Otto-Friedrich-University Bamberg

Professorship for Computer Science,
Communication Services, Telecommunication,
Systems and Computer Networks

Foundation of Internet Communication

Assignment-05 :Intradomain Dynamic Routing Protocols

Submitted by:
Group J

Reem Eslam Mohamed Mekky Khalil
Shivasharan Reddy
Azar Ghadami
Reema Miranda
Manjunath B Marigoudar

Supervisor: Prof. Dr. Udo Krieger

Bamberg, July, 2020
Summer Term 2020

Contents

List of Figures

Chapter 1

Interior Routing with OSPF

OSPF is a link state routing protocol, in which each router sends information on the cost metric of its network interfaces to all other routers in the network. The information about the interfaces is sent in messages that are called link state advertisements (LSAs). LSAs are disseminated using flooding, each router maintains a link state database of all received LSAs, which provides the router with complete information about the network topology.

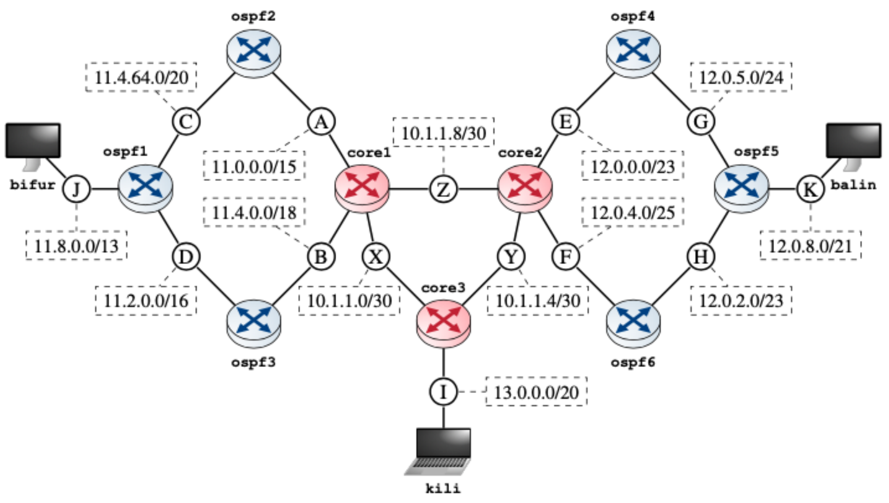


Figure 1.1: ospf lab network topology

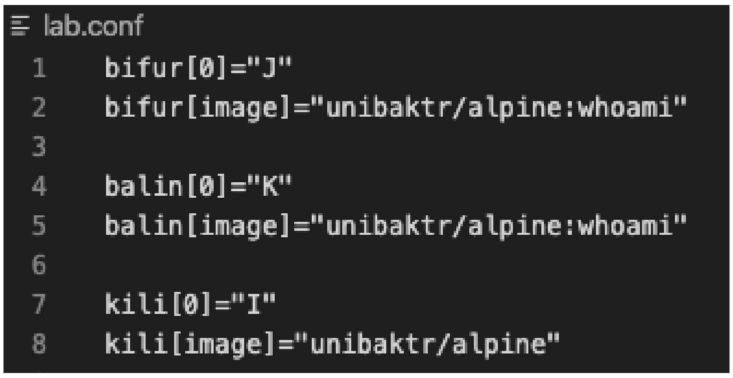
In this section we build the topology as shown in the above figure. hence, we have startup files for all the devices shown in the above network and we create folders for the routers as shown in the below example.

- example:- core1-etc-frr-daemons and osfpd.conf

1.1 Build the topology depicted in Figure 1.1 with Kathara.

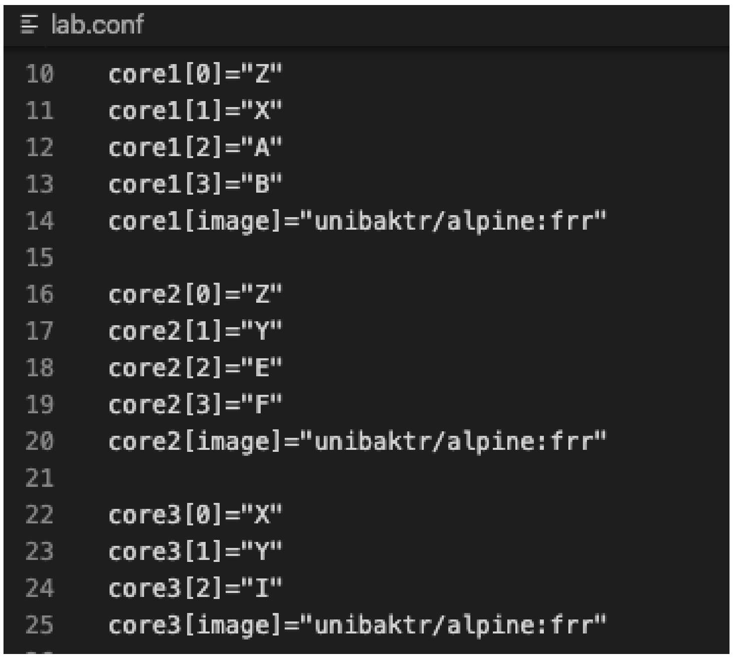
To build the above topology we need configure lab.conf file and all the configuration in startup files as shown below.

- lab conf file



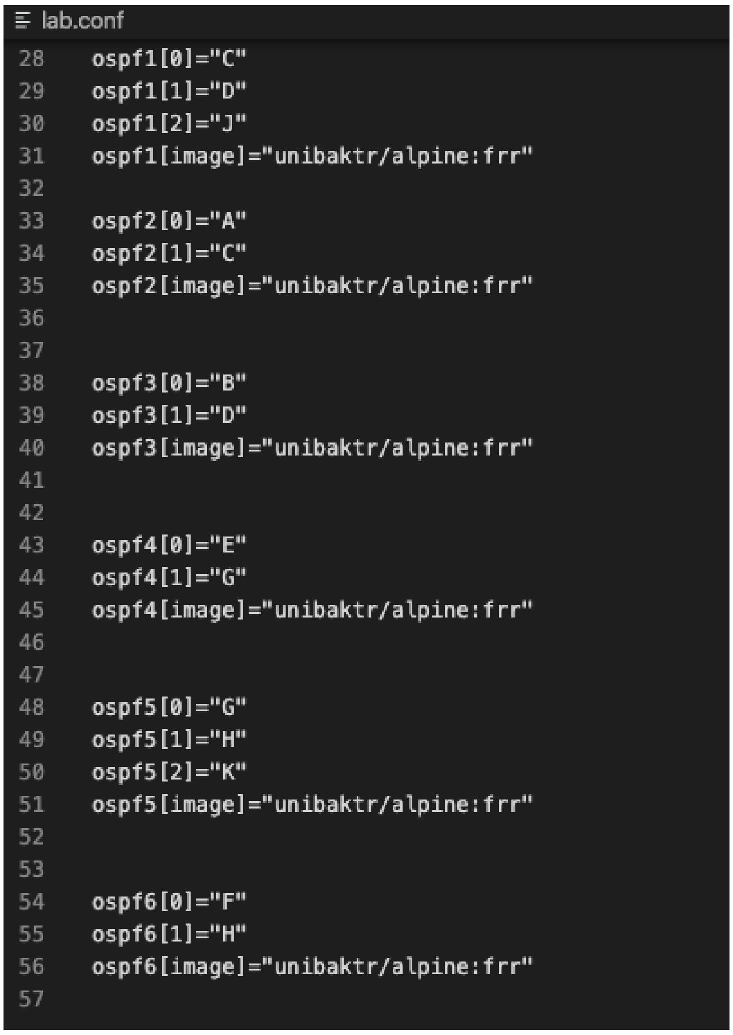
```
lab.conf
bifur[0]="J"
bifur[image]="unibaktr/alpine:whoami"

balin[0]="K"
balin[image]="unibaktr/alpine:whoami"

kili[0]="I"
kili[image]="unibaktr/alpine"
```

Figure 1.2: Bifur,balin and kili lab conf file



```
lab.conf
core1[0]="Z"
core1[1]="X"
core1[2]="A"
core1[3]="B"
core1[image]="unibaktr/alpine:frr"

core2[0]="Z"
core2[1]="Y"
core2[2]="E"
core2[3]="F"
core2[image]="unibaktr/alpine:frr"

core3[0]="X"
core3[1]="Y"
core3[2]="I"
core3[image]="unibaktr/alpine:frr"
```

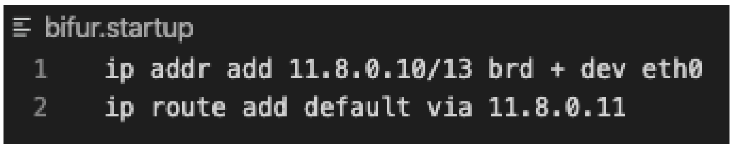
Figure 1.3: core routers lab conf file



```
lab.conf
ospf1[0]="C"
ospf1[1]="D"
ospf1[2]="J"
ospf1[image]="unibaktr/alpine:frr"

ospf2[0]="A"
ospf2[1]="C"
ospf2[image]="unibaktr/alpine:frr"


ospf3[0]="B"
ospf3[1]="D"
ospf3[image]="unibaktr/alpine:frr"


ospf4[0]="E"
ospf4[1]="G"
ospf4[image]="unibaktr/alpine:frr"


ospf5[0]="G"
ospf5[1]="H"
ospf5[2]="K"
ospf5[image]="unibaktr/alpine:frr"


ospf6[0]="F"
ospf6[1]="H"
ospf6[image]="unibaktr/alpine:frr"

```

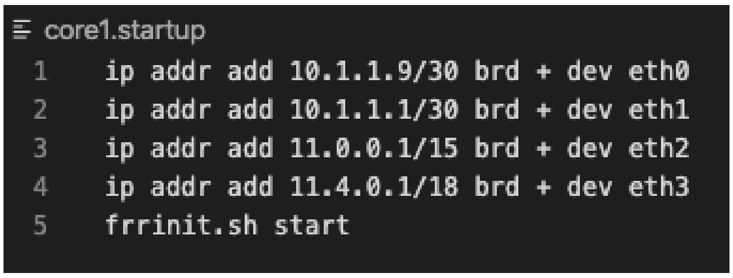
Figure 1.4: osdpf routers lab conf file

- startup files of all devices.



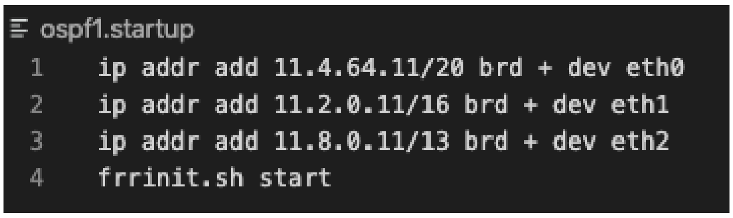
```
bifur.startup
ip addr add 11.8.0.10/13 brd + dev eth0
ip route add default via 11.8.0.11
```

Figure 1.5: Bifur startup configuration as an example the rest of the hosts will be the same.



```
core1.startup
ip addr add 10.1.1.9/30 brd + dev eth0
ip addr add 10.1.1.1/30 brd + dev eth1
ip addr add 11.0.0.1/15 brd + dev eth2
ip addr add 11.4.0.1/18 brd + dev eth3
frrinit.sh start
```

Figure 1.6: core-01 startup configuration as an example the others cores startup files will be the same.



```
ospf1.startup
ip addr add 11.4.64.11/20 brd + dev eth0
ip addr add 11.2.0.11/16 brd + dev eth1
ip addr add 11.8.0.11/13 brd + dev eth2
frrinit.sh start
```

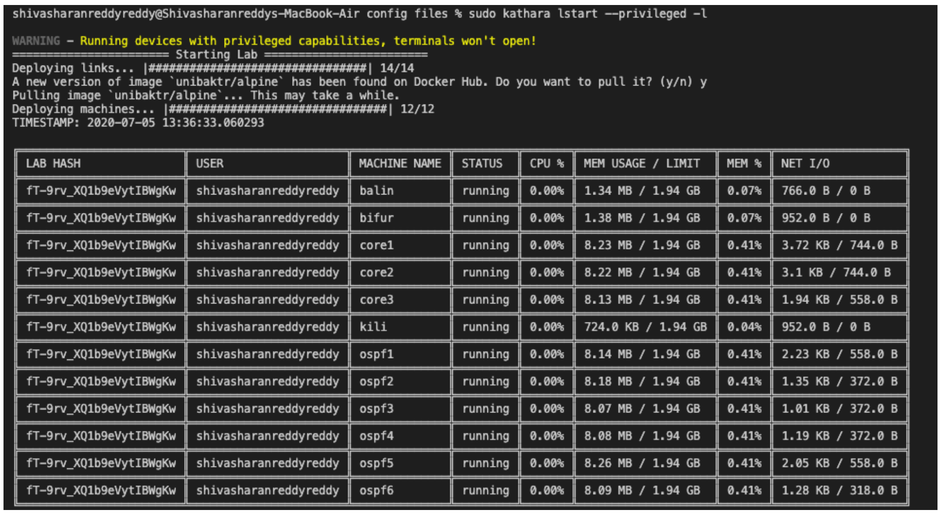
Figure 1.7: ospf-01 startup configuration as an example the rest of ospf startup files will be the same.

