Name 1: THIERCELIN Marin
Name 2: CADOUX Cyril

TCP/IP NETWORKING

LAB EXERCISES (TP) 4
DYNAMIC ROUTING AND SDN BASICS

Maaz Mohiuddin

November 12, 2017

1 LAB ORGANIZATION AND INSTRUCTIONS

1.1 LAB ORGANIZATION

In this TP you will learn how to configure "link-state" routing protocols that typically run inside the network of an autonomous system (AS). The protocols automatically set up network routes that ensure shortest path between two points in the network with respect to a predefined metric (such as number of hops). During the lab you will configure a fully functional network using Cisco-like routers. You will learn how to configure a network of routers and through some interesting scenarios, see how the routing works in the internet where the network is constantly changing.

You will also learn about software-defined networking (SDN) and study how it can be used to create forwarding rules on switches. The part on SDN is left as a bonus exercise. However, you are strongly advised to do this part as it gives you a good idea of how enterpise networks are managed today.

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 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. That 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 22 at 23:55.

2 QUAGGA: SOFTWARE ROUTING SUITE

```
BOFH excuse #427:
network down, IP packets
delivered via UPS
```

BOFH

The Internet core is run by powerful routers that can handle large amounts of traffic built by companies such as Cisco, Huawei or Juniper. Even in a large company, or on 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 the following labs we will give a flavor of router configuration.

Ideally, we would like to run IOS in a virtual environment. While this is technically possible, it is not very legal, as Cisco or Juniper do not allow their OS to be run on a device other than their routers.

Instead we will use Quagga, which is 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 WORKS?

Quagga is a routing software suite that provides implementations of several routing protocols (namely OSPF, RIP and BGP-4) for Unix platforms. It consists of a handful of processes that can be run in the background as daemons. Four 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). 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.
- *quagga*: The main service, which is used to call the three daemons above.

The architecture of Quagga is as shown in Figure 1. Quagga runs one instance of each of these daemons per host. To have a clear separation between the daemons of each host, we will use MiniNExT, a Mininet extension that supports PID namespaces [1].

```
+----+  +----+  +-----+  +-----+
|bgpd|  |ripd|  |ospfd|  |zebra|
+----+  +----+  +-----+  +-----+
         Quagga Processes     |
+-------------------------|--+
|                         v  |
|  UNIX Kernel  routing table |
|                            |
+----------------------------+
```

Figure 1: Quagga Architecture

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

2.2 Starting a Quagga Process

Each daemon has its own `<daemon_name>.conf` file that needs to exist, even if it is empty, in order for Quagga to work properly. In the TCPIP-VM, Quagga is configured as a Linux service. To start the service, we need to configure two files: `daemons` and `debian.conf`. In the `daemons` file you should see something like this:

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

where you need to specify which processes would you like Quagga to enable. Note that `zebra` process should always be enabled.

In the `debian.conf` file, you should see something like:

```
vtysh_enable=no
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"
```

where:

- `--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

Once each `conf` file is properly configured, we need to start the Quagga service like any other Linux service:

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

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 first method as it can be better adopted to Mininet's scripting nature. 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
```

ENVIRONMENT VARIABLES

In this lab, you'll need to navigate to the config folders of each router. Instead of tediously typing `cd /home/lca2/Desktop/shared/r1/...`, you use environment variables in Linux such as `cd` $\$r1$. To create a variable, open the file `~/.bashrc` as the super use using the command

```
sudo leafpad ~/.bashrc
```

At the end of this file, add the following line.

