Name 1:

Name 2:

TCP/IP NETWORKING

LAB EXERCISES (TP) 4
DYNAMIC ROUTING (OSPF) AND SDN BASICS
With Solutions

November, 2020

Solution: Known bugs in this lab:

- *For some weird reason, some students have issues with OpenVSwitch switches [2019].*
  - *Symptom: If they run `sudo mn` (that runs the mininet default network), the console is stuck at step `Adding switches` or `Starting 1 switches`.*
  - *Reason: It means that they are unable to use openVSwitch, the specific reason is unknown.*
  - *Solution: These students have the choice: either use the laptop of their team-mate for all labs using mininet or they can use linux bridges, the following way:*
    - *In all Python scripts they need to:*
      - *add the line `from mininet.nodelib import LinuxBridge`*
      - *change the line `net = Mininet()` by `net = Mininet(switch=LinuxBridge)`*
    - *We don't know any alternative to the OVS commands used in **6 BONUS: SOFTWARE DEFINED NETWORKING**, we added a note to ask the students to use another laptop.*
- *Some students had permissions issues with the lab4 folder on the desktop of the guest machine [2019].*
  - *Symptom: "I have all ospfd processes running on routers, but none exchange of topology information happens, and all ospf databases remain empty." https://moodle.epfl.ch/mod/forum/discuss.php?d=22443*
  - *Reason: These students have made the `lab4` folder shared with their host machine. The intended procedure for the `lab4` folder was: place the `lab4` folder in the shared folder, THEN move it to the desktop of the guest machine, so that the resulting `lab4` folder in the desktop of the guest machine is NOT shared with the host machine. For students that made the `lab4` folder on the guest desktop shared with the host machine, it results in a permission issue that does not allow ospf to write files in the folder, making ospf hang.*
  - *Solution: The `lab4` folder in the guest desktop should not be a shared folder with the host machine.*

1 Lab Organization and Instructions

1.1 Lab Organization

In this lab, you will learn how to configure the OSPF routing protocol, which typically runs inside the network of an autonomous system (AS). The protocol automatically sets up network routes that ensure shortest path between two points in the network with respect to a predefined metric (such as number of hops). You will configure a fully functional network using "Cisco-like" **emulated** routers. You will learn how to configure a network of routers. Optionally (bonus exercise), you can learn about software-defined networking (SDN) and study how it can be used to create forwarding rules on switches. You are strongly advised to do this part as it gives you a good idea of how enterprise networks are managed today.

For this lab, you will be using the same Mininet on the virtual machine provided on Moodle (see Lab 0 for installation instructions if need be).

The lab is meant to be done sequentially. Random access might give you different answers. We advise you to do a section in one sitting. Restarting Mininet within a section might make the analysis difficult.

1.2 Lab Report

Type your answers directly in this document. We recommend you to use the latest or an updated version of Adobe Reader to open this PDF, as other readers (such as SumatraPDF, but also older versions of Adobe!) don't support saving HTML forms. This will be your Lab report (one per group). When you finish, save the report and upload it on Moodle. Don't forget to write your names on the first page of the report.

The deadline is Wednesday, November 25 at 23:55.

2 FRRouting: Software Routing Suite

The Internet core is run by powerful routers that can handle large amounts of traffic and are built by companies such as Cisco, Huawei, or Juniper. Even in a large company, or in large university campuses (such as EPFL), these routers are present. They use proprietary operating systems (such as Cisco IOS, or JunOS) and can be accessed via control terminals tailored for network configuration, with commands that are quite different from those you may encounter in a UNIX/Linux console. In this lab and in lab 6, we will give you a flavour of router configuration.

Ideally, we would have liked to run Cisco IOS in a virtual environment. It is technically possible but not legal, as Cisco or Juniper do not allow their OSs to be run on a device other than their routers. Therefore, we will use FRR, a free software that implements and manages various IPv4 and IPv6 routing protocols. It accepts similar commands to the ones in Cisco's IOS.

2.1 What is FRR and how does it work?

FRR is a routing software suite that provides implementations of several routing protocols (namely OSPF, RIP, and BGP-4) for Unix platforms. The architecture of FRR is shown in Figure 1.

As depicted in the figure, it consists of a handful of processes that can be run in the background as daemons. Three FRR processes (daemons) are important for executing this lab:

- *zebra*: is used to manage the network interfaces of a machine (in our case, each router will run in the virtual machine). It allows you to configure them (using IPv4 and/or IPv6 addresses), to monitor their states, and it provides a more detailed view of the routing tables than the `route -n` command. In a

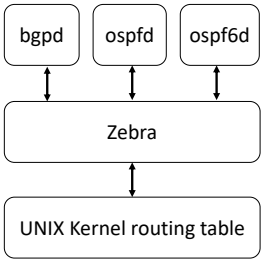

Figure 1: FRR Architecture

way, *zebra* is a replacement for the Linux networking commands used during the first three labs (*i.e.,* `ifconfig`, `route`, etc.).
- *ospf*: handles OSPF version 2 implementation.
- *ospf6d*: handles OSPF routing for IPv6.

3 SCENARIO: A GAME OF ROUTERS

With the impending attack of the White Walkers on Westeros, all the kingdoms must unite in the fight for the living. As the lands of men stretch from Winterfell in the North to Valyria down in the South, the wise maesters at the Citadel have realized that their communication network with carrier ravens will not be sufficient to communicate for large distances in short time.