With all the configuration's done as shown above we can start the kathara lab, for starting the kathara lab we also have to pull the new image with command

- Docker pull unibaktr/alpine:frr

And we can then start the lab with command.

- sudo kathara lstart –privileged -l



```
shivasharanreddyreddy@Shivasharanreddys-MacBook-Air config files % sudo kathara lstart --privileged -l

WARNING - Running devices with privileged capabilities, terminals won't open!
======================= Starting Lab ========================
Deploying links... |################################| 14/14
A new version of image `unibaktr/alpine` has been found on Docker Hub. Do you want to pull it? (y/n) y
Pulling image `unibaktr/alpine`... This may take a while.
Deploying machines... |################################| 12/12
TIMESTAMP: 2020-07-05 13:36:33.060293

| LAB HASH               | USER                  | MACHINE NAME | STATUS  | CPU % | MEM USAGE / LIMIT  | MEM % | NET I/O            |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | balin        | running | 0.00% | 1.34 MB / 1.94 GB  | 0.07% | 766.0 B / 0 B      |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | bifur        | running | 0.00% | 1.38 MB / 1.94 GB  | 0.07% | 952.0 B / 0 B      |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | core1        | running | 0.00% | 8.23 MB / 1.94 GB  | 0.41% | 3.72 KB / 744.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | core2        | running | 0.00% | 8.22 MB / 1.94 GB  | 0.41% | 3.1 KB / 744.0 B   |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | core3        | running | 0.00% | 8.13 MB / 1.94 GB  | 0.41% | 1.94 KB / 558.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | kili         | running | 0.00% | 724.0 KB / 1.94 GB | 0.04% | 952.0 B / 0 B      |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | ospf1        | running | 0.00% | 8.14 MB / 1.94 GB  | 0.41% | 2.23 KB / 558.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | ospf2        | running | 0.00% | 8.18 MB / 1.94 GB  | 0.41% | 1.35 KB / 372.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | ospf3        | running | 0.00% | 8.07 MB / 1.94 GB  | 0.41% | 1.01 KB / 372.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | ospf4        | running | 0.00% | 8.08 MB / 1.94 GB  | 0.41% | 1.19 KB / 372.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | ospf5        | running | 0.00% | 8.26 MB / 1.94 GB  | 0.41% | 2.05 KB / 558.0 B  |
| fT-9rv_XQ1b9eVytIBWgKw | shivasharanreddyreddy | ospf6        | running | 0.00% | 8.09 MB / 1.94 GB  | 0.41% | 1.28 KB / 318.0 B  |
```

Figure 1.8: Kathara lab of OSPF network

1.2 Enable and configure OSPF on all routers. Let the coreX routers form the backbone area, place ospf1 to ospf3 in one stub area and ospf4 to ospf6 in an- other one. Explain the concept of different areas in OSPF. What is the purpose to use different areas?

In this section for enabling OSFP we first need to assign ID's for all the routers and also we need to assign area ID's for all the OSPF areas.
We have nine routes in the network and we assign each router its on ID

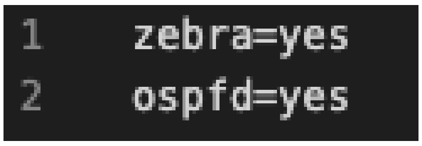
- Router ID'S
  ospf1 = 1.0.0.1
  ospf2 = 1.0.0.2
  ospf3 = 1.0.0.3
  ospf4 = 1.0.0.4
  ospf5 = 1.0.0.5
  ospf6 = 1.0.0.6
  core1 = 1.0.0.7
  core2 = 1.0.0.8

In the network shown above we have three stub OSPF areas and one backbone area and we place ospf1 to 0spf3 in one area and 0spf4 to ospf6 in one area. We assign area ids for all the four areas present in the network

- AREA-01 (ospf01 to ospf03 in one stub area)
  ospf1 = 1.1.1.1
  ospf2 = 1.1.1.1
  ospf3 = 1.1.1.1
  CORE1 = 1.1.1.1
- AREA-02 (ospf04 to ospf06 in another stub area)
  ospf4 = 2.2.2.2
  ospf5 = 2.2.2.2
  ospf6 = 2.2.2.2
  core2 = 2.2.2.2
- AREA-03 We also have a third area in which core3 is present.
  Core3 = 3.3.3.3
- Apart from the three areas in the network we also have a back bone network.
  Backbone area = 0.0.0.0

Now after we assign all the areas and router ID's in the given network we configure OSPF on all the routers and for doing that we create folders for all the given routers in the following order
example:- core1- etc- frr- Demons and ospfd files

- Daemons File
  Daemons file for all the routers will be same as we configure permission in daemons file so that ospfd file shall run in the lab.

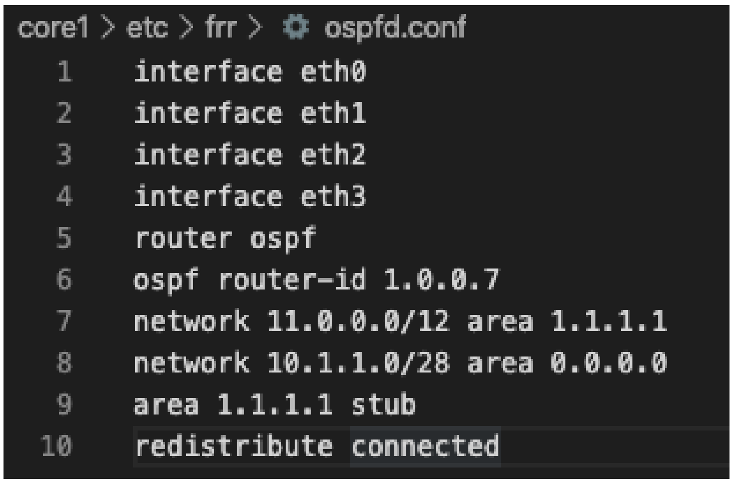


```
zebra=yes
ospfd=yes
```

Figure 1.9: Daemons file for all the routers

- OSPF configuration on all the devices.
  we configure ospf configurations inside the ospfd file of all the routers as shown below.

- CORE-01 ospf configuration
  Core-01 is inside two areas, Area-1 as-well as Backbone area hence we have the below configuration

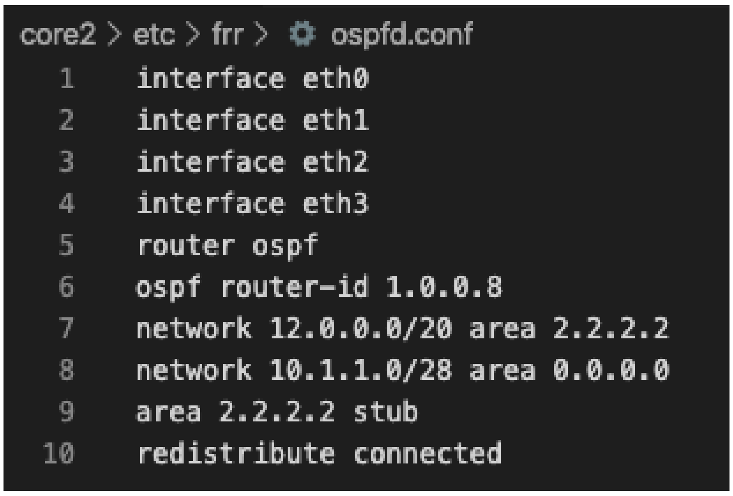


core1 > etc > frr > ospfd.conf

```
interface eth0
interface eth1
interface eth2
interface eth3
router ospf
ospf router-id 1.0.0.7
network 11.0.0.0/12 area 1.1.1.1
network 10.1.1.0/28 area 0.0.0.0
area 1.1.1.1 stub
redistribute connected
```

Figure 1.10: Core01 Ospf configuration

- CORE-02 ospf configuration
  Core-02 is inside two areas, Area-2 as-well as Backbone area hence we make the below configuration



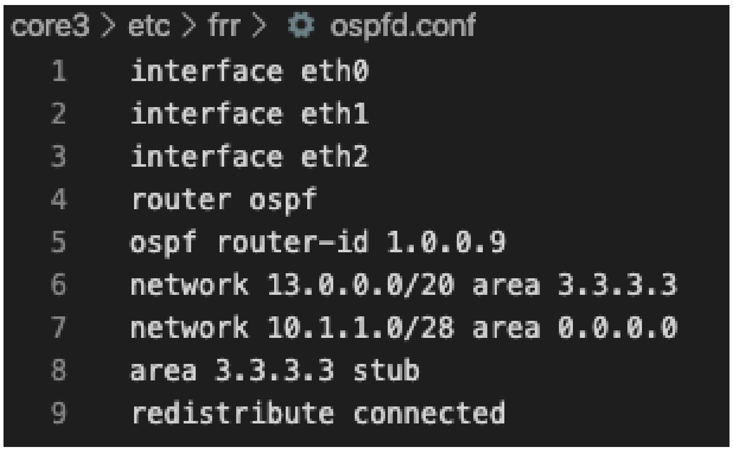
core2 > etc > frr > ospfd.conf

```
interface eth0
interface eth1
interface eth2
interface eth3
router ospf
ospf router-id 1.0.0.8
network 12.0.0.0/20 area 2.2.2.2
network 10.1.1.0/28 area 0.0.0.0
area 2.2.2.2 stub
redistribute connected
```

Figure 1.11: Core02 Ospf configuration

- CORE-03 ospf configuration

ore-03 is inside two areas, Area-3 as-well as Backbone area hence we have the below configuration



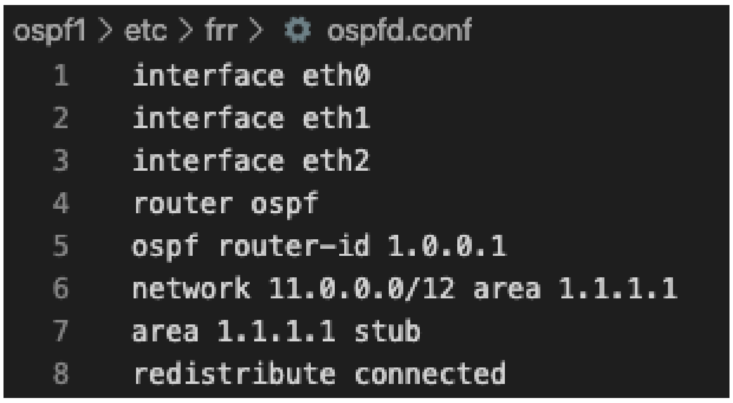
core3 > etc > frr > ospfd.conf

```
interface eth0
interface eth1
interface eth2
router ospf
ospf router-id 1.0.0.9
network 13.0.0.0/20 area 3.3.3.3
network 10.1.1.0/28 area 0.0.0.0
area 3.3.3.3 stub
redistribute connected
```

Figure 1.12: Core03 Ospf configuration

OSPF-01, OSPF-02 And OSPF-03 are inside one stub area with AREA-id 1.1.1.1 hence we have the below configurations.

