Software-Defined Networking

Lab 5
Midterm Lab

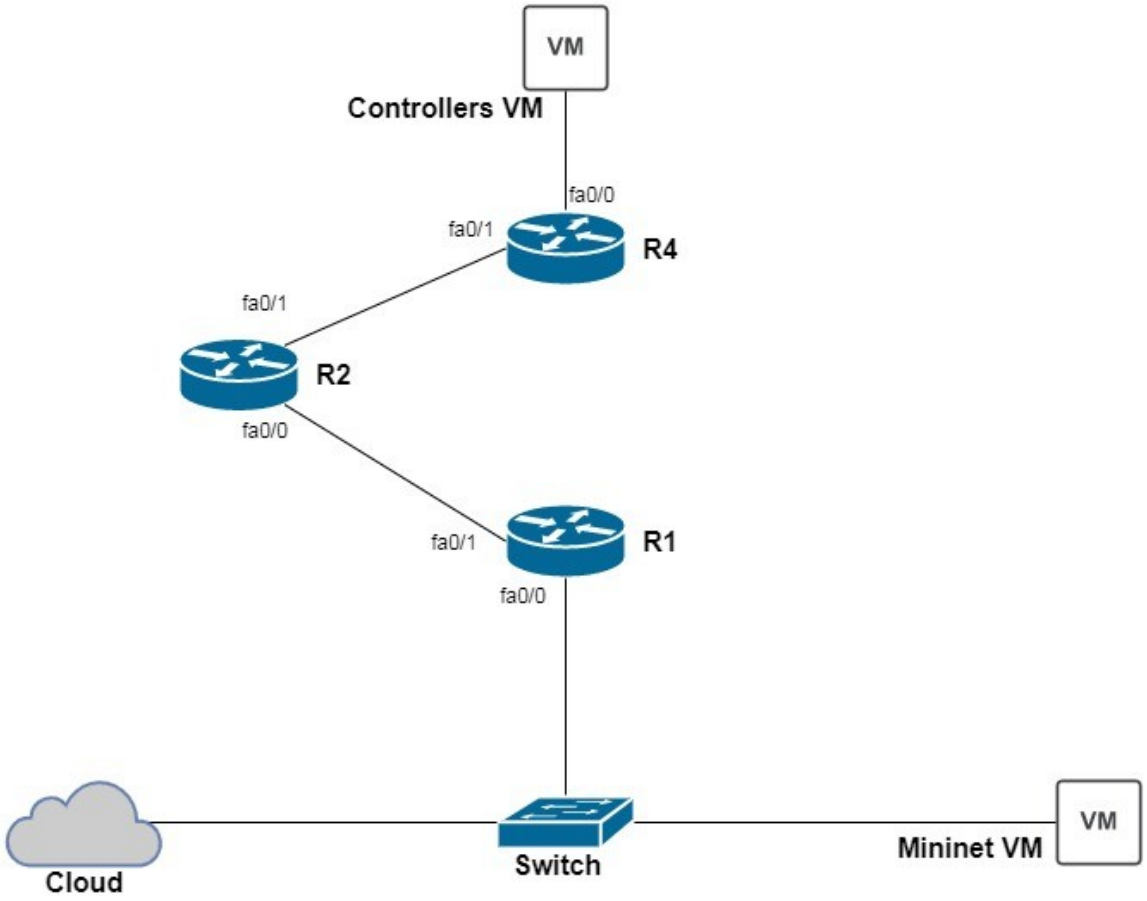
University of Colorado Boulder
Department of Computer Science

Professor Levi Perigo, Ph.D.

Summary

The objective of this lab is to recall and apply all the knowledge you have gained so far in this course. You will utilize the knowledge of traditional networking, software-defined networking, virtual switches, controllers, OpenFlow, packet capturing, GNS3 and Python. Students are encouraged to expand on the topics for additional learning and experiments.

