End of Term Assignment – A Network on a Laptop

CS456/656 Computer Networks

Spring 2020

1) Objective

The goal of this programming assignment is to get hands-on experience by simulating a virtual network using Mininet. This will be an opportunity to comprehensively review the Internet Protocol (IP) and its forwarding fabric in a virtual Software Defined Network (SDN). Specifically, we will experiment with routing configurations and the OpenFlow protocol, which are the building blocks of a SDN. You will begin with setting up Mininet and then implement a virtual network topology, using Mininet's Python API. Next, the switches in the virtual SDN will be configured to forward packets according to given rules and specifications. If the configurations are correct, the virtual hosts inside the virtual network will be able to communicate with each other. This end of term programming assignment is divided into three parts: Part A, B and C, with a bonus Part D. Graduate, especially CS, students are encouraged to attempt Part D. Out of department CS graduate students can consider incorporating and, or extending Part D to be their graduate project for this course.

2) Background

You will find necessary background material in the following textbook chapter sections and articles:

a. Chapter 4 Section 4.4 Generalized Forwarding and SDN
b. Chapter 5 Section 5.5 The SDN Control Plane
c. Mininet Documentation at http://mininet.org/overview/
d. More background material here

3) Specifications

To setup and install Mininet.

i. For Ubuntu users, install the latest version of Mininet in Ubuntu, using `Option 2: Native Installation from Source`.

ii. For Windows and OS X users, use `Option 1: Mininet VM Installation (easy, recommended)`. Windows and OS X users, need to use a Virtualbox VM available at the official releases page or an Ubuntu VM of their own choosing (≥14.04) since Mininet uses OS-level virtualization technologies only available on Linux. Please make sure to test your installation by running the test topology "`sudo mn --test pingall`" as illustrated in the last step in Option 2.

Troubleshooting Installation

If the command complains that Open vSwitch is not installed, download the snap-shot for Open vSwitch (OVS) 2.10. Unpack and install OVS 2.10. Continue to follow the instructions to unpack, build and install your version of OVS. To ensure successful installation, please follow through the post install steps according to the tutorial linked above:

```
ovs-vsctl show
modinfo openvswitch | grep version
```

to make sure that Open vSwitch and its kernel module are installed.

Please note that if ovs-controller is not installed, running the test topology "`sudo mn --test pingall`" will fail. The controller is the brain of SDN. However, as we will be working *controller-less* with static flow table entries, you can ignore the controller by using command "`sudo mn --test pingall --controller=none`"

These components are pre-installed in the newer distributions of Mininet and on the VM. If your setup is complete and correct with Open vSwitch and ovs-controller installed, `pingall` should succeed.

a. **Part A: Hands on Mininet:** Mininet allows you to simulate arbitrary network topologies. To define these custom topologies, Mininet use Python API that are installed with the Mininet package. Download the sample topology python script and run it in Mininet using command "`sudo python topology_filename.py`" to create the 10 host and 10 switch topology. Once this topology is running, you can open a terminal (`xterm`[1]) for any VM (hosts/switches) and run an application, such as `ping`, in it.

iii. To do this, run the `xterm` command from inside the Mininet shell

```
mininet> xterm h0
mininet> h0 ping h1
```

iv. Note, if you try to `ping h1` i.e. try "`ping 10.0.1.2`", using IP address of `h1` as defined in the topology, it will fail. Note, hosts `h0` and `h1` are on different networks and there are no routing entries in switches `s0` and `s1` to route the ping packets between these hosts. Refer to the Mininet walk through for more Mininet commands.

v. Our goal is to setup the routing entries to enable host `h0` ping host `h1`. In Mininet CLI, open `xterm` terminal for the switch s0, and type the command "`ovs-vsctl set bridge s0 protocols=OpenFlow13`" to

[1] The `xterm` commands requires a graphical interface. If you are accessing the machine running Mininet through SSH, you should either (i) use X11 forwarding (ii) use the command format "`h0 bash`" instead, which opens a shell on h0.

enable OpenFlow 1.3 protocol in the OVS switches. Now, add following flow table entries to switch `s0`

```
ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6634
in_port=1,ip,nw_src=10.0.0.2,nw_dst=10.0.1.2,actions=mod_dl_src:0A:00
:0A:01:00:02,mod_dl_dst:0A:00:0A:FE:00:02,output=2

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6635
in_port=2,ip,nw_src=10.0.0.2,nw_dst=10.0.1.2,actions=mod_dl_src:0A:00
:01:01:00:01,mod_dl_dst:0A:00:01:02:00:00,output=1
```

