Network Softwarization: Technologies and Enablers (Winter 2019)

Module 2.3: Network Function Virtualization

In recent years, a growing number of software tools offer SDN and service chaining through high-level graphical interfaces that often hide the particular details of their implementation under-the-hood. This lab builds upon the skills and knowledge you have acquired from past modules, and is designed for you to gain greater insight to the low-level inner-workings of SDN and NFV service chaining in a realistic deployment setting. You will build on the previous lab and deploy your own network topology as an SDN-controlled overlay network on top of the SAVI testbed. You will then deploy a basic application on the overlay (in this case, WordPress), and use SDN principles to chain a firewall service to protect it.

This lab will use 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 file transfer tool for transferring files to/from Windows and Unix 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

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: Create a Private Overlay with a Custom Topology

The *saviOverlay* script which you completed in the previous lab allows you to dynamically create SDN-enabled overlay networks of your choice on any cloud infrastructure that supports OpenStack APIs. In this first part of the lab, you will boot up an overlay network and simply test its connectivity via ICMP pings.

The topology you will create is shown in Figure 1 below. Altogether, this overlay requires 6 VMs to be created, plus 1 more for the SDN controller. Note that it is possible to simply run the controller from within one of the 6 VMs, but we include it separately to demonstrate that the controller is architecturally not part of the overlay topology.

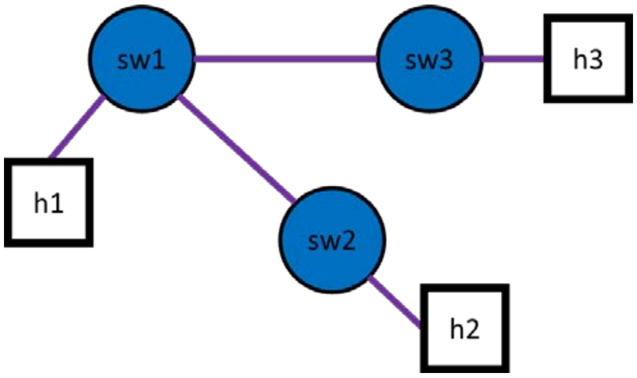

Figure 1: SDN Overlay Topology for Part 1

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-nfv-init*

Double-check the security group, which was created with your username, to ensure it has rues for opening TCP ports 22, 6633, and 80, UDP port 4789, and enabling all ICMP.

Tasks to Complete

1. To serve as 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 this 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:

    1. A L2 learning switch application defining the behaviour of connected switches;
    2. A topology discovery application for learning how connected switches are inter-connected (if at all), and learning the ingress ports of MAC addresses; and
    3. A number of RESTful services for querying the topology and installing flows
4. Update the *overlay_config.py* file from the previous lab:
    - Implement the topology shown above in Figure 1. You may pick the overlay addresses for the hosts, but they must be within the same CIDR or subnet. The subnet should also not overlap with the underlay network. It is recommended that you make a backup copy of the file before overwriting it with your new topology.
    - The *contr_addr* variable should be modified to use the IP address of the VM you just created in step 1.
5. To deploy the overlay, simply execute your *saviOverlay* script:
    - *./saviOverlay deploy*

    In a separate window, you can type ‘*nova list*’ to check the status of the VMs as they build.

6. SSH into *h1*, and verify the connectivity of the overlay by pinging the other hosts as specified in the *overlay_config.py* file.
    - **Troubleshooting:** If there is no connectivity when using the overlay addresses, check to see if the OVS switches in the switch VMs are connected to the controller. Also double-check to see if the Ryu controller is running in the controller VM.

In your report, show the results of the ping tests from *h1* to *h2* and to *h3*.

