Name 1:
Name 2:

TCP/IP Networking

Lab Exercises (TP) 4
Dynamic Routing (OSPF) and SDN Basics
With Solutions

Jagdish Achara

November, 2018

1 Lab Organization and Instructions

1.1 Lab Organization

In this TP, you will learn how to configure 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 and, with some interesting scenarios, you will see how the routing works in the internet where the network is constantly changing. 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 a Mininet extension, MiniNExT [1]. MiniNExT currently only supports an old version (2.1.0) of Mininet[2]. **Therefore, you need to download[3] and use another virtual disk image for this lab**. Also, for the last lab (lab 6), you will again use the MiniNExT and therefore, we ask you to continue using this new virtual machine for all the labs from now onwards. For how to create a virtual machine from a virtual disk image, please follow the instructions from lab 1. If you want to have shared folder between your host and virtual machine, please set the folder name as "share" and folder path as the path that you want to have shared. Also, select the "Auto-mount" checkbox.

Please download the lab4 folder from the Moodle. Depending on where you put the lab4 folder, you might need to change the paths for clean.sh script and different config files (zebra.conf, ospfd.conf, and ospd6d.conf) of each router.

[1] https://github.com/USC-NSL/miniNExT

[2] A project or thesis is available if you want to work on either incorporating MiniNExT features into Mininet or on developing a MiniNExT version that supports the latest version of Mininet. Please contact jagdish.achara@epfl.ch.

[3] https://drive.google.com/open?id=0ByarHGTFm1KISUozUWpTMDJDVG8

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 November 21 at 23:55.

2 Quagga: 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 the lab 6, we will give you a flavor 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 Quagga, a free routing software suite running on Linux. Quagga provides a control terminal that accepts similar commands to the ones in Cisco's IOS.

2.1 What is Quagga and how does it work?

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

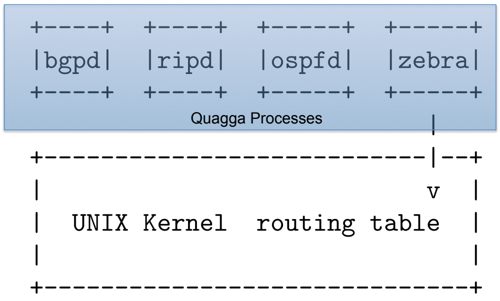

Figure 1: Quagga Architecture

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

- *zebra*: is used to manage the network interfaces of a machine (in our case, each router we 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 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.

As mentioned earlier, we use MiniNExT , a Mininet extension, in this lab. The extension supports mount and PID namespaces[4] and therefore, enables to have a clear separation between the daemons of each host/router running in the virtual machine.

2.2 Starting a Quagga Process

In the VM you downloaded from Moodle, Quagga is configured as a Linux service. To start the service, we first need to configure two files, `daemons` and `debian.conf`, present in the directory `/etc/quagga/`.

In the `daemons` file, you need to specify which processes you would like Quagga service to enable. Note that `zebra` process should always be enabled.

An example `daemons` file, where `zebra` and `ospfd` are enabled, is shown below:

```
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
isisd=no
```

Below is an example `debian.conf` file:

```
vtysh_enable=yes
zebra_options=" --daemon -A 127.0.0.1 -u quagga -g quagga"
bgpd_options=" --daemon -A 127.0.0.1 -u quagga -g quagga"
ospfd_options=" --daemon -A 127.0.0.1 -u quagga -g quagga"
ospf6d_options="--daemon -A ::1 -u quagga -g quagga"
ripd_options=" --daemon -A 127.0.0.1 -u quagga -g quagga"
ripngd_options="--daemon -A ::1 -u quagga -g quagga"
isisd_options=" --daemon -A 127.0.0.1 -u quagga -g quagga"
```

- `vtysh_enable`: Enables the user interface shell called `vtysh`.
- `--daemon`: Makes the process run in the background.
- `-A`: Specifies through which IP address the configuration mode can be accessed. By default only from localhost.
- `-u`: Username to start the process. Default is quagga.
- `-g`: Specify the privilege group for the user. Default is quagga.