Objective 1 – Set up topology in GNS3



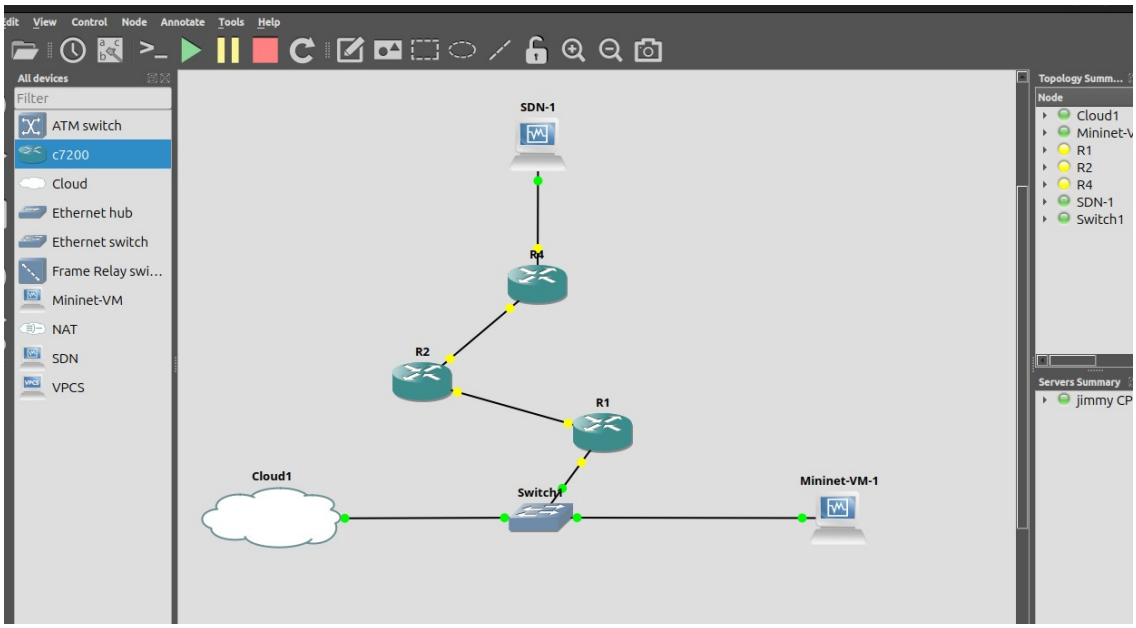
Device	Interface	IP address
R1	fa0/0	192.168.100.1/24
	fa1/0	192.168.200.1/24

R2	fa0/0	192.168.200.2/24
	fa1/0	172.16.100.2/24
R4	fa0/0	10.20.30.1/24
	fa1/0	172.16.100.1/24
Controllers VM		10.20.30.2/24

1. Configure the above topology in GNS3 interconnecting traditional routers, virtual machines, cloud and host laptop.
2. Configure IP addresses on the routers and the Controllers VM as given. Do not configure any IP address on the Mininet VM.
3. Configure DHCP server on R1 to provide an IP address to the Mininet VM.

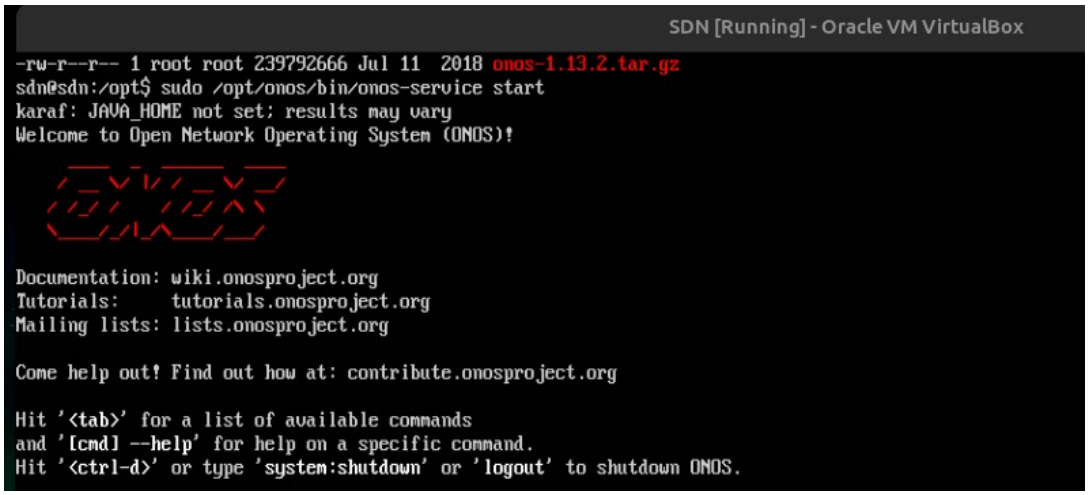
```
mininet@mininet-vm:~$ sudo dhclient -v eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/08:00:27:84:36:92
Sending on   LPF/eth0/08:00:27:84:36:92
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3 (xid=0xffb11632)
DHCPOFFER of 192.168.122.80 from 192.168.122.1
DHCPREQUEST for 192.168.122.80 on eth0 to 255.255.255.255 port 67 (xid=0x3216b1ff)
DHCPACK of 192.168.122.80 from 192.168.122.1 (xid=0xffb11632)
RTNETLINK answers: File exists
bound to 192.168.122.80 -- renewal in 1367 seconds.
mininet@mininet-vm:~$
```

