Laboratory 2 Dynamic routing using RIP. Iptables.

Part1. Dynamic Routing

Introduction

Static routing has the advantage that it is simple, requires no computing power in router for determining routes (this work is done by the administrator) and does not generate network traffic (routers do not change the routing information among them), but has the following disadvantages:

- In case of changes, management is difficult;
- Do not adjust automatically in case of malfunction; a route that became inaccessible remains inaccessible until the administrator intervention.

Dynamic routing protocols – shortly called **routing protocols**- are classified according the algorithm determining the routing information in: ***link state*** protocols and ***distance vector*** protocols.

A further classification, which considers the coverage of the routing protocols, divide them in:

- **internal routing protocols** (IGP - Interior Gateway Protocols), which operated inside a routing domain consisting of a number routers under a single administration and defining a so-called autonomous system (AS - Autonomous System
- **external routing protocols** (EGP - Exterior Gateway Protocols) used to interconnect autonomous systems.

Whatever routing method is chosen, IP routing is of "next hop" type, meaning that each node only specify the next hop node, as opposed to "source routing" stating the entire way forward between source and destination. Source routing is not used in the Internet because it requires knowledge of the entire network centrally and could not easily adapt to unforeseen changes.

RIP

RIP (*Routing Information Protocol*) is a distance vector protocol. It uses Belman Ford distributed routing algorithm to find the shortest path between network nodes. RIP features are:

RIP (RFC 1058, 2453.)

- Metric used in path selection is the number of hops;
- Max. 15 hops
- Updates sent by default, every 30 seconds.

RIP v2

- updates are sent to the broadcast address 255.255.255.255
- Transmit packets to neighbors and update network mask, to support classless networks and VLSM (Variable Length Subnet Mask)
- Support authentication

RIP protocol configuration

Some commands that allow the study of routing tables and other issues associated routing are listed in the table below.

command	*description*
`show ip route`	shows routing table; Networks are classified by type: those directly connected, the static ones and learned through routing protocols.
`show ip route` *`destination_network`*	Shows information about a route to a specific destination.
`show ip protocols`	Shows the routing protocols that run on the router.
`show ip rip database`	Shows information about the RIP table.

`debug ip rip`
removed with
`undebug all`
	Enables debug messages be printed to the console: each time an event occurs in RIP (eg an update is sent or received). These messages are printed to the console only, not via telnet!

Routes learned by the router can be erased with the command:

```
Router# clear ip route *
```

where * is a “wildcard” meaning “all routes”. Alternatively, you can specify one route instead of *.

Each route has an associated administrative distance; if there are several routes available, the router will choose the route with the lowest administrative distance. Administrative distances are represented in the routing table in the following format (value in bold in parentheses; the other value after "/" is the route metric measured in the number of hops up to the destination network; for other protocols, the metric is different):

```
Router# sh ip route

     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
R       10.0.0.0/8 [120/2] via 172.16.1.2, 00:00:05, FastEthernet1/0
C       10.1.1.0/24 is directly connected, FastEthernet0/0
     172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
R       172.16.0.0/16 [120/3] via 172.16.1.2, 00:00:05, FastEthernet1/0
C       172.16.1.0/24 is directly connected, FastEthernet1/0
R       172.16.2.0/24 [120/1] via 172.16.1.2, 00:00:05, FastEthernet1/0
```

```
S       172.16.3.0/24 [1/0] via 10.1.1.2
R    192.168.1.0/24 [120/2] via 172.16.1.2, 00:00:05, FastEthernet1/0
R    193.1.1.0/24 [120/2] via 172.16.1.2, 00:00:05, FastEthernet1/0
```

Static routes have an administrative distance 1 while the routes learned through RIP have 120. Therefore, the router will prefer static routes, and use RIP only if they become unavailable. To configure a backup static route, which is not preferred over RIP, one will configure a greater distance, eg 130, for it:

```
Router0(config)#ip route 192.168.1.0 255.255.255.0 172.16.1.2 130
```

Routes summarization (route aggregation)

Route summarization, also called route aggregation, is a method of minimizing the number of routing tables in an IP (Internet Protocol) network. It works by consolidating selected multiple routes into a single route advertisement, in contrast to flat routing in which every routing table contains a unique entry for each route. This requires addresses continuity in an are - if the subnets of the same net are randomly distributed across multiple routers can not make summarization.