<u>Part 2: Traffic through the Overlay</u>

You may have noticed in the previous part that the hosts serving as hosts still access the internet (and actually, any network other than the overlay IP range) through their original IP via eth0. The overlay network is only used if an application running in the VM specifies other hosts via their overlay IP. There may be circumstances in which you want to force the VM's default traffic to go through your overlay network, perhaps so that you can control and monitor its behaviour via SDN principles, or perhaps to pass the traffic through custom middle-box software designed to modify the packets. In this part of the lab, you will set host *h1* to act as a gateway router performing NAT, and set *h2* and *h3* to use *h1*'s overlay address as its default gateway.

<u>Tasks to Complete</u>

1. Configure the VM serving as *h1* as a router to perform either **IP masquerading** or **source NAT**, treating the overlay network as the "private network". You will need to make the following changes in `iptables`, the Linux network filter.
    - Change the default policy of the `iptables` `FORWARD` chain to accept all packets
    - Enable either masquerading or SNAT
2. Now you need to set *h2* and *h3* to use *h1*'s overlay address as the **default gateway**.
    - **NOTE**: When accessing *h2* and *h3* to make your modifications, you should first SSH into *h1* then SSH into *h2* or *h3* via their overlay address. If you SSH directly to them via their underlay network IP and change the default route, your connection will be cut off.
        - Recall that the default username/password for the **ECE1508-overlay** image (which your overlay nodes should be using) is *ubuntu/savi*

- First add the new default gateway
  - *sudo route add default gw {new gateway_ip}*
- Then delete the old gateway to avoid potential conflicts
  - *sudo route del default gw {old gateway_ip}*

Verify all the traffic to and from the internet from either *h2* or *h3* go through the VM serving *h1*. The best way to prove this is to use the Linux `tcpdump` utility to listen to the *eth0* interface in *h1* (e.g. *tcpdump –n –i eth0 icmp*) and observe the packets from the other hosts while they're pinging some public address (e.g. Google).

In your report, include screenshots clearly showing the correctness of your implementation: a routing table from either *h2* or *h3*, and the relevant `iptables` entry for turning *h1* into a NAT router.

Part 3: Deploying the WordPress and Firewall

In this part of the lab, you will use the private overlay to deploy an application. In this instance, that application will be a basic WordPress service deployed in *h2*. In addition, you will deploy a firewall service in *h3* using *Snort*, an open-source network intrusion-detection system, to protect the WordPress service. You will use container technology to simplify the deployment of both the WordPress and firewall services. We use containers as they pre-package all the required software and dependencies (e.g. libraries) into a minimalistic isolated environment within the OS.

Tasks to Complete

1. Deploy the WordPress service in *h2* using Docker containers. We choose Docker as it allows users to upload custom container images to the public cloud, which enables us to easily re-use existing images. The image used to boot up the VMs for the overlay came with Docker pre-installed. Deploying the WordPress service requires two components, 1) An HTTP server with WordPress; and 2) A MySQL database server for persistent storage.
   - SSH to *h2* (via *h1*) and run the following commands:
     - *docker run --name wp-mysql -e MYSQL_ROOT_PASSWORD=dbpass -d -p 3306:3306 mysql:5.7*

        - *docker run --name wp-serv --link wp-mysql:mysql -p 80:80 -d wordpress*

    The *docker run* command automatically downloads the container images for you and deploys them. For more information about what the command-line flags are doing, see: https://docs.docker.com/engine/reference/commandline/run/#options