4. Do not configure any routing commands manually.
5. Paste screenshot of the topology created in GNS3. [**15 points**]



6. Start the Ryu app simple_switch_13.py on the controllers VM.

   I wanted to use ONOS on this one, so I put the ONOS controller on my VirtualBox VM that I created for this lab.



Objective 2 – Python script

1. Write a script in Python which runs on your laptop to achieve the following objectives (Please read all the objectives before you begin to write your script)-

a. SSH into R1 and find the IP address leased out to the Mininet VM. Paste relevant screenshots. [**20 points**]

In the Mininet network with my lab setup, I had R1, my VM (Cloud1), and the Mininet VM. The interface that I was using on my VM to communicate with the topology was the virbr0 interface and by default that is running a DHCP server. I had some odd behavior initially until I found that the Mininet VM was getting an IP address from that DHCP server instead of R1, so there were two DHCP servers on the network instead of one. I decided not to fight the virbr0 DHCP server and allow the Mininet VM to get an IP address from my 'NMAS' VM instead of R1. Instead of skipping this part of the lab, and possibly missing out on points for this assignment, I wrote a function to find the IP address from R1 in my code, but it uses a couple of ways instead of just 'show ip dhcp binding'. The code will start with this check, if there are no bindings, the code will then check arp entries for the Mininet VM's MAC, which always returns something.

```
jimmy@jimmy:~/JDV/Semester5$ sudo ./sdn_midterm_2.py
Connecting to R1 (192.168.122.2) to find Mininet VM IP address
Found ARP entry 192.168.122.80 -> 0800.2784.3692
Using Mininet IP: 192.168.122.80
```

b. SSH into the Mininet VM using the IP address found in the previous step and initialize the default Mininet topology (sudo mn). Paste relevant screenshots. [**20 points**]

This part was a little tricky. My code needs to initialize Mininet, keep the topology running and alive, but also be able to run ovs commands in the background or logout and log back in to do show commands (it's not ideal or practical to do one big Paramiko session for this whole assignment). In order to achieve this flexibility of having a session alive and go back to it when needed, but also push it to the background when I want, I used the screen package in Linux. This creates a separate session going beyond logout, so Paramiko can come and go as needed to initialize Mininet, then log in again to

add the controller, then log in again later and do the ovs show commands to check on everything.

```
# Launch mininet topology in a detached screen
cmd = "sudo screen -dmS mininet sudo mn --switch ovs,protocols=OpenFlow13"
paramiko_send_command(mininet_ip, mn_cfg['username'], mn_cfg['password'], cmd)
```

Above is the command that I used (snip is from my code).

c. Configure the OvS on the Mininet VM to connect to the controller. Paste relevant screenshots. [**20 points**]

Here is a screenshot that shows the output for my code. First the code initializes Mininet as shown above, then a function runs to check if s1 exists. The switch won't appear immediately because it takes a second to load, so my program will just check to see if it exists over and over until it appears. Once s1 is found, the code will do "ovs-vsctl show" and displays the output showing that the controller set to default (localhost), then it will set the controller, and do the same show command again to show that the new controller is set:

```
jimmy@jimmy:~/JDV/Semester5$ sudo ./sdn_midterm_2.py
Connecting to R1 (192.168.122.2) to find Mininet VM IP address
Found ARP entry 192.168.122.80 -> 0800.2784.3692
Using Mininet IP: 192.168.122.80
s1

s1 exists
e7a21c84-4464-4b53-9d84-7ac031b48c46
    Bridge s1
        Controller "ptcp:6654"
        Controller "tcp:127.0.0.1:6653"
        fail_mode: secure
        Port s1-eth2
            Interface s1-eth2
        Port s1
            Interface s1
                type: internal
        Port s1-eth1
            Interface s1-eth1
    ovs_version: "2.13.1"

e7a21c84-4464-4b53-9d84-7ac031b48c46
    Bridge s1
        Controller "tcp:10.20.30.2:6653"
        fail_mode: secure
        Port s1-eth2
            Interface s1-eth2
        Port s1
            Interface s1
                type: internal
        Port s1-eth1
            Interface s1-eth1
    ovs_version: "2.13.1"
```

