Network Softwarization: Technologies and Enablers (Winter 2019)

Module 2.2: Programmable Networks

In this lab, you will learn how to programmatically deploy your own custom network topology as an overlay network on top of the SAVI testbed. Similar to Mininet, you will be able to specify the number of end-hosts and switches in your network, as well as how they are connected together to create dynamic topologies. This is the first lab of a two-part series, so your work in this lab will lay a critical foundation for the next lab.

This lab will use both the command line interface (CLI) tools for interacting with the SAVI testbed. For Windows users, a few tools (free to download) that may be helpful for interfacing with Linux-based virtual machines are listed here as follows:

- PuTTY: Free terminal emulator for Windows, for logging into Unix-based systems;
  - Tutorial: http://www.electrictoolbox.com/article/applications/ssh-putty/
- WinSCP: Free Windows SCP client for transferring files to/from Unix-based systems.
  - Tutorial: https://sysmincomputing.wordpress.com/2011/01/22/winscp/

If you're using a Mac or Linux based operating system, similar tools should already exist and ready for you to use from your Terminal interface. If anyone needs help in using these tools, please feel free to post a message in the discussion board on Piazza.

Also, a helpful hint for anyone unfamiliar with Unix-based terminals: most terminals have a history buffer, which stores a list of previous commands. You can scroll through this history via the up and down arrow keys on your keyboard.

IMPORTANT: Since you are sharing the same project/tenant with others in the class, please **do not** alter or delete any virtual machines that does not belong to you. Also, whenever you are finished using a virtual machine, please terminate the instance and release any floating IPs you may have allocated. If you are temporarily suspending your work, please disassociate and release the floating IP so it can be re-used by others in the meantime.

Prerequisite: Accessing the SAVI Command-Line Client Tools

You have all been assigned a guest account on SAVI. To find your SAVI credentials, please access the following link: https://tinyurl.com/ece1508-2019s

A few scripts have been made and are provided to you in order to simplify some of the tasks in this lab. There is a gateway server set up with all the CLI client tools needed to access SAVI, and all the scripts already installed. You can log into the server via SSH:

- *ssh {username}@client1.savitestbed.ca*
    - The *username* is your SAVI username, and the default password is the same as your username (or whatever you changed it to).

After you have logged in, you must load the proper credentials into your environment variables. A script has been prepared for you in your home directory called *savi*. The usage of the script is as follows:

- *source savi {username} {project} {node}*
    - The *username* is your SAVI username
    - The *project* is the project/tenant that was assigned to you
    - Specify the *node* depending on the region assigned to you in the credentials sheet

You are now ready to use the SAVI testbed. To test, please type *nova image-list* to see if you can view the list of available VM images.

Part 1: Basic Overlay Network Orchestrator

You will complete a script, named *saviOverlay*, which will allow you to dynamically create SDN-enabled overlay networks of your choice on any cloud infrastructure that supports OpenStack APIs. Overlays are a way of realizing logical topologies that exist on top of an underlying network topology[1]. You can execute the script (written in Python) from within any Linux environment, but for this lab you will be using the SAVI client server. In this part of the lab, the end-goal is to be able to boot up a simple overlay network and test its connectivity via ICMP pings.

The topology you will create is shown in Figure 1 below. Altogether, this overlay requires 4 VMs to be created. By default, the switches (Open vSwitch or OVS) function as simple L2 Ethernet learning switches, so connectivity should be plug-and-play.

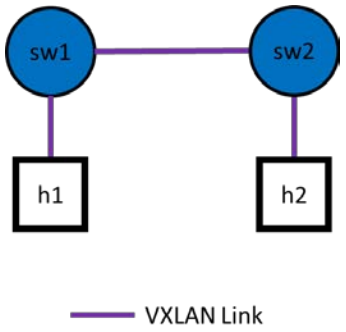

Figure 1: Overlay Topology for Part 1

The overlay connections are created using VXLAN tunnels. VXLAN is a protocol that enables the virtualization of networks by supporting packet encapsulations of Layer 2 frames within Layer 4 UDP packets. This can be seen in Figure 2 below.

