CS 6480: Advanced Computer Networks - Fall 2020

Lab Assignment 1:
Cloud Computing and Network Function Virtualization

August 31, 2020

1 Overview

There are three goals associated with this assignment. First is to introduce students to the POWDER testbed infrastructure [9, 7]. The second is to explore aspects of cloud computing by experimenting with the Docker platform as a service framework [1]. The third is to use the Docker environment together with the Quagga open source routing suite [6] to explore basic network functions virtualization (NFV) concepts [2, 3]. A side effect of the third goal will be a review and practical application of basic routing concepts.

2 Assignment Details

2.1 Overview

Cloud computing has radically changed the way data centers and their associated resources (compute, storage and network) are managed and used. The same underlying concepts and technologies are now changing the way networks are realized and managed through a concept called network functions virtualization (NFV). In short, NFV allows network operators to move away from the conventional model where network elements were realized as hardware appliances with fixed functionality, to an environment where network elements can be dynamically realized through software instantiated on (possibly) commodity hardware. Such an approach offers great flexibility and the promise of making it easier for network operators to introduce new services and service features and to evolve their networks based on network conditions and user requirements.

There are, however, a number of challenges associated with realizing the NFV vision. First is how to realize virtualized network elements with performant data forwarding capabilities [11, 8]. Another open question is how network management practices will need to evolve in this new environment. (In essence NFV makes the already difficult task of network management even more complex because in an NFV-enabled network things will be more dynamic than in a conventional network.) Finally, despite ongoing efforts in this space [5, 4], exactly how NFV-enabled networks will be realized, orchestrated and managed remains an open question.

Our goal with this assignment is not to solve all (or indeed any) of these problems, but rather to gain a hands-on understanding of the potential and challenges associated with NFV. For the final part of this assignment you will be required to realize a simple NFV-enabled routed network and to develop a basic OSPF NFV "orchestrator"' to perform dynamic, zero-impact, network modifications.

The essence of what is required is depicted in Figure 1. Specifically you will be required to:

- Dynamically instantiate a simple three node router topology between two end hosts HostA and HostB. (Figure 1(a).) Both the hosts and the routers will be Docker containers in a Linux environment. You will use Docker configuration files and commandline functions to create this initial topology dynamically (including IP subnet assignments suitable for a routed network topology).

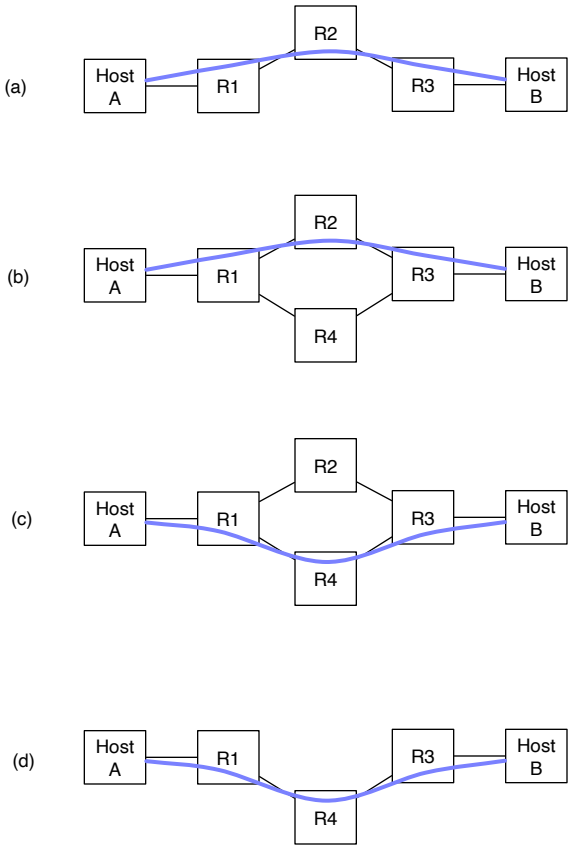

Figure 1: Zero impact network modification

- The routers in this topology (R1, R2 and R3) will run the Quagga routing software. You will have to create configuration files for these routers to enable them to run OSPF. At this stage the two hosts should be able to communicate via the R1-R2-R3 path as shown in Figure 1(a).
- You will then have to introduce another router into the topology, as shown in Figure 1(b), via Docker commands and configuration files. You will have to programmatically configure R4 to take part in OSPF with the other routers.
- Our eventual goal is to remove R2 from the topology. However, we want to do this without causing **any** packet drops. With OSPF this can be achieved by setting the OSPF links weights such that router R1 will prefer the path via R4 towards HostB, and similarly R3 will prefer the path via R4 towards HostA. You will have to programmatically set the link weights between R1 and R2 and between R2 and R3 to realize this. The desired change in traffic flow is depicted in Figure 1(c).
- Finally you will have to programmatically remove R2 from the topology to arrive at the topology shown in Figure 1(d).