After several sleepless nights in the Citadel, Maester Illyrio Pycell and his intern Samwell Tarly have designed the communication mechanism, called TCP/IP. Together, they setup routers and switches at strategic locations in Westeros as shown in Figure 2. There are a total of 5 routers r1-r5 placed at Winterfell, Braavos, Valyria, Casterly Rock, and King's Landing, respectively. Samwell has connected the routers through 7 switches as shown in the figure. He has also configured the routers with the IP addresses as shown in the figure. Note that, all IP Addresses for router rx end in x.

Sam's configuration scripts are available in the lab folder. To be consistent with the paths in this document and in the scripts, please place the uncompressed folder directly in the Desktop of the virtual machine and be sure to name it **lab4** (case-sensitive). Run `lab4_network.py` as root from a terminal of your virtual machine, to recreate the established topology. See the output of the `net` command in the Mininet prompt and verify whether the connections in the created network correspond to the ones of Figure 2. Use the `pingall` command in the mininet prompt.

Q1/ What percentage of the 78 possible combinations of connections (between 8 hosts and 5 routers) in Westeros are functional (approximated to the nearest integer)? **Explain why you can or cannot ping some or all of them.** Hint: Check the Python script that creates the network topology.

Solution: *22% (17 out of 78)*
In our python script, we are only configuring/assigning one IP address per router and host. In the case of h1 and r1, we assigned the ip addresses that belong to the same LAN. We don't have other connectivities because we have not configured all the interfaces of different hosts and routers (except one interface at each host and router) and we have not enabled static or dynamic routing at the routers.

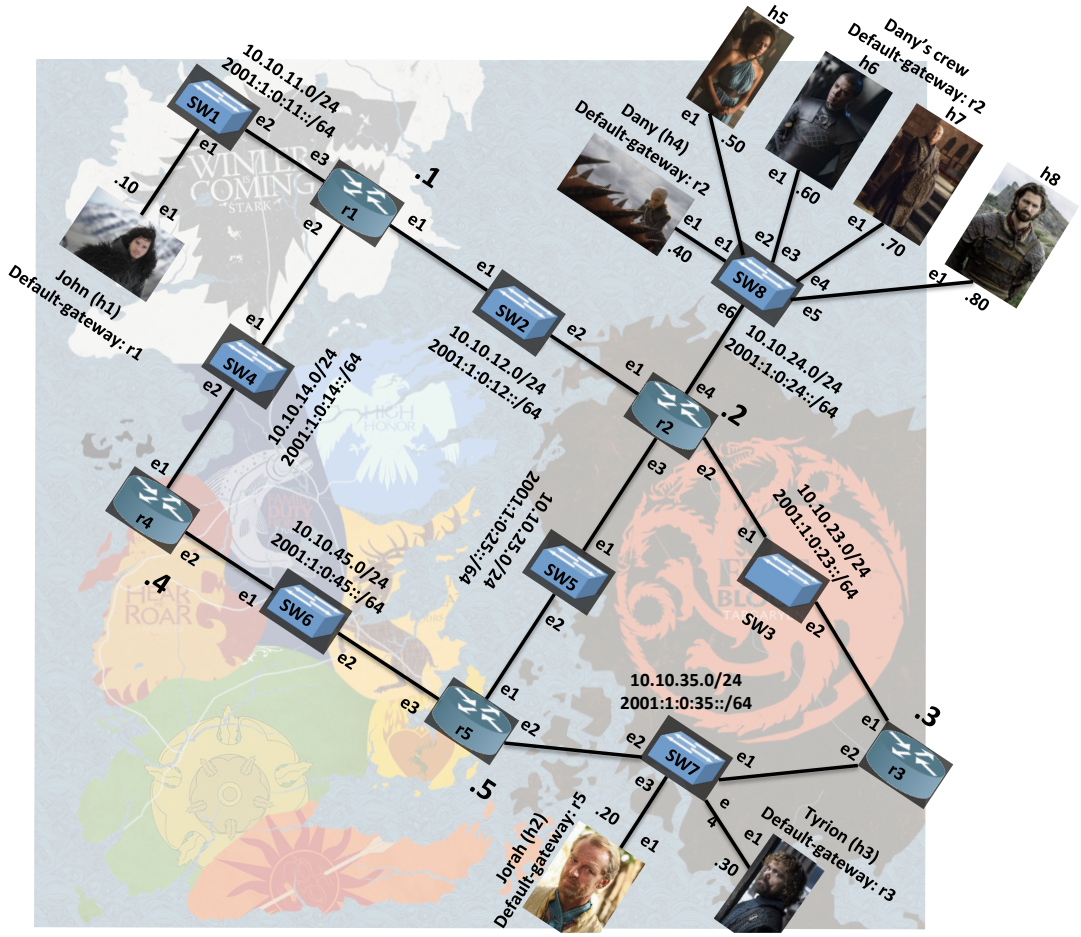

Figure 2: Networking in Westeros

In the next section, your job is to help Sam with the configuration of the network so that all kingdoms can talk to each other.

Each router is, in essence, a Linux machine, thus they can be configured the same way. This means that you could reuse the same set of commands that you used for previous labs, e.g., to configure network interfaces, to monitor their states, to inspect the contents of the routing tables, etc. However, instead of the set of Linux networking commands, in this lab, you will be using the tool suite from *FRR*. Documentation can be found on the FRRouting website (*www.http://docs.frrouting.org*).