To implement route summarization in IP Version 4 (IPv4), Classless Inter-Domain Routing (CIDR) must be used. All IP addresses in the route advertisement must share identical high-order bits. The length of the prefix must not exceed 32 bits.

Route summarization offers several important advantages over flat routing:

- Reduce the size of routing tables;
- If a network "behind" a router, which is summarize in a supernet, "fall" due to technical problems (and eventually recovers flashing), this is not propagated in the routing table, so the other routers do not need to change their routing tables every time. This remains a problem "internal" router respectively;
- Can minimize the latency in a complex network.

Example 1:

- Are given four networks: 192.168.0.0/24, 192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24
- It is desired to combine them in a supernet with a mask **/22** (4 networks occupy 2 bits)
- Supernet written in binary (bold) will be 192.168.00000**000.00000000** ie 192.168.0.0/22 - so four independent network of 256 addresses can be written as a supernet with 1024 addresses.
-

Netfilter. Iptables.

Introduction [2]

The Linux kernel comes with a packet filtering framework named **netfilter**. It allows you to allow, drop and modify traffic leaving in and out of a system. A tool, **iptables** builds upon this functionality to provide a powerful firewall, which you can configure by adding rules. In addition, other programs such as fail2ban also use iptables to block attackers.

Iptables is just a command-line interface to the packet filtering functionality in netfilter. However, to keep this article simple, we won't make a distinction between iptables and netfilter in this article, and simply refer to the entire thing as "iptables".

The packet filtering mechanism provided by iptables is organized into three different kinds of structures: **tables, chains** and **targets**. Simply put, a table is something that allows you to process packets in specific ways. The default table is the filter table, although there are other tables too.

Again, these tables have chains attached to them. These chains allow you to inspect traffic at various points, such as when they just arrive on the network interface or just before they're handed over to a process. You can add rules to them match specific packets — such as TCP packets going to port 80 — and associate it with a target. A target decides the fate of a packet, such as allowing or rejecting it.

When a packet arrives (or leaves, depending on the chain), iptables matches it against rules in these chains one-by-one. When it finds a match, it jumps onto the target and performs the action associated with it. If it doesn't find a match with any of the rules, it simply does what the default policy of the chain tells it to. The default policy is also a target. By default, all chains have a default policy of allowing packets.

Tables [2]

As we've mentioned previously, tables allow you to do very specific things with packets. On modern Linux distributions, there are four tables:

- The ***filter*** table: This is the default and perhaps the most widely used table. It is used to make decisions about whether a packet should be allowed to reach its destination.
- The ***mangle*** table: This table allows you to alter packet headers in various ways, such as changing TTL values.
- The ***nat*** table: This table allows you to route packets to different hosts on NAT (Network Address Translation) networks by changing the source and destination addresses of packets. It is often used to allow access to services that can't be accessed directly, because they're on a NAT network.
- The **raw** table: iptables is a stateful firewall, which means that packets are inspected with respect to their "state". (For example, a packet could be part of a new connection, or it could be part of an existing connection.) The raw table allows you to work with packets before the kernel starts tracking its state. In addition, you can also exempt certain packets from the state-tracking machinery.