- Ospf-01 ospf configuration OSPF-01 is inside one stub area with AREA-id 1.1.1.1 hence we have the below configurations.



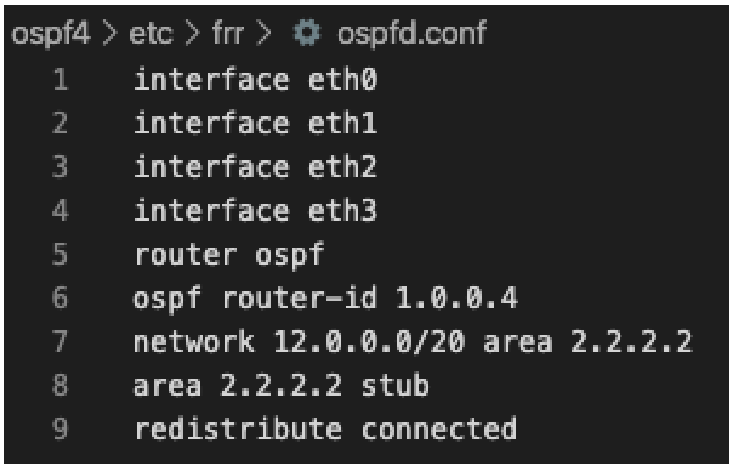
ospf1 > etc > frr > ospfd.conf

```
interface eth0
interface eth1
interface eth2
router ospf
ospf router-id 1.0.0.1
network 11.0.0.0/12 area 1.1.1.1
area 1.1.1.1 stub
redistribute connected
```

Figure 1.13: ospf01 Ospf configuration as an example for ospfd and ospf2 and ospf3 will have the same area and same way of configuration except there router ids

OSPF-04, OSPF-05 And OSPF-06 are inside another stub area with AREA-id 2.2.2.2 hence we have the below configurations.

- ospf-04 ospf configuration OSPF-04 is inside another stub area with AREA-id 2.2.2.2 hence we have the below configurations.

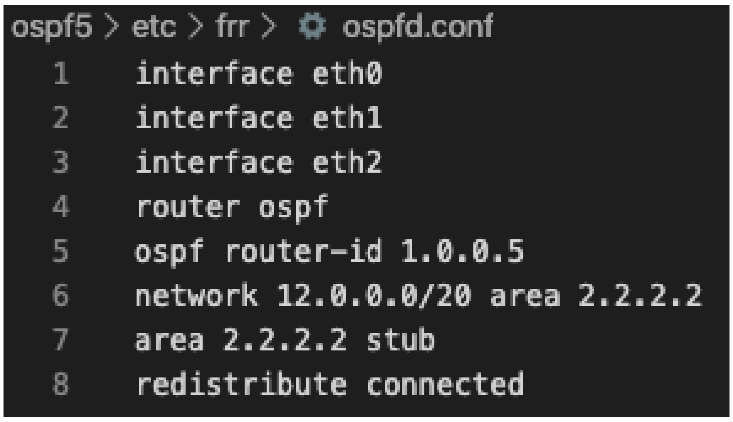


ospf4 > etc > frr > ospfd.conf

```
interface eth0
interface eth1
interface eth2
interface eth3
router ospf
ospf router-id 1.0.0.4
network 12.0.0.0/20 area 2.2.2.2
area 2.2.2.2 stub
redistribute connected
```

Figure 1.14: ospf04 Ospf configuration as an example for ospfd and ospf5 and ospf6 will have the same area and same way of configuration except there router ids

- ospf-05 ospf configuration OSPF-05 is inside another stub area with AREA-id 2.2.2.2 hence we have the below configurations.

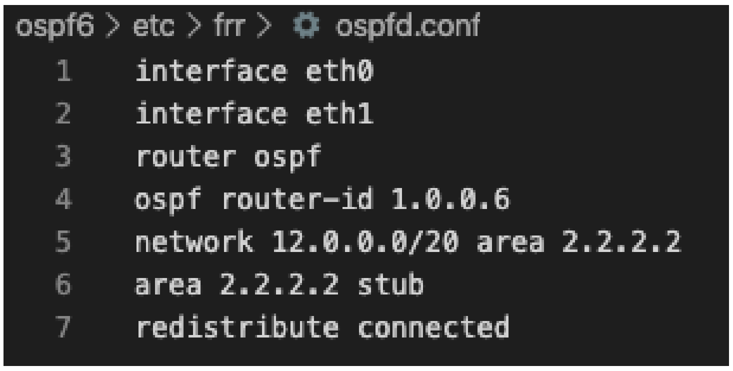


ospf5 > etc > frr > ospfd.conf

```
interface eth0
interface eth1
interface eth2
router ospf
ospf router-id 1.0.0.5
network 12.0.0.0/20 area 2.2.2.2
area 2.2.2.2 stub
redistribute connected
```

Figure 1.15: ospf05 Ospf configuration

- ospf-06 ospf configuration OSPF-06 is inside another stub area with AREA-id 2.2.2.2 hence we have the below configurations.



```
ospf6 > etc > frr > ospfd.conf
interface eth0
interface eth1
router ospf
ospf router-id 1.0.0.6
network 12.0.0.0/20 area 2.2.2.2
area 2.2.2.2 stub
redistribute connected
```

Figure 1.16: ospf06 Ospf configuration

Explain the concept of different areas in OSPF. What is the purpose to use different areas
OSPF stands for Open Shortest Path First is a routing protocol commonly routing protocol used all over the world. It is a type of interior gateway dynamic routing protocol. Interior gateway protocol is used within an organisation also known as "autonomous system". Dynamic routing protocol is nothing but the protocol which decides the best path for forwarding packets between source and destination networks
OSPF uses areas to simplify administration and optimize traffic and resource utilization.
An area is simply a logical grouping of contiguous networks and routers. All routers in the same area have the same topology table and don't know about routers in the other areas.
The main benefits of using areas in an OSPF network are

- The routing tables on the routers are reduced.
- less time is required to run the SPF algorithm, since routers need to recalculate their link-state database only when there's a topology change within their own area.
- Routing updates are reduced.

Each area in an OSPF network must be connected to the backbone area(also known as area 0).
All routers inside an area must have the same area ID in order to become OSPF neighbors.
A router that has interfaces in more than one area (area 0 and area 1, and area 3) is known as an Area Border Router (ABR). A router that connects an OSPF network to other routing domains

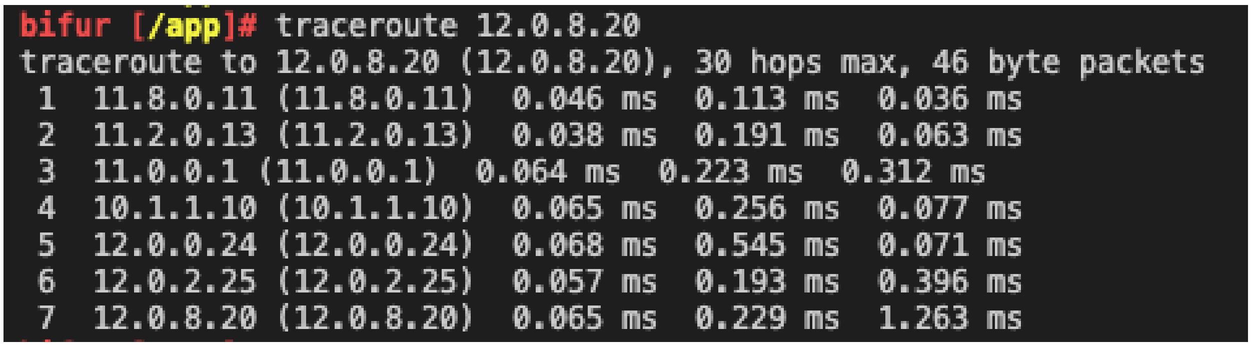
1.3 With wire shark start to capture traffic on CD Y

In this section we are supposed to capture traffic on collision domain Y. There is a problem in capturing by wireshark in mac book from Kathara lab so I tried to use tcpdump but unfortunately the image doesnot recognise tcpdump. hence we could not attach the required screenshot in this section.

1.4 From bifur, run a trace route to balin, Determine wether the path includes ospf2 or ospf3 and ospf4 or ospf6

In this section we will try to trace route from bifur to balin and determine whether the path includes the mentioned OSPF.
For doing the trace route from bifur to balin we connect to bifur first and then we use the command traceroute and the destination IP address

- traceroute 12.0.8.20



```
bifur [/app]# traceroute 12.0.8.20
traceroute to 12.0.8.20 (12.0.8.20), 30 hops max, 46 byte packets
 1  11.8.0.11 (11.8.0.11)  0.046 ms  0.113 ms  0.036 ms
 2  11.2.0.13 (11.2.0.13)  0.038 ms  0.191 ms  0.063 ms
 3  11.0.0.1 (11.0.0.1)  0.064 ms  0.223 ms  0.312 ms
 4  10.1.1.10 (10.1.1.10)  0.065 ms  0.256 ms  0.077 ms
 5  12.0.0.24 (12.0.0.24)  0.068 ms  0.545 ms  0.071 ms
 6  12.0.2.25 (12.0.2.25)  0.057 ms  0.193 ms  0.396 ms
 7  12.0.8.20 (12.0.8.20)  0.065 ms  0.229 ms  1.263 ms
```

Figure 1.17: Trace route from bifur to balin

When we trace route from bifur to balin we can see it goes through series of routers that also shown below.
so, when we trace route from bifur first it goes to OSPF-01 then ospf01 router foreword's the packet to OSPF-03 then it is been forwarded to CORE-01 And then Core-02 And then the packet is forwarded to OSPF-04 and then to OSPF-05 and finally it reaches balin
so we can confirm that the path includes**SPF3 and OSPF6**

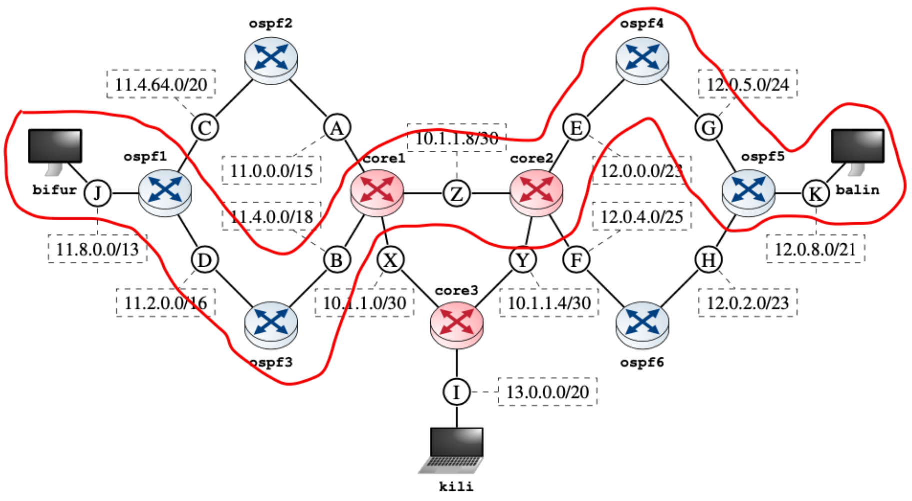
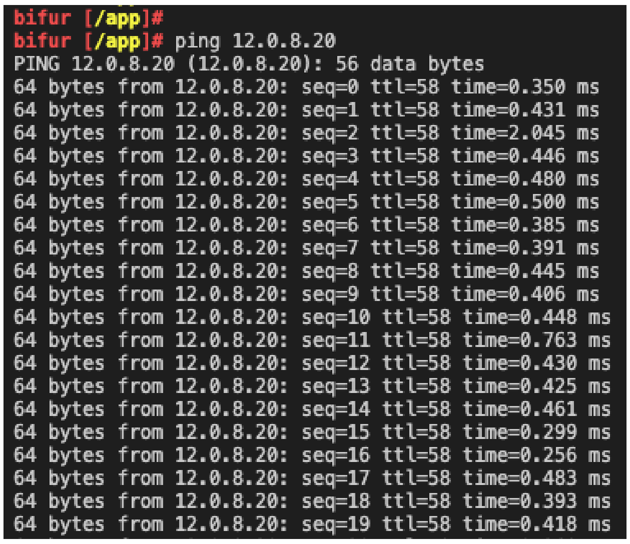