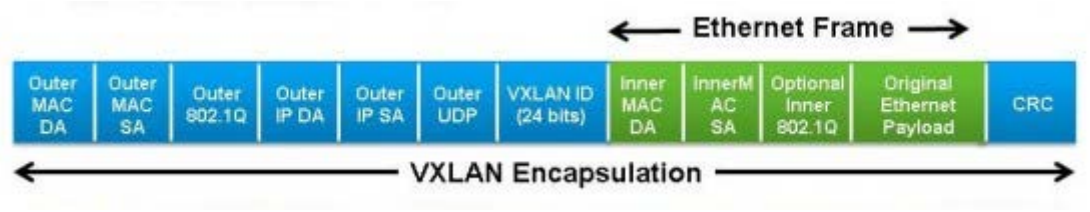

Figure 2: VXLAN Encapsulation[2]

[1] See this figure for example: http://www.ktword.co.kr/img_data/3481.gif
[2] From Intel's blog about VXLAN: https://software.intel.com/en-us/blogs/2015/01/29/optimizing-the-virtual-networks-with-vxlan-overlay-offloading

To help you get started by setting up the necessary SSH key-pairs, security group rules, and download the required files for this lab, you can execute the initialization script:

- *netsoft-overlay-init*

You should see a new *programmable_networks_lab* directory within your home directory, and within that, there should exist the *saviOverlay* script, as well as a file named *overlay_config.py*. Note that this filename is important for the script to work (i.e. the script searches for a file named *overlay_config.py*). If you change the name of this file, you will have to change the script accordingly. You may want to open the files and observe the contents inside in order to learn how they work together. You should also double-check the security group, which was created with your username, to ensure it has rues for opening TCP ports 22, and 6633, UDP port 4789, and enabling all ICMP.

Tasks to Complete

1. Go into the *programmable_networks_lab* directory, if you haven't already, and try executing the *saviOverlay* script. You should see a help section printed out.
    - Three functional options are available as a second argument for the script. They are briefly described here below:
        - **deploy**: Creates the overlay. This involves the full operation of creating the VMs, and SSH'ing into them to configure OVS to set up the overlay connections;
        - **list**: Lists the VMs belonging to your overlay; and
        - **cleanup**: Deletes the VMs belonging to your overlay.
    - Right now, these options won't do anything except print out a simple message.
2. The initialization script you ran earlier has already configured the *overlay_config.py* file for you. Open it and double-check the settings.
    - The default topology creates the 4-VM (2 host and 2 switches) topology as shown in class. Keep this for now, and when your script fully works, you can test with other topologies.
    - **NOTE**: There is a field called *contr_addr* which will be used in the second part of this lab. You can ignore this field for now.

3. Open up the *saviOverlay* script in whichever editor you prefer, and study the code.
    - This script is a skeleton, meaning it has the basic structure already set up. It is your job to understand the code written thus far, and complete it.
    - **NOTE**: To open the script, you can use either command-line editors like *vim* or *nano*, or copy the file to/from your computer and edit it locally. If you choose the file copying method, keep in mind that you need to copy twice (i.e. once from the VM to *client1*, then from client1 to your computer).
4. There are 6 functions within the script which you must complete, some of them may already include some code within them. The 6 functions are:
    - **bootVM()**: Creates a new virtual machine. The existing code within the function will pre-pend your username to any VM name that is passed into the function, this is how you can identify which VM belongs to you and your overlay;
    - **setOverlayInterface()**: SSH into a given VM and creates an internal interface on the OVS, and sets the overlay IP address on the interface;
    - **connectNodes()**: Establishes bi-directional VXLAN connectivity between the OVS switches within two VMs. This involves SSH'ing into the VMs and creating the VXLAN interfaces on the OVS;
    - **deployOverlay()**: Called directly from the script's main function. Reads the topology from the *overlay_config.py* file and calls the appropriate functions to create and configure the overlay network;
    - **listOverlay()**: Called directly from the script's main function. Lists the VMs belonging to the overlay network, by printing out the VM's names, UUIDs, and underlay IP addresses; and
    - **cleanupOverlay()**: Called directly from the script's main function. Finds the VMs belonging to the overlay network and deletes them.