Chains [2]

Now, each of these tables are composed of a few default chains. These chains allow you to filter packets at various points. The list of chains iptables provides are:

- The PREROUTING chain: Rules in this chain apply to packets as they just arrive on the network interface. This chain is present in the nat, mangle and raw tables.
- The INPUT chain: Rules in this chain apply to packets just before they're given to a local process. This chain is present in the mangle and filter tables.
- The OUTPUT chain: The rules here apply to packets just after they've been produced by a process. This chain is present in the raw, mangle, nat and filter tables.
- The FORWARD chain: The rules here apply to any packets that are routed through the current host. This chain is only present in the mangle and filter tables.
- The POSTROUTING chain: The rules in this chain apply to packets as they just leave the network interface. This chain is present in the nat and mangle tables.

Targets [2]

As we've mentioned before, chains allow you to filter traffic by adding rules to them. So for example, you could add a rule on the filter table's INPUT chain to match traffic on port 22. But what would you do after matching them? That's what targets are for — they decide the fate of a packet.

Some targets are terminating, which means that they decide the matched packet's fate immediately. The packet won't be matched against any other rules. The most commonly used terminating targets are:

- ACCEPT: This causes iptables to accept the packet.
- DROP: iptables drops the packet. To anyone trying to connect to your system, it would appear like the system didn't even exist.
- REJECT: iptables "rejects" the packet. It sends a "connection reset" packet in case of TCP, or a "destination host unreachable" packet in case of UDP or ICMP.

On the other hand, there are non-terminating targets, which keep matching other rules even if a match was found. An example of this is the built-in LOG target. When a matching packet is received, it logs about it in the kernel logs. However, iptables keeps matching it with rest of the rules too.

Sometimes, you may have a complex set of rules to execute once you've matched a packet. To simplify things, you can create a custom chain. Then, you can jump to this chain from one of the custom chains.

Examples [2]

The most common use for a firewall is to block IPs.

- `iptables -t filter -A INPUT -s 59.45.175.62 -j REJECT`

The `-t` switch specifies the table in which our rule would go into — in our case, it's the `filter` table.

The `-A` switch tells iptables to "append" it to the list of existing rules in the INPUT chain.

The `-s` switch simply sets the source IP that should be blocked.

Finally, the `-j` switch tells iptables to "reject" traffic by using the REJECT target. If you want iptables to not respond at all, you can use the DROP target instead.

- `iptables -A INPUT -s 59.45.175.62 -j REJECT`

To block a range of IP addresses one can ue the CIDR notation.

- `iptables -A INPUT -s 59.45.175.0/24 -j REJECT`

Iptables can be used to block protocols, too. For example, to block all incoming ICMP traffic, one simply need to specify the protocol.

- `iptables -A INPUT -p icmp -j DROP`

A more complex example:

- `iptables -A INPUT -p tcp -m tcp --dport 22 -s 59.45.175.0/24 -j DROP`

It blocks SSH access for an IP range. First match all TCP traffic, In order to check the destination port, you should first load the `tcp` module with `-m`. Next, you can check if the traffic is intended to the SSH destination port by using `--dport`.

Part 2: Experimental part

Build basic network topologies and network nodes configuration for a better understanding of the RIP protocol. Learn to filter traffic using access lists.

Remark:

Do not forget to set the Netkit environment variables and check your configuration:

- `export NETKIT_HOME=~/netkit`
- `export MANPATH=:$NETKIT_HOME/man`
- `export PATH=$NETKIT_HOME/bin:$PATH`

Check your configuration:

- `./check_configuration.sh`

Study of RIP protocol

In this part a topology consisting of five routers will be implemented as in Figure 1. RIP protocol will be activated and configured on each router. RIP functionalities will be investigated in several scenarios.

Network topology

The network topology is presented in Figure 3.