```
export r1="absolute path to the folder r1"
```

The path needs to be absolute. For example, /home/lca2/Desktop/shared/configs/r1. You can add one variable for each router. Once you save and close the file, type `source ~/.bashrc` for the variables to take effect.

3 Scenario: A Game of Routers

Valar Morghulis!

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 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 Tarly Communication Process by Illyrio Pycell, TCP/IP in short. 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.

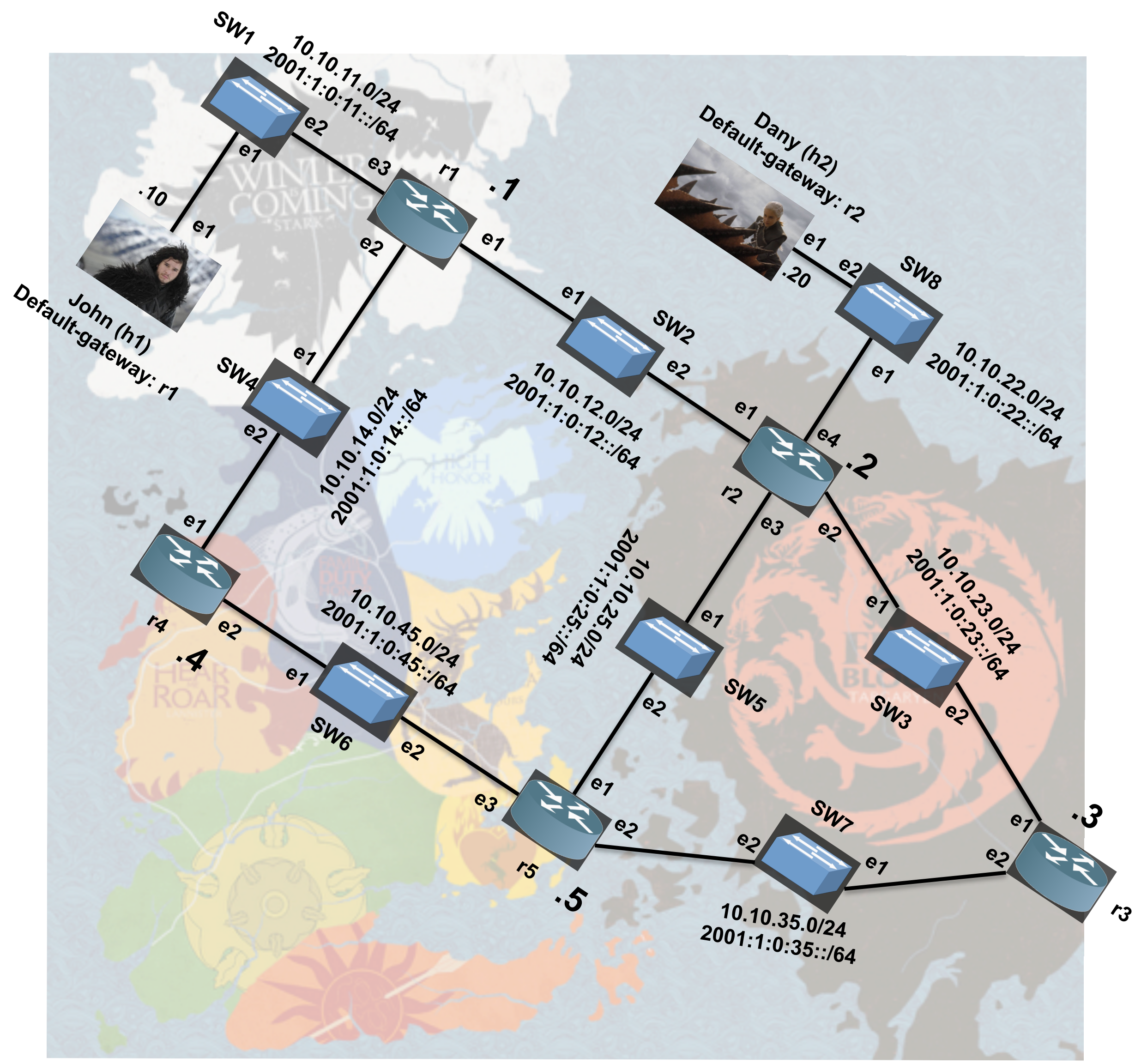

Figure 2: Networking in Westeros

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 42 possible connection in Westeros are functional?. Explain why you can or cannot.

[A1] With a pingall, we see that 2 connections (between h1 and r1) are functional. This represents roughly 5% of the possible connections.

Your job is to help Sam with configure 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, 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 good references can also be found 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;
- `password`: this will be password used to log in to the zebra process to view the changes
- `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 as this 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;
- `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.

Note: The scripts that we provided expect that the `configs` folder we provided is located at `/home/lca2/Desktop/shared/lab4/configs/`. 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). Thus, on all your `daemons.conf` file, make sure that only the zebra process is enabled, all other processes should be set to "no".

Note: Make sure that the permissions for the `zebra.log` files allow you to read-write them. To make a script read-write for anyone, you can type the following command in the terminal window (running with root privileges):