Figure 1.18: Packet flow from bifur to balin with ospf

1.5 Now, start to continuously ping balin from bifur

In this session we start to ping balin from bifur and we wont stop until we shut down the eth0 on core1

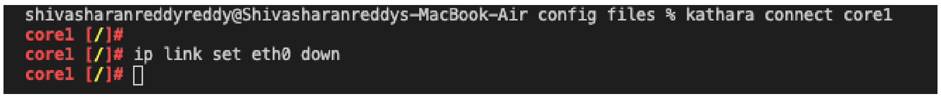


```
bifur [/app]#
bifur [/app]# ping 12.0.8.20
PING 12.0.8.20 (12.0.8.20): 56 data bytes
64 bytes from 12.0.8.20: seq=0 ttl=58 time=0.350 ms
64 bytes from 12.0.8.20: seq=1 ttl=58 time=0.431 ms
64 bytes from 12.0.8.20: seq=2 ttl=58 time=2.045 ms
64 bytes from 12.0.8.20: seq=3 ttl=58 time=0.446 ms
64 bytes from 12.0.8.20: seq=4 ttl=58 time=0.480 ms
64 bytes from 12.0.8.20: seq=5 ttl=58 time=0.500 ms
64 bytes from 12.0.8.20: seq=6 ttl=58 time=0.385 ms
64 bytes from 12.0.8.20: seq=7 ttl=58 time=0.391 ms
64 bytes from 12.0.8.20: seq=8 ttl=58 time=0.445 ms
64 bytes from 12.0.8.20: seq=9 ttl=58 time=0.406 ms
64 bytes from 12.0.8.20: seq=10 ttl=58 time=0.448 ms
64 bytes from 12.0.8.20: seq=11 ttl=58 time=0.763 ms
64 bytes from 12.0.8.20: seq=12 ttl=58 time=0.430 ms
64 bytes from 12.0.8.20: seq=13 ttl=58 time=0.425 ms
64 bytes from 12.0.8.20: seq=14 ttl=58 time=0.461 ms
64 bytes from 12.0.8.20: seq=15 ttl=58 time=0.299 ms
64 bytes from 12.0.8.20: seq=16 ttl=58 time=0.256 ms
64 bytes from 12.0.8.20: seq=17 ttl=58 time=0.483 ms
64 bytes from 12.0.8.20: seq=18 ttl=58 time=0.393 ms
64 bytes from 12.0.8.20: seq=19 ttl=58 time=0.418 ms
```

Figure 1.19: continuously ping balin from bifur

1.6 Disconnect the interface eth0 of the core1

In this section we simultaneously connect core1 and shut down the eth0 on core1, when the ping is still going on from bifur to balin as shown in below figure.



```
shivasharanreddyreddy@Shivasharanreddys-MacBook-Air config files % kathara connect core1
core1 [/]#
core1 [/]# ip link set eth0 down
core1 [/]# 
```

Figure 1.20: shutting down eth0 on core1

1.7 Now,OSPF should update the routing tables .Examine the OSPF messages captured on CD Y to answer the following questions

- **How quickly are OSPF messages sent after the interface was shutdown.**
  before shutting down the interface the ttl of the packets was ttl=58 but after the interface was shutdown the ttl is ttl=57 as shown in the figure

- **How many OSPF messages are sent?**
  The total OSPF message sent can be calculated by seeing the capture but there is a problem in MAC and windows devices were we are not able to capture

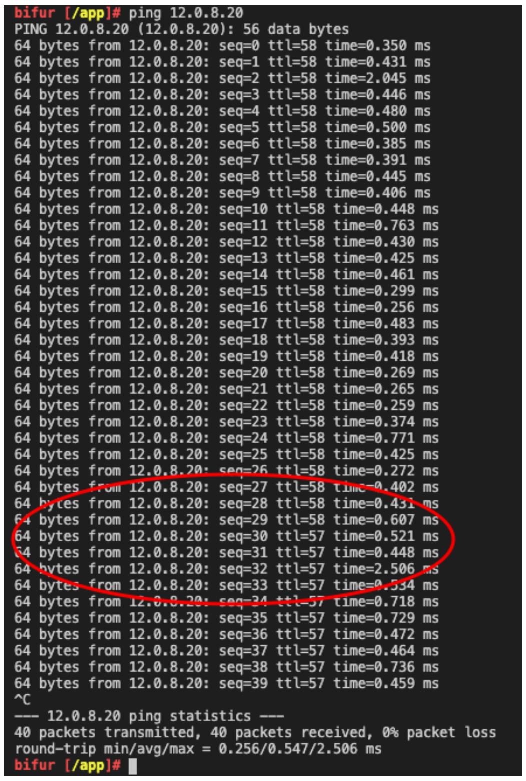
- **Which type of OSPF packet is used for flooding link state information?**
  ospf packets type 4

- **Which transport protocol is used?**
  OSPF does not carry data via a transport protocol, such as the User Data gram Protocol (UDP) or the Transmission Control Protocol (TCP),instead, OSPF forms IP data grams directly

- **What is the destination address of OSPF packets?**
  destination address of ospf packet is 12.0.8.20

1.8 Wait until the ttl (Time to Live) value of ping changed, then, stop the capture and save its output.

In this section we will capture the output of the ping from bifur to balin, when interface on core1 is shut down the ttl valve of ping changes, hence we capture that output and save it as shown in the below screenshot.



```
bifur [/app]# ping 12.0.8.20
PING 12.0.8.20 (12.0.8.20): 56 data bytes
64 bytes from 12.0.8.20: seq=0 ttl=58 time=0.350 ms
64 bytes from 12.0.8.20: seq=1 ttl=58 time=0.431 ms
64 bytes from 12.0.8.20: seq=2 ttl=58 time=2.045 ms
64 bytes from 12.0.8.20: seq=3 ttl=58 time=0.446 ms
64 bytes from 12.0.8.20: seq=4 ttl=58 time=0.480 ms
64 bytes from 12.0.8.20: seq=5 ttl=58 time=0.500 ms
64 bytes from 12.0.8.20: seq=6 ttl=58 time=0.385 ms
64 bytes from 12.0.8.20: seq=7 ttl=58 time=0.391 ms
64 bytes from 12.0.8.20: seq=8 ttl=58 time=0.445 ms
64 bytes from 12.0.8.20: seq=9 ttl=58 time=0.406 ms
64 bytes from 12.0.8.20: seq=10 ttl=58 time=0.448 ms
64 bytes from 12.0.8.20: seq=11 ttl=58 time=0.763 ms
64 bytes from 12.0.8.20: seq=12 ttl=58 time=0.430 ms
64 bytes from 12.0.8.20: seq=13 ttl=58 time=0.425 ms
64 bytes from 12.0.8.20: seq=14 ttl=58 time=0.461 ms
64 bytes from 12.0.8.20: seq=15 ttl=58 time=0.299 ms
64 bytes from 12.0.8.20: seq=16 ttl=58 time=0.256 ms
64 bytes from 12.0.8.20: seq=17 ttl=58 time=0.483 ms
64 bytes from 12.0.8.20: seq=18 ttl=58 time=0.393 ms
64 bytes from 12.0.8.20: seq=19 ttl=58 time=0.418 ms
64 bytes from 12.0.8.20: seq=20 ttl=58 time=0.269 ms
64 bytes from 12.0.8.20: seq=21 ttl=58 time=0.265 ms
64 bytes from 12.0.8.20: seq=22 ttl=58 time=0.259 ms
64 bytes from 12.0.8.20: seq=23 ttl=58 time=0.374 ms
64 bytes from 12.0.8.20: seq=24 ttl=58 time=0.771 ms
64 bytes from 12.0.8.20: seq=25 ttl=58 time=0.425 ms
64 bytes from 12.0.8.20: seq=26 ttl=58 time=0.272 ms
64 bytes from 12.0.8.20: seq=27 ttl=58 time=0.402 ms
64 bytes from 12.0.8.20: seq=28 ttl=58 time=0.431 ms
64 bytes from 12.0.8.20: seq=29 ttl=58 time=0.607 ms
64 bytes from 12.0.8.20: seq=30 ttl=57 time=0.521 ms
64 bytes from 12.0.8.20: seq=31 ttl=57 time=0.448 ms
64 bytes from 12.0.8.20: seq=32 ttl=57 time=2.506 ms
64 bytes from 12.0.8.20: seq=33 ttl=57 time=0.534 ms
64 bytes from 12.0.8.20: seq=34 ttl=57 time=0.718 ms
64 bytes from 12.0.8.20: seq=35 ttl=57 time=0.729 ms
64 bytes from 12.0.8.20: seq=36 ttl=57 time=0.472 ms
64 bytes from 12.0.8.20: seq=37 ttl=57 time=0.464 ms
64 bytes from 12.0.8.20: seq=38 ttl=57 time=0.736 ms
64 bytes from 12.0.8.20: seq=39 ttl=57 time=0.459 ms
^C
--- 12.0.8.20 ping statistics ---
40 packets transmitted, 40 packets received, 0% packet loss
round-trip min/avg/max = 0.256/0.547/2.506 ms
bifur [/app]#
```

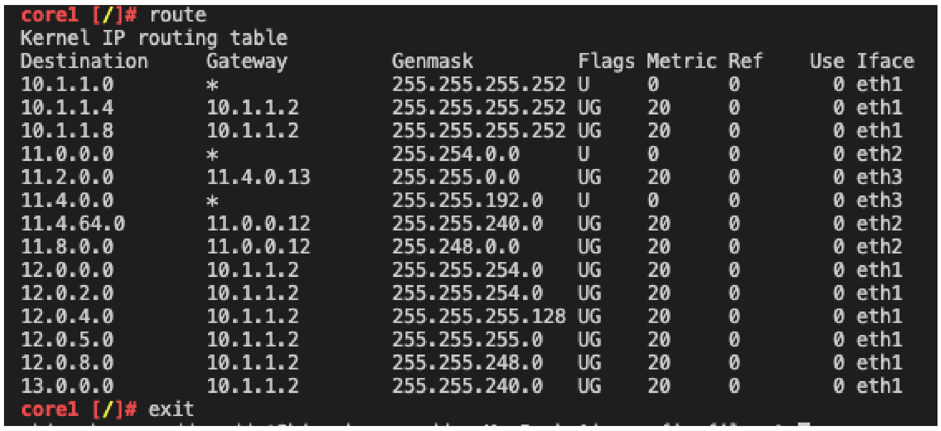
Figure 1.21: Changed ttl value when interface was shut down in eth0

1.9 Save and compare the routing tables of the core routers. Are they all identical?

In this section we capture all the routing table entries and compare them. we have three screenshots of core routers core1,core2 and core3 to compare with as shown below