4 HOST NETWORK CONFIGURATION WITH FRR

In principle, you could create a topology without FRR, launch a terminal window for each router, and configure its network interfaces using the `ip addr` command (following the scheme shown in Figure 2). However, in this lab, we will be using the `zebra` daemon instead.

Note: Never try to configure the network interfaces on a machine using both the `ip addr` command and *zebra* daemon, as interaction between the two is not always clear and the outcome may be uncertain.

4.1 CONFIGURING INTERFACES USING CONFIGURATION FILES

Before running your topology script, a configuration file must be created and edited at least for the `zebra` process. Script 1 is an example of such a file written for router `r3`. This file can be found among setup files available on Moodle in the `lab4/configs` folder.

Script 1: Zebra configuration file for r3: `zebra_r3.cfg`

```
hostname r3
password zebra
enable password zebra

log file /home/lca2/Desktop/lab4/logs/zebra_r3.log
debug zebra packet

ip forwarding
ipv6 forwarding

interface r3-eth1
no shutdown
ip address 10.10.23.3/24
ipv6 address 2001:1:0:23::3/64

interface r3-eth2
no shutdown
ip address 10.10.35.3/24
ipv6 address 2001:1:0:35::3/64
```

Let's examine the content of this configuration file:

- `!`: the lines starting with `!` are comments, they are ignored;
- `enable password`: this creates a password for "on the fly" configuration of the zebra process, in this case the password is set to `zebra`;
- `log file`: Allows you to specify the file to which zebra related information is logged (adding, deleting routes in principle). **Make sure you write the full path to the file as it could prevent the FRR service from starting**;
- `debug zebra packet`: more detailed debugging, *i.e.,* allows you to see when routes are added or deleted from the routing table;
- `ip forwarding`: This instructs the zebra process to enable IPv4 routing on the router;
- `ipv6 forwarding`: This instructs the zebra process to enable IPv6 routing on the router;
- `interface <interface name>`: this command starts the interface configuration mode;
- `ip address`: assigns an IPv4 address to the selected interface;
- `ipv6 address`: assigns an IPv6 address to the selected interface;

We already created all the configuration scripts for the routers $r3$, $r4$, and $r5$, you can find them in the folder we provided. **You can now create the configuration scripts for the routers $r1$ and $r2$.**

You should now have all the following files: `zebra_r1.cfg`, `zebra_r2.cfg`, `zebra_r3.cfg`, `zebra_r4.cfg`, `zebra_r5.cfg`. For consistency with the paths of the proposed commands and with the paths of the log files, it is highly recommanded that you copy your `lab4` folder to your Desktop. This folder contains the `configs` folder, the topology script, an empty `logs` folder and an empty `run` folder. Each time you launch the simulation, the `logs` and `run` folders must be empty and the `configs` folder must contain only the `.cfg` files. The topology script erases the content of the `logs` and `run` before creating the topology. If you wish to stop the mininet simulation, type `exit` in the mininet prompt and clear the cache with the following command:
`sudo mn -c`.

Activate the zebra daemon at router `r1` with the following command:

```
zebra -d -f /home/lca2/Desktop/lab4/configs/zebra_r1.cfg -i
/home/lca2/Desktop/lab4/run/zebra_r1.pid -z /home/lca2/Desktop/lab4/run/frr_r1.api
-u root -k
```

Note that in order to paste a copied command in a terminal, you should click on the middle-button of your mouse on the terminal. Further note that an error can be due to copy-pasting characters from a pdf (empty spaces may include hidden invisible characters which the terminal does not understand). Do the same for all routers.

We have manually started the zebra daemon on each router. In order to prevent re-writing this command at each router at each simulation, you may update the python script that creates the network topolgy `Lab4_network.py` in order to make the process start automatically at all routers. This can be done by updating the script with commands as such:

```
<routerID>.cmd('<command>')
```

where <routerID> is $r1$, $r2$, $r3$, $r4$ or $r5$ and <command> is the command you would type on the router's terminal.

Check that zebra is running for all routers by checking the list of all running processes in the VM: you can do so with the following command: `ps -A`, if you wish to restrict the list to only zebra processes type the following: `ps -aux | grep zebra`. Observe that each process has a PID, if you wish to stop a process , the command is: `kill < PID >`. If you want to stop all zebra processes, you can do so with the following: `kill $(ps -aux | pgrep zebra)`, pgrep directly extracts the PIDs. Note that processes running for mininet entities such as $r1$, are in fact running on the VM, hence by terminating the mininet simulation, processes don't stop automatically. Make sure that you kill them all before launching new simulations, you can add this command in the script to be executed at the beginning of each new simulation.

4.1.1 MONITORING MODE

In order to monitor the activity of a running process, e.g., `zebra`, you can enter the monitoring mode by connecting to it using the following command in the terminal of a router.

```
telnet localhost zebra
```

Th password is `zebra`. Let us see what information can be obtained now. To inspect the contents of the IPv4 and IPv6 routing tables type `show ip route` and `show ipv6 route` respectively. At all times, you can view the entire list of the commands at your disposal using `list`.

Q2/ What subnets can be found in the two routing tables on $r3$ at this point?

Solution:
10.10.23.0/24
10.10.35.0/24
2001:1:0:23::/64
2001:1:0:35::/64
fe80::/64 (r3-eth1 link local)
fe80::/64 (r3-eth2 link local)
These are the same routes that you find if you do a ip route *or* ip -6 route *commands from the linux terminal.*

Next, check the status of the network interfaces by typing `show interface` command.