```
chmod 666 <script name>
```

Start quagga service on each router. To do so, use the following command from the terminal of that router:

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

You may also set Quagga to start automatically in the `lab4_topo.py` script from now onwards.

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`.

Q2/ What subnets can be found in the two routing tables on $r1$ at this point?

[A2] We can see these subnets for IPv4:
10.10.11.0/24
10.10.12.0/24
10.10.14.0/24

We can see these subnets for IPv6:
2001:1:0:11::/64
2001:1:0:12::/64
2001:1:0:14::/64

Next, check the status of the network interfaces on router $r3$ by typing `show interface`.

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 same as the one you used.

4.1.2 "On the fly" configuration

Let us now consider how a running configuration can be modified "on the fly", without changing the configuration files. Remember that this is not the recommended approach. Here we will consider an example where "on the fly" configuration could be useful.

You might have noticed that the *zebra* configuration file for the router $r4$ has one line commented out, *i.e.,* the interface `r4-eth1` is not assigned an IPv4 address. You could of course stop the *quagga* service on this router, modify the line in the configuration file and restart the service. However, we will fix this problem without stopping the service. For this purpose, we have to enter Quagga configuration mode on the router $r4$ and enter the *interface configuration mode*.

To enter the *interface configuration mode* follow the steps below:

- First, enter the zebra monitoring mode using command `telnet localhost zebra`.
- Next, enable the configuration mode by using `enable`.
- Next, move to the *configuration mode* by typing the command `configure terminal`. The cursor remains the same, but the name before the cursor changes to `r4(config)`.
- Finally, in the *configuration mode* type `interface r4-eth1`. The name before the cursor changes to `r4(config-if)`, which indicates that you entered the *interface configuration mode*.

Now, you can assign an IPv4 address to the interface `r4-eth1`, by using the same command that is used in the configuration file for this purpose. Simply type `ip address 10.10.14.4/24`. Exit the *interface configuration mode* and the *configuration mode*, by typing `exit` twice and verify that the modification took effect with the `show interface` command.

Important: Before leaving this section, make sure to change the `zebra.conf` file of $r4$, to remove the comment on interface `r4-eth1` as "on the fly" configuration does not save changes to the configuration file.

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", as illustrated in Section 4.1.2. Note that the `zebra` process must be enabled with the `ospfd` process in the `daemons` configuration file, as they work together.

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 to create similar files for the routers $r3$ and $r5$.

The inevitable steps when configuring the OSPF protocol on a router are:

- enabling OSPF;
- choosing which networks should be advertised via OSPF;
- specifying the interfaces that take part in the exchange of routing information.

Some optional steps include debugging, logging, route filtering, etc. 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 events`: 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`: You will learn about this command in Section 5.1

- `redistribute`: enables the announcement of prefixes learned from a given source. The syntax is:

```
redistribute protocol
```

where `protocol` denotes the source from which the prefixes have been learned. It can be:

(1) a routing protocol: such as `bgp` or `rip` (*e.g.,* if you want to redistribute the routes from these protocols into OSPF),
(2) `static`: if you want to redistribute the static routes
(3) `connected`: if you want to announce only the prefixes the router learned from the configuration of its own interfaces (*i.e.,* redistribution of the directly connected networks).

- `no login` disables the password for monitoring mode. The password for configuration mode is set as `quagga` in line 5.

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 events` and `debug ospf packet all`). In this section, you must have `zebra` running on all routers and `ospfd` running on all routers except $r4$.

In the `zebra.conf` of $r4$, be sure to uncomment the line concerning the configuration of interface `r4-eth1`. Then, stop `quagga` on all routers and clear all logs. For this you can use the script `clean.sh` available for 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..

The routing information are exchanged by using multicast, since the destination address of the Hello messages are 224.0.0.5.

This is verified by the line :
2017/11/10 14:31:40 OSPF: interface 10.10.11.1 [179] join AllSPFRouters Multicast group.

[A3]

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

r1 receives a Hello from r2.

[A4]

4.2.1 MONITORING COMMANDS

We have seen before that we can use command `show ip route` 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 neighbors 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
```