Now, repeat the command for `s1`, to enable the OpenFlow Protocol 1.3 in the OVS switch and add the flow table entries

```
ovs-vsctl set bridge s1 protocols=OpenFlow13

ovs-ofctl -O OpenFlow13 add-flow
tcp:127.0.0.1:6635
in_port=1,ip,nw_src=10.0.1.2,nw_dst=10.0.0.2,actions=mod_dl_src:0A:00
:0A:FE:00:02,mod_dl_dst:0A:00:0A:01:00:02,output=2

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6634
in_port=2,ip,nw_src=10.0.1.2,nw_dst=10.0.0.2,actions=mod_dl_src:0A:00
:00:01:00:01,mod_dl_dst:0A:00:00:02:00:00,output=1
```

You can review the commands installed in each switch using the command "`ovs-ofctl dump-flows s1`" (replace `s1` with the switch name, for other switches).

vi. Now retry the ping. Does it work from `h1` to `h0` too? Explain what the flow entries that you entered do.

vii. Now we want the following pairs to be able to ping each other:
`h2` ↔ `h4`
`h1` ↔ `h6`
`h0` ↔ `h3`

b. **Part B: Custom Topologies:** Simulate the network illustrated in Figure 1 using Mininet. You need to demonstrate that ***"Alice"*** and ***"Carol"*** can **ping** each other. You will need to submit your python script for creating and setting up the corresponding topology, switch flow commands to enable ping between ***"Alice"*** and ***"Carol"***, and ping snap-shots. Note that in this topology, hosts and routers of each subnet are connected to the same bus. To implement the bus behaviour in Mininet, you can instead use an L2 switch in the middle (which does not necessarily have to be OpenFlow enabled) and connect all devices of each subnet to that switch.

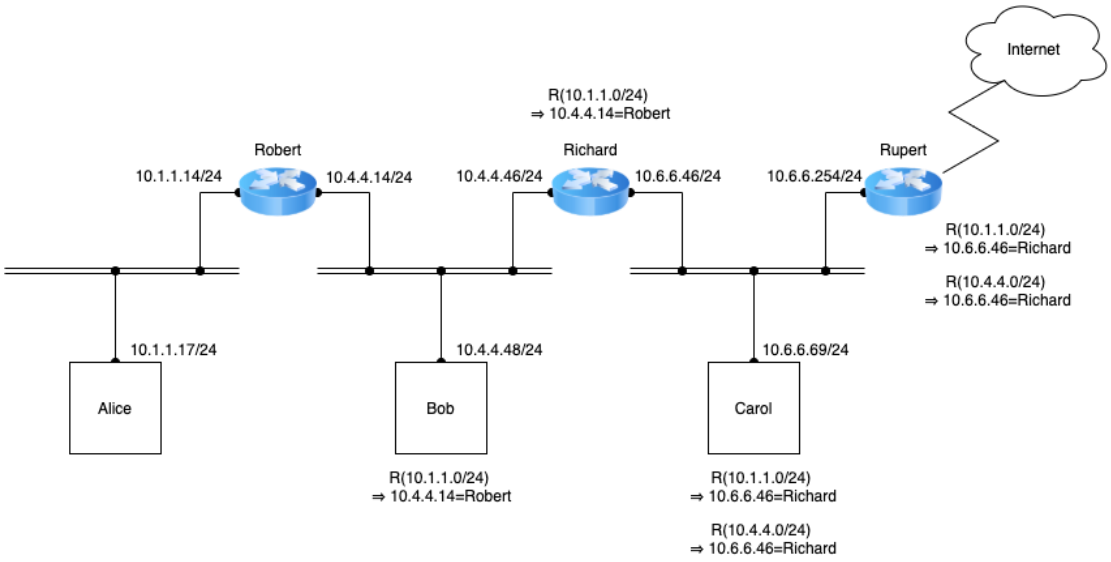


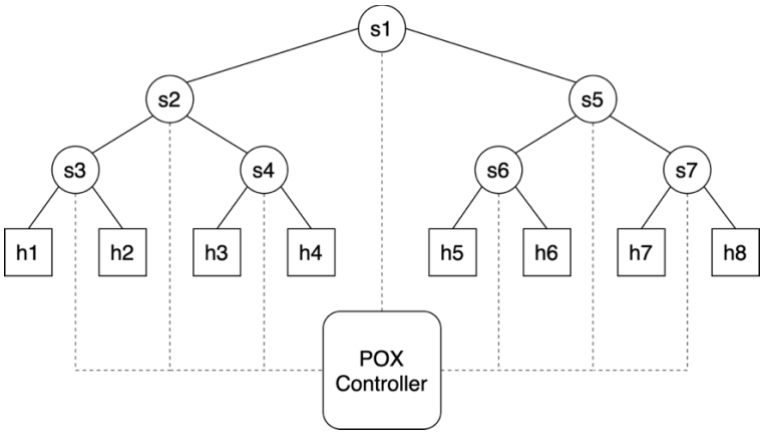
Figure 1. Topology for Part B: Custom Topologies.[2] The hosts and routers are shown using squares and blue cylinders respectively, with each black dot on them indicating an NIC.

