.gz <project directory>
```

Note: Only one student in the group needs to submit that tarball.

You can submit the file using the following command:

```
submitece361s <lab_number> <filename>
```

Example: **`submitece361s 5 lab5.tar.gz`**

You can verify if you have submitted successfully by using the following command: **`submitece361s -l 5`**

For more information regarding the command, please refer to it's man page: **`submitece361s`**

The submitted files will be used to verify your findings and check for plagiarism.

Marking

This lab is worth a total of **3** marks with the following breakdown:

- Part 1.3 Code for performing network tap: **0.5** mark (marked as group)
- Part 1.4 Code for blocking traffic: **0.5** mark (marked as group)
- Part 2.3 Code for installing path: **1** mark (marked as group)
- Questions from TAs: **1** marks (marked individually)

All marks will be assigned by the end of the lab session.