1 Overview

The primary goal of this assignment is to develop a simple network management orchestrator that can perform basic traffic management tasks in a small network of OSPF capable routers.

In order to realize that we will: (i) Create a small network topology by using the Docker framework (https://www.docker.com) to create Linux containers to act as routers, (ii) Use the FRRouting Project (https://frrouting.org) to provide OSPF routing in the network and (iii) Create a python orchestrator to manipulate OSPF weights to seamlessly "move" traffic from one network path to another.

2 Assignment Description

2.1 Overview

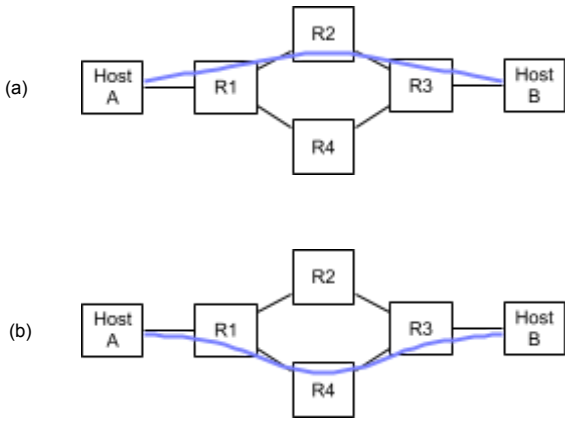

Figure 1: Zero impact traffic path change

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

- Create a network topology involving a simple four node router topology, *R*1 through *R*4, between two end hosts, *HostA* and *HostB*. Both the hosts and the routers will be Docker containers executing in a Linux virtual machine (VM), and the containers themselves will also execute Linux.. You will use Docker configuration files and commandline functions to create this topology (including IP subnet assignments suitable for a routed network topology).
- Deploy the open source FRR routing software in your routed network, execute the FRR OSPF daemon and create appropriate configuration files to turn the network into an OSPF routed domain. Once

configured in this way, the two hosts should be able to communicate with each other via the routed network.

- The traffic management goal of the assignment is to create a python orchestrator that can move the traffic between the hosts from either the northern path (i.e., going through router *R*2 as shown Figure 1 (a)), to the southern path (i.e., going through router *R*4 as shown Figure 1 (b)), or vice versa. Critically, this change in the traffic path should be orchestrated so as to occur without causing **any** packet drops.

  Assuming the traffic is being moved as shown in Figure 1 (a) to (b), i.e., from the northern path to the southern path, this can be achieved by setting the OSPF links weights such that router *R*1 will prefer the path via *R*4 towards *HostB*, and similarly *R*3 will prefer the path via *R*4 towards *HostA*. Your orchestrator will have to programmatically set the link weights between *R*1 and *R*2 and between *R*2 and *R*3 to realize this.

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 two parts. You will have to demonstrate (and will be graded on) your progress for each part:

1. Full topology with OSPF routed functionality:
   - We will use the POWDER platform and you will instantiate a single Linux VM on which the assignment will be done.
   - You will be provided with the following:
     - A script with which to install Docker tools.
     - Docker configuration files with which to create a simple three node topology: *HostA* < − > *R*1 < − > *HostB*. (Including appropriate prefix/address assignment and ensuring *R*1 is able to forward IP traffic between the two hosts.)
   - You will use the provided material to familiarize yourself with Docker and basic router function- ality.
   - You will then have to:
     - Create Docker configuration files to create the complete topology shown in Figure 1.
     - Install and configure FRR OSPF in this topology to enable routing between the two hosts. (Including appropriate prefix/address assignment suitable for routing.)
     - Demo: Show your Docker setup, show that your OSPF configuration is working correctly and demonstrate being able to successfully *ping* between the two hosts.
2. Orchestrated traffic movement:
   - Develop an orchestrator to automate the movement of traffic by manipulating OSPF link weights.
   - Demo: Show your Docker setup and demonstrate the use of your orchestrator to move traffic from the *R*1 < − > *R*2 < − > *R*3 path to the *R*1 < − > *R*4 < − > *R*3 path, or vice versa, without incurring any packet loss.

3 Assignment details

3.1 Part 1: Full topology with OSPF routed functionality

- Install Docker, verify basic network setup, become familiar with Docker configuration files:
  - Once your node is ready: On the ”List View” tab, click on the gear icon in the node row and select “Open VNC window” to access a Linux desktop on your VM.
  - Run the following commands to instantiate your three node network: HostA < − > R1 < − >

HostB

need to run docker commands as root sudo
bash
start up Docker containers as specified by config files docker compose
up -d

Other useful Docker commands:

see all containers and status docker ps
see all networks and status docker
network ls
see more details of specific network docker
network inspect <network name>
to attach to a running container, e.g., to execute commands in container docker attach
<container name>
to detach from container and leave it running Ctrl+P Ctrl+Q
to take down all containers docker
compose down

- In this single router topology, the router will be able to route between its directly connected subnets without running a routing protocol. You will however, need to set up routes on the two hosts to point them towards the routed network. E.g., the provided configuration files use the subnet 10.0.14.0/24 between *HostA* and *R*1, and the subnet 10.0.15.0/24 between *R*1 and *HostB*; On the 10.0.14.0/24 subnet *HostA* uses the IP address 10.0.14.3 and *R*1 uses 10.0.14.4; On the 10.0.15.0/24 subnet *HostB* uses the IP address 10.0.15.3 and *R*1 uses 10.0.15.4.