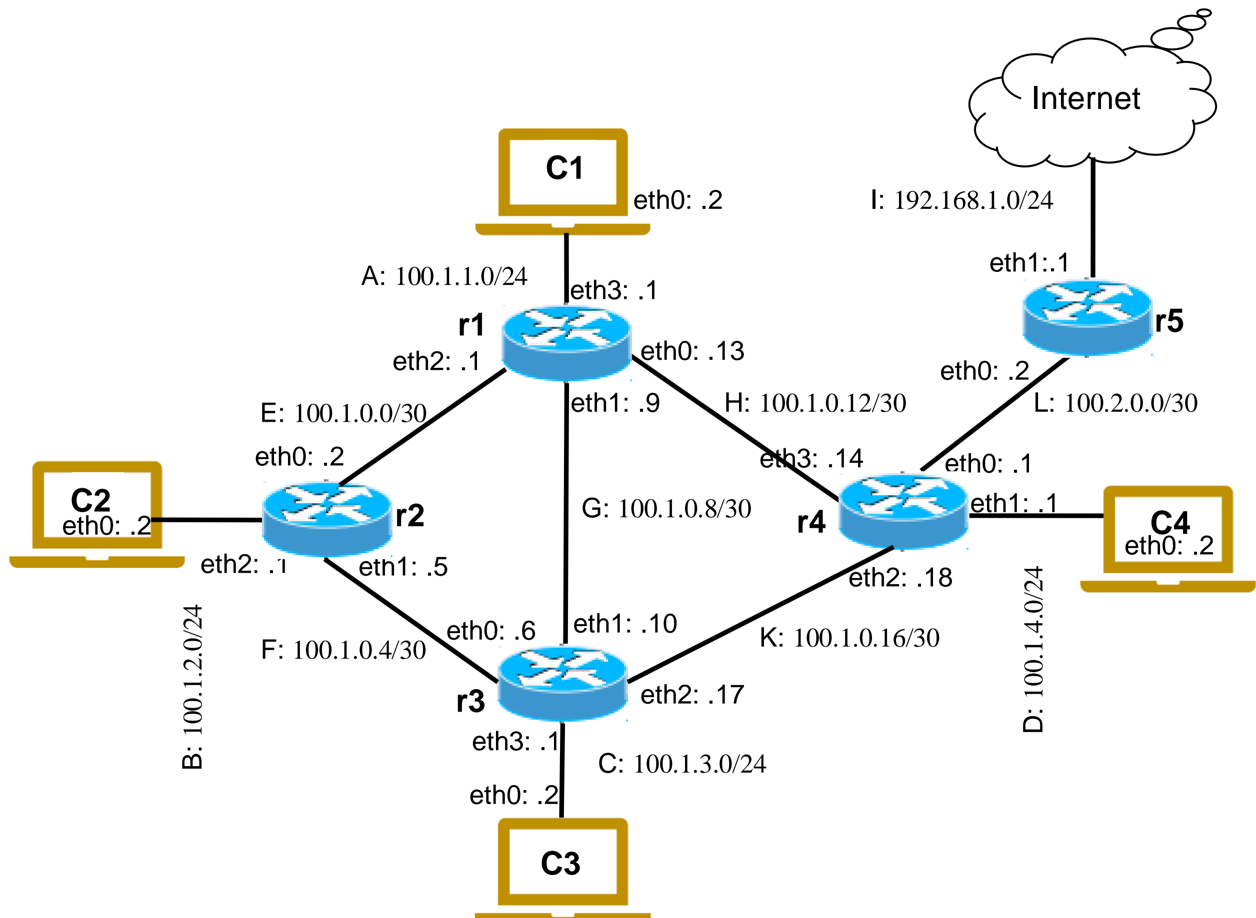

Figure 1: Network topology

1. Creating the topology lab

Inside the `netkit` folder create a new folder `lab3`. Inside the `lab3` folder create the structure of subdirectories and files associated with the topology.