To view the current running configuration, you need to first enable the configuration mode using

```
enable
```

The password for the configuration mode is `zebra`. Finally, inspect the running configuration of the router $r3$ using the `show running-config` command and verify if it is the same as the one you used.

4.2 CONFIGURING OSPF

The `ospfd` process implements OSPFv2 that is defined in RFC 2328. Similar to the `zebra` process, the `ospfd` process can be configured by editing a configuration file or "on the fly". However, in this lab, we will not configure it "on the fly". Like the `zebra` configuration files, we are providing the "ready-to-use" `ospfd_ri.cfg, i=3,4,5` configuration files. We give an example of `ospfd_r3.cfg` configuration file in Script 2.

Script 2: Ospfd configuration file for r3: `ospfd_r3.cfg`

```
hostname r3
password ospfd
enable password ospfd

log file /home/lca2/Desktop/lab4/logs/ospfd_r3.log

debug ospf event
debug ospf packet all

router ospf




network 10.10.23.0/24 area 0
network 10.10.35.0/24 area 0
```

Let's have a closer look at the commands in the file above (you are already familiar with some of them):

- `log file`: Same as with `zebra`, it specifies the file to which the OSPF related information is logged;
- `debug ospf event`: less detailed debugging, *i.e.,* only the coarse events, such as sending and receiving OSPF updates, can be seen;
- `debug ospf packet all`: more detailed debugging, *i.e.,* allows you to see the content of the sent and received OSPF updates;
- `router ospf`: enables the *ospfd* process;
- `network`: assigns an area to a particular network segment.

Using the example configuration file shown above and the address scheme of Figure 2, **create the *ospfd* configuration files for the routers $r1$ and $r2$ and place them in the same folder `/home/lca2/Desktop/lab4/configs`**

In this section, you must have `zebra` running on all routers and `ospfd` running on all routers except $r4$. In order to start the `ospfd` process at router `r1` type the following command at `r1`'s terminal:

```
ospfd -d -f /home/lca2/Desktop/lab4/configs/ospfd_r1.cfg -i
/home/lca2/Desktop/lab4/run/ospfd_r1.pid -z /home/lca2/Desktop/lab4/run/frr_r1.api
-u root
```

Again, make sure that all processes are running by checking the list of running processes.

Q3/ Open wireshark on router $r2$. By looking at the exchange of packets, how is the routing information between routers exchanged (*i.e.,* by using broadcast, unicast or multicast)? Open the *lab4/logs/ospfd.log* file on router $r2$ and compare to what you see in Wireshark. Write the line from the log file that confirms your observation in Wireshark.

Solution: *The packets are exchanged by multicast.*
`2019/08/19 15:06:09 OSPF: Hello sent to [224.0.0.5] via [r2-eth1:10.10.12.2].`

Q4/ Check the *lab4/logs/ospfd.log* file on router $r2$. How does $r2$ identify $r1$?

Solution: *Through its OSPF Router-ID: 10.10.12.1 using the hello protocol. (This answer could be different for different students, but it has to be one of the IP addresses of $r1$.)*

4.2.1 MONITORING COMMANDS

We have seen before that we can use command `ip route show` to display the IPv4 routing table. Now, we will see how this information is stored and managed by OSPF. OSPF stores all information about the networks it knows about in the OSPF database. An OSPF router also maintains a list of neighbours from whom, it expects periodic updates called link state advertisements (LSAs).

We will start by inspecting the neighbors of $r3$. For this, you can enter the monitoring mode of the `ospfd` daemon. This is done in the same way as for the *zebra* process (*i.e.,* `telnet localhost ospfd`). The password is `ospfd`. After that, you can display the content of the OSPF database using the command:

```
show ip ospf neighbor
```

Q5/ How many OSPF neighbors does $r3$ have? What are their ids? Identify which of those are designated routers (DRs) and which ones are Backup designated routers (BDRs)?

Solution: *Two neighbours.*
10.10.23.2
10.10.45.5 Depends. For me, all of them were DRs.

Q6/ What does the column Dead Time represent? *Hint: Launch the above command several times*. How is it related to the hello timer? Can you infer the values of the dead time and the hello timer?

Solution: *Dead time is the time after which the neighbour is considered dead and all its LSAs are removed from the database. Hello timer is the time after which it expects a hello message. 4 consecutive hellos lost means dead.*
40 s and 10 s.

Next, we will look at the LSAs in the database of routers. For this, you can use the following set of commands.

```
show ip ospf database
show ip ospf database network
show ip ospf database router
show ip ospf database self-originate
```

Q7/ What are the network LSAs advertised by $r3$? How is this related to the answer to question 5?

Solution: *This answer depends on Q5. Basically, $r3$ advertises the network LSAs for which it is a DR, hence in our case, no network LSA is advertised by $r3$.*

Q8/ What are the prefixes that $r1$ advertises to $r2$ in its router LSA? Explain the difference.

Solution: *To find this out, first check the type of the self-originating LSAs at $r1$. It will be a router LSA. Then look at the router LSA database of $r1$. You will find three networks: 10.10.11.0, 10.10.12.1, and 10.10.14.0. One is a transit and the other two are stub. It's because $r4$ is not yet an OSPF router as we have not yet run the ospf daemon on it.*

Q9/ How many network LSAs and router LSAs are present in the OSPF database of $r3$? Explain why?

Solution: *There are 4 router LSAs and 4 network LSAs.*

Each router advertises 1 LSA. As there are 4 OSPF routers, we have 4 router LSAs.
Each transit network has one LSA that is advertised by the designated router of the LSA. As there are 4 transit networks, there are 4 network LSAs.