  Under these conditions, the following command will instruct HostA to attempt to reach 10.0.15.0/24 via the appropriate R1 interface:

  If you are “Docker attached” to HostA:

  route add -net 10.0.15.0/24 gw 10.0.14.4

  Or, you can execute the commend directly from the VM command line to execute it inside the HostA container:

  docker exec -it part1-ha-1 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.

  Verify that you can ping across the router: If you are “Docker attached” to HostA:

  ping 10.0.15

- Create the four node network topology:

- Once you have verified the above single router functionality, you should study the provided Docker configuration files and Docker documentation to allow you to create a set of configuration files to create the topology depicted in Figure 1.

  Helpful pointers:

  https://docs.docker.com/get-started/docker-overview/

  https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/

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

  Helpful notes:

  - *docker − compose.yaml* can be extended/recreated to specify the containers and networks in the more complex network setup.
  - Creating per-node *Dockerfile* files (and indicating that in the *docker − compose.yaml* file) might simplify subsequent steps

- Install and configure FRR OSP daemon:

  - Once your four-router network is set up, attach to each of the router nodes *dockerattach < containername >* and execute the following commands to install FRR (from: https://deb. frrouting.org):

    ```
    # remember to deattach with Ctrl+P Ctrl+Q to keep the container running... apt -y install curl
    apt -y install gnupg
    curl -s https://deb.frrouting.org/frr/keys.gpg | \
            tee /usr/share/keyrings/frrouting.gpg > /dev/null apt install
    lsb-release
    FRRVER="frr-stable"
    echo deb '[signed-by=/usr/share/keyrings/frrouting.gpg]' https://deb.frrouting.org/frr \
            $(lsb_release -s -c) $FRRVER | tee -a /etc/apt/sources.list.d/frr.list apt update && apt -y
    install frr frr-pythontools
    ```

    While still being connected to the router container in question, enable the OSPF daemon by **editing** */etc/frr/daemons* to have:

    ospfd=yes

    While still being connected to the router container in question, restart FRR by executing:

    service frr restart

    Verify that the OSPF daemon is running:

    ps -ef | grep ospf

    Study the FRR OSPF configuration guide (https://docs.frrouting.org/en/latest/ospfd. html) then run vtysh to enter FRR command line configuration system and configure the router:

    vtysh

Configuration notes:

* You will need to tell the router which of its attached networks should be advertised using OSPF (all the networks attached to each router should be advertised), and which OSPF area to use (using the OSPF backbone area, i.e., 0.0.0.0, should work fine). E.g., for a router connected to 10.0.10.0/24 and 10.0.13.0/24 your OSPF configuration might look like this:

  router ospf
  ospf router-id 192.168.1.1 network
  10.0.10.0/24 area 0.0.0.0 network
  10.0.13.0/24 area 0.0.0.0 exit

* You will need to assign an OSPF link cost to each of the router interfaces connected to other routers. E.g., the configuration snippet to set the OSPF weight on interface eth0 might look like this:

  interface eth0 ip
  ospf cost 5 exit

Note:

* You will need to repeat the above steps on all four of your routers, of course with slightly different configurations for each, in order to get fully functional OSPF in your network.
* Once you have things figured out you would want to create appropriate configuration files and scripts and add that to your Docker config files to simplify and automate the basic OSPF network setup.
* It is possible to execute *vtysh* commands from the container commandline. For example:

  # to show the current running configuration vtysh -c 'show
  running'
  # to set the OSPF weight for interface eth0 to 10
  vtysh -c 'configure terminal' -c 'interface eth0' -c 'ip ospf cost 10' -c 'end' # to store the current running config to file
  vtysh -c 'write memory'

* It is also possible to execute these command from the Docker host... For example, to show the running config for FRR executing on a container called *class* − *r*1 − 1, execute the following from the Docker host commandline (i.e., your VM):

  docker exec -it class-r1-1 vtysh -c 'show running

3.2 Part 2: Orchestrated traffic movement

Given the understanding you have developed in part 1 of the assignment, you should now finalize the assignment according to the description in Section 2.

- **Orchestrator.** You can package your orchestrator code in any way you please, but it should have a single executable python script 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 network topology shown Figure 1 (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. Ability to move traffic between the "north" path (*R*1, *R*2, *R*3), and the "south" path (*R*1, *R*4, *R*3). (Figure 1). Or, vice versa.

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

- **Evaluating your orchestrator.**

  For evaluation you will have to demonstrate the above mentioned functionality of your orchestrator. 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.