Name 1:
Name 2:

TCP/IP NETWORKING

LAB EXERCISES (TP) 4
DYNAMIC ROUTING

November 2, 2015

Abstract

In this TP you will learn how to configure "distance-vector" routing protocols that typically run inside the network of an autonomous system (AS). The protocols set up automatically network routes that ensure shortest path between two points in the network with respect to a predefined metric (such as number of hops).

1 TP ORGANIZATION AND INSTRUCTIONS

1.1 TP ORGANIZATION

In this lab, you will be designing a fully functional network using Cisco-like routers and switches. 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.

1.2 TP REPORT

Type your answers in this document. We recommend you use Adobe Reader XI 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 TP 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 18 at 23:59.

2 SETUP AND ENVIRONMENT

```
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 and 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) 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 flavour of router configuration.

Ideally, we would like to run IOS in a virtual machine in GNS3. While this is technically possible, it is not very legal, as Cisco does not allow its OS to be run on a device other than a Cisco router.

Instead we will use free router software, Quagga, running on SliTaz Linux. Quagga provides a control terminal that accepts similar commands to the ones in Cisco's IOS. Quagga is already installed in your SliTaz image. You can reuse the same virtual machines for the this lab. However, we will use the symbol of a router instead of a PC and call them by the name of "virtual router" or simply "router".

2.1 SETUP TESTING

In GNS3, change the settings of the virtual routers so that each router has 3 interfaces and uses 384 MB of memory. Furthermore, for each router, unselect `Reserve first NIC for VirtualBox NAT to host OS`. Then, create a configuration with six ethernet switches and five virtual routers as shown on Figure 1.

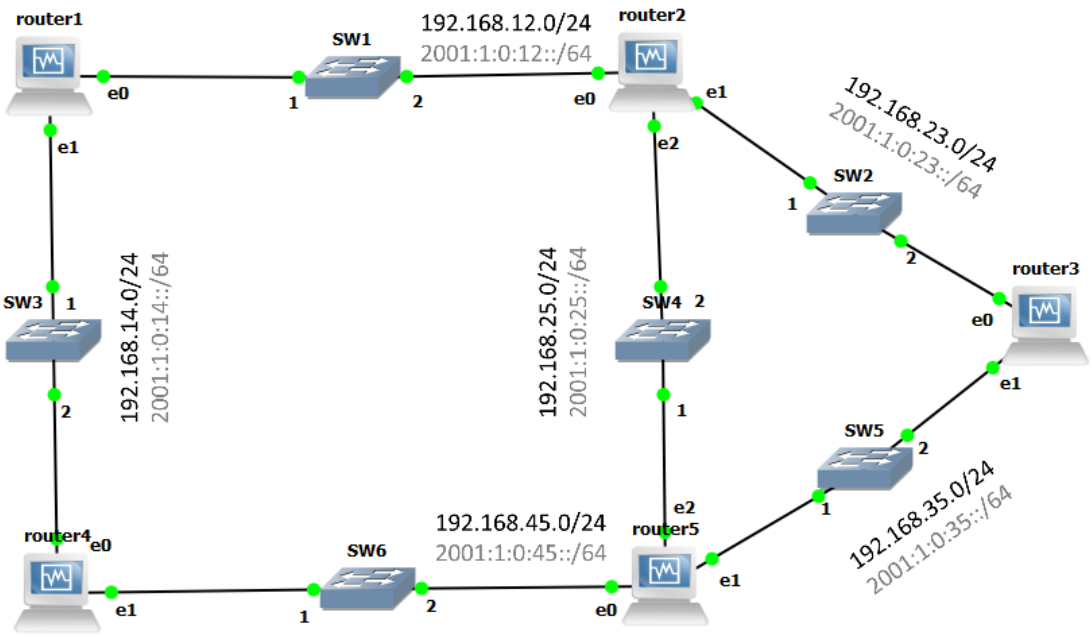

Figure 1: Lab topology

Ensure that all the interfaces on all routers do not have an IP address. The IP address in this picture are only shown for you reference. We will use Quagga to configure the routers in this lab. Also, enable IP forwarding

for both IPv4 and IPv6 in all routers with following commands.

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo sysctl -w net.ipv6.conf.all.forwarding=1
```

You will have to enter these commands in each time you restart a router. For you convenience, we have put these in the file `enableIPForward.sh` which is available on the moodle. To be able to run a script, you have to first own the required privileges by

```
chmod 755 enableIPForward.sh
```

3 HOST NETWORK CONFIGURATION WITH QUAGGA

The virtual routers can be configured in pretty much the same way as the physical Linux machines that you worked with. 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 this set of Linux networking commands, in this lab, you will be using the software suite called *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.