John wants to see how far they are from their friends Jorah ($h2$) and Tyrion ($h3$) currently travelling through Valyria, accessible from network 10.10.35.0/24. From router $r1$, use the `show ip ospf route` command.

Q10/ What is the cost from $r1$ to reach the Valyrian network? For this cost, what are the possible routes?

Solution: *The cost from $r1$ to the Valyrian network is 30, it goes through $r2$ and then either $r3$ or $r5$.*

Note that $h2$ and $h3$ are both on the Valyrian network and that $h1$ is on network 10.10.11.0/24, thus packets between $h1$ and hosts from the Valyrian network should behave in a similar fashion. Do traceroutes from $h1$ to all hosts on Valyrian network.

Q11/ Do all traceroutes show the same path? Give proportions and explain your observations.

Solution: *All traceroutes don't show the same path, some routes go through $r3$ while others go through $r5$, in our case the proportions are exactly $50 - 50\%$ but they could be different for each student. We can also observe that the traceroutes from John to $h2$ and $h3$ don't take the same route (this also depends, might not be the case for all). This is because of Equal Cost Multipath (ECMP) behavior in Linux machines. A (uniform) hash function is applied on the $\{$ ip source, ip destination$\}$ tuple to select the next hop. Hence each tuple is randomly attributed to a next hop, which ensure some load balancing of the two equal cost paths. See https://codecave.cc/multipath-routing-in-linux-part-2.html*

4.3 CONFIGURING OSPF6D

OSPF for IPv6 was defined in the RFC 2740 and is known as OSPFv3. It is an IPv6 reincarnation of the OSPF protocol.

As multiple routing protocols can run in parallel on the majority of routers, FRR allows you to configure OSPFv3 in parallel to OSPFv2. Just like `zebra` and `ospfd` processes, the `ospf6d` process can be configured by editing a configuration file or via telnet.

As in the case of `ospfd`, we are providing the *ospf6d* configuration files for the routers $r3$, $r4$ and $r5$. The configuration file of `ospf6d` is a little different from that of `ospf`. Below, we explain the file `ospf6d.conf` on $r3$.

Script 3: Example of an ospf6d.conf file

```
hostname r3
password ospf6d
enable password ospf6d

log file /home/lca2/Desktop/lab4/logs/ospf6d_r3.log
debug ospf6 neighbor
```

```
debug ospf6 interface

interface r3-eth1
 ipv6 ospf6 instance-id 1

interface r3-eth2
 ipv6 ospf6 instance-id 1

 router ospf6

 interface r3-eth1 area 0.0.0.0
  area 0.0.0.0 range 2001:1:0:23::/64

 interface r3-eth2 area 0.0.0.0
  area 0.0.0.0 range 2001:1:0:35::/64
```

To configure `ospf6d`, you need to first specify the interfaces on which you wish to enable OSPFv3. Then, you need to configure the OSPF area and network those interfaces belong to. The commands are.

- `interface <interface-name>`: Enable OSPFv3 on this interface.
- `interface <interface-name> area <A.B.C.D>`: Assigns the interface to a given area. Notice that although it is an IPv6 service, it uses a 32-bit area code that is represented in the dotted decimal format.
- `area <A.B.C.D> range <IPv6 network with mask>`: Specifies which networks belong to a given area.

The configuration scripts of $r1$ and $r2$ are left for you to create. You should place them in the same folder as earlier `/home/lca2/Desktop/lab4/configs/`.

In this section, you must have `zebra` running on all routers, and `ospfd` and `ospf6d` running on all routers except $r4$. In order to launch `ospf6d` on $r1$, you can type the following command at the terminal of the router:

```
ospf6d -d -f /home/lca2/Desktop/lab4/configs/ospf6d_r1.cfg -i
/home/lca2/Desktop/lab4/run/ospf6d_r1.pid -z /home/lca2/Desktop/lab4/run/frr_r1.api
-u root
```

Again, make sure that all processes are running by checking the list of running processes.

4.3.1 MONITORING COMMANDS

To inspect the `ospf6d` database, you should execute the command: `telnet ::1 ospf6d`. To display the content of the OSPF routing table:

```
show ipv6 ospf6 route
show ipv6 ospf6 route detail
```

Now, explore by yourself, the available monitoring commands. Recall that you can press `?` at any time to list the possible commands at an instant.

Q12/ Check the SPF tree of $r3$. Write the used command. In `ospf6d`, what are the costs for a packet to go from a network to a router and a router to a network?

Solution: *The command to check the spf tree is* `show ipv6 ospf6 spf tree` *and we observe that:*
Network to router: 0
Router to network: 10.

5 OSPF PLAYGROUND

In this section you will be confronted with a number of situations that might arise in an OSPF network. Answering the questions will allow you to better understand `ospfd` and `ospf6d`, and dynamic routing in general. You can use two terminals, one for the FRR process commands and one for the Linux commands. To have a second `xterm` window for `xterm r1`, you can type: `xterm r1` in the `mininet>` prompt.

5.1 UNDERSTANDING NEIGHBORS IN OSPF

In this section, we will learn about how neighbors interact within OSPFv2 (`ospfd`).

Now, restart the Mininet network using the `python lab4_network.py` command. Don't forget to clear the mininet cache before restarting the network. If you did not enable the automatic start of zebra at all routers, manually do so at all routers. Activate `ospfd` at all routers except `r4`.