2.2 Approach

This is a non-trivial assignment which will acquaint you with a variety of concepts, several of which might be new. To ensure you follow a staged approach the assignment will be broken into three parts. You will have to demonstrate (and will be graded on) your progress for each part:

1. Basic router functionality with simple topology:
   - Gain access to POWDER platform and start up an experiment with a single Linux host.
   - Install Docker tools.
   - Use Docker to create a simple three node topology: HostA $<->$ R1 $<->$ HostB. (Including appropriate prefix/address assignment and ensuring R_1 is able to forward IP traffic between the two hosts.)
   - Demo: Show your Docker setup and demonstrate being able to successfully *ping* between the two hosts.
2. Full topology with OSPF routed functionality:
   - Use Docker to create the complete topology shown in Figures 1(b/c).
   - Install and configure OSPF in this topology to enable routing between the tow hosts.
   - Demo: Show your Docker setup, show that your OSPF configuration is working correctly and demonstrate being able to successfully *ping* between the two hosts.
3. Fully orchestrated realization of assignment:
   - Develop the orchestrator to automate the steps associated with the assignment described above.
   - Demo: Show your Docker setup and demonstrate the use of your orchestrator to move traffic from the R1 $<->$ R2 $<->$ R3 path to the R1 $<->$ R4 $<->$ R3 path without incurring any packet loss.

3 Assignment details

3.1 Part 1: Basic router functionality with simple topology

- Request access to POWDER[1]:
  - Go to: https://powderwireless.net
  - Click on "Sign Up".
  - Fill out the form on the left. **Do create an ssh key and upload the public part.** It simplifies things later on.
  - On the right, select "Join Existing Project". **Use "cs6480-2020" for the project name.**
- Note: POWDER is built on and extends the Emulab environment. Which means that a basic working knowledge of Emulab is useful to make use of the platform:
  - Scan the first three sections of the original Emulab paper [10].
  - Scan Sections 1 through 5 of the Emulab manual: `http://docs.emulab.net`

[1]POWDER is a platform enabling mobile and wireless research. For this assignment we will not make use of any of mobile/wireless features. Rather we will only use compute nodes which might reside in the Emulab or CloudLab platforms.

- **Important:** Note that any changes you make on the nodes allocated to you is considered ephemeral. I.e., these changes will be lost when your experiment is terminated/swapped out! And your experiment by default terminate in approximately 16 hours. (Note: Once your experiment is instantiated, you can use the POWDER portal to request an extension.) **You should maintain notes, configuration files, scripts etc., that you create on a separate persistent storage facility. E.g., a git repo.**

- Once you have access to the POWDER platform you can use this profile to instantiate a single physical compute node running Ubuntu Linux:

  `https://www.powderwireless.net/p/cs6480-2020/single-node`

- Once your single node has been instantiated you can ssh into the node from your laptop/work station to access it remotely.

- Install Docker tools (Docker Engine and Docker Compose) on your node by following appropriate instructions from here:

  `https://docs.docker.com/engine/install/ubuntu/`

  and

  `https://docs.docker.com/compose/install/`

- You will now have to read appropriate Docker documentation in order to set up the topology needed for creating the functionality needed for this paty of the assignment. (**This is the most difficult piece of this part of the assignment and will likely require a fair amount of exploration and experimentation.**)

  Some hints:

  - Each node in the topology (HostA $<->$ R1 $<->$ HostB) will be a Linux Ubuntu container.
  - You will need to create a Dockerfile to tell Docker how you want each container to be set up.
  - You will need to create a docker-compose.yaml file to tell docker-compose the topology you want, the network configuration you want and how each container connects to the networks in question etc.
  - In this single router topology, the router will be able to route between its directly connected subnets. You will however, need to set up routes on the two hosts to point them towards the routed network. E.g., suppose the subnet between HostA and R1 is 10.0.14.0/24, the subnet between R1 and HostB is 10.0.15.0/24, on the 10.0.14.0/24 subnet HostA uses the IP address 10.0.14.3 and R1 uses 10.0.14.4. Under these conditions, the following command will instruct HostA to attempt to reach 10.0.15.0/24 via the appropriate R1 interface:

    ```
    route add -net 10.0.15.0/24 gw 10.0.14.4
    ```

    Note that you will need a reciprocal configuration on HostB for the return direction.

3.2 Part 2: Full topology with OSPF routed functionality

For this part of the assignment you will set up the complete topology shown in Figures 1(b/c) as Linux Ubuntu containers and networks using Docker and then use the Quagga Routing suite to enable OSPF routing functionality in that topology.

- Changing the topology should be a fairly straight forward extension to the docker-compose.yaml you created for the earlier part of the assignment.