d. SSH into the traditional routers R1, R2 and R3 to configure routing to establish the OpenFlow connectivity between the OvS and the controller. Paste relevant screenshots. [**20 points**]

Summarizing the functionality: what I did to accomplish this part was use a Netmiko function set to configure a Cisco IOS router, then used a list of dictionaries to feed information to the function. The list of dictionaries is on the next page, along with the output of my code. The dictionary for each router contained the name, IP address, and a list of commands (static routes) to achieve full connectivity in the topology.

```
    # Configure routers
    routers = [
        {'name': 'R1', "ip": "192.168.122.2", "config": ["ip route 10.20.30.0 255.255.255.0 192.168.200.2", "ip route 172.16.100.0 255.255.255.0
192.168.200.2"]},
        {'name': 'R2', "ip": "192.168.200.2", "config": ["ip route 10.20.30.0 255.255.255.0 172.16.100.1", "ip route 192.168.122.0 255.255.255.0
192.168.200.1"]},
        {'name': 'R4', "ip": "172.16.100.1", "config": ["ip route 192.168.122.0 255.255.255.0 172.16.100.2"]}
    ]
```

Here is the output of when the code is ran:

```
192.168.122.2 configured
Configured R1 at 192.168.122.2
192.168.200.2 configured
Configured R2 at 192.168.200.2
172.16.100.1 configured
Configured R4 at 172.16.100.1
```

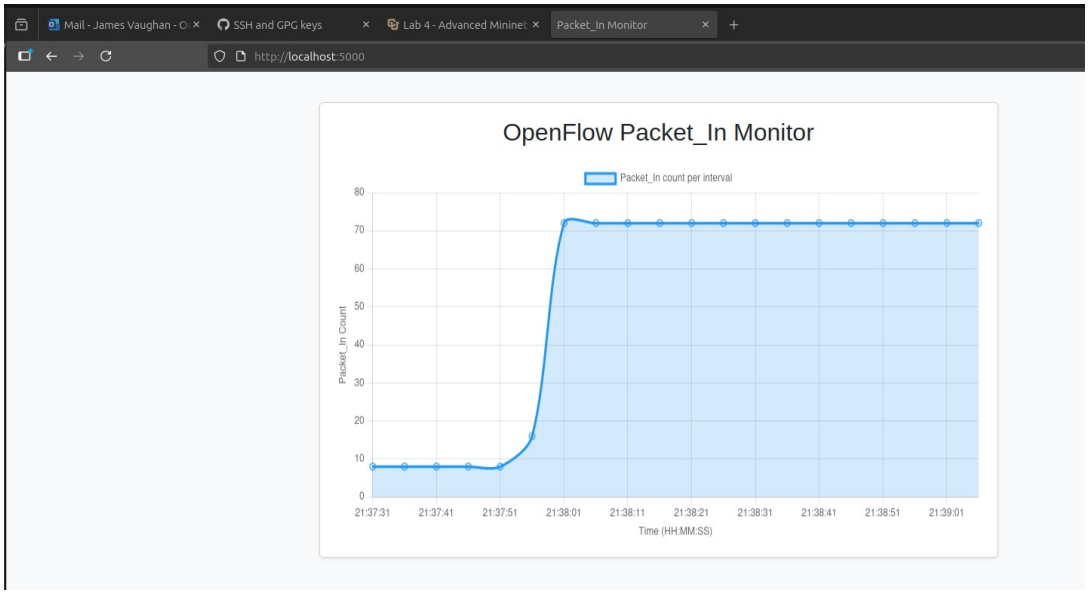
e. Verify and display that the successful OpenFlow connectivity between the OvS and the controller. Paste relevant screenshots. [**10 points**]