    The first three functions are helper functions. The last three are functions directly invoked by the script's main, and can call the helper functions if need be.

    NOTE: It is highly recommended that you study the comments of these functions before starting, as there may be hints on how you should use them or complete them.

5. You will undoubtedly run the script numerous times during your testing process. Between each iteration, please clean-up your previous VMs to avoid needlessly taking up resources (the cloud is not an infinite pool of resources). To help in this process, it is recommended that you complete the *cleanupOverlay()* function as soon as possible.

   NOTE: You are sharing the tenant with other people in the course. Be careful not to delete other people's VMs!

   You can verify the deletion of your VMs by typing *nova list* and seeing if any VMs related to your overlay still exist.
6. Modify the topology to ensure that your script can handle arbitrary configurations. The new topology is entirely up to you. The only limitation is to ensure that you do not introduce loops into the topology.
7. At the end, clean up and delete your overlay network by simply running:
   - *./saviOverlay cleanup*

Part 2: SDN-Enabled Overlay Network

By default, the overlay works without having to install any flows into the switches due to the fact that the default behaviour of OVS is to function as a basic L2 Ethernet learning switch. In this part of the lab, you will connect the OVS within the switch nodes of your overlay topology to connect to a remote SDN controller, as depicted in Figure 3 below.

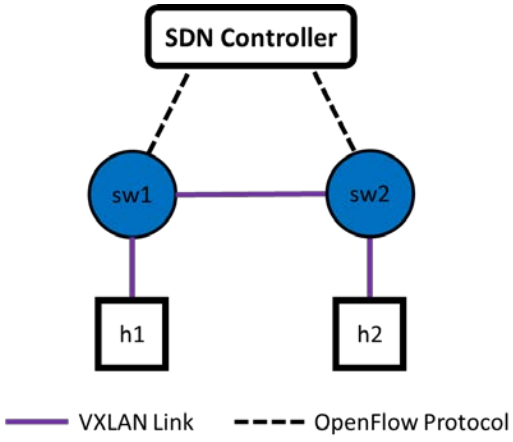

Figure 3: Overlay Topology Controlled by an SDN Controller

Tasks to Complete

1. To serve the OpenFlow controller of the overlay network we will create, you need to first boot up a VM based on the "**ece361**" image using the "**m1.small**" flavor. Use the key and security group that was created for you when you ran the init script.
    - If you don't remember the key and security groups that were created for you by the init script, you can find them by running:
        - *nova keypair-list*
        - *nova secgroup-list*
    - Refer back to the lecture slides if you're unsure how to create a VM.
    - The ece361 image comes with a light-weight Python-based OpenFlow controller, named Ryu, already installed. It also includes a Python library which you will use later on in the next lab.
2. Access the controller VM via SSH. The default username/password is: *ubuntu*/*ece361*
    - To update the controller code, run *ece361-update* once you've logged into the VM
3. To start up the Ryu OpenFlow controller, execute the following command:
    - *ece361-start-controller switch*

    When you run the command, it starts Ryu within a *screen* in the background. You can search online how to attach and detach to this screen. Ryu is launched using the settings written into the configuration file specified (you don't need to worry about this) and loads a L2 learning switch application defining the behaviour of connected switches.
4. The initialization script you ran earlier has already configured most of the settings in *overlay_conf.py* file for you. However, you need to modify it further to specify the IP address and port of the SDN controller.
    - The *contr_addr* variable should be modified and set. For example, if the VM created has an IP of 10.2.0.3 and the SDN controller is listening on port 6633 (which it should be), then set the variable to *"10.2.0.3:6633"*.
5. Open the *saviOverlay* script and note the pre-defined function called *setController()*. Modify your *deployOverlay()* code to call *setController()* such that the OVS' in the switch nodes of the topology are connected to the controller.
6. Re-deploy your overlay and test that it still has connectivity from h1 to h2.

7. At the end, clean up and delete your overlay network by simply running:
    - *./saviOverlay cleanup*

Post-Lab: Cleaning Your Activity

After you are done the lab, please delete any leftover VMs you have created (overlay and SDN controller), as well as any security groups you may have created.