- Create an empty folder for each device in the topology:C1 folder, C2 folder, …r1 folder, r2 folder, ….
- Create the `lab.conf` file where the topology is described:
  - `c1[0]=A`
  - …
  - `r1[0]=H`
  - `r1[1]=G`
  - `r1[2]=E`
  - `r1[3]=A`
  - …
- Create the `.startup` files for each device were the initial configuration is specified:
  - `c1.startup`:
    - `ifconfig eth0 100.1.1.2 netmask 255.255.255.0 up`
    - `route add default gw 100.1.1.1`
  - `c2.startup`
    - …
  - …
    - …
  - `r1.startup`
    - `ifconfig eth0 100.1.0.13 netmask 255.255.255.252 up`
    - `ifconfig eth1 100.1.0.9 netmask 255.255.255.252 up`
    - `ifconfig eth2 100.1.0.1 netmask 255.255.255.252 up`
    - `ifconfig eth3 100.1.1.1 netmask 255.255.255.252 up`
  - `r2.startup`
    - …
  - …
    - …

Start the lab with `lstart` command:

- **~/netkit$** `lstart –d ~/netkit/lab3`

2. Start Zebra daemon

On each router activate the RIP daemon and start zebra routing software.

To activate the RIP daemon edit the `daemons` file located in `/etc/zebra`.

- `r1:~# cd /etc/zebra` #go to zebra directory
- `r1:~# nano daemons` #edit daemons file with nano editor

To start the zebra daemon the following command will be used:

- `r1:~# /etc/init.d/zebra start`

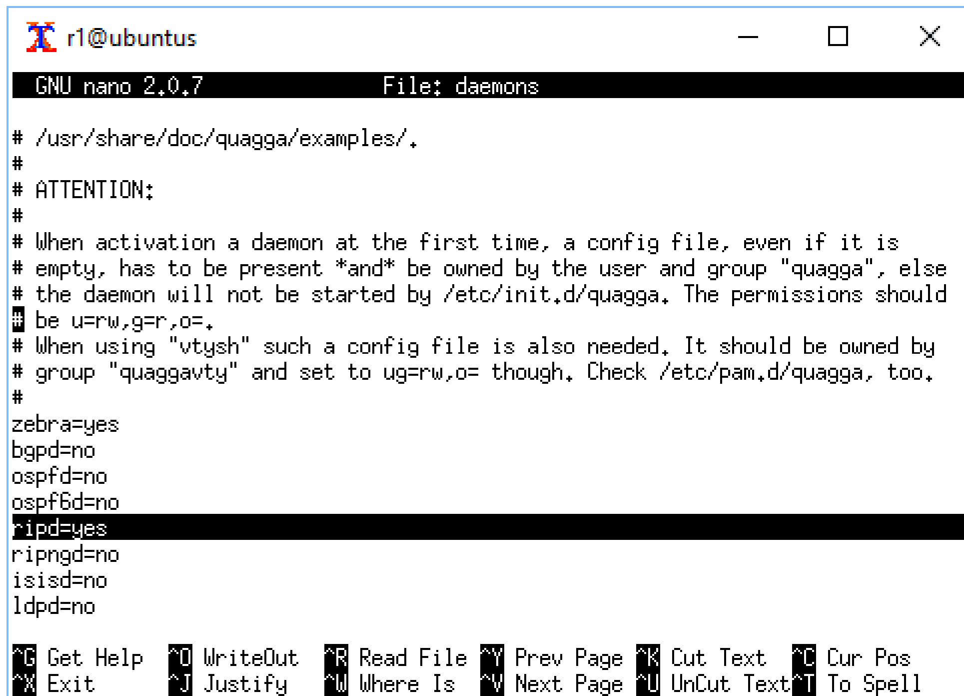

Figure 2: Activate RIPdaemon in `daemons` file

- Use `ping` command to test connectivity between the lab nodes.
- Try to ping a remote destination. Does `ping` work? Why
- Inspect routing tables with `route` command. What types of routes do exist in the routing table?

3. Configure RIP protocols

In order to establish the network connectivity between any devices in the topology, the RIP protocol will be configured.

On each of the routers r1 to r4 connect to the main zebra daemon using `vtysh`:

- `r1:~# vtysh`

Configure RIP protocol on each of the routers r1 to r4 as in Figure 3. To configure RIP protocol one must specify all its directly connected networks.