The next couple of questions will require the comparison of the OSPF database on $r5$ before and after launching the `ospfd` process on $r4$. Take a snapshot of the database for your reference. Specifically, `show ip ospf database`, `show ip ospf database router`, and `show ip ospf database network`. Now, we will see how OSPF routers are synchronised. Now, launch the `ospfd` and `ospf6d` processes on $r4$. This will make $r4$ an OSPF router. Let this time be $t1$.

Q13/ Does $r4$ appear in the neighbor list of $r5$? What's $r4$'s neighbor state at $r5$? What are the possible states of an OSPF neighbor finite state machine?

Solution: *Yes, with state Full.*
In total, there are 8 states: DOWN, ATTEMPT, INIT, 2-WAY, EX-START, EXCHANGE, LOADING and FULL. https://cyruslab.net/2012/04/01/ospf-finite-state-machine/

Q14/ What changes do you notice in the OSPF database at $r5$? Explain the reason for each of the changes.

Solution: *We see three new router LSAs from $r1$, $r4$, and $r5$. $r1$ and $r5$ advertise a change in their connected networks, i.e., the change in networks 10.10.14.0 and 10.10.45.0 from stub networks to transit networks. $r4$ on the other hand is just announcing its presence for the first time.*
As there are two new transit networks, we also see two network LSA for networks 10.10.14.0 and 10.10.45.0.

5.2 MODIFIED LINK METRIC

OSPF works by constructing a shortest path tree. By default, in OSPFv3, the directly connected subnets are treated as having the distance equal to 10. Each additional "hop" (router) adds 10 to this distance metric. Nevertheless, `ospf6d` offer you the possibility to modify the metric of an arbitrary link, changing the desirability of the routes that contain this link.

Cersei (at King's Landing in $r1$), being the paranoid queen, wants to know if Dany is talking to Jorah behind her back. So, she orders that the (one way) IPv6 traffic from $h4$ to $h2$ to go via $r1$ instead of the direct route. To fulfill this task, you need to modify the cost of the link between $r2$ and $r5$ so that the total cost of routing through $r2 \rightarrow r1 \rightarrow r4 \rightarrow r5$ is less than the direct route. The cost of an interface is set in the interface setup. The syntax for specifying cost is

```
ipv6 ospf6 cost x
```

Q15/ Describe the minimal set of changes to the configuration of router $r2$, router $r5$, or both of them, to achieve the desired affect. Write the modified configuration file(s).

Solution: *We need to make two changes to* $r2$ *on* $r2 - eth2$ *and* $r2 - eth3$
For interface $r2 - eth2$ *add* `ipv6 ospf6 cost x`
For interface $r2 - eth3$ *add* `ipv6 ospf6 cost x`
where `x` *must be a value strictly higher than 30.*

5.3 BROKEN LINK

John's dragon Drogon sometimes gets restless and burns things down. In one such incident, he burnt down the link between $SW2$ and $r1$. To simulate the broken link between $r1$ and $r2$, shutdown the *r1-eth1* interface on $r1$. To do this, type on $r1$ the following commands:

```
telnet localhost zebra
enable
configure terminal
interface r1-eth1
shutdown
```

Note that it is convenient to use the "on the fly configuration" here.

Q16/ Observe the changes in the OSPF database at $r2$. Explain what happens.

Solution: *After the expiration of the* Dead timer *of 40s, the network LSA from* $10.10.12.0/24$ *is marked invalid by setting its age to maxAge of 3600 because this network is no longer a transit network. As there is a change in the connected links of* $r1$*, it sends a new LSA which is updated in the router LSA section of* $r2$*. Moreover, the network LSA for* $r1$ *is also removed from the database of* $r2$ *after 40-60s (Wait timer).*

The crash of an OSPF process also has a similar behaviour but on all interfaces instead of just one.

Q17/ Can you ping the IPv4 address of $r1 - eth1$? Explain.

Solution: *No, we can't ping as the interface is down.*

Q18/ Instead of a broken link, if we were to comment out (using "!" at the beginning of the line) the `network 10.10.12.0/24` from $r1$ (in its `ospfd` configuration file), could you then ping the IPv4 address of $r1 - eth1$. Explain.

Solution: *Yes, it pings. Indeed,* $r1$ *is not announcing anymore a route toward* `10.10.12.0/24`*, but* $r2$

continue announcing that it has route toward this network. Thus, thanks to OSPF, $r1 - eth1$ *is reachable through* $r2$*.*

Q19/ In the case of not breaking the link and simply commenting out `network 10.10.12.0/24` from $r1$, conclude how does the OSPF protocol sees the connection between $r1$ and $r2$? What paths do packets from $r2$ to $h1$ and $r3$ to $h1$ take?

Solution: *OSPF sees two different routers with a common prefix (`10.10.12.0/24`), and these two routers have no communication between each other. Therefore,* $r2$ *has to take the route announced by* $r5$ *as the only choice to go to any of* $r1$*'s networks (other than `10.10.12.0/24`, as* $r2$ *has it directly connected) and installs the networks to `10.10.11.0/24` and `10.10.14.0/24` through* $r5$*.*
$r2 \rightarrow r5 \rightarrow r4 \rightarrow r1 \rightarrow h1$
$r3 \rightarrow r5 \rightarrow r4 \rightarrow r1 \rightarrow h1$

6 BONUS: SOFTWARE DEFINED NETWORKING

For students who could not run openVSwitch switches in the previous labs (and were using LinuxBridges), please do this part with your teammate's computer or contact the TAs to set up an alternative.