Next step is to configure the conf files corresponding to the processes we enabled in the `daemons` file. There are two ways for configuring the Quagga processes: with the `<daemon_name>.conf` file or using "on the fly" configuration. In this lab, we insist on configuring the Quagga processes using the `<daemon_name>.conf` file as it can be better adopted to Mininet's scripting nature.

[4] https://en.wikipedia.org/wiki/Linux_namespaces

Each daemon has its own `<daemon_name>.conf` file that needs to exist in `/etc/quagga/`, even if it is empty, in order for Quagga to work properly. The sample conf files for different processes/daemons can be found in the following directory: `/usr/share/doc/quagga/examples/`.

Once all the `conf` files are properly configured, we start the Quagga service like any other Linux service:

```
/etc/init.d/quagga start
```

You can check the status with

```
/etc/init.d/quagga status
```

In case of above `daemons` and `debian.conf` conf files, you should see both `zebra` and `ospf` daemon running. For more information, you can always consult its documentation online at `https://www.quagga.net/docs.html`.

When "on the fly" (while running) configuration is used, for each process you need to enter the corresponding configuration mode using the command:

```
telnet localhost <process>
```

or for IPv6 routing protocols:

```
telnet ::1 <process>
```

Once logged into the Quagga daemon, you can type `list` to print all possible commands in that mode and can press `?` to list the possible actions available at that instant and use Tab to auto-complete. To close a telnet connection, you can type `quit` or press `Ctrl + C`.

2.3 Killing a Quagga process

Quagga processes can be killed using two methods:

1. **Stop quagga service**: Using the traditional command to stop a process in Linux terminal:

```
/etc/init.d/quagga stop
```

2. **Killing individual processes**: Sometimes you want to simulate a crash on a router, and you require to kill a specific process. To kill the individual Quagga processes that are running on a router, type the following command in the Linux terminal window:

```
killall <process name 1> <process name 2> ...  <process name N>
```

   where `<process name 1> <process name 2> ... <process name N>` are the Quagga processes that are running on the router (*e.g., zebra*, *ospfd*, *ospf6d*, etc). Note that if you want to start the individual process again, you will need to restart the `quagga` service, as this is the only option for starting the Quagga processes.

You can check whether a process is running on a router by observing the list of running processes. To do this, use the following command:

```
ps –A
```

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 the intern Samwell Tarly have designed the communication mechanism, called TCP/IP. Together, they setup routers and switches at strategic locations in Westeros as show 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 show 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.