QQ5/ 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)?

r2 has 3 neighbors :
10.10.14.1 which is BDR
10.10.35.3 which is DR
10.10.45.5 which is DR

[A5]

QQ6/ What does the column Dead Time represent? How is it related to the hello timer? Can you infer the values of the dead timer and the hello timer?

This is a timer that indicates the duration of validity of the last received information.
We can see that the Dead Timer is roughly 40 seconds and the Hello Timer is roughly 10 seconds.

[A6]

Next, we will look at the LSAs in the database of routers. For this, use 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
```

QQ7/ What are the network LSAs advertised by $r2$? How is this related to the answer to question 5?

[A7]

LS age : 273
Options : 0x2 : *|-|-|-|-|-|E|*
LS Flags : 0x3
LS Type : network-LSA
Link State ID : 10.10.12.2 (address of Designated Router)
Advertising router : 10.10.25.2
LS Seq Number : 80000001
Checksum : 0xefc6
Length : 32

QQ8/ What are the prefixes that $r1$ advertises to $r2$ in its router LSA? What is the difference between them? Explain.

[A8]

r1 advertises :
10.10.12.0/24
10.10.14.0/24

The first 2 are Transit Networks whereas the last one is a Stub Network

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

[A9]

There are 4 network LSAs and 4 router LSAs.

The 4 routers LSAs come from the routers 1,2,3,5 that are engaged in ospf

Since there is a network LSA for each transit ethernet LAN, we obtain 4 network LSAs (12,25, 53, 23)

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 in a router, you can either use `ip route` or `show ip ospf route` in the monitoring mode of ospfd.

2 10.10.23.3 2.631ms 10.10.35.5 2.683ms 10.10.23.3 2.038ms

We see that the route goes through router 3 even if it would be shorter in terms of hop count to go directly to router 5 via the 25 LAN

[A10]

QQ11/ Does `ospfd` perform per-flow load-balancing?

Entering the previous traceroute command several times, we see that the packets go either through router 3 and 5 with equal probability. Thus we can say that ospfd does not perform per-flow load balancing.

[A11]

John wants to ask Dany if he can ride one of her dragons, Viserion. So, he sends her a long message explaining how he is a capable rider and will return Viserion as soon as possible. Sadly, he does not get a reply. He is wondering if Dany did not receive his message or she actually ignored the request. Can you help him?

QQ12/ Why does John ($h1$) not receive a request from Dany ($h2$)? How can you fix it? What is the average round-trip time between them?

John receives no request from Dany since R1 is not advertising the LAN between h1 and R1.

We could fix it by decommenting the line in r1/ospfd.conf allowing r1 to advertise this LAN.
The average roundtrip time is 1.112 ms

[A12]

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 OSDF 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 *ospfd* 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 an 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 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.

QQQ13/ 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?

The cost to go from a router to a network is 1.

The cost to go from a network to a router is 0 (we can see that reaching a network after a router has the same cost).

[A13]

If you notice the routing table at $r5$, you will see that it has two routes to the network 2001:1:0:12::/64. One is labeled intra-area and another one is external. This is because $r1$ is advertises the connected networks twice, once as a nework LSA and once again as an external LSA. This behavior is in fact a bug in the OSPFv3 implementation in Quagga, which was also present at an earlier time in Cisco IOS [2]. Fortunately, it has been fixed in the Cisco IOS version, but not in the open-source version. This is a typical example of the practical difficulty in networking which arises due to half-baked implementations, specially in IPv6 networking stack.

[2] https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/18722-redist-conn.html?referring_site=RE&pos=1&page=https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/113339-ospf-connected-net.html

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`, `ospf6d` and dynamic routing in general.

Notes: It might be a good idea to use two terminals, one for the Quagga process commands and one for Linux commands. To have a second `xterm` window for a particular router, you can type: `xterm h1` in the `mininext>` prompt.

5.1 Understanding neighbors in OSPF

In this section we will learn about how do neighbors interact within OSPFv2 (`ospfd`). Make sure the following configurations are done properly.

1. In `ospfd.conf` at $r1$, uncomment the line `network 10.10.11.0/24`.
2. In `zebra.conf` at $r4$, uncomment the IPv4 configuration of `r4-eth1`.
3. In `lab4_topo.py`, set Quagga to start automatically.

In this section, you must have `zebra` running on all routers, `ospfd` and `ospf5d` running on all routers except $r4$. OSPF daemons on $r4$ will be started after Q15.

Now, restart the Mininet network using `python lab4_network.py`.

QQ14/ Can $h1$ be able to update its routing information with $r1$'s LSAs?. Explain your answer

[A14] The only information that h1 needs is its default gateway router. Thus there is no reason for h1 to update its routing information.

Let us look at the routes in $r1$. For this, enter the monitoring mode of `zebra` on $r1$ and type `show ip route`.

QQ15/ What is the metric for networks 10.10.35.0/24 and 10.10.45.0/24?

[A15] For 10.10.35.0/24 : [110/30]
For 10.10.45.0/24 : [110/30]

The next couple of questions will require 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 synchronized. Recall that OSPF is not yet enabled on $r4$. In the file `daemons` at $r4$, 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 $t1$.

Q16/ 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 is the total number of states in the OSPF neighbor finite state machine?

R4 appears in the neighbor list of r1. R4 is BDR.

The state of R4 changes from Init to Loading to Full

2017/11/14 11:37:33 OSPF: nsm_change_state(10.10.45.4, Loading -> Full): scheduling new router-LSA origination

There are 8 states in the OSPF neighbor finite state machine.

[A16]

Q17/ What changes do you notice in the OSPF database at $r1$? Explain the reason for each of the changes.

There is one more router LSA (the one corresponding to R4)
There are 2 more network LSAs (for connections 14 and 54)

[A17]

Q18/ In the `zebra.log` file at $r1$, how many ZEBRA_IPV4_ROUTE_ADD messages you see after Time $t1$? Explain.
Hint: Check the routes in the `zebra` daemon.

There are 7 messages ZEBRA_IPV4_ROUTE_ADD

Demander indice aux assistants

[A18]

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 that she is, 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
```