By default (i.e., without further configuration) RIPalready propagates information about directly connected subnets.attached to RIP-speaking interfaces only. The command `redistribute connected` forces RIP to propagate information about all connected subnets. So, another approach to configure rip is to force it to propagate information about all connected subnets as in Figure 4.

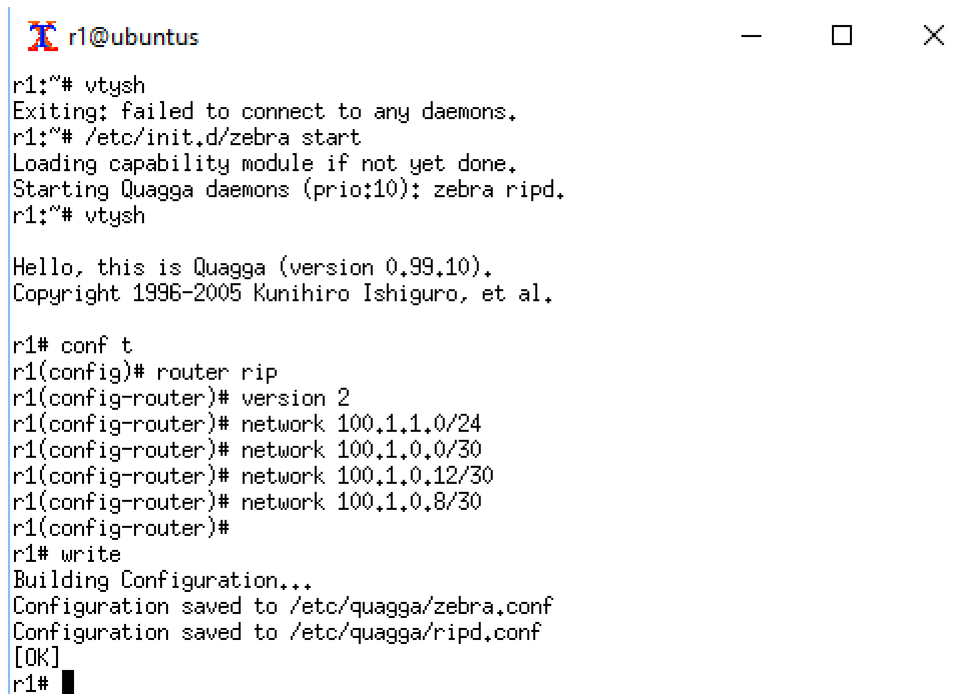

Figure 3: RIP configuration on router `r1`

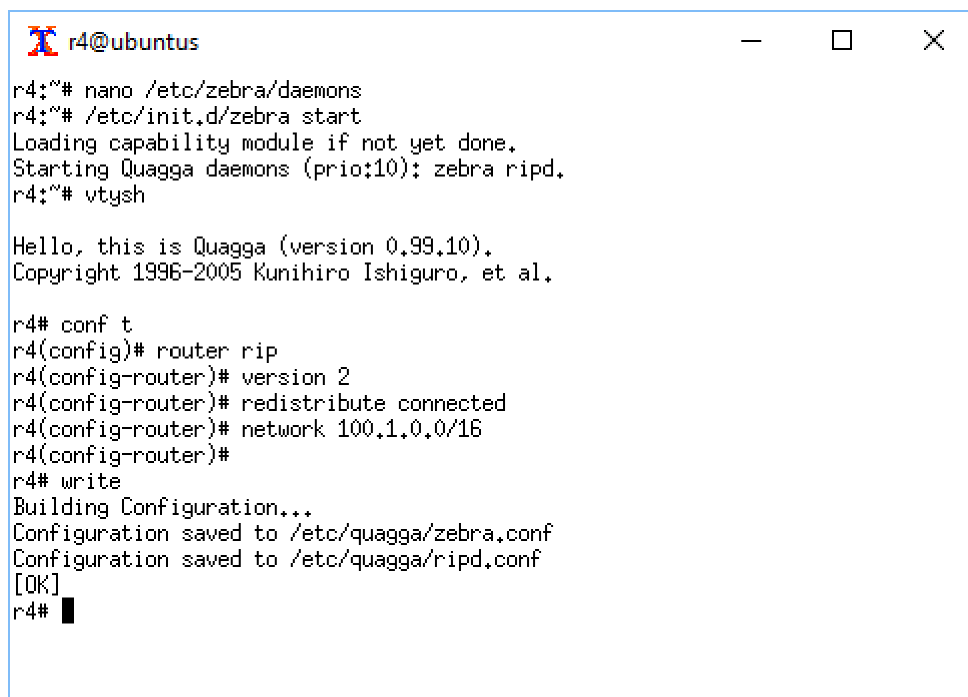

Figure 4: RIP configuration on router `r4` using `redistribute connected`

- After routers `r1` to `r4` are configured check that there exists connectivity between any two nodes.
- Check the routing table on each router with `show ip route` command.

Capture the RIP packets with tcpdump to visualize the RIP messages:

- `r4:~# tcpdump -i eth2 -v -n -s 1518`

-v: to display packet details

-n: don't resolve numbers to name

-s 1518 sniff entire Ethernet packet. By default, only the first 68 bytes are captured.

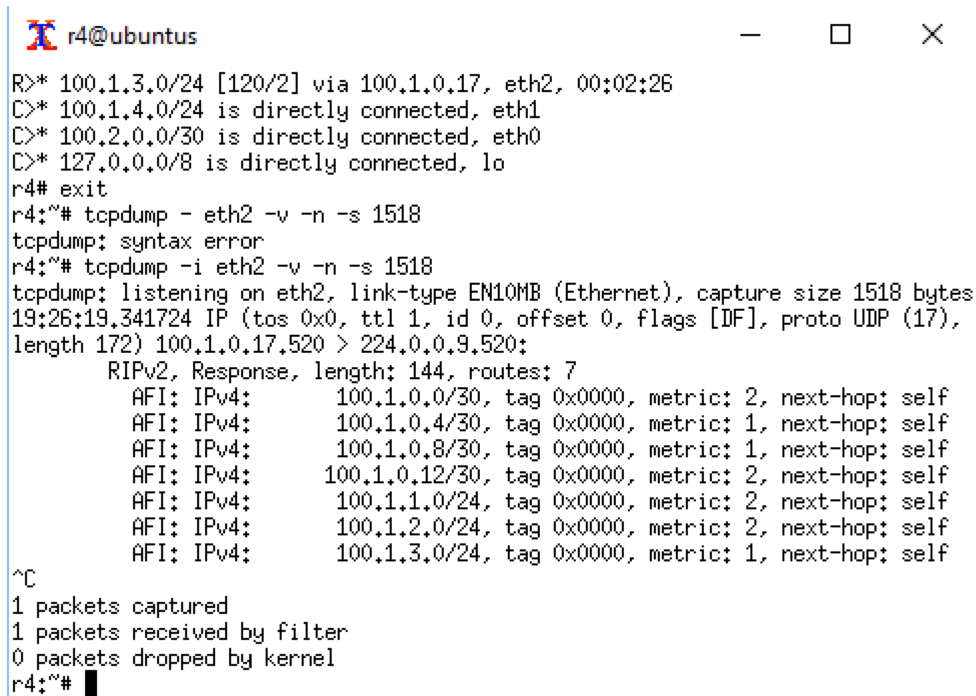

Figure 4: Sniffing RIP packets with `tcpdump`

4. Static routing configuration on `r5` router

Router `r5` can be seen as the gateway to Internet. The network `100.1.0.0/16` is a stub network for router `r5`. So, static routes are enoughfor connecting it to the Internet.

- Add a static route on `r5` to the network `100.1.0.0/16.`
- Check connectivitybetween `r5` and the other nodes on the network.

5. Emulating an Internet link

Interface `eth1` on router `r5` is used to emulate an Internet link. Try to `ping` IP address on `eth1`from one of the routers `r1` to `r4`. Does ping work? Why?