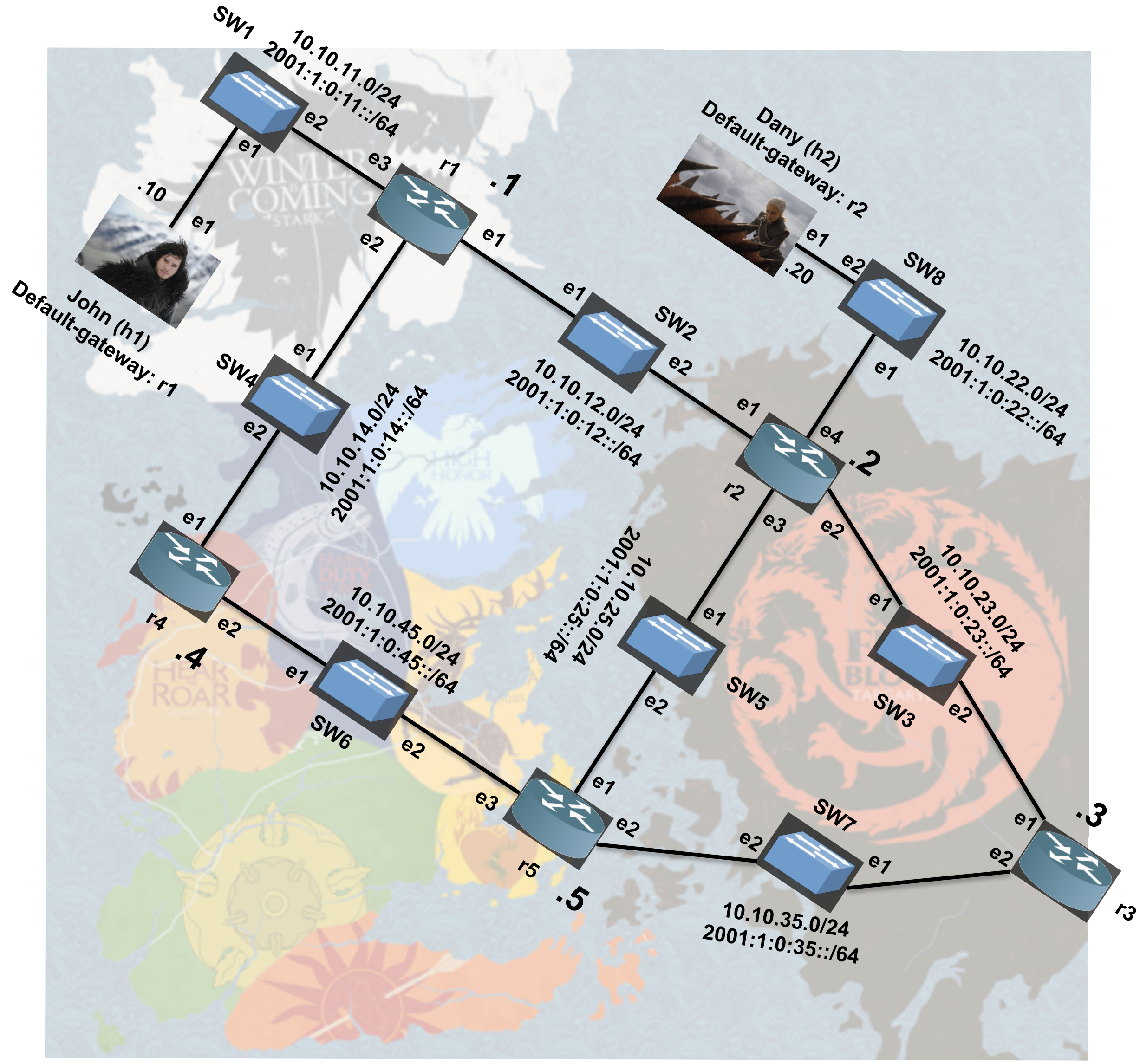

Figure 2: Networking in Westeros

Sam's configuration scripts are available in the lab folder. Run `lab4_network.py` to recreate the established topology. See the output of the `net` command in the MiniNExT prompt and verify whether the connections in the created network are same as in the Figure 2.

QQ1/ What percentage (correct to nearest integer) of the 21 possible combinations of connections (between 2 hosts and 5 routers) in Westeros are functional? **Explain why you can or cannot ping some or all of them.** Use pingall command in the mininext prompt. Hint: Check Python scripts that create the network topology.

Solution: *5% (1 out of 21)*
In our python script (lab4_topo.py), we are only configuring/assigning one IP address per router and host. In case of h1 and r1, we assigned the ip addresses that belong to the same LAN. This is not the case with host h2 and router r2. Router r2 is assigned an ip in some other 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.

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

Each router is, in essence, the same as the physical 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 *Quagga*. An advantage of Quagga is that its commands represent a subset of commands used to configure Cisco networking equipment. Thus, documentation can be found on the Quagga website (*www.nongnu.org/quagga/*), but also, you can find good references on the Cisco website.

4 Host network configuration with Quagga

In principle, you could create a topology without Quagga, 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 the router `r1`. This file can be found among setup files available on Moodle.

Script 1: Example of a zebra.conf file

```
!
! Zebra configuration file for r1
!
hostname r1
enable password quagga

log file /home/lca2/Desktop/shared/lab4/configs/r1/logs/zebra.log
debug zebra packet
debug zebra events

ip forwarding
ipv6 forwarding

interface r1-eth1
no shutdown
ip address 10.10.12.1/24
ipv6 address 2001:1:0:12::1/64

interface r1-eth2
no shutdown
ip address 10.10.14.1/24
ipv6 address 2001:1:0:14::1/64

interface r1-eth3
no shutdown
ip address 10.10.11.1/24
ipv6 address 2001:1:0:11::1/64


line vty
no login
```

Let's examine the content of this configuration file:

- `!`: the lines starting with `!` are comments, they are ignored;
- `enable password`: this will be password used to enable "on the fly" configuration of zebra process;
- `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 and Quagga has permission to write to that file as these issues could prevent the Quagga 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;
- `ipv4 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;
- `line vty`: enables *telnet* access to the process;
- `no login`: disables the password for monitoring mode. The password for configuration mode is set as `quagga` in line 5.