c. **Part C: Introducing the Controller:** In this part, we explore the SDN controller and its value for configuring networks from a centralized component. In parts A and B, you have seen how configuring networks at a low level, even for achieving very simple requirements, can become complex and exhaustive. However, in real world, it is often not a human operator that comes up with those OpenFlow rules! The controller is a centralized software that acts as the brains of the SDN network. Using the SDN controller, you can instruct high-level policies, such as "host A should be able to access host B", and let the controller figure out how that translates to low-level OpenFlow commands and deploy it on the network switching devices. POX is one such software which is going to be used as the network controller in this assignment. Install and run POX by cloning its git repository:

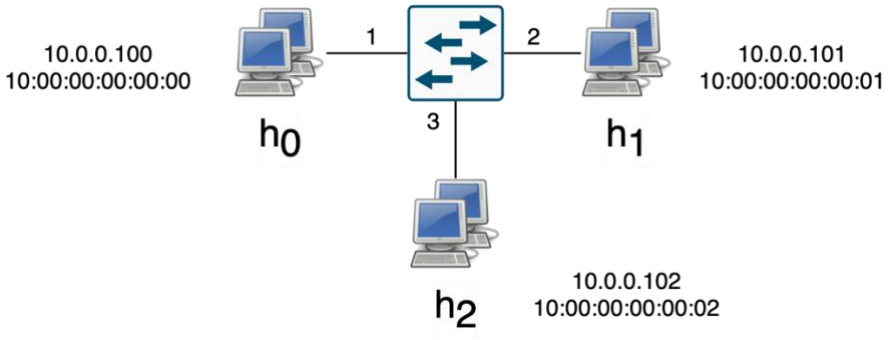
```
git clone https://github.com/noxrepo/pox.git
cd pox
git checkout dart
./pox.py log.level --DEBUG forwarding.l2_learning # note that it is
the lowercase L in l2, not to be mistaken with number 1
```

In a separate terminal, use the command, "`mn --topo=tree,depth=3 --controller=remote --mac --switch ovs`" to quickly create the tree-like topology below, connected to the POX controller started above.

[2] The notation **R(x.x.x.x/x) ⟹ y.y.y.y** means that the packets going towards the **x.x.x.x/x** subnet should be routed via **y.y.y.y**.



d. **Part D: Middle Box [optional]:** A Middle Box is a real or virtual device that is placed inside the network and is used the perform some network function. A middle box network function might modify the network traffic passing through the middle box or passively collect information about it. For instance, a firewall is a network function that matches the traffic passing through it against a set of configured rules and filters out parts of the traffic e.g. dropping all traffic for a certain TCP port or allowing traffic only from certain IP addresses. Imagine a network with only three hosts connected by a single switch. Suppose we want the payloads of all UDP packets going from `h0` to `h1` to be automatically encrypted using Caesar's code by a network function running on `h2`. Note that this network function should be transparent to `h0` and `h1` i.e. the packets arrive at their destinations as if the communication was directly between `h0` and `h1` and the encryption took place at the origin or source host.



4) Deliverables

a. For Part A, submit a file named `question_a.md`, with the OVS rules to enable the following pairs of hosts to ping each other and only each other.

`h2` ↔ `h4`
`h1` ↔ `h6`
`h0` ↔ `h3`

It is important to note that other host pairs should not be able to access each other. Include the full OVS commands (like the ones in the instructions above) and be clear about which switch each command is supposed to be run on.

b. For Part B, submit the Python code for designing the topology for Part B, in a file named `custom_topology.py`. Write the OVS rules to allow pinging between Alice and Carol and place it in a file named `question_b.md`. Include the full OVS commands and be clear about which switch is each command is supposed to be run on.

c. For Part C, submit a file named `question_c.md` that answers the following questions:

   i. Try pinging from `h1` to `h5`. The ping should succeed, despite the fact that you have not installed any rules on the switches yourself. Study the output of the POX program, what has the controller done? Include the `DEBUG` output as well as your interpretation of it.