Here is a screenshot of the output from my code that shows the output of ‘ovs-vsctl show’. The output shows that is_connected is true and the controller is 10.20.30.2:6653:

```
Verifying OpenFlow connectivity...
e7a21c84-4464-4b53-9d84-7ac031b48c46
    Bridge s1
        Controller "tcp:10.20.30.2:6653"
            is_connected: true
        fail_mode: secure
        Port s1-eth1
            Interface s1-eth1
        Port s1-eth2
            Interface s1-eth2
        Port s1
            Interface s1
                type: internal
    ovs_version: "2.13.1"

tcp:10.20.30.2:6653
```

f. Capture the number of OpenFlow Packet_In messages sent from the switch to the controller and visualize this through an interactive graph on a webpage. You can use your favorite Python web-framework like Flask, Django, etc. to set up the webpage. The graphs should be displayed in real-time i.e. they should get refreshed automatically after a periodic interval of time (say 5 seconds) without manually reloading the webpage. Paste relevant screenshots. [**30 points**]

Here is a screenshot of the graph that I made:



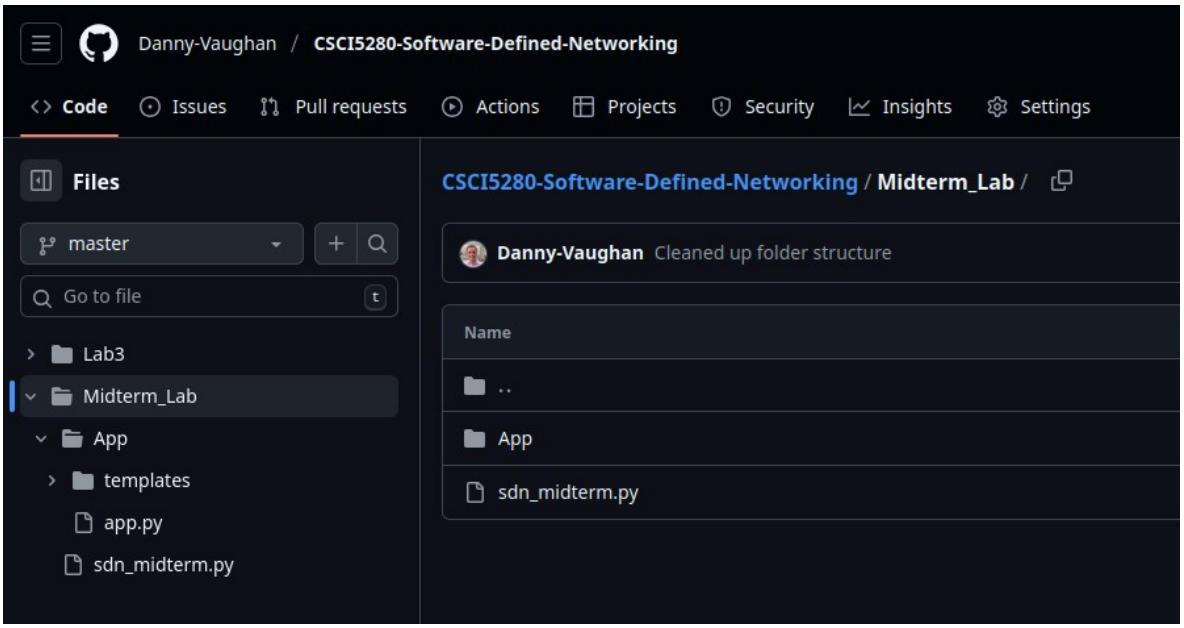
Instead of working with a packet capture and parsing the sniffed packets, I just pulled packets received from the ONOS API. I then ran the app and generated some traffic to see it refresh. I will include a video as well of the data.

The way I had it refresh every 5 seconds was in my index.html file. I used Javascript in the html file to fetch the data periodically and update the page:

```
    // Refresh every 5 seconds
    setInterval(fetchData, 5000);
    fetchData();
</script>
```

g. At the end, push your Python script to a new private repo 'SDN-Midterm' in your GitHub account. Paste relevant screenshots. [**5 points**]

Here is a screenshot of my GitHub repo with this midterm lab in there:



Total Score = _____ / 140