Note: The Python scripts that we provided expect that the `configs` folder we provided is located in the repository where the script (in our case, `lab4_network.py`) resides. If you have a different directory structure, you will need to change the paths in the scripts.

We already created all the configuration scripts for the routers $r1$, $r2$, and $r4$, and you can find them in files we provided. **The creation of the configuration scripts for the routers $r3$ and $r5$ is left to you.** You should now start configuring the interfaces on the routers (assignment of IP addresses) by correctly creating zebra.conf file for $r3$ and $r5$ routers. Before running the Quagga service on each router, make sure that only the zebra process is enabled in `daemons` file of each router (all other processes should be set to "no"). This is because we have not yet configured other config files, e.g., for ospf. Also, make sure the path of `zebra.log` files exist and you have the permissions to read-write to these files.

Once you have finished creating the `zebra.conf` files for routers $r3$ and $r5$, start quagga service on each router. To do so, use the following command from the terminal of that router:

```
/etc/init.d/quagga start
```

In above steps, we have manually started the quagga on each router. Instead you should now set Quagga to start automatically (while creating the topology) on each router. For this, you need to enable `autoStart` flag in the `lab4_topo.py` script. With this setting, next time when you will create the topology by running `lab4_topo.py` script, you will not need to start the Quagga manually on each router.

4.1.1 Quagga Monitoring Mode

In order to monitor the activity of a running Quagga 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
```

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`.

QQ2/ What subnets can be found in the two routing tables on $r1$ at this point?

Solution:

127.0.0.0/8 (lo)
10.10.11.0/24
10.10.12.0/24
10.10.14.0/24

::1/128 (lo)
2001:1:0:11::/64
2001:1:0:12::/64
2001:1:0:14::/64
fe80::/64 (r2-eth1 link local)
fe80::/64 (r2-eth2 link local)
fe80::/64 (r2-eth3 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 `quagga`. 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

In Quagga, 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.conf` configuration files. However, they are available only for the routers $r1$, $r2$, and $r4$. Similar to configuration for zebra, these files should be created in the same folder as the `zebra.conf`. We left the creation of the ospfd.conf configuration files for the routers $r3$ and $r5$ to you. Below, we explain the steps needed to configure OSPF on a router. These steps should allow you to understand the content of the `ospfd.conf` configuration files on the routers $r1$, $r2$, and $r4$, and then, to create similar files yourself for the routers $r3$ and $r5$. Please continue reading to understand how to create these files before actually starting to create them ;-). We give an example of `ospfd.conf` configuration file in Script 2.

Script 2: Example of an ospfd.conf file

```
!
! OSPF configuration file for r1
!
hostname r1
enable password quagga

log file /home/lca2/Desktop/shared/lab4/configs/r1/logs/ospfd.log
!
debug ospf event
debug ospf packet all
!
router ospf
```

```
!
network 10.10.11.0/24 area 0
network 10.10.14.0/24 area 0
network 10.10.12.0/24 area 0
!
line vty
no login
```

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 in Figure 2, **create the *ospfd* configuration files for the routers** $r3$ **and** $r5$ (configuration files for $r1$, $r2$, and $r4$ are available in the lab folder). Make sure that both debugging modes are enabled for OSPF (*i.e.,* `debug ospf event` and `debug ospf packet all`). In this section, you must have `zebra` running on all routers and `ospfd` running on all routers except $r4$.

Now, stop the `quagga` service on each router, if it is running, and clear all the logs. For clearing the logs and restarting Quagga, you can use the script `clean.sh` available for each router. However, depending on where your files are located, you may need to change the path in the `clean.sh` file of each router.

QQ3/ Open wireshark on router $r1$. 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 *configs/logs/ospfd.log* file on router $r1$ 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.*
`2017/10/13 21:45:09 OSPF: Hello sent to [224.0.0.5] via [r1-eth2:10.10.14.1].`

QQ4/ Check the *configs/logs/ospfd.log* file on router $r1$. How does $r1$ identify $r2$?

Solution: *Through its OSPF ROUTER-ID: 10.10.25.2 using the hello protocol. (This answer could be different for different students, but it has to be one of the IP addresses of* $r2$*.)*

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 $r2$. For this, you can enter to Quagga monitoring mode of the `ospfd` daemon. This is done in the same way as in the case of *zebra* process (*i.e.,* `telnet localhost ospfd`). After that, you can display the content of the OSPF database using the command:

```
show ip ospf neighbor
```

QQQ5/ How many OSPF neighbors does $r2$ have? What are their ids? Identify which of those are designated routers (DRs) and which ones are Backup designated routers (BDRs)?

Solution: *Three neighbours.*
10.10.14.1
10.10.35.3
10.10.45.5 Depends. For me, all of them were backups.

QQQ6/ What does the column Dead Time represent? 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 form the database. Hello timer is the time after which the 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
```

QQQ7/ What are the network LSAs advertised by $r2$? How is this related to the answer to question 5?

Solution: *As $r2$ is the DR for each of the three networks it is a part of, it advertises the network links of all. This answer depends on Q5. Basically, $r2$ advertises the network LSAs for which it is a DR.*

QQQ8/ 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, 10.10.12.0, 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 using Quagga.*

QQ9/ How many network LSAs and router LSAs are present in the OSPF database of $r2$? 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.

Dany wants to see how far she is from her ancestral home of Valyria. Help her by doing a traceroute from $h2$ to the IP address of Valyria, 10.10.35.5.

QQ10/ Write the output of the second hop. Do you see something unusual? Explain.
Hint: To see the routing tables, you can use `show ip ospf route` in the monitoring mode of ospfd.

Solution: *traceroute to 10.10.35.5 (10.10.35.5), 64 hops max*
1 10.10.12.2 1.413ms 0.418ms 0.415ms
2 10.10.35.5 1.561ms 10.10.23.3 1.012ms 10.10.35.5 0.481ms
The packets take one of the two paths: either through $r3$ or $r5$. This is because of equal-cost multi-path routing in OSPF.

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, Quagga 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. Again, the *zebra* process must be enabled along with the *ospf6d* process in the `daemons` configuration file.

As in the case of `ospfd`, we are providing the *ospf6d* configuration files for the routers $r1$, $r2$ and $r4$. The configuration file of `ospf6d` is a little different from that of `ospf`. Below, we explain the file `ospf6d.conf` on $r1$.

Script 3: Example of an ospf6d.conf file

```
!
! OSPF6D configuration file for r1
!
hostname r1
enable password quagga

log file /home/lca2/Desktop/shared/lab4/configs/r1/logs/ospf6d.log
!
! Interface setup
!
interface r1-eth1
 ipv6 ospf6 instance-id 1

interface r1-eth2
 ipv6 ospf6 instance-id 1
!
```

```
! Router setup
!
router ospf6

interface r1-eth1 area 0.0.0.0
 area 0.0.0.0 range 2001:1:0:12::/64

interface r1-eth2 area 0.0.0.0
 area 0.0.0.0 range 2001:1:0:14::/64

interface r1-eth3 area 0.0.0.0
 area 0.0.0.0 range 2001:1:0:11::/64
!
line vty
no login
```

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.
- `ipv6 ospf6 instance-id <1-255>`: Runs an instance of ospf6d with the given instance-id. The instance-ids for different interfaces may be same.
- `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 $r3$ and $r5$ are left for you to create. You should make sure that you log the updates of *ospf6d* in a separate file than the *ospfd* events/packets.

In this section, you must have `zebra` running on all routers, and `ospfd` and `ospf6d` running on all routers except $r4$.

4.3.1 Monitoring commands

To inspect the `ospf6d` database, you should execute `telnet ::1 ospf6d` command. 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.

QQQ11/ Check the SPF tree of $r5$. In `ospf6d`, what are the costs for a packet to go from a network to a router and a router to a network?

Solution: *Network to router: 0*
Router to network: 1.