2. Configure *h1* to act as a reverse-proxy for the WordPress service (i.e. by accessing *h1* at TCP port 80, the traffic will be redirected to the WordPress service). Using a reverse-proxy (e.g. nginx, HAProxy, etc.) is a common deployment strategy for services that may require load balancing multiple backend replicas.
    - For this lab, since we are only deploying a single WordPress server, a simple `iptables` rule to re-route the destination IP should suffice:
        - *sudo iptables -t nat -A PREROUTING -d {h1 underlay IP}/32 -p tcp -m tcp --dport 80 -j DNAT --to-destination {h2 overlay ip}:80*
    - To test your setup, you will need to be able to access *h1*'s underlay address directly from your local computer's browser. One way to do this is using SSH port forwarding to forward localhost port 80 to *h1*'s port 80:
        - On Mac or Linux-based systems, open up a new terminal and SSH'ing to client1 with the following flags:
            - *ssh {username}@client1.savitestbed.ca –L 80:{h1's underlay IP}:80*
        - On Windows using PuTTY, first fill in the username and address for client1 like you normally do. Before opening the connection, go to *Connection* → *SSH* → *Tunnels* and:
            - Input **80** for the *Source port* field
            - Input ***{h1's underlay IP}*:80** for the *Destination* field
            - Then click *Add*, and you can now *Open* the connection
        - Once this is done, you can test the port forwarding by using `tcpdump` on *eth0* of *h1*, listening on port 80 (e.g. *tcpdump –n –i eth0 port 80*) and directing your local browser to: **http://localhost**
    - If your port-forwarding setup was correct, and your WordPress deployment was correct, you should now be able to view your WordPress setup page.

    - Go ahead and enter the required fields to finish setting up your WordPress
- Once WordPress is set up, change the URL to: **http://localhost/?inject**
    - You should observe that you can still view the same home page. This is because the firewall service has not yet been deployed and chained through.

3. You will now deploy *Snort* into *h3*. Since we intend to use *Snort* as a stand-alone firewall, it will operate in *in-line* mode. This mode requires two interfaces, one for listening for packets on (*Snort* ingress interface), and the other for forwarding packets to (*Snort* egress interface). You will add these interfaces to the same Open vSwitch within *h3* that was created by your *saviOverlay* script. Specifically, you will create and add *internal* interfaces, which are interfaces connected to the bridge and exposed internally within the OS.
    - Use the following command to create an internal interface:
        - *sudo ovs-vsctl add-port {switch name} {interface name} -- set interface {interface name} type=internal*

      The switch name should be that of the switch created by your *saviOverlay* script, and the interface name is up to you. You will need to create two of these.
    - Verify the interfaces have been created via:
        - *sudo ovs-ofctl show {switch name}*
        - **NOTE**: Ignore the MAC addresses you see in the output of *ovs-ofctl show*, they are not the same as the MAC addresses for the internal interfaces.

      It is recommended you figure out which port number on the switch is associated with these interfaces, as you will need this information later on.
    - Bring up both of the interfaces:
        - *sudo ifconfig {interface name} up*
    - You will need to manually install five flow rules into the OVS in *h3* using the *ovs-ofctl* command. These rules are given to you with placeholders for you to fill:
        1. *in_port={port num of vxlan iface},priority={something high},dl_dst={mac of internal iface created by saviOverlay},actions=output:{port num of internal iface created by saviOverlay}*

2. *in_port={port num of vxlan iface},priority={something high},dl_dst=01:00:00:00:00:00/01:00:00:00:00:00,actions=output:{ port num of internal iface created by saviOverlay }*
3. *in_port={port num of internal iface created by saviOverlay},actions=output:{port num of vxlan iface}*
4. *in_port={port num of vxlan iface},priority={something lower than before},actions=output:{port num of snort ingress iface}*
5. *in_port={port num of snort egress iface},priority={something lower than before},actions=output:{port num of vxlan iface}*

In your report, explain what each of the above rules do and why they are needed. Be specific for each rule. It may help to accompany the explanation with a figure. In addition, include a dump of the OVS flow-table after you have installed the rules, as well as the output of *sudo ovs-ofctl show {bridge name}*.

4. Deploy the *Snort* intrusion-detection system using a Docker container. This can be achieved as follows:
   - *docker run -it --net=host --name wp-firewall linton/docker-snort /bin/bash*
   - The above command will launch a container with *Snort* installed, and automatically drop you into a bash shell. You are now *within* the container.
     - **NOTE:** Avoid typing **exit** as that will stop the container
     - To detach from container without stopping it, use **Ctrl+p+q**
     - To re-attach to the container, type: *docker attach wp-firewall*
   - Create a configuration file defining a rule for *Snort* to use. From within the container, create a new file called *snort.conf*. Enter the following line into the file:
     - *reject tcp any any -> any 80 (content:"inject"; nocase; msg:"accessed forbidden pages!!"; sid:5000000;)*

   In your report, explain what each component of the rule shown above are for, as well as the syntax for defining the rule.