Software defined networking (SDN) is an approach to TCP/IP networking that allow network administrators to manage services through flexible interfaces provided by a control-layer. A typical SDN network consists of routers or switches interconnected, each of them with a "listener" daemon running in which they receive forwarding-decision rules, called flows, from one (or many) controller(s) using, for example, the OpenFlow protocol. The network administrators make changes to the controller and these changes are disseminated through the control-layer (typically another IP-based connection) so that appropriate routing/switching decisions can be applied.

The Mininet virtual environment used in this lab is compatible with SDN. In fact, the openVSwitch (OVS), which is the switch we have been using so far in this lab, is a software listener switch that is controlled using the OpenFlow protocol. The central controller that comes with Mininet has no special features or policies, which make the openVSwitches used in the virtual environment, to behave as a basic L2-switch.

In order to explore SDN, we need a new controller. We will use the RYU controller [1], which is an open-source controller written in Python. In the `lab4/SDN` folder, unzip the file `ryu.zip` where you will find the source files of RYU. If you followed the path consistency instructions, the command

```
cd /home/lca2/Desktop/lab4/SDN && unzip ryu.zip
```

should unzip the folder. Otherwise, make sure the uncompressed folder is located in the `SDN` folder and its name is `ryu` (case-sensitive). Run the script `install_ryu.sh` as root on your virtual machine to install RYU. You can check if RYU is properly installed by trying out the command `ryu-manager -h`. It should print the help menu without any errors.

In Mininet, the configuration changes for using an external controller are minimum. Basically we need to call the `RemoteController` class by importing it from `mininet.node`, and then when we call the constructor `Mininet`, we pass the attribute `controller=RemoteController`. With this configuration, the switches will look on the VM's loopback address (default port 6633) for the controller.

The topology for this section is composed of 5 switches and is depicted in Figure 3. The script `lab4_sdn.py` (available on Moodle) has been created for you to match the topology described.

Q20/ How many subnets are there in Figure 3?. What would be the correct network mask for each IP address?

Solution: *There is only one single subnet, thus all IP addresses should share a common prefix. The one used in this section is `/16`, but any prefix up to `/21` is valid.*

6.1 CONTROLLING THE OPENFLOW SWITCH IN STANDALONE MODE

The OVS switch comes with the `ovs-ofctl` utility, which uses the terminal prompt to send basic OpenFlow messages, for basic configuration and retrieving important information such as installed flows, port information, dump statistics, etc. To display a complete list of the commands (with a brief description of what it does), type the `ovs-ofctl -h` command from a terminal prompt. Note that Mininet also comes

[1] https://osrg.github.io/ryu/

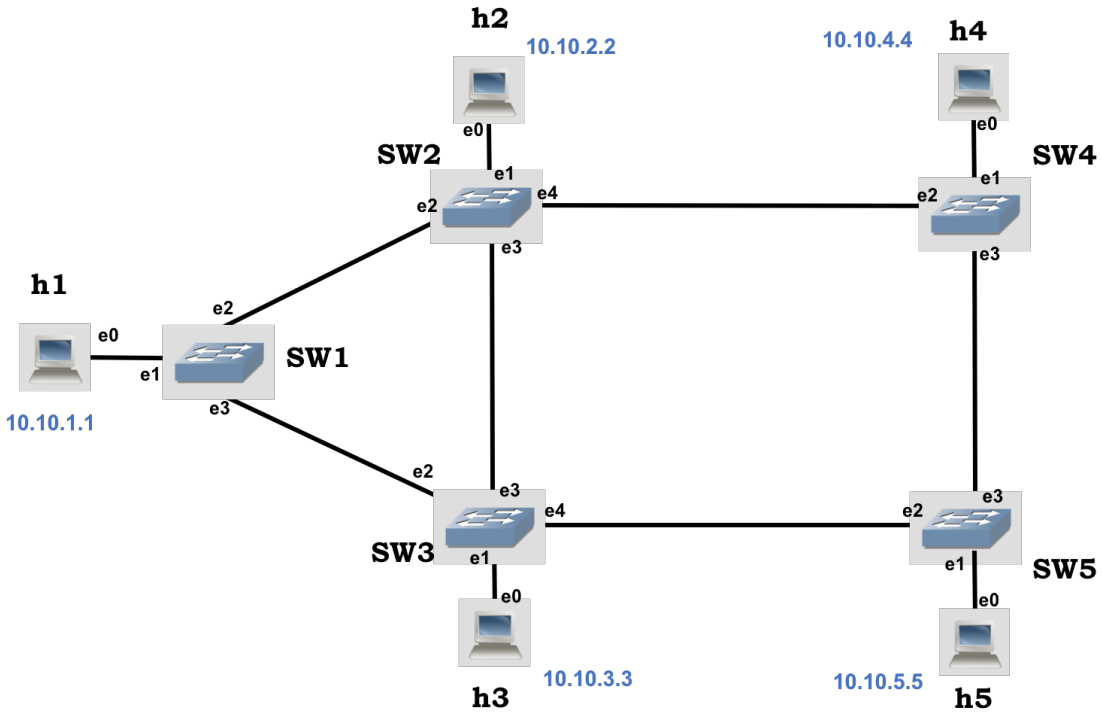

Figure 3: Lab 4 SDN topology

with a similar utility called `dpctl`, which has a short version of the `ovs-ofctl` utility. Unlike the latter, `dpctl` is available through the `mininet>` prompt.