5 OSPF Playground

In this section you will be confronted with a number of situations that might arise in a 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 Quagga process commands and one for Linux commands. To have a second `xterm` window for `xterm r1`, you can type: `xterm r1` in the `mininext>` prompt.

5.1 Understanding neighbors in OSPF

In this section, we will learn about how neighbors interact within OSPFv2 (`ospfd`). You must have `zebra` running on all routers and `ospfd` and `ospf6d` running on all routers except $r4$. Now, restart the Mininet network using `python lab4_network.py` command. The next couple of questions will require the comparison of the OSPF database on $r1$. Take a snapshot of the database for your reference. Specifically, `show ip ospf database`, `show ip ospf database router`, and `show ip ospf network`. Now, we will see how OSPF routers are synchronised. Recall that OSPF is not yet enabled on $r4$. In the file `daemons` at $r4$, now set `ospfd` and `ospf6d` to yes. Then, restart the Quagga service on $r4$ using script `clean.sh`. This will make $r4$ an OSPF router. Let this time be $t1$.

QQQ12/ Does $r4$ appear in the neighbor list of $r1$? What's $r4$'s neighbor state at $r1$? Look at the `osfpd.log` of $r1$ and identify the line that indicates the change of state from X to current state. What is X? What are the states in the OSPF neighbor finite state machine?

Solution: *Yes, with state Full.*
nsm_change_state(10.10.45.4, Loading $->$ Full): scheduling new router-LSA origination
X is Loading.
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/

QQQ13/ What changes do you notice in the OSPF database at $r1$? 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 1. Each additional "hop" (router) adds 1 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 $r5$), being the paranoid queen, wants to know if Dany is talking to John behind her back. So, she orders that the (one way) IPv6 traffic from $h2$ to $h1$ to go via $r5$ instead of the direct route. To fulfill this task, you need to modify the cost of the link between $r1$ and $r2$ so that the total cost of routing through $r2 \rightarrow r5 \rightarrow r4 \rightarrow r1$ 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
```

QQQ14/ Describe the minimal set of changes to the configuration of router $r1$, router $r2$, or both of them, to achieve the desired affect. Write the modified configuration file(s).

Solution: *We only need to make one change to* $r2$*.*
At line 13, for interface $r2 - eth1$ *add* `ipv6 ospf6 cost 5`*.*

5.3 Broken link

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