QQ19/ Describe the minimal set of changes to the configuration of routers $r1$ and $r2$ needed to achieve the desired affect. Write the modified configuration file(s).

[A19]

We have to set the cost of the link '12' to 5.

With that change, the direct route from Dany to John costs 7, whereas the redirected route through King's Landing only costs 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.

QQ20/ Observe the changes in the OSPF database at $r1$. Explain what happens.

In the new database, the advertising router for the link 12 is now R1 (it was R2 before the dragon destroyed it).

Since R1 was the backup designated router and since R2 is now no more connected to the link 12, R1 becomes the designated advertising router for this link.

[A20]

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

QQ21/ Can you ping the IPv4 address of $r2 - eth1$? Explain.

No because this interface is not connected to anything.

[A21]

QQ22/ 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.

It is now possible to ping this interface. Even if R2 does advertise the fact that it is connected to this subnet, R1 does it. So the packets of the ping request can follow the ospf route through R1 and arrive à R2.

[A22]

QQ23/ 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 $r4$ to $h2$ and $r5$ to $h2$ take?

The OSPF protocol can see the connection between R1 and R2 because R1 advertises.Moreover the OSPF sees the LAN 12 as a stub network.

The packets from r4 to h2 go through r5 and r2
The packets from r5 to h2 go through r2

[A23]

5.4 Security in OSPF

We noticed in the output of `show ip ospf` that the configuration of the `ospfd` process had no authentication enabled. As a result, a rogue process can inject fake OSPF packets and alter the routing protocol. In order to avoid this, OSPF allows for authentication.