3.1 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 deamons. Three Quagga processes (deamons) are important for executing this lab:

- *zebra*: is used to manage the network interfaces of a machine (physical or virtual). 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. If you want, *zebra* is in a way a replacement for the Linux networking commands used during the first three labs (*i.e.,* `ifconfig`, `route`, etc.).
- *ripd*: handles RIP routing.
- *ripngd*: handles RIP routing for IPv6 (*RIP new generation*).

3.1.1 STARTING A QUAGGA PROCESS

Here we give a short reference on two ways of router configuration. You can treat this as a reminder to which you can come back later if needed.

Each Quagga process can be launched and run independently from the others. The syntax for running any of the Quagga processes is the same:

```
<process name> -d -f <configuration file>
```

where:

- `<process name>` is the name of the process you want to run (zebra, ripd or ripngd);
- `-d` allows to run the process in the daemon mode (*i.e.,* in the background);
- `<configuration file>` is the name of the configuration file. They should be created in the /usr/local/quagga directory where you can find empty configuration files.

In this lab, we insist on configuring the Quagga processes by editing the configuration files and by running the processes using these files. In principle, the processes can also be configured "on the fly" (while running) by entering the Quagga configuration mode using the command

```
configure terminal
```

If you want to go back from Quagga configuration mode you should type

```
exit
```

The first method is often preferred, as it offers a clear view of the active configuration of your router. Thus, we will predominantly use this method.

Although not recommended, changing the configuration of a running process "on the fly" is sometimes used to introduce minor changes to the running configuration (without stopping the process). The `show` commands (explained in Section 3.2.2) let you verify that the changes took effect. As modifying the running configuration without stopping the processes will be desirable at some stages of this lab, in Section 3.2.3 we explain how the configuration of a virtual host (router) can be modified "on the fly".

3.1.2 KILLING A QUAGGA PROCESS

Sometimes you want to kill a Quagga process that you have previously started. For example, you might want to modify the *zebra* configuration file and start the *zebra* process again. To kill the Quagga processes that are running on a host, type the following command in the 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 host (*e.g., zebra*, *ripd*, *ripngd*, etc). You should always kill a process before starting it again. Never start a process that is already running. You can check whether a process is running on a host, by observing the list of running processes. To do this, use the following command:

```
ps -A
```

3.2 CONFIGURING NETWORK INTERFACES

It is time to try out in practice some of the concepts introduced in Section 3.1. For this purpose, we will be using the scenario from Figure 1. In principle, you could open the terminal window on the virtual router R1 and configure its network interfaces using the `ip addr` command (following the scheme shown in Figure 1). However, in this lab, we will be using the *zebra* deamon instead.

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

3.2.1 CONFIGURING INTERFACES USING CONFIGURATION FILES

Before running the `zebra` process, a configuration file must be created and edited. Below is an example of such a file written for the router R1. This file can be found among setup files available on Moodle. The configuration file is a text file located in the /usr/local/quagga directory along with the configuration files for other daemons. In the terminal, enter the following command to open the configuration file.

```
leafpad /usr/local/quagga/zebra.conf
```

Edit the configuration file to look like this and save the changes.

```
!
!  zebra configuration file for R1
!
hostname R1
password zebra
enable password zebra
!
log file /usr/local/quagga/zebra.log
debug zebra packet
!
interface eth0
no shutdown
ip address 192.168.12.1/24
ipv6 address 2001:1:0:12::1/64
!
interface eth1
no shutdown
ip address 192.168.14.1/24
ipv6 address 2001:1:0:14::1/64
!
!
line vty
!
```

Let's examine the content of the configuration file above:

- `!`: 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
- `debug zebra packet`: more detailed debugging, *i.e.,* allows you to see when routes are added or deleted from the routing table;
- `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.