- Core1 routing table

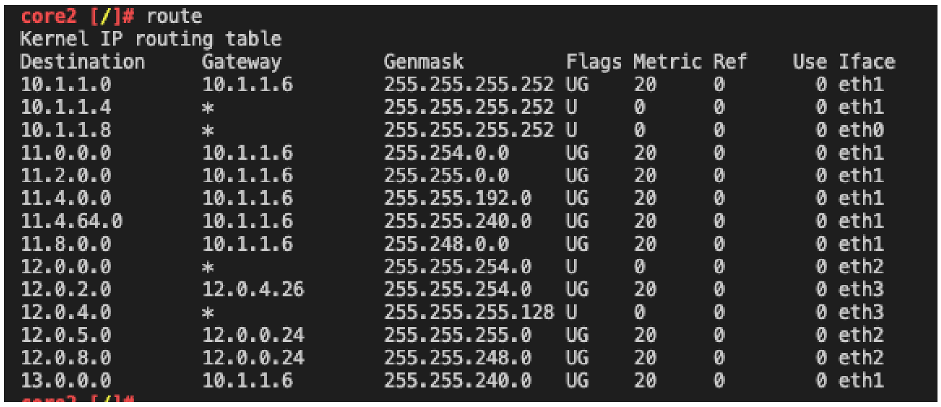
when we look at the routing table of core1 we can see that there is not entry of interface eth0 because we have disconnected it.



```
core1 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.1.0        *               255.255.255.252 U     0      0        0 eth1
10.1.1.4        10.1.1.2        255.255.255.252 UG    20     0        0 eth1
10.1.1.8        10.1.1.2        255.255.255.252 UG    20     0        0 eth1
11.0.0.0        *               255.254.0.0     U     0      0        0 eth2
11.2.0.0        11.4.0.13       255.255.0.0     UG    20     0        0 eth3
11.4.0.0        *               255.255.192.0   U     0      0        0 eth3
11.4.64.0       11.0.0.12       255.255.240.0   UG    20     0        0 eth2
11.8.0.0        11.0.0.12       255.248.0.0     UG    20     0        0 eth2
12.0.0.0        10.1.1.2        255.255.254.0   UG    20     0        0 eth1
12.0.2.0        10.1.1.2        255.255.254.0   UG    20     0        0 eth1
12.0.4.0        10.1.1.2        255.255.255.128 UG    20     0        0 eth1
12.0.5.0        10.1.1.2        255.255.255.0   UG    20     0        0 eth1
12.0.8.0        10.1.1.2        255.255.248.0   UG    20     0        0 eth1
13.0.0.0        10.1.1.2        255.255.240.0   UG    20     0        0 eth1
core1 [/]# exit
```

Figure 1.22: Core1 routing Table entries

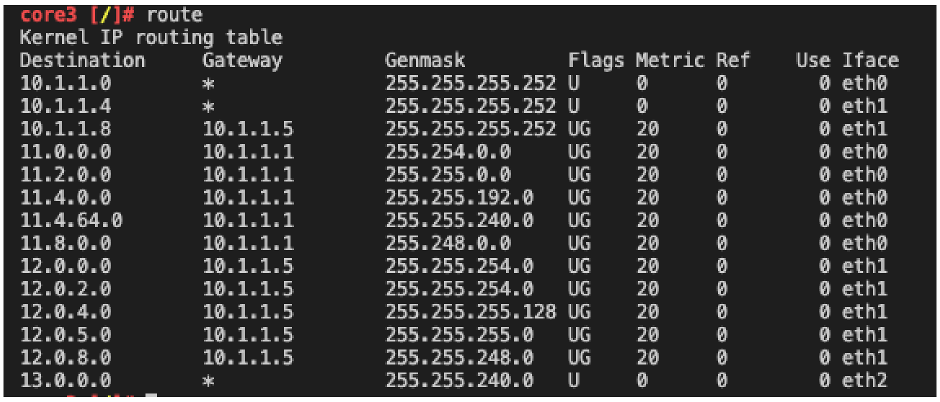
- Core2 routing table
  when we look at routing entries of core2 we can see that we have an entry of interface eth0 because we have not disconnected it.



```
core2 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.1.0        10.1.1.6        255.255.255.252 UG    20     0        0 eth1
10.1.1.4        *               255.255.255.252 U     0      0        0 eth1
10.1.1.8        *               255.255.255.252 U     0      0        0 eth0
11.0.0.0        10.1.1.6        255.254.0.0     UG    20     0        0 eth1
11.2.0.0        10.1.1.6        255.255.0.0     UG    20     0        0 eth1
11.4.0.0        10.1.1.6        255.255.192.0   UG    20     0        0 eth1
11.4.64.0       10.1.1.6        255.255.240.0   UG    20     0        0 eth1
11.8.0.0        10.1.1.6        255.248.0.0     UG    20     0        0 eth1
12.0.0.0        *               255.255.254.0   U     0      0        0 eth2
12.0.2.0        12.0.4.26       255.255.254.0   UG    20     0        0 eth3
12.0.4.0        *               255.255.255.128 U     0      0        0 eth3
12.0.5.0        12.0.0.24       255.255.255.0   UG    20     0        0 eth2
12.0.8.0        12.0.0.24       255.255.248.0   UG    20     0        0 eth2
13.0.0.0        10.1.1.6        255.255.240.0   UG    20     0        0 eth1
```

Figure 1.23: Core2 routing Table entries

- core3 routing table
  when we look at routing entries of core3 we can see all the interfaces because we have not made any changes on core 3.



```
core3 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.1.1.0        *               255.255.255.252 U     0      0        0 eth0
10.1.1.4        *               255.255.255.252 U     0      0        0 eth1
10.1.1.8        10.1.1.5        255.255.255.252 UG    20     0        0 eth1
11.0.0.0        10.1.1.1        255.254.0.0     UG    20     0        0 eth0
11.2.0.0        10.1.1.1        255.255.0.0     UG    20     0        0 eth0
11.4.0.0        10.1.1.1        255.255.192.0   UG    20     0        0 eth0
11.4.64.0       10.1.1.1        255.255.240.0   UG    20     0        0 eth0
11.8.0.0        10.1.1.1        255.248.0.0     UG    20     0        0 eth0
12.0.0.0        10.1.1.5        255.255.254.0   UG    20     0        0 eth1
12.0.2.0        10.1.1.5        255.255.254.0   UG    20     0        0 eth1
12.0.4.0        10.1.1.5        255.255.255.128 UG    20     0        0 eth1
12.0.5.0        10.1.1.5        255.255.255.0   UG    20     0        0 eth1
12.0.8.0        10.1.1.5        255.255.248.0   UG    20     0        0 eth1
13.0.0.0        *               255.255.240.0   U     0      0        0 eth2
```

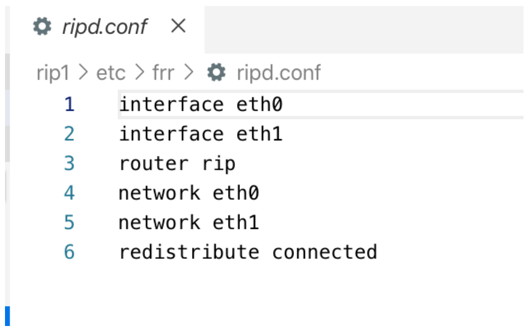
Figure 1.24: Core3 routing Table entries

When we compare all the three core routers, all of them are not similar because we have shut down interface between core1 and core3 i.e eth0 on core1 so core1 will not show the entry of core1 Hence when compared all the route tables and conclude that they are not identical.

Chapter 2

Dynamic Routing with RIP

In this section we will use **Routing information protocol** (RIP) instead of OSPF. RIP is a dynamic routing protocol which uses hop count as a routing metric to find the best path between the source and the destination network. In contrast to OSPF, in RIP We do not need to assign id or area to routers we just need to assign each router its network interfaces as we can see in 2.1. Now we are going to go through RIP step by step to check how easier it is than OSPF.

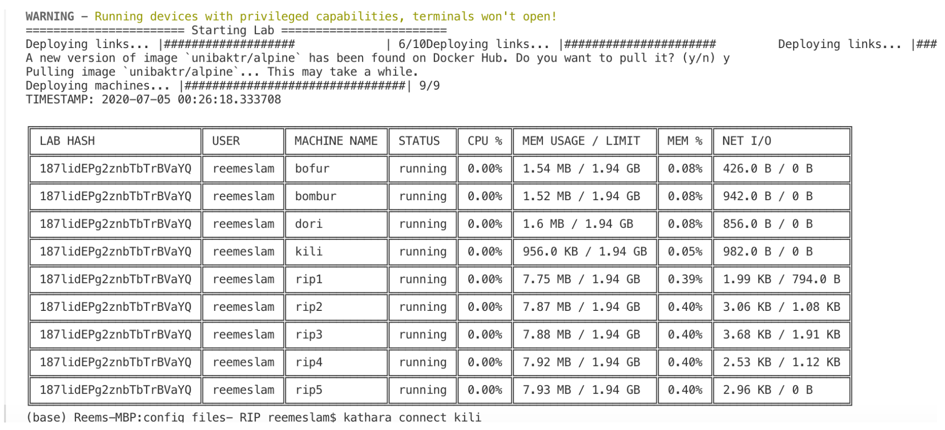


```
interface eth0
interface eth1
router rip
network eth0
network eth1
redistribute connected
```

Figure 2.1: RIP conf file for router rip1

2.1 Deploying The topology and ensuring connectivity between hosts

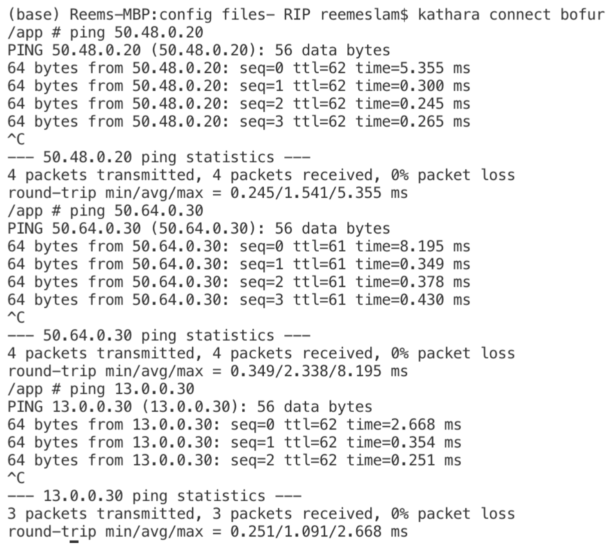
2.1.1 Deploying the topology



```
WARNING - Running devices with privileged capabilities, terminals won't open!
====================== Starting Lab ======================
Deploying links... |###################            | 6/10Deploying links... |######################        Deploying links... |###
A new version of image `unibaktr/alpine` has been found on Docker Hub. Do you want to pull it? (y/n) y
Pulling image `unibaktr/alpine`... This may take a while.
Deploying machines... |###############################| 9/9
TIMESTAMP: 2020-07-05 00:26:18.333708

| LAB HASH             | USER      | MACHINE NAME | STATUS  | CPU % | MEM USAGE / LIMIT  | MEM % | NET I/O            |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | bofur      | running | 0.00% | 1.54 MB / 1.94 GB  | 0.08% | 426.0 B / 0 B      |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | bombur     | running | 0.00% | 1.52 MB / 1.94 GB  | 0.08% | 942.0 B / 0 B      |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | dori       | running | 0.00% | 1.6 MB / 1.94 GB   | 0.08% | 856.0 B / 0 B      |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | kili       | running | 0.00% | 956.0 KB / 1.94 GB | 0.05% | 982.0 B / 0 B      |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | rip1       | running | 0.00% | 7.75 MB / 1.94 GB  | 0.39% | 1.99 KB / 794.0 B  |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | rip2       | running | 0.00% | 7.87 MB / 1.94 GB  | 0.40% | 3.06 KB / 1.08 KB  |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | rip3       | running | 0.00% | 7.88 MB / 1.94 GB  | 0.40% | 3.68 KB / 1.91 KB  |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | rip4       | running | 0.00% | 7.92 MB / 1.94 GB  | 0.40% | 2.53 KB / 1.12 KB  |
| 187lidEPg2znbTbTrBVaYQ | reemeslam | rip5       | running | 0.00% | 7.93 MB / 1.94 GB  | 0.40% | 2.96 KB / 0 B      |
(base) Reems-MBP:config files- RIP reemeslam$ kathara connect kili
```

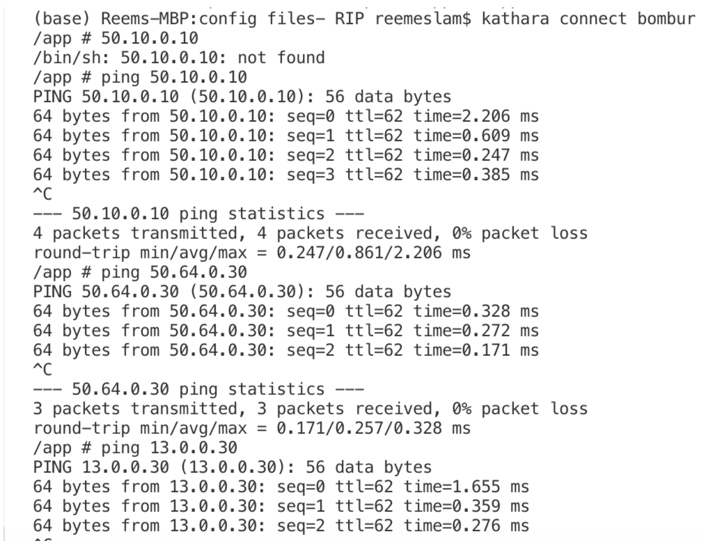
Figure 2.2: Deploying The topology of the network using Rip with the FR-Routing framework.