QQ24/ What are the authentication options provided by `ospfd` of Quagga? Are they secure?

Quagga provides an MD5 authentication which is obsolete. It is not very secure.

[A24]

QQ25/ As `ospfd` uses symmetric keys per link, a network with 500 links would require the administrator to generate and manage 500 keys. This is cumbersome to achieve manually. Propose a method to manage the keys on all routers.

We could use a central authority that provides the keys with anyone that wants to connect to the subnet.

If the client is trusted, the key could be distributed and then broadcasted to other routers.

[A25]

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 [3], which is an open-source controller written in Python. Unzip the file `ryu.zip` where you will find the csource files of RYU. Run the script `install_ryu.sh` as root on your virtual machine to install RYU. You can check if RYU is property 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.

QQ26/ How many subnets are there in Figure 3?. What would be the correct network mask for each IP address?

[A26]

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.

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

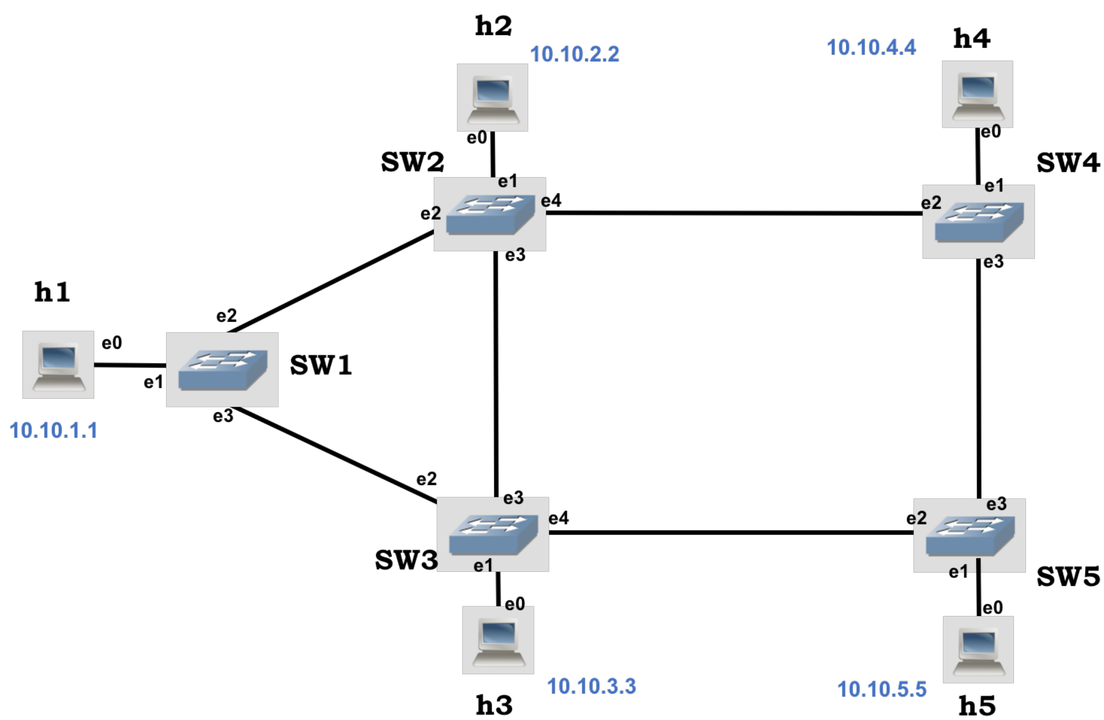

Figure 3: Lab 4 SDN topology

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 $h2$:

```
mininet>h1 ping -c 1 h2
```

As you probably thought, the pings were 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 $h2$ to 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.

QQ27/ 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

[A27]

6.2 Controlling the OpenFlow Switch using RYU

Now it is time 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 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.

QQ28/ Why do you think ping was unsuccessful?

[A28]

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.

QQ29/ What changes are made to the ports of the switches?

[A29]

QQ30/ 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 $h2$.

[A30]

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.

QQ31/ Enumerate all the changes to the flow tables of the switches and the spannig tree. What is the new path from h2 to h3?

[A31]