Let's analyze the behavior of the OVS switch in the absence of the controller. Run the `lab4_sdn.py` script. Now, from $h4$ try to make three pings to $h5$:

```
mininet>h4 ping -c 3 h5
```

The pings are unsuccessful as the switches in the network don't know what to do with the incoming packet. Now, let's use the `ovs-ofctl` utility to help $h4$ and $h5$ communicate. From a new terminal prompt (from Linux, not Mininet), type the following commands:

```
sudo ovs-ofctl add-flow SW4 in_port=1,actions=output:3
sudo ovs-ofctl add-flow SW5 in_port=3,actions=output:1
```

From the commands above:

- `add-flow`: refers to the command we are sending to the OVS switch, other options are *del-flow, dump-flows, dump-port, mod-port, etc.*
- `<switch>`: refers to the OVS switch we would like to send the message to. By default it resolves the names of the switches that are in the same machine, for others we need to specify IP address and TCP port.
- `in_port=1`: we tell the OVS switch on which port it should apply the flow to, in the inward direction.
- `actions=output:3`: we tell the OVS switch, what are we doing with packets matching the flow statement. In this case, we just send it out through a particular interface. Other options include flooding to all ports, setting a new *next-hop*, changing a particular field in the IP packet (priority, TTL, flags), etc.

Q21/ Apply the `ovs-ofctl` commands and test the ping command again. Does it work? Explain what is happening and which commands (if any) would help you fix the problem. **Hint**: use wireshark to check for packet arrival

Solution: *In $h5$, we see ARP request arriving from $h4$ and we see that $h5$ replies to these packets. In $h4$ we do not see any ARP reply, only ARP requests. We can suspect there is traffic flowing in one direction thus we are missing flow configuration in the reverse path. By applying the following commands, the ping starts to succeed:*

```
sudo ovs-ofctl add-flow SW5 in_port=1,actions=output:3
sudo ovs-ofctl add-flow SW4 in_port=3,actions=output:1
```

6.2 CONTROLLING THE OPENFLOW SWITCH USING RYU

Now it is time to use a remote controller. Before continuing, make sure you erase all flows from $SW4$ and $SW5$ by issuing the command `ovs-ofctl del-flows <switch>` from a Linux's terminal window (as root).

Stop the simulation in Mininet and start the RYU controller using a separate Linux's terminal window:

```
sudo ryu-manager ryu.app.simple_switch_13 ryu.app.ofctl_rest
```

The above command runs the RYU controller with the `simple_switch_13` applications. Leave it running on a separate terminal. Applications are "functionality add-ons" that are loaded onto the RYU controller, as the controller itself doesn't do much. In particular, the `simple_switch` application is the same as the one that comes as default component in Mininet's controller implementation, makes the OpenFlow switches to act as a "type" of layer-2 learning switch. The application learns MAC addresses, and the flows it installs are exact-matches on as many fields as possible. Therefore, for different TCP/UDP connections between two hosts, you will have different flows being installed.

We have also specified the use of application `ofctl_rest`. This application is used by RYU to access the flow tables of the switches. This allows RYU to write to the switches and read from them. The RYU controller comes with a few other applications that are located in the

`/home/lca2/Desktop/lab4/SDN/ryu/ryu/app` folder.

For all cases, start the RYU controller first, and then start Mininet by running `lab4_sdn.py`. In this way, you can see the progress of all logs from the switches in the RYU controller console, which will be helpful to answer many questions in this section.

With the controller started, the OVS switches should now have automatic flows installed for every packet and all hosts should be reachable. From the `mininet>` prompt, do a `h4 ping -c 1 h5` command.

Q22/ Why do you think ping was unsuccessful?

Solution: *The topology includes loops. In normal switches loops are handled by the `Spanning-Tree`*

Protocol (STP). As the simple_switch_13 application does not implement any STP, the loop in the network makes packets to multiply exponentially, causing a congestion.

Use the `ovs-ofctl show <switch>` and `ovs-ofctl dump-flows <switch>` commands (from Linux's terminal) and notice the flows in the switches. Now, stop the RYU controller by using `Ctrl + C` in the Linux terminal window. Next, let's explore a different application.

Now, start the RYU controller using the following command:

```
sudo ryu-manager ryu.app.simple_switch_stp_13 ryu.app.ofctl_rest
```

Wait around one minute and keep looking at the output from the RYU controller. Do `pingall` from the Mininet prompt.

Does it work now? Use the `ovs-ofctl show <switch>` and `ovs-ofctl dump-flows <switch>` commands (from Linux's terminal) and try to discover how the `simple_switch_stp` component solves the previous problem.

Q23/ What changes are made to the ports of the switches?

Solution: *The app implements the STP which uses the switch with lowest `dpid` as the root and from there builds a spanning tree. Then, any link that does not belong to the tree is given the command `no-flood`, which logically breaks the loops. Note that in this case, it is the RYU controller that is computing the spanning-tree protocol (STP), as opposed to enterprise-class switches where they perform the STP locally, and only then they try to reach the controller for further configuration.*

Q24/ What is the root of the spanning tree? How many ports of the switches are closed? Verify if the tree is a valid minimum spanning tree. Write the path that flows take to reach $h4$ from $h5$.

Solution: *The root of the spanning tree is $SW1$, you can find it by searching the logs. 2 ports are closed. $SW2--SW3$ and $SW4--SW5$. The path is $h4 \rightarrow SW4 \rightarrow SW2 \rightarrow SW1 \rightarrow SW3 \rightarrow SW5 \rightarrow h5$*