Start the *zebra* process using this file by typing in the terminal window on the router R1:

```
zebra -d -f /usr/local/quagga/zebra.conf
```

To ensure that the same Quagga process is not started twice, you should use a shell script similar to the one shown below.

```
#!/bin/sh

#uncomment whatever lines you need

rm /usr/local/quagga/zebra.log
touch /usr/local/quagga/zebra.log
chmod 666 /usr/local/quagga/zebra.log
rm /usr/local/quagga/ripd.log
touch /usr/local/quagga/ripd.log
chmod 666 /usr/local/quagga/ripd.log
rm /usr/local/quaggarip/ngd.log
touch /usr/local/quagga/ripngd.log
chmod 666 /usr/local/quagga/ripngd.log

killall zebra ripd ripngd
zebra -d /usr/local/quagga/zebra.conf
#ripd -d /usr/local/quagga/ripd.conf
#ripngd -d /usr/local/quagga/ripngd.conf
```

The script assumes that your configuration files for `zebra`, `ripd` and `ripngd` are called `zebra.conf`, `ripd.conf` and `ripngd.conf` and that they can be found in the `/usr/local/quagga/` directory. In the same directory you can save the above script as `start.sh`.

We already created all the scripts for the routers R1, R2 and R4 and you can find them on Moodle too. The creation of the scripts for the routers R3 and R5 is left to you. In fact they are exactly the same as the files names are the same.

The first group of commands (*i.e.,* `touch`, `chmod` and `rm`) create the log files for `ripd` and `ripngd` processes that are discussed later in this lab. They also permit writing to these files and they delete the old log files. As the configuration files for `ripd` and `ripngd` processes are still not created, the lines in the script starting these processes are commented out, using the # character. Hence, the log files for these two processes remain empty for the moment.

You should now start configuring the interfaces on the routers (assignement of IP addresses). Thus, start the zebra processes on all five routers by using the command provided above or the `start.sh` scripts.

Note: Make sure that the permissions for the `start.sh` files allow you to execute them. To make a script executable you can type the following command in the terminal window:

```
chmod 755 <script name>
```

Finally, you execute the script by navigating to `/usr/local/quagga/` directory and typing the script name preceded by "`./`".

3.2.2 QUAGGA MONITORING MODE

In order to monitor the activity of a running Quagga process, say `zebra`, you can enter the monitoring mode by connecting to it using

```
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`.

Q1/ What subnets can be found in the two routing tables on R1 at this point? Explain.

[A1]

Next, check the status of the network interfaces on router R3 by typing `show interface`.

Q2/ How many interfaces with IP addresses are shown? Write those IPv4 and IPv6 addresses.

[A2]

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

```
enable
```

Finally, inspect the running configuration of the router R3 using the `show running-config` command.

3.2.3 "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 eth0 is not assigned an IPv4 address. You could of course kill the *zebra* process on this router, modify the line in the configuration file and restart the process. However, we will fix this problem without stopping the process. 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 eth0`. 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 *eth0*, by using the same command that is used in the configuration file for this purpose. Simply type `ip address 192.168.14.4/24`. Exit the *interface configuration mode* and the *configuration mode*, by typing `exit` twice and verify that the modification took effect (`show interface` command).

3.3 CONFIGURING RIPV2

Similarly to the `zebra` process, the `ripd` process can be configured by editing a configuration file or "on the fly", as illustrated in Section 3.2.3. Note that the *zebra* process must be invoked before invoking the *ripd* process.

Like with the *zebra* configuration files, we are providing the "ready-to-use" *ripd* 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 */usr/local/quagga/ripd.conf*. We left the creation of the *ripd* configuration files for the routers R3 and R5 to you.

Below, we explain the steps needed to configure RIPv2 on a router. These steps should allow you to understand the content of the *ripd* 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 RIP protocol on a router are:

- enabling RIP, and choosing its version;
- choosing which networks should be advertised via RIP;
- specifying the interfaces that take part in the exchange of routing information.

Some optional steps include debugging, logging, etc. Before actually starting necessary processes with `start.sh` we first give an example of a `ripd` configuration file:

```
!
!  ripd configuration file for RX
!
hostname RX
password ripd
enable password ripd
!
log file /usr/local/quagga/ripd.log
```

```
debug rip events
debug rip packet
!
router rip
version 2
redistribute connected
network 192.168.xx.0/24
!
line vty
!
```

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

- `log file`: allows you to specify the file to which the RIP related information is logged;
- `debug rip events`: less detailed debugging, *i.e.,* only the coarse events, such as sending and receiving RIP updates, can be seen;
- `debug rip packet`: more detailed debugging, *i.e.,* allows you to see the content of the sent and received RIP updates;
- `router rip`: enables the *ripd* process;
- `version`: allows you to specify the version of RIP the *ripd* process will run;
- `network`: all the router's interfaces whose IP addresses belong to the network prefix declared with the `network` command will participate in the exchange of RIP updates. Additionally, the prefix that follows the command will be advertised via RIP. If you do not want a particular prefix to be exchanged over RIP, add `no` at the beginning of the command. The syntax is:

```
(no) network (network prefix)
```

- `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 `ospf` (*e.g.,* if you want to redistribute the routes from these protocols into RIP),
  (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).

  Note: You must always use the command `redistribute` in conjunction with the command `network`, in order to tell the router which interfaces participate in the exchange of RIP updates.

In the example above, we advertise via RIP all the directly connected subnets (prefixes), but we allow only to the router's interfaces that belong to the subnet 192.168.xx.0/24 to send and receive RIP advertisements.

Using the example configuration file shown above and the address scheme in Figure 1, create the *ripd* configuration files for the routers R3 and R5 (configuration files for R1, R2 and R4 can be downloaded from Moodle). Make sure that both debugging modes are enabled for RIP (*i.e.,* `debug rip events` and `debug rip packet`). Use the `start.sh` scripts to start the *zebra* and *ripd* processes on all routers in the following order: R3, R5, R2, R1, then wait around 10s before executing `start.sh` on R4.

Q3/ Open the */usr/local/quagga/ripd.log* file on router R1. How is this information exchanged between the routers (*i.e.,* by using broadcast, unicast or multicast)? Copy the line from the log file that illustrates this.

[A3]

Q4/ Check the */usr/local/quagga/ripd.log* file on router R1. What is the time (t_1) when R1 receives the first routing update from R2 and what are the networks that are advertised?

[A4]

Q5/ Check again the same file on router R1. What is the time (t_2) when R1 receives the first routing update from R4 and what are the networks that are advertised?

[A5]

Q6/ Now open the */usr/local/quagga/zebra.log* file on router R1. Find entries that correspond to time t_1 ($\pm 1s$) and look for ZEBRA_IPV4_ROUTE_ADD messages? How many are there? Explain.

[A6]

Q7/ Now check the */usr/local/quagga/zebra.log* file on router R1 at time t_2 ($\pm 1s$) and look for ZEBRA_IPV4_ROUTE_ADD messages? How many are there? Explain.

[A7]

3.3.1 MONITORING COMMANDS

We have seen before that we can use command `show ip route` to display the IPv4 routing table. However, at this point we should introduce yet another concept, i.e. RIP database (RIP Routing Information Base). RIP database is the place where networks known to RIP are stored. They are candidates for becoming part of the IPv4 routing table. We might have other routing protocols with alternative routes for the same networks. The decision which routing protocol to choose is made based on the routing protocol preference. In our case, we have only RIP running in our network. As a result all the entries from the RIP database will become part of the routing table.

To inspect the RIP database (that contains the routes known to RIP), you can enter to Quagga configuration mode. This is done in the same way as in the case of *zebra* process (*i.e.,* `telnet localhost ripd`). After that, you can display the content of the RIP database using the command:

```
show ip rip
```

Q8/ Write the content of the RIP database on the router R1. Can you infer the value of the *update timer* from the RIP database? If so, what is its value? Is it constant? In the case of the network prefixes that are not directly connected to the router (learned via RIP), what does the column *Time* represent?

[A8] 

3.4 CONFIGURING RIPNG

RIPng (RIP next generation), defined in RFC 2080, is an extension of RIPv2. It is an IPv6 reincarnation of the RIPv2 protocol. RIPng sends updates on UDP port 521 using the multicast group FF02::9.

As multiple routing protocols can run in parallel on the majority of routers, Quagga allows you to configure RIPng in parallel to RIPv2. Just like `zebra` and `ripd` processes, the `ripngd` process can be configured by editing a configuration file or via telnet. Again, the *zebra* process must be invoked before invoking the *ripngd* process.

As in the case of `ripd`, we are providing the *ripngd* configuration files for the routers R1, R2 and R4. It is left to you to create similar files for the routers R3 and R5. Like in the case of RIPv2, you should advertise via RIPng all the directly connected IPv6 subnets.

The main configuration steps required to configure RIPng on a router do not differ from those required to configure RIP. In terms of commands used to configure *ripngd*, very few differ from those used to configure *ripd*. The few exceptions are listed below:

- `router ripng`: this command is used instead of `router rip` to declare the routing protocol
- `version`: this command does not exist in the case of RIPng
- `debug ripng events` and `debug ripng packet`: used instead of the `debug rip events` and `debug rip packet` commands

You should make sure that you log *ripng* events/packets into a separate file.

3.4.1 MONITORING COMMANDS

To inspect the RIPng database (RIPng Routing Information Base), which contains the routes to IPv6 subnets known to RIPng, you should execute `telnet localhost ripngd` command. To display the content of the RIPng database type:

```
show ipv6 ripng
```

Finally, to display the contents of the IPv6 routing table type the following in the terminal.

```
ip -6 route show
```

Q9/ Check the contents of the IPv6 routing table on router R1. What addresses are shown as the next hops for the routes learned via RIPng? Explain.

[A9]

4 RIP AND RIPNG PLAYGROUND

In this section you will be confronted with a number of situations that might arise in a RIPv2 or a RIPng network. Answering the questions that accompany each problem should help you not only pointwise, but it should also allow you to better understand RIPv2, RIPng and dynamic routing in general.

Note: It might be a good idea to use two terminals, one for the Quagga process commands and one for Linux commands.

4.1 BROKEN LINK

Do the following section of actions. Kill ripd processes on router R1, R2 and R4. Now execute script `start.sh` on the following order: R1, R2 and then R4. It is assumed that all processes are already running on R3 and R5.

Display the content of the routing table and the RIP database of R1 (`show ip route` and `show ip rip` commands). You can write it down for later comparison.

To simulate broken link between R1 and R2 shutdown the *eth0* interface on R2. To do this, type on R2 the following commands:

```
telnet localhost zebra
enable
configure terminal
interface eth0
shutdown
```

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

Q10/ Observe the changes in the routing table and the RIP database of R1. Explain what happens.

[A10]

After 4-5 minutes, bring the *eth0* interface on R2 up using the command:

```
no shutdown
```

Q11/ Once the *eth0* interface on R2 is up, wait for the things to converge and then check the content of R1's routing table. Is it the same as in the beginning (before we brought down the *eth0* interface on R2)?. Explain.

[A11]

4.2 MODIFIED LINK METRIC

Before starting this part, put the interface *eth0* on R2 up again and wait for the routing tables to converge. RIP and RIPng are distance vector protocols. By default, the directly connected subnets are treated as having the distance equal to 1. Each additional "hop" (router) adds 1 to this distance metric. Nevertheless, RIP and RIPng offer you the possibility to modify the metric of an arbitrary link, changing the desirability of the routes that contain this link.

If you want to change the metric of the link that connects the routers R_i and R_j **from the point of view of** R_i, there are two things that you have to do.

- First, R_i has to announce to their neighbors the new metric value for the subnet the shared link R_iR_j belongs to. To do this, you have to use a `route-map`.
  Here is an example that shows how to add a route map to the *ripngd* configuration file of the router R_i. It changes the metric of the link connected to its *eth1* interface from the default value 1 to 5.