- Next you will need to install Quagga on the routers in your topology. (Simplest way to do this is to update your Dockerfile(s) to install Quagga with the Linux apt-get utility.)
- Useful resources to learn about the Quagga routing suite:
  - `https://www.quagga.net`
  - `https://ixnfo.com/en/installing-quagga-on-ubuntu-server-18.html`
- Next you will need to make the appropriate Quagga configurations to make OSPF work in your topology. (**This is the most difficult piece of this part of the assignment and will likely require a fair amount of exploration and experimentation.**)

  Some hints:

  - You will need to have both the Zebra daemon and the OSPF daemon operational.
  - As described for the earlier part of the assignment, you will need to set up routes to point the hosts in your topology to the routed network.
  - You will need to configure each of the routers that will be part of your OSPF routed network. E.g., for a router connected to networks 10.0.11/24 and 10.0.12/24 your OSPF configuration will look like this:

    ```
    !
    router ospf
     network 10.0.11.0/24 area 0.0.0.0
     network 10.0.12.0/24 area 0.0.0.0
    !
    ```

  - To get OSPF working you will not need to assign any OSPF weight to links in the routed network. To realize "traffic moving" goal of the assignment, however, will require you to change OSPF weight in a sequenced manner so that the appropriate path through the network is preferred.

    An appropriate configuration snippet to set OSPF link/interface weights might look like this:

    ```
    interface eth0
     ip ospf cost 10
    !
    interface eth1
     ip ospf cost 10
    !
    ```

3.3 Part 3: Fully orchestrated realization of assignment

Given the understanding you have developed you should now finalize the assignment according to the description in Section 2.

- **OSPF NFV orchestrator.** You can package your orchestrator code in any way you please, but it should have a single executable script (e.g., bash or python) that drives all the orchestrator functions. Running your script with a "-h" commandline option should show "help" information for your orchestrator and its options.

  Your orchestrator should have options to realize the following functions:

  1. Construct the initial topology shown Figure 1(a) (including appropriate address allocations etc.).
  2. Start up OSPF daemons, with appropriate configurations, in the routers in the routed topology.
  3. Install routes on each host/endpoint connected to your routed topology.

4. Be able to add R4 to the routed topology and start OSPF up on it. (Figure 1(b))
5. Ability to move traffic between the "north" path (R1, R2, R3), and the "south" path (R1, R4, R3). (Figure 1(c)). Or, vice versa.
6. Remove router R2 from the topology (Figure 1(d)).

The generality of your orchestrator design will be considered as part of the evaluation.

Your orchestrator script should execute on the physical machine on which the containers are executing.

- **Submit your OSPF NFV orchestrator code via Cade.**
- **Evaluating your orchestrator.**

  For evaluation you will have to demonstrate the above mentioned functionality of your orchestrator to the instructor. I.e., you will have to execute the orchestrator with the appropriate option and parameters and then invoke the necessary other Linux and/or Docker commands to show that the intended result was achieved.

4 Grading and evaluation

4.1 What to hand in

Completed Assignment Your completed assignment should consist of a single executable script which implements your orchestrator. (You will likely have other files and scripts that you need in order to complete the assignment. You do not have to submit these files, only your orchestrator script needs to be submitted.)

You should submit your orchestrator script via handin on Cade.

You should use the following naming convention for your orchestrator script file:

`Firstname_Lastname_UID e.g.,`
`Joe_Doe_u0000000`

Turning in assignments using handin on Cade Your submission file must be submitted on a CADE machine using the handin command. To electronically submit files while logged in to a CADE machine, use:

`% handin cs6480 assignment_name name_of_orchestrator_script`

where `cs6480` is the name of the class account and `assignment_name` (la1, la2, or la3) is the name of the appropriate subdirectory in the handin directory. **Use la1 for this assignment.**

4.2 Grading

Criteria	Points
Part 1 demo	25
Part 2 demo	25
Part 3 demo	40
Code structure, generality and inline documentation	10
Total	100

References

[1] Docker. `https://www.docker.com`.

[2] Network Functions Virtualisation. `https://portal.etsi.org/NFV/NFV_White_Paper.pdf`.

[3] Network Functions Virtualization (NFV); Architectural Framework. `http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf`.

[4] Onos: Open Network Operating System. http://onosproject.org.

[5] Opnfv. https://www.opnfv.org.

[6] Quagga Routing Suite. http://www.nongnu.org/quagga/.

[7] Breen, J., Eide, E., Lewis, E., Reading, D., Buffmire, A., Hibler, M., Maas, D., Ricci, R., Duerig, J., Johnson, D., Orange, A., Schurig, D., Dutt, K., Kasera, S. K., Patwari, N., Stoller, L. B., Van der Merwe, J., Webb, K., and Wong, G. Powder: Platform for Open Wireless Data-driven Experimental Research. In *ACM WiNTECH proceedings* (September 2020).

[8] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., and Huici, F. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2014), NSDI'14, USENIX Association, pp. 459–473.

[9] The POWDER Team. Powder (the Platform for Open Wireless Data-driven Experimental Research). https://www.powderwireless.net, 2018.

[10] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002), USENIX Association, pp. 255–270.

[11] Wood, T., Ramakrishnan, K., Hwang, J., Liu, G., and Zhang, W. Toward a software-based network: integrating software defined networking and network function virtualization. *Network, IEEE 29*, 3 (May 2015), 36–41.