   - From within the container, start *Snort* using the newly created configuration file:

        - *snort -Q --daq afpacket -i {snort ingress interface name}:{snort egress interface name} -c snort.conf -A console*

    The above command will start *Snort* in the foreground, attached to the screen. You can detach from the container to leave it running, and re-attach to the container at any time to stop/restart *snort* or adjust the rule file.

<u>Part 4: Chain HTTP Traffic through Firewall</u>

In this part of the lab, you will use the OpenFlow controller to install flows that will redirect all HTTP traffic going from *h1* to *h2* through the firewall service located at *h3*. While the Ryu controller that you started has RESTful APIs that you can work with, you can instead leverage the `ryu_ofctl` Python library that abstracts the APIs for you. You can use this library to modify the flow tables of switches in the network, as well as query basic information regarding the topology of the network. This library is pre-installed in the same VM as the controller.

<u>Tasks to Complete</u>

1. Read through the sample workflow for the `ryu_ofctl` library documented here:
    - https://github.com/t-lin/ryu_ofctl/blob/master/README.md
2. SSH into the VM you created serving as the controller of your overlay network. From here, you can simply start a basic Python shell (i.e. run: *python*) and start using the library
3. Collect the overlay MAC addresses of *h1*, *h2*, and *h3* (i.e. from the internal interfaces created by your *saviOverlay* script)
    - Using the topology APIs, you can discover the DPIDs of the switches that are directly connected to the hosts
    - **Troubleshooting**: If you find that the APIs do not return any information regarding this, try 1) restarting the Ryu controller; and 2) Make *h1* ping *h2* and *h3*
        - To restart the Ryu controller, simply call: *ece361-stop-controller* followed by *ece361-start-controller switch*
        - You need to make *h1* ping *h2* and *h3* so that the restarted controller will learn of the MAC to port associations

4. Using the collected information regarding the DPIDs and the overlay MACs, install the appropriate flows that redirects HTTP traffic (and only HTTP traffic) from *h1* destined for *h2* to go through *h3*.
   - Each time you install a flow into a switch, it is recommended that you SSH into the VM (in a separate terminal) serving as the switch and validate your actions by running: *sudo ovs-vsctl dump-flows {switch name}*
   - You can observe the counters of the flows to see which ones are triggered.

   In your report, include a dump of the flow-tables from the switches you modified. In addition, provide all the Python commands you ran to install the flows **in an appendix** section of the report.
5. Test to see if the firewall is doing its job.
   - If the flows were correctly installed and *Snort* is working, and you previously successfully set up WordPress, you should be able to still view your WordPress home page by open your browser and going to: **http://localhost**
   - Now change the URL and add a */?inject* at the end
     - e.g. **http://localhost/?inject**

     The page should now be unable to load, as *Snort* should be dropping the GET requests containing the "*inject*" string.
   - Change the URL in your browser back (without the */?inject*) and refresh it, you should now be able to load the page again.

   In your report, include screenshots of your browser before and after adding */?inject* to show the working firewall implementation.

Post-Lab: Cleaning Your Activity

After you are done the lab, please delete any leftover VMs you have created (overlay and SDN controller). The quickest way to delete the VMs participating in your overlay is by running:

- *./saviOverlay cleanup*

You will need to manually delete the VM serving as the OpenFlow 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. Remember to include any screenshots and other items asked of you in the above sections. Feel free to include any comments regarding where you may have encountered difficulty during the lab. **Submit the report (as a .pdf file)** in a compressed file (e.g. zip, rar, or tar) and post it on Piazza under hw6.