What to Submit

Submit a report describing 1. what you have done; and 2. what you have learned. It should be clear such that someone not from this course can understand it. Feel free to include any comments on where you may have encountered difficulty during the lab. We will be examining not just the correctness and efficiency of the *saviOverlay* script, but also the clarity of the report. **Submit the report (as a .pdf file) as well as your completed *saviOverlay* script** in a compressed file (e.g. zip, rar, or tar) and post it on Piazza under hw5.

Appendix A: Nova Client Walkthrough

```
import novaclient.v1_1.client as novaClient

# Setting credentials
username = 'your_username'
password = 'your_password'
tenant_name = 'your_tenant_project'
reg = 'your_assigned_region'
auth_url = 'http://iam.savitestbed.ca:5000/v2.0/'

nova = novaClient.Client(username, password, tenant_name, auth_url,
                                    region_name=reg, no_cache=True)

# Equivalent to "nova list" in command-line
nova.servers.list()

# Equivalent to "nova flavor-list" in command-line
nova.flavors.list()

# Note how they return lists of objects

# Get an item (the first object in the list) and inspect it
a = nova.flavors.list()[0]

dir(a) # Alternatively, type "a." then press tab

# We see the object has a member variable called 'name'

# Let's use the find() function to search based on 'name'
flavor = nova.flavors.find(name = "m1.small")

# Equivalent to "nova image-list" in command-line
nova.images.list()

# Get an item (the first object in the list) and inspect it
a = nova.images.list()[0]

dir(a) # Alternatively, type "a." then press tab

# We see the object has a member variable called 'name'

# Let's use the find() function to search based on 'name'
image = nova.images.find(name = 'ECE1508-overlay')

# Nova Python client can do limited network commands too...
# Equivalent to "neutron net-list" in command-line
nova.networks.list()

# Get an item (the first object in the list) and inspect it
a = nova.networks.list()[0]

dir(a) # Alternatively, type "a." then press tab
```

```
# There's no member variable called name... :(
# But wait! We see the object has a member variable called 'label'
a.label # It's the name!

# Get our network object
# In SAVI, we named all our networks as just the tenant name + '-net'
net_name = tenant_name + '-net'

# Let's use the find() function to search based on 'label'
net = nova.networks.find(label = net_name)

# How to create a server?
nova.servers.create() # Will tell you it needs 4 arguments

# Let's inspect the function
import inspect
inspect.getargspec(nova.servers.create)

# NICs just specifies what network to connect the VM to
vm = nova.servers.create("myTestVM", image, flavor, key_name='tlin-client',
                         security_groups=['default'], nics=[{'net-id':
net.id}])

# See if it's active
vm.status

# Right now will have static object, won't be refreshed automatically
vm.get() # Re-sync state from Nova


# We need to do this programmatically... let's loop and wait
import time

while vm.status != 'ACTIVE':
    print "waiting for status active, sleeping..."
    time.sleep(3) # Sleep 3 seconds
    vm.get()

# to SSH into the server and run commands...
import paramiko

# More docs here: http://docs.paramiko.org/en/2.4/api/client.html
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

# Need to get the IP of the VM
ip = vm.networks.get(net_name)[0]

# Let's SSH!
ssh.connect(ip, username = 'ubuntu', password = 'savi')
```

```
# If the VM's SSH daemon isn't up yet, connect() will throw an exception
# We need to be able to catch this... and try again
while True:
    try:
        # The ECE1508-overlay image's default username and password
        ssh.connect(ip, username = 'ubuntu', password = 'savi')
        break
    except Exception as e:
        print e
        print "Unable to connect. Trying again...."
        time.sleep(3)

# Now we've established an SSH connection, we can run commands
stdin, stdout, stderr = ssh.exec_command("ifconfig")

# Get all the output at once
out = stdout.readlines()

# Kinda ugly, it's array of strings, one line per element
print out

# Let's joing them all into one single long string
out = ''.join(out)
print out


# Once you're done, delete the VM
vm.delete()
```