2.2 ensuring connectivity between bofur, bombur, kili and dori.



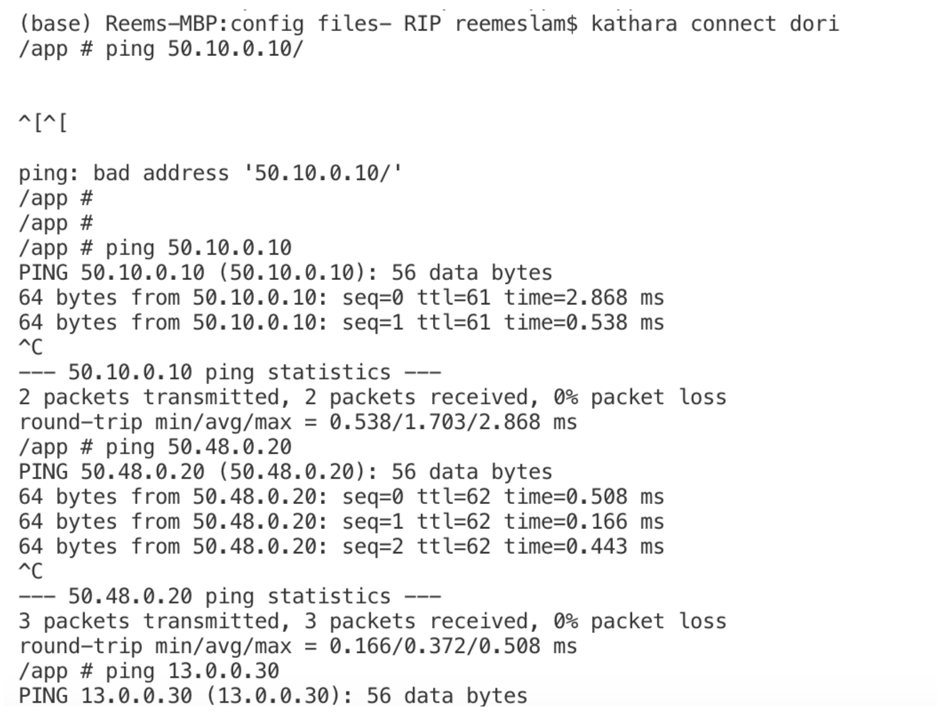
```
(base) Reems-MBP:config files- RIP reemeslam$ kathara connect bofur
/app # ping 50.48.0.20
PING 50.48.0.20 (50.48.0.20): 56 data bytes
64 bytes from 50.48.0.20: seq=0 ttl=62 time=5.355 ms
64 bytes from 50.48.0.20: seq=1 ttl=62 time=0.300 ms
64 bytes from 50.48.0.20: seq=2 ttl=62 time=0.245 ms
64 bytes from 50.48.0.20: seq=3 ttl=62 time=0.265 ms
^C
--- 50.48.0.20 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.245/1.541/5.355 ms
/app # ping 50.64.0.30
PING 50.64.0.30 (50.64.0.30): 56 data bytes
64 bytes from 50.64.0.30: seq=0 ttl=61 time=8.195 ms
64 bytes from 50.64.0.30: seq=1 ttl=61 time=0.349 ms
64 bytes from 50.64.0.30: seq=2 ttl=61 time=0.378 ms
64 bytes from 50.64.0.30: seq=3 ttl=61 time=0.430 ms
^C
--- 50.64.0.30 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.349/2.338/8.195 ms
/app # ping 13.0.0.30
PING 13.0.0.30 (13.0.0.30): 56 data bytes
64 bytes from 13.0.0.30: seq=0 ttl=62 time=2.668 ms
64 bytes from 13.0.0.30: seq=1 ttl=62 time=0.354 ms
64 bytes from 13.0.0.30: seq=2 ttl=62 time=0.251 ms
^C
--- 13.0.0.30 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.251/1.091/2.668 ms
```

Figure 2.3: Pinging from Bofur to other hosts.



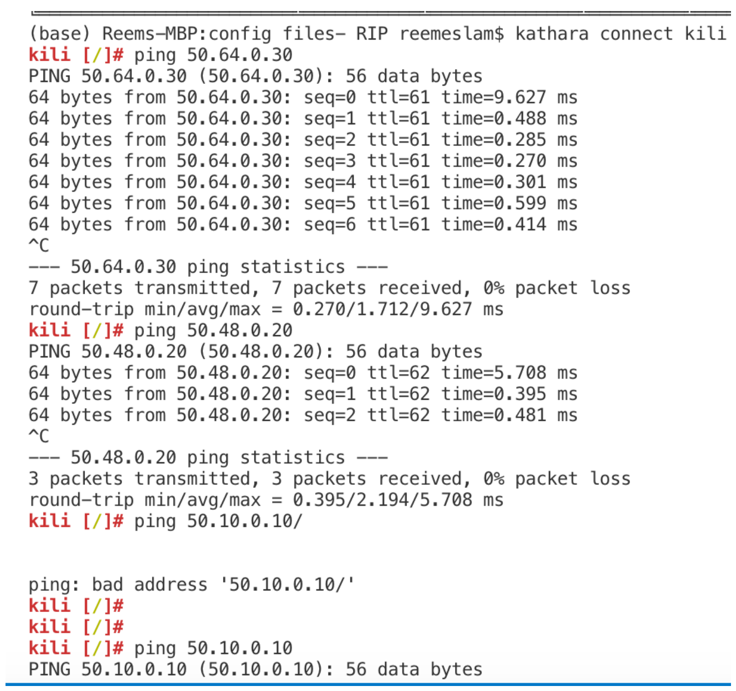
```
(base) Reems-MBP:config files- RIP reemeslam$ kathara connect bombur
/app # 50.10.0.10
/bin/sh: 50.10.0.10: not found
/app # ping 50.10.0.10
PING 50.10.0.10 (50.10.0.10): 56 data bytes
64 bytes from 50.10.0.10: seq=0 ttl=62 time=2.206 ms
64 bytes from 50.10.0.10: seq=1 ttl=62 time=0.609 ms
64 bytes from 50.10.0.10: seq=2 ttl=62 time=0.247 ms
64 bytes from 50.10.0.10: seq=3 ttl=62 time=0.385 ms
^C
--- 50.10.0.10 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.247/0.861/2.206 ms
/app # ping 50.64.0.30
PING 50.64.0.30 (50.64.0.30): 56 data bytes
64 bytes from 50.64.0.30: seq=0 ttl=62 time=0.328 ms
64 bytes from 50.64.0.30: seq=1 ttl=62 time=0.272 ms
64 bytes from 50.64.0.30: seq=2 ttl=62 time=0.171 ms
^C
--- 50.64.0.30 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.171/0.257/0.328 ms
/app # ping 13.0.0.30
PING 13.0.0.30 (13.0.0.30): 56 data bytes
64 bytes from 13.0.0.30: seq=0 ttl=62 time=1.655 ms
64 bytes from 13.0.0.30: seq=1 ttl=62 time=0.359 ms
64 bytes from 13.0.0.30: seq=2 ttl=62 time=0.276 ms
```

Figure 2.4: Pinging from Bombur to other hosts.



```
(base) Reems-MBP:config files- RIP reemeslam$ kathara connect dori
/app # ping 50.10.0.10/


^[^[

ping: bad address '50.10.0.10/'
/app #
/app #
/app # ping 50.10.0.10
PING 50.10.0.10 (50.10.0.10): 56 data bytes
64 bytes from 50.10.0.10: seq=0 ttl=61 time=2.868 ms
64 bytes from 50.10.0.10: seq=1 ttl=61 time=0.538 ms
^C
--- 50.10.0.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.538/1.703/2.868 ms
/app # ping 50.48.0.20
PING 50.48.0.20 (50.48.0.20): 56 data bytes
64 bytes from 50.48.0.20: seq=0 ttl=62 time=0.508 ms
64 bytes from 50.48.0.20: seq=1 ttl=62 time=0.166 ms
64 bytes from 50.48.0.20: seq=2 ttl=62 time=0.443 ms
^C
--- 50.48.0.20 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.166/0.372/0.508 ms
/app # ping 13.0.0.30
PING 13.0.0.30 (13.0.0.30): 56 data bytes
```

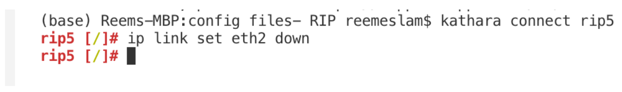
Figure 2.5: Pinging from Dori to other hosts.



```
(base) Reems-MBP:config files- RIP reemeslam$ kathara connect kili
kili [/]# ping 50.64.0.30
PING 50.64.0.30 (50.64.0.30): 56 data bytes
64 bytes from 50.64.0.30: seq=0 ttl=61 time=9.627 ms
64 bytes from 50.64.0.30: seq=1 ttl=61 time=0.488 ms
64 bytes from 50.64.0.30: seq=2 ttl=61 time=0.285 ms
64 bytes from 50.64.0.30: seq=3 ttl=61 time=0.270 ms
64 bytes from 50.64.0.30: seq=4 ttl=61 time=0.301 ms
64 bytes from 50.64.0.30: seq=5 ttl=61 time=0.599 ms
64 bytes from 50.64.0.30: seq=6 ttl=61 time=0.414 ms
^C
--- 50.64.0.30 ping statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 0.270/1.712/9.627 ms
kili [/]# ping 50.48.0.20
PING 50.48.0.20 (50.48.0.20): 56 data bytes
64 bytes from 50.48.0.20: seq=0 ttl=62 time=5.708 ms
64 bytes from 50.48.0.20: seq=1 ttl=62 time=0.395 ms
64 bytes from 50.48.0.20: seq=2 ttl=62 time=0.481 ms
^C
--- 50.48.0.20 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.395/2.194/5.708 ms
kili [/]# ping 50.10.0.10/


ping: bad address '50.10.0.10/'
kili [/]#
kili [/]#
kili [/]# ping 50.10.0.10
PING 50.10.0.10 (50.10.0.10): 56 data bytes
```

Figure 2.6: Pinging from Kili to other hosts.

2.3 Disabling eth2 interface of rip5



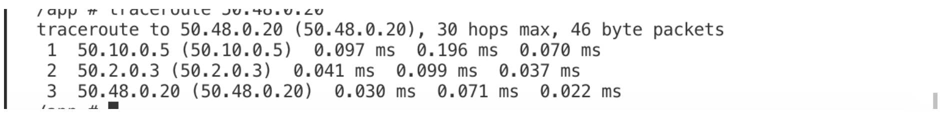
```
(base) Reems-MBP:config files- RIP reemeslam$ kathara connect rip5
rip5 [/]# ip link set eth2 down
rip5 [/]# █
```

Figure 2.7: Shutting down eth2 in Rip5.

2.4 Wireshark capture on CD O

There is a problem in capturing by wireshark in macbook from Kathara lab so I tried to use tcpdump but unfortunately the image doesnot recognise tcpdump.