To provide Internet access to the routers `r1` to `r4` default gateway will be configured on `r4` and then instruct RIP on `r4` to redistribute the default route to the other routers in the network.

- Configure default route to `r4` towards router `r5` gateway.
- Configure router `r4` to propagate the default route into RIP with command:
  - `r4(config-router)#route 0.0.0.0/0`
- Check that default route was transmitted to the other routers.
- Check `ping` connectivity towards Internet (IP address on `r5 eth1` interface).

- Check that packets with external network destinations are forwarded to r5 router
  - From r1 ping an external network
    - r1:~#ping 8.8.8.8
  - Start tcpdump on r5
    - r1:~#tcpdump -i eth0 -n -s 1518
  - Check that ping packets are arriving at r5.

6. Reroute traffic when an interface is down

We will check that RIP will reroute traffic when an interface along the path towards the destination goes down.

- On C2 use traceroute to find the route towards C4
  - C2:~# traceroute 100.1.4.2
- On r1 inspect the routing table to check that it corresponds with the route followed by tcpdump packets.
- On C2 start a ping command towards C4
  - C2:~# ping 100.1.4.2
- Does ping work?
- Shut down an interface on the intermediate router along the path. For example, if traceroute shows that the packets from C2 are following the path r2, r3, r4, shut down intereface eth2 on r3.
  - r3:~# ifconfig eth2 down
- Is ping working on C2? Why?
- After ping starts working again check the new route with tcpdump on C2.
- On r1 inspect the routing table to check that it was updated.
- Rwstart the interface that was putted down:
  - r3:~# ifconfig eth2 up

Study of Iptables

Traffic filtering with Iptables will be configured in this section.

1. Basic traffic filtering

In this section IP traffic originated from an IP address will be rejected. The traffic filter will be installed on C3 machine. It will reject the packets coming from C1 machine.

- On C1 machine ping the C3 machine
  - C1:~# ping 100.1.3.2
- Does ping work?

On C3 build a script that will install an iptrables rule to reject the traffic coming from C1.

The script will be named filter-script.sh and will contain the lines:

- #!/bin/sh
- iptables -A INPUT -s 100.1.3.2 -j REJECT

To edit the file use nano editor (or other available editors):

- C3:~#nano filter-script.sh

After the file is edited and saved make it executable with the command:

- `C3:~#chmod a+rx filter-script.sh`

Before running the script inspect the iptables with the command:

- `C3:~#iptables -L -v`

Run the script with the command:

- `C3:~#./filter-script.sh`

Check that the filter is working:

- On `C1` machine `ping` the `C3` machine
  - `C1:~# ping 100.1.3.2`
- Does `ping` work? Why?

Remove the iptables rules using the script `remove_rules.sh`. Edit the script using `nano` editor:

- `C3:~#nano remove_rules.sh`

Insert the following lines into `remove_rules.sh` script:

- `#!/bin/bash`
- `iptables -F FORWARD`
- `iptables -nL`

After the file is edited and saved make it executable with the command:

- `C3:~#chmod a+rx remove_rules.sh`

Run the script with the command:

- `C3:~#./remove_rules.sh`

2. Filtering protocols

More complex filtering rules can be implemented with iptables. For example, protocols' packets ca be filtered using iptables. In this section the ICMP packets will be filtered on C4 machine.

Implement a filtering rule to filter on C4 the ICMP packets coming from networks 100.1.2.0/24 and 100.1.1.0/24.

- Check that the filtering rule is working using ping command from C1 and C2.

Implement a filtering rule to block ssh access from network 100.1.3.0/24 on C4

- Check that the filtering rule is working.

References:

1. http://wiki.netkit.org/netkit-labs/netkit-labs_basic-topics/netkit-lab_rip/netkit-lab_rip.pdf
2. https://www.booleanworld.com/depth-guide-iptables-linux-firewall/