```
telnet localhost zebra
enable
configure terminal
interface r2-eth1
shutdown
```

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

QQQ15/ Observe the changes in the OSPF database at $r1$. 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* $r2$*, it sends a new LSA which is updated in the router LSA section of* $r1$*. Moreover, the network LSA for* $r2$ *is also removed from the database of* $r1$ *after 40-60s (Wait timer).*

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

QQQ16/ Can you ping the IPv4 address of $r2 - eth1$? Explain.

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

QQQ17/ 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 $r2$, could you then ping the IPv4 address of $r2 - eth1$. Explain.

Solution: *In the latter, yes, we can ping because the connectivity is independent of OSPF.*

QQQ18/ In the case of not breaking the link and simply commenting out `network 10.10.12.0/24` from $r2$, 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

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. Note that in this section, we will work only with Mininet (and not MiniNExT) as we will apply flow-policy in switches (not in routers). SDN routing policies are only available with proprietary hardware and controller (Cisco, Huawei, Juniper, etc.)

The Mininet virtual environment used in this lab is compatible with SDN. In fact, the openVSwitch (OVS), which is the switch we have been used so far through the whole 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 [5], which is an open-source controller written in Python. Unzip the file `ryu.zip` where you will find the source files of RYU. 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`, and 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 in Moodle) has been created for you to match the topology described. Note that in this section we use Mininet instead of MiniNExT as we don't have a requirement for PID namespaces.

QQQ19/ 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 with a similar utility called `dpctl`, which has a short version of the `ovs-ofctl` utility. Unlike the former, `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 $h1$ try to make three pings to $h2$:

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

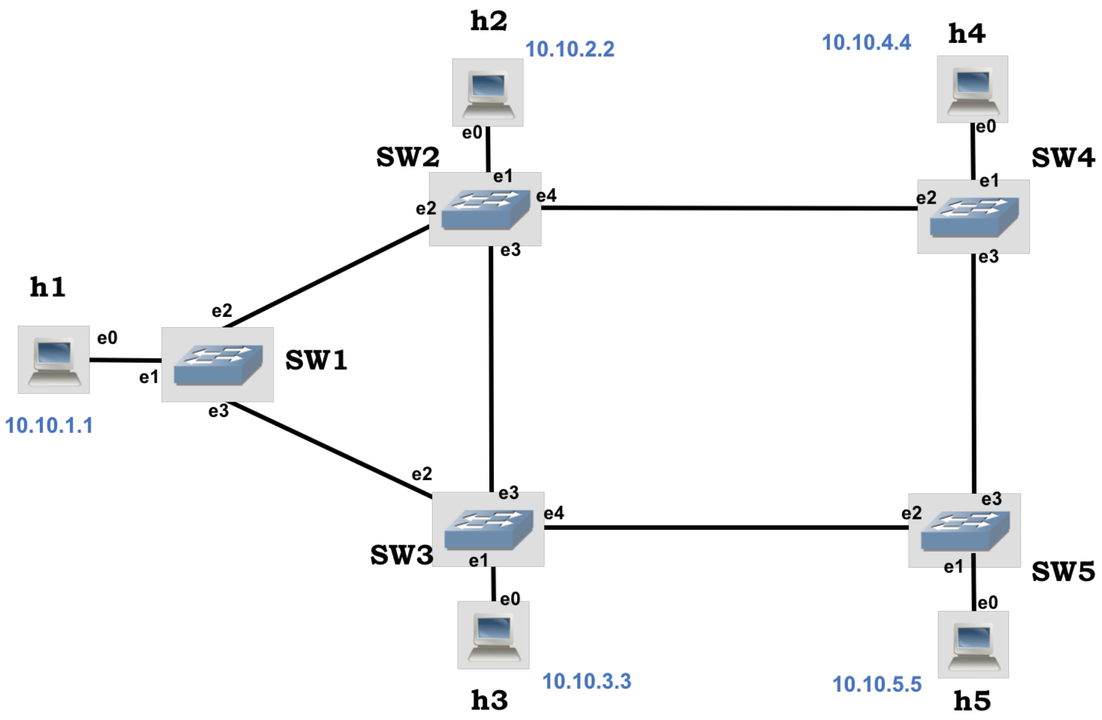

Figure 3: Lab 4 SDN topology

```
mininet>h1 ping -c 3 h2
```

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 $h1$ and $h2$ communicate. From a new terminal prompt (from Linux, not Mininet), type the following commands:

```
sudo ovs-ofctl add-flow SW1 in_port=1,actions=output:2
sudo ovs-ofctl add-flow SW2 in_port=2,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:2`: 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.

QQ20/ 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 $h2$, we see ARP request arriving from $h1$ and we see that $h2$ replies to these packets. In $h1$ 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 SW2 in_port=1,actions=output:2
sudo ovs-ofctl add-flow SW1 in_port=2,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 $SW1$ and $SW2$ 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 ryu.app.ofctl_rest
```

The above command runs the RYU controller with the `simple_switch` applications. 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/ryu/ryu/app` folder.

For all cases, start the RYU controller first, and then start Mininet. 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 `h1 ping -c 1 h2` command.

QQQ21/ 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` 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 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.

QQQ22/ 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.*

QQQ23/ What is the root of the spanning tree? How many ports of the switches are closed? Verify if the tree is a valid mininmum spanning tree. Write the path that flows take to reach $h3$ from $h2$.

Solution: *The root of the spanning tree is $SW1$. 2 ports are closed. $SW2 - -SW3$ and $SW4 - -SW5$. The path is $h2 \rightarrow SW2 \rightarrow SW1 \rightarrow SW3 \rightarrow h3$*

Now, use the command `link SW1 SW2 down` from the `mininet>` prompt to bring down the link between SW1 and SW2. Wait for 1 minute.

QQQ24/ Enumerate all the changes to the flow tables of the switches and the spannig tree. What is the new path from h2 to h3?

Solution: *The link is detected as broken a new STP computation is triggered. $SW2$ is now the root. The new path is form $h2 \rightarrow SW2 \rightarrow SW3 \rightarrow h3$.*