2.5 Now lets traceroute to explore the route path between hosts using RIP

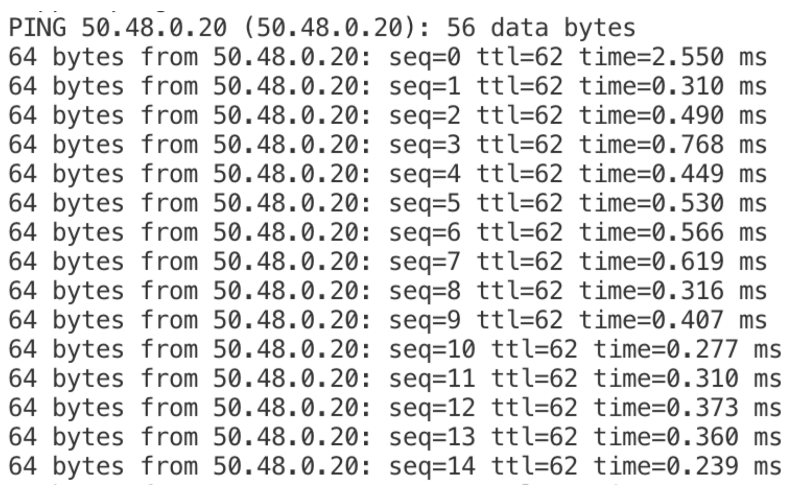


```
/app # traceroute 50.48.0.20
traceroute to 50.48.0.20 (50.48.0.20), 30 hops max, 46 byte packets
 1  50.10.0.5 (50.10.0.5)  0.097 ms  0.196 ms  0.070 ms
 2  50.2.0.3 (50.2.0.3)  0.041 ms  0.099 ms  0.037 ms
 3  50.48.0.20 (50.48.0.20)  0.030 ms  0.071 ms  0.022 ms
/app # █
```

Figure 2.8: Traceroute from bofur to bombur.

As we can see from figure 2.8 RIP uses hop count to figure out the path that has lowest number of routers between source and destination.

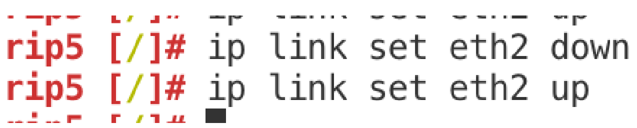
2.6 Start pinging from bofur to bombur



```
PING 50.48.0.20 (50.48.0.20): 56 data bytes
64 bytes from 50.48.0.20: seq=0 ttl=62 time=2.550 ms
64 bytes from 50.48.0.20: seq=1 ttl=62 time=0.310 ms
64 bytes from 50.48.0.20: seq=2 ttl=62 time=0.490 ms
64 bytes from 50.48.0.20: seq=3 ttl=62 time=0.768 ms
64 bytes from 50.48.0.20: seq=4 ttl=62 time=0.449 ms
64 bytes from 50.48.0.20: seq=5 ttl=62 time=0.530 ms
64 bytes from 50.48.0.20: seq=6 ttl=62 time=0.566 ms
64 bytes from 50.48.0.20: seq=7 ttl=62 time=0.619 ms
64 bytes from 50.48.0.20: seq=8 ttl=62 time=0.316 ms
64 bytes from 50.48.0.20: seq=9 ttl=62 time=0.407 ms
64 bytes from 50.48.0.20: seq=10 ttl=62 time=0.277 ms
64 bytes from 50.48.0.20: seq=11 ttl=62 time=0.310 ms
64 bytes from 50.48.0.20: seq=12 ttl=62 time=0.373 ms
64 bytes from 50.48.0.20: seq=13 ttl=62 time=0.360 ms
64 bytes from 50.48.0.20: seq=14 ttl=62 time=0.239 ms
```

Figure 2.9: Start pinging from bofur to bombur.

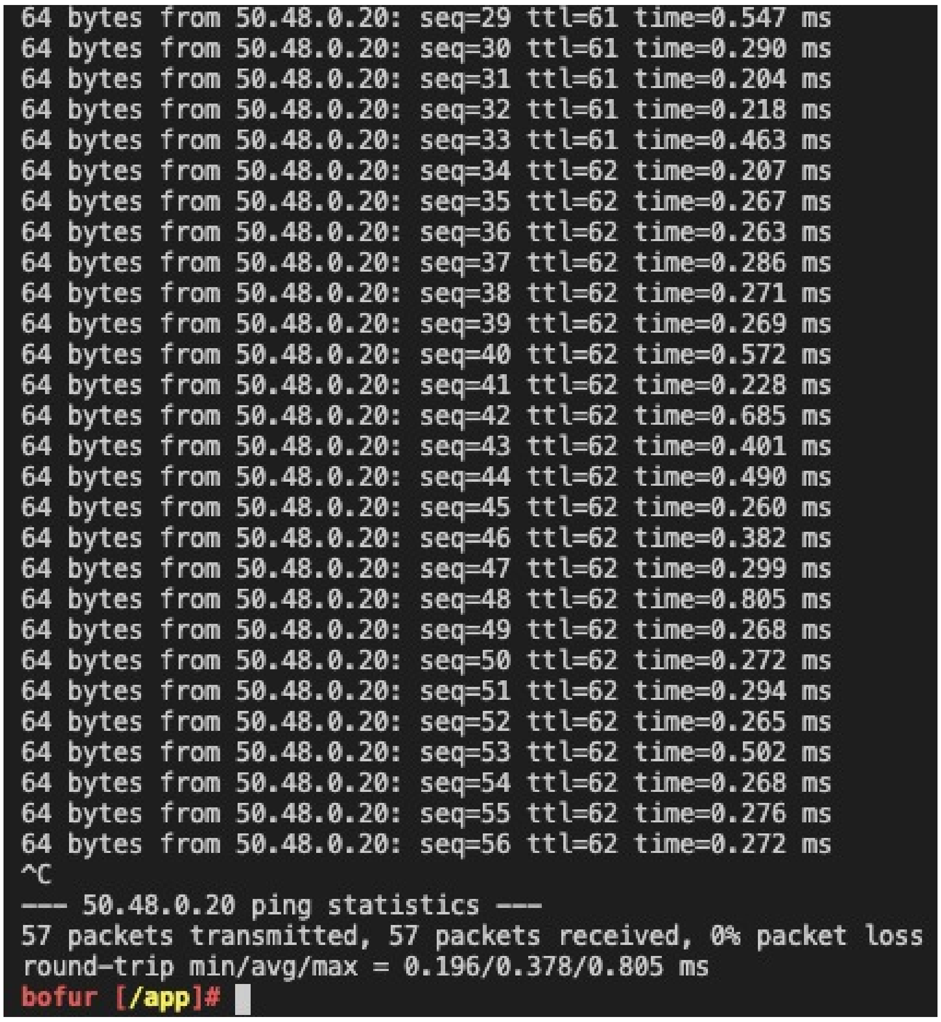
2.7 Enable eth2 again in rip5



```
rip5 [/]# ip link set eth2 down
rip5 [/]# ip link set eth2 up
```

Figure 2.10: Enabling eth2 in rip5 again.

2.7.1 Checking change in TTL



```
64 bytes from 50.48.0.20: seq=29 ttl=61 time=0.547 ms
64 bytes from 50.48.0.20: seq=30 ttl=61 time=0.290 ms
64 bytes from 50.48.0.20: seq=31 ttl=61 time=0.204 ms
64 bytes from 50.48.0.20: seq=32 ttl=61 time=0.218 ms
64 bytes from 50.48.0.20: seq=33 ttl=61 time=0.463 ms
64 bytes from 50.48.0.20: seq=34 ttl=62 time=0.207 ms
64 bytes from 50.48.0.20: seq=35 ttl=62 time=0.267 ms
64 bytes from 50.48.0.20: seq=36 ttl=62 time=0.263 ms
64 bytes from 50.48.0.20: seq=37 ttl=62 time=0.286 ms
64 bytes from 50.48.0.20: seq=38 ttl=62 time=0.271 ms
64 bytes from 50.48.0.20: seq=39 ttl=62 time=0.269 ms
64 bytes from 50.48.0.20: seq=40 ttl=62 time=0.572 ms
64 bytes from 50.48.0.20: seq=41 ttl=62 time=0.228 ms
64 bytes from 50.48.0.20: seq=42 ttl=62 time=0.685 ms
64 bytes from 50.48.0.20: seq=43 ttl=62 time=0.401 ms
64 bytes from 50.48.0.20: seq=44 ttl=62 time=0.490 ms
64 bytes from 50.48.0.20: seq=45 ttl=62 time=0.260 ms
64 bytes from 50.48.0.20: seq=46 ttl=62 time=0.382 ms
64 bytes from 50.48.0.20: seq=47 ttl=62 time=0.299 ms
64 bytes from 50.48.0.20: seq=48 ttl=62 time=0.805 ms
64 bytes from 50.48.0.20: seq=49 ttl=62 time=0.268 ms
64 bytes from 50.48.0.20: seq=50 ttl=62 time=0.272 ms
64 bytes from 50.48.0.20: seq=51 ttl=62 time=0.294 ms
64 bytes from 50.48.0.20: seq=52 ttl=62 time=0.265 ms
64 bytes from 50.48.0.20: seq=53 ttl=62 time=0.502 ms
64 bytes from 50.48.0.20: seq=54 ttl=62 time=0.268 ms
64 bytes from 50.48.0.20: seq=55 ttl=62 time=0.276 ms
64 bytes from 50.48.0.20: seq=56 ttl=62 time=0.272 ms
^C
--- 50.48.0.20 ping statistics ---
57 packets transmitted, 57 packets received, 0% packet loss
round-trip min/avg/max = 0.196/0.378/0.805 ms
bofur [/app]#
```

Figure 2.11: Enabling eth2 in rip5 again.

However Rip concept is easier than ospf as we do not need to define id or area for routers it took longer time for ttl to change. This is expected as rip will figure out everything by itself so it will take longer time to figure out the change.

2.8 Shutdown the link between rip5 and rip3 again by disabling the particular interface. What is the effect on the routing tables of the other routers? Show the routing tables of each router again. Can you still reach bombur from bofur? Is it the same path?

In this section we disable the link between rip5 and rip3 that is cD o and on rip5 it is eth2 and on rip3 its is eth1