```
route-map incLinkMetric permit 10
match interface eth1
```

```
set metric 5
route-map incLinkMetric permit 20
```

In order for the route map `incLinkMetric` to be taken into account, modify the `redistribute connected` command in the *ripngd* configuration file, as shown below:

```
redistribute connected route-map incLinkMetric
```

- Second, you have to add this extra metric to all routes that contain the link R_iR_j so that R_i can increase the cost for 5 and not 1 before propagating the information. To do this, use the `offset-list` command. For more details on `route-map`, `offset-list` and `access-list` commands, check the documentation on the Quagga website. Here is an example of how to use the `offset-list` command:

```
offset-list addExtraMetric in 4 eth0
ipv6 access-list addExtraMetric permit any
```

The commands above add the extra metric of 5 to each route that contains the link R_iR_j $(1 + 4 = 5)$ that are propagated by R_i. Note that the R_i's *eth1* interface (in the first command) is on the shared link R_iR_j. Also, the given configuration will create **assymetric** behavior as nothing has been changed for R_j. In order to have the symmetric behavior the commands above have to be added to the *ripngd* configuration file on both R_i and R_j. **Finally, it is important to note that *offset-list* and *access-list* commands have to come before the *route-map* in the *ripngd* configuration file in order to avoid runtime errors.**

Now, concretely, let's assume we want to make R1 prefer the route to *2001:1:0:45::/64* via R2 and R5, instead of the route via R4. Apply the changes described above to ripngd.conf of R1. As a result you should have:

```
!
!  ripngd configuration file for R1
!
hostname R1
password ripngd
enable password ripngd

log file /usr/local/quagga/ripngd.log
debug ripng events
debug ripng packet

router ripng
redistribute connected route-map incLinkMetric
network 2001:1:0:12::/64
network 2001:1:0:14::/64

offset-list addExtraMetric in 4 eth1
ipv6 access-list addExtraMetric permit any

route-map incLinkMetric permit 10
match interface eth1
```

```
set metric 5
route-map incLinkMetric permit 20

line vty
```

Restart all the processes on R1 with `start.sh` and verify whether the desired routing from R1 to *2001:1:0:45::/64* is indeed put in place.

The changes you did should also affect routing from R2 to the network 2001:1:0:14:: . Find in the log file on the router R1 time t_0 when you activated RIPng routing process with the new settings.

Q12/ In the RIPng log file on R2 find the lines that correspond to the packets received after t_0 that contain information about network 2001:1:0:14:: . Are there some events in zebra log file that correspond to the times of reception of identified packets? How many changes of the R2 routing table occured and why?.

[A12]

If all the changes are done correctly a packet from R2 to 2001:1:0:14::4 (eth0 interface of R4) should go through R5 (not R1). Now let's imagine that we want to enforce that packets in the opposite direction take the alternative route. Concretely, the packet sent from R4 to 2001:1:0:25::2 (eth2 interface of R2) should go through R1 (not R5).

Q13/ Propose changes on R5 similar to those seen on R1 to achieve the policy explained above. Write down the resulting content of the configuration file you changed.

[A13]

4.3 PROCESS CRASH

At this point it is assumed that *zebra*, *ripd* and *ripngd* processes are running on all five routers and that the network is configured according to the scheme shown in Figure 1. Revert all the changes you did, i.e. all links should have distance metric of 1 again!

Now, kill only the *ripd* processes on routers R2 and R5. All other processes (*i.e., zebra* and *ripngd*) running on these two routers, as well as the *zebra*, *ripd* and *ripngd* processes on the remaining three routers, should continue running.

Observe the changes in the IPv4 routing table and the RIP database at routers R1 and R4, over the period of a few minutes. The right way to do this is to refresh (display) the content of both tables every 10-20 seconds. Focus on the entries that contain routes to the prefixes that are directly connected to R3 and pay special attention to the `Time` column in the RIP database.

Q14/ After how much time do the prefixes for the subnets that are directly connected to R3 disappear from the routing table of R1 and R4? How do you explain this?

[A14]

Q15/ After how much time do the prefixes for the subnets that are directly connected to R3 disappear from the RIP database of R1 and R4? Explain.

[A15]

Q16/ Can you ping the IPv4 addresses of the R3's ethernet interfaces from the router R1? How about the IPv6 addresses assigned to the same interfaces (use *ping ipv6* from Quagga configuration mode or *ping ipv6*

if you are not in Quagga configuration mode)? Explain.

[A16]