   ii. Record the ping `RTT` times for the first 10 ping messages. Compare the `RTT` of the first ping message with the subsequent ones. Is there a difference? Why or why not?

   iii. Open a terminal in `s1` and dump the flow rules installed on the switch using this command, "`ovs-ofctl dump-flows s1`." These rules are automatically installed by the POX controller. Keep in mind that the OpenFlow rules installed on each switch expire in a few seconds, so if the command returns empty, retry after another ping command (unless there are actually no rules ever installed on the switch). Dump the flow rules on all the switches and include them in your answer (make sure you change `s1` in the command above to the proper switch name. Be clear about which rule pertains to which switch). Are they similar to the rules you defined in part A? Briefly explain what they collectively achieve. Note that before opening the terminal to dump flows of each switch, it's best to repeat the ping for h1 to h5 to make sure the flow rules are not expired.[3]

d. For Part D, submit a file named `question_d.md`. Be brief, but clear in describing your approach.

   i. Describe how OpenFlow protocol can be used to place `h2` as a middle box for modifying the UDP traffic between `h1` and `h0`.

   ii. Write the OpenFlow rules to implement your approach.

[3] Some switches might really have empty flow rule tables, even after the ping. Can you guess which ones?

iii. Explain what the network function running in `h2` will look like i.e. what actions does it perform when it receives a new packet. You can outline the program running in `h2` in a pseudo-code[4]

5) Instructions for Submission

Submit all your files in a single compressed file (.zip, .tar etc.) using End of Term Assignment drop box in LEARN. Remember, you cannot share your program or work with any other student or in groups. This programming assignment is to be completed individually. You **will** lose marks if your answers are not legible.

You must hand in the following files / documents:

- Source code file from Part B, `custom_topology.py`
- Answers to questions in Parts A, B, C and D (optional) in files named, `question_a.md`, `question_b.md`, `question_c.md`, `question_d.md` `(optional)`, respectively
- **README** file must contain instructions on how to run your program and what version of `make` and compilers you are using.

There is **no** late submission for the End of Term Assignment. You are encouraged to start early and install and setup Mininet as soon as possible, so that you have plenty of time to seek help in installation and setup.

6) Additional Notes

a. These examples **linuxrouter.py** and **topo-2sw-2host.py** are very useful as a starting point.
b. Be clear about all the questions that you are answering, keeping answers brief and concise.
c. For answering the OpenFlow questions in Parts A, B and C, include the full rule command. The commands will be tested and if they fail, you will lose the respective marks.
d. This assignment does not require a `Makefile`. Yet, `custom_topology.py` is expected to run seamlessly in the VMs provided by the Mininet project. No dependencies should be required other than the **mininet** package itself which is provided automatically when installing Mininet.
e. Using the `Markdown` syntax in the answer files is not required but encouraged.

[4] Do not use the pseudo code to implement the Caesar's code itself. Assume that it is available to you and you can simply use it as a function.

7) Grading Rubric

No	Scenario Description	Failure Penalty
Part A: Hands on OpenFlow (25%)		
1	Question A-I: Explain the purpose of the given OpenFlow rules clearly and correctly	5%
2	Question A-II: Correct structure and executability of the commands	5%
3	Question A-II: Achieving at least one-way communication of the host pairs, and correct connectivity in adjacent hops	5%
4	Question A-II: Achieving all required pairwise communications and nothing more than that	10%
Part B: Custom Topologies (35%)		
5	Question B-I: Correct specification of the network links and topology	10%
6	Question B-I: Correct specification of the hosts and IP assignments	10%
7	Question B-II: Appropriate structure of OVS rules	2%
8	Question B-II: Achieving at least one-way connectivity or partial connectivity in adjacent nodes	3%
9	Question B-II: Achieving end-to-end connectivity	10%
Part C: Introducing the Controller (30%)		
10	Question C-I: Correct `DEBUG` logs	2%
11	Question C-I: Explaining what the controller is doing	5%
12	Question C-II: Ping output and observation	2%
13	Question C-II: `RTT` difference justification	6%
14	Question C-III: Correct flows from all switches	10%
15	Question C-III: Explaining the collective effect of the installed flows concisely in simple language	5%

General (10%)		
16	README briefly specifies the running environment and set-up steps	5%
17	Coding style, comments, readability	5%
Part D: Middle Box [Bonus] (+25%)		
18	D-I: Strategy described for placing middleware on `h2` during `h0-h1` UDP communication	5%
19	D-II: Correct OpenFlow rules for averting the traffic to `h2`	15%
20	D-III: Middle-ware procedure is sound and well-structured	10%