- On RIP5 we disable eth2

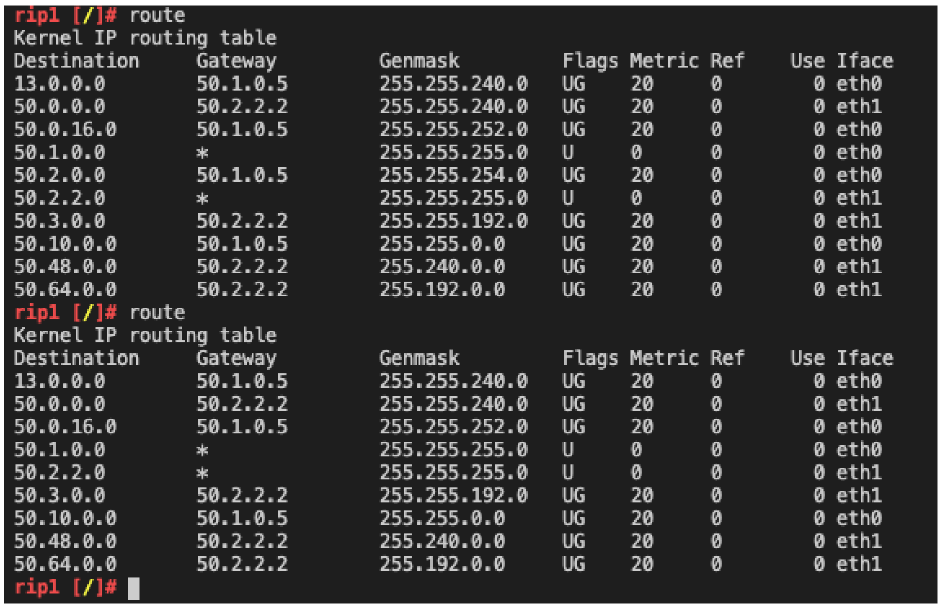
- On RIP3 we disable eth1

we disable the interfaces with thw following commnds on rip5 and rip3

- ip link set eht2 down

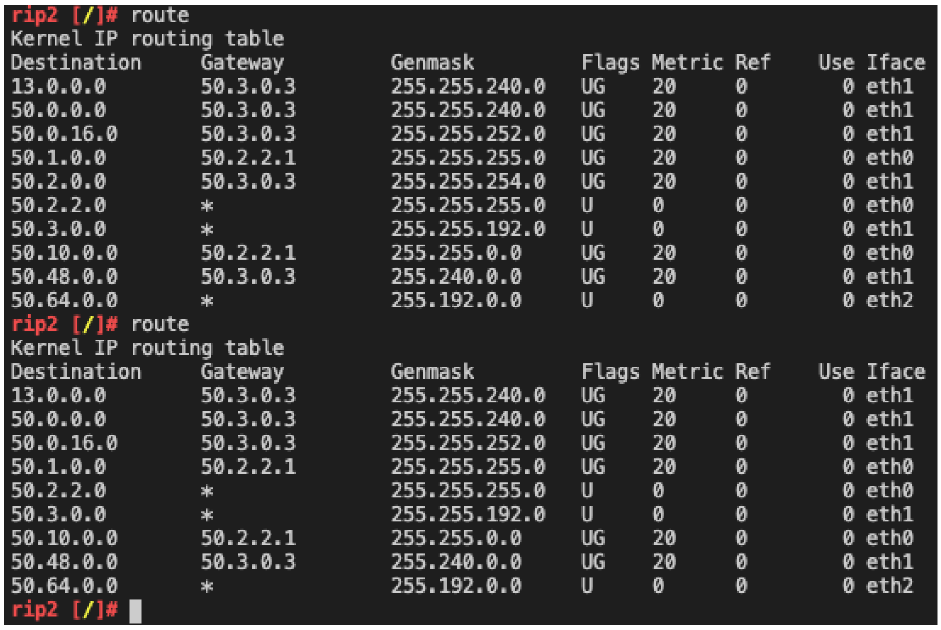
- ip link set eth1 down

After shut down the particular interface on both the routers we check the effect on other routers like rip1,rip2 and rip4 routing tables as shown below.



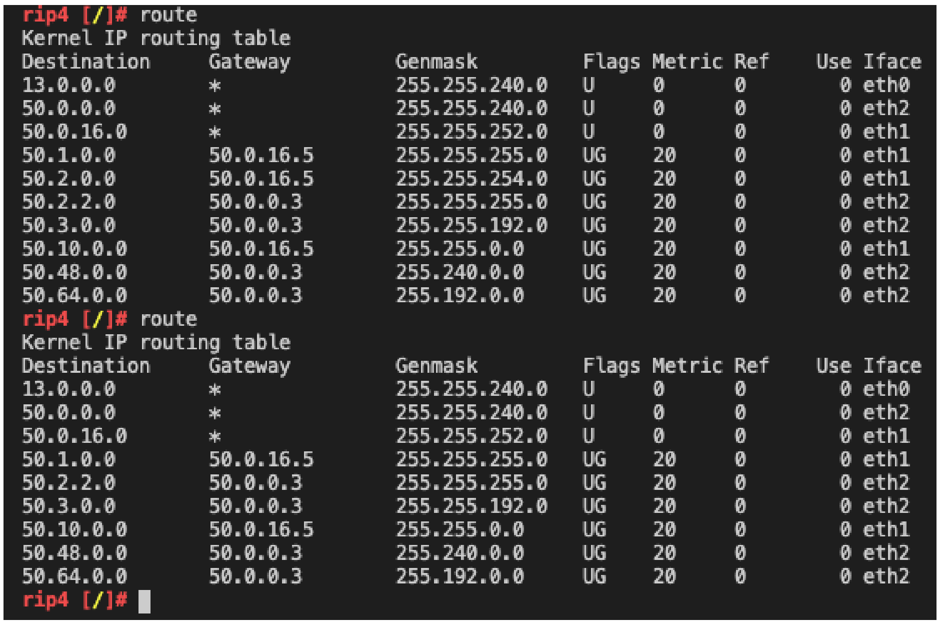
```
rip1 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
13.0.0.0        50.1.0.5        255.255.240.0   UG    20     0        0 eth0
50.0.0.0        50.2.2.2        255.255.240.0   UG    20     0        0 eth1
50.0.16.0       50.1.0.5        255.255.252.0   UG    20     0        0 eth0
50.1.0.0        *               255.255.255.0   U     0      0        0 eth0
50.2.0.0        50.1.0.5        255.255.254.0   UG    20     0        0 eth0
50.2.2.0        *               255.255.255.0   U     0      0        0 eth1
50.3.0.0        50.2.2.2        255.255.192.0   UG    20     0        0 eth1
50.10.0.0       50.1.0.5        255.255.0.0     UG    20     0        0 eth0
50.48.0.0       50.2.2.2        255.240.0.0     UG    20     0        0 eth1
50.64.0.0       50.2.2.2        255.192.0.0     UG    20     0        0 eth1
rip1 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
13.0.0.0        50.1.0.5        255.255.240.0   UG    20     0        0 eth0
50.0.0.0        50.2.2.2        255.255.240.0   UG    20     0        0 eth1
50.0.16.0       50.1.0.5        255.255.252.0   UG    20     0        0 eth0
50.1.0.0        *               255.255.255.0   U     0      0        0 eth0
50.2.2.0        *               255.255.255.0   U     0      0        0 eth1
50.3.0.0        50.2.2.2        255.255.192.0   UG    20     0        0 eth1
50.10.0.0       50.1.0.5        255.255.0.0     UG    20     0        0 eth0
50.48.0.0       50.2.2.2        255.240.0.0     UG    20     0        0 eth1
50.64.0.0       50.2.2.2        255.192.0.0     UG    20     0        0 eth1
rip1 [/]#
```

Figure 2.12: Before and after routes of RIP1 after shutting down the interface on rip5 and rip3



```
rip2 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
13.0.0.0        50.3.0.3        255.255.240.0   UG    20     0        0 eth1
50.0.0.0        50.3.0.3        255.255.240.0   UG    20     0        0 eth1
50.0.16.0       50.3.0.3        255.255.252.0   UG    20     0        0 eth1
50.1.0.0        50.2.2.1        255.255.255.0   UG    20     0        0 eth0
50.2.0.0        50.3.0.3        255.255.254.0   UG    20     0        0 eth1
50.2.2.0        *               255.255.255.0   U     0      0        0 eth0
50.3.0.0        *               255.255.192.0   U     0      0        0 eth1
50.10.0.0       50.2.2.1        255.255.0.0     UG    20     0        0 eth0
50.48.0.0       50.3.0.3        255.240.0.0     UG    20     0        0 eth1
50.64.0.0       *               255.192.0.0     U     0      0        0 eth2
rip2 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
13.0.0.0        50.3.0.3        255.255.240.0   UG    20     0        0 eth1
50.0.0.0        50.3.0.3        255.255.240.0   UG    20     0        0 eth1
50.0.16.0       50.3.0.3        255.255.252.0   UG    20     0        0 eth1
50.1.0.0        50.2.2.1        255.255.255.0   UG    20     0        0 eth0
50.2.2.0        *               255.255.255.0   U     0      0        0 eth0
50.3.0.0        *               255.255.192.0   U     0      0        0 eth1
50.10.0.0       50.2.2.1        255.255.0.0     UG    20     0        0 eth0
50.48.0.0       50.3.0.3        255.240.0.0     UG    20     0        0 eth1
50.64.0.0       *               255.192.0.0     U     0      0        0 eth2
rip2 [/]#
```

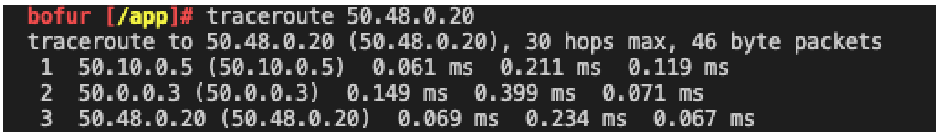
Figure 2.13: Before and after routes of RIP2 after shutting down the interface on rip5 and rip3



```
rip4 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
13.0.0.0        *               255.255.240.0   U     0      0        0 eth0
50.0.0.0        *               255.255.240.0   U     0      0        0 eth2
50.0.16.0       *               255.255.252.0   U     0      0        0 eth1
50.1.0.0        50.0.16.5       255.255.255.0   UG    20     0        0 eth1
50.2.0.0        50.0.16.5       255.255.254.0   UG    20     0        0 eth1
50.2.2.0        50.0.0.3        255.255.255.0   UG    20     0        0 eth2
50.3.0.0        50.0.0.3        255.255.192.0   UG    20     0        0 eth2
50.10.0.0       50.0.16.5       255.255.0.0     UG    20     0        0 eth1
50.48.0.0       50.0.0.3        255.240.0.0     UG    20     0        0 eth2
50.64.0.0       50.0.0.3        255.192.0.0     UG    20     0        0 eth2
rip4 [/]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
13.0.0.0        *               255.255.240.0   U     0      0        0 eth0
50.0.0.0        *               255.255.240.0   U     0      0        0 eth2
50.0.16.0       *               255.255.252.0   U     0      0        0 eth1
50.1.0.0        50.0.16.5       255.255.255.0   UG    20     0        0 eth1
50.2.2.0        50.0.0.3        255.255.255.0   UG    20     0        0 eth2
50.3.0.0        50.0.0.3        255.255.192.0   UG    20     0        0 eth2
50.10.0.0       50.0.16.5       255.255.0.0     UG    20     0        0 eth1
50.48.0.0       50.0.0.3        255.240.0.0     UG    20     0        0 eth2
50.64.0.0       50.0.0.3        255.192.0.0     UG    20     0        0 eth2
rip4 [/]# 
```

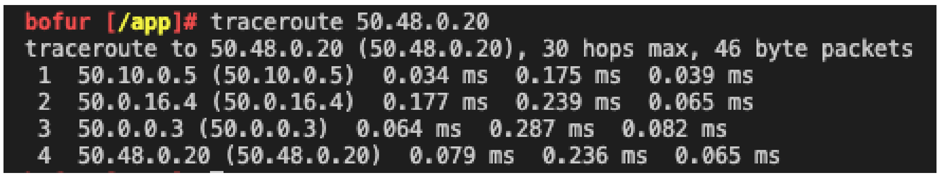
Figure 2.14: Before and after routes of RIP4 after shutting down the interface on rip5 and rip3

And yes, we can still reach bombur from bofur but from other path because of the shutting down of the path between rip5 and rip3 as shown below.



```
bofur [/app]# traceroute 50.48.0.20
traceroute to 50.48.0.20 (50.48.0.20), 30 hops max, 46 byte packets
 1  50.10.0.5 (50.10.0.5)  0.061 ms  0.211 ms  0.119 ms
 2  50.0.0.3 (50.0.0.3)  0.149 ms  0.399 ms  0.071 ms
 3  50.48.0.20 (50.48.0.20)  0.069 ms  0.234 ms  0.067 ms
```

Figure 2.15: Path before Interface was up between rip5 and rip3



```
bofur [/app]# traceroute 50.48.0.20
traceroute to 50.48.0.20 (50.48.0.20), 30 hops max, 46 byte packets
 1  50.10.0.5 (50.10.0.5)  0.034 ms  0.175 ms  0.039 ms
 2  50.0.16.4 (50.0.16.4)  0.177 ms  0.239 ms  0.065 ms
 3  50.0.0.3 (50.0.0.3)  0.064 ms  0.287 ms  0.082 ms
 4  50.48.0.20 (50.48.0.20)  0.079 ms  0.236 ms  0.065 ms
```

Figure 2.16: Path after we disconnect interfact between rip5 and rip3

fig:2.11

2.9 RIP and security

However Rip is simple and easy. Attacking it, is also easy as Whenever there is change in the network, running rip router will send any updates to its neighbour routers so if the attacker send fake rip response messages. The other routers will eventually insert the malicious entries in there tables and start broadcasting the changes to neighbours and that will make it easier to convergence to know the topology.