Homework-3: Networking - Report

CSE534 - Fundamentals of Computer Networks

- Bhavesh Goyal (111482386)

A1.

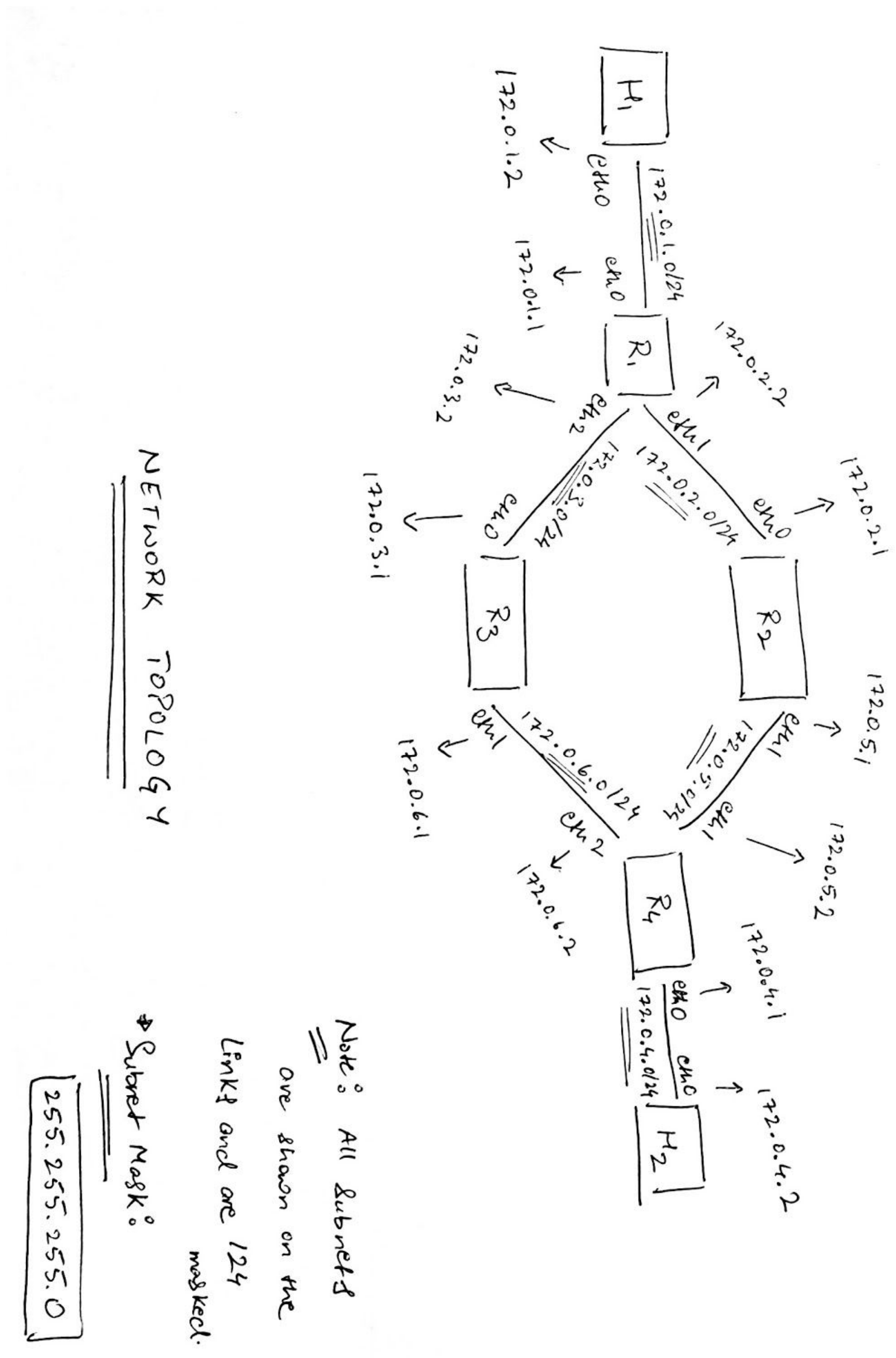


Fig A.1.1

A2.

The static routes were added by configuring Quagga topo and start files. First, the quagga hosts were assigned a default ip on their eth0 interface, while adding them to the host controller. The H1, and H2 were given the default gateway R1 and R2 respectively to forward all the default traffic to the connected routers. All the routers were configured on a different subnet and only the hosts and directly connected routers were set on the same network. Then, the quagga links were added to connect all the routers to each other according to the topology given above. Next, we had to configure all the routers, assigning ips to each of their network interfaces and connecting the neighbors to each other.

First, we enabled ip forwarding at each of the network components using sysctl command. Next, we assign a ip and subnet address to each non-default(non eth0) interface in the routers using

```
$> Ifconfig <Routerlabel>-<ifacelabel> [Iface IP/SubnetSuffix]
```

Here, we also verify that the adjacent routers are assigned a same subnet and follow the topology given above.

Next, we add static routes to the routing tables for each of the routers.
For each router we add,

```
$> ip route add [Destination IP Subnet] via [next hop(neighbour) to reach the destination] dev [Iface to reach the neighbour]
```

Thus, for each router we make sure every router has a route to reach every other subnet which is not directly connected to it. Note, we only add routes to the public interfaces (`eth0`) to each of the other routers and next we make sure that NAT rules are setup to forward packets between private(`eth1/eth2`) and public(`eth0`) interfaces.

For this, we configure iptables for each of the routers.

First we setup NAT so that each router masquerades its outgoing packets as if coming from its public interfaces. (In other words, change source address for all outgoing packets). This is done using

```
$> iptables -t nat -A POSTROUTING (For outgoing packets) -o
<Routerlabel-eth0> -j MASQUERADE
```

Then, we configure iptables to allow the routers to forward packets back and forth between public and private interfaces. We first allow all internal packets to traverse outside of the router through its public interface. This is done using

```
$>iptables -A FORWARD -i <Routerlabel-internalinterface> -o
<Routerlabel-outgoinginterface> -j ACCEPT
```

Since the router netfilter is a stateless machine, we also need to make sure to allow incoming traffic for already established connections. This is done using

```
$>iptables -A FORWARD -i <Routerlabel-publicinterface> -o
<Routerlabel-internalinterface> -m state --state ESTABLISHED,RELATED -j
ACCEPT
```

At this point, we have all the necessary NAT rules setup and we just need to enable forwarding on the router box. This is done using the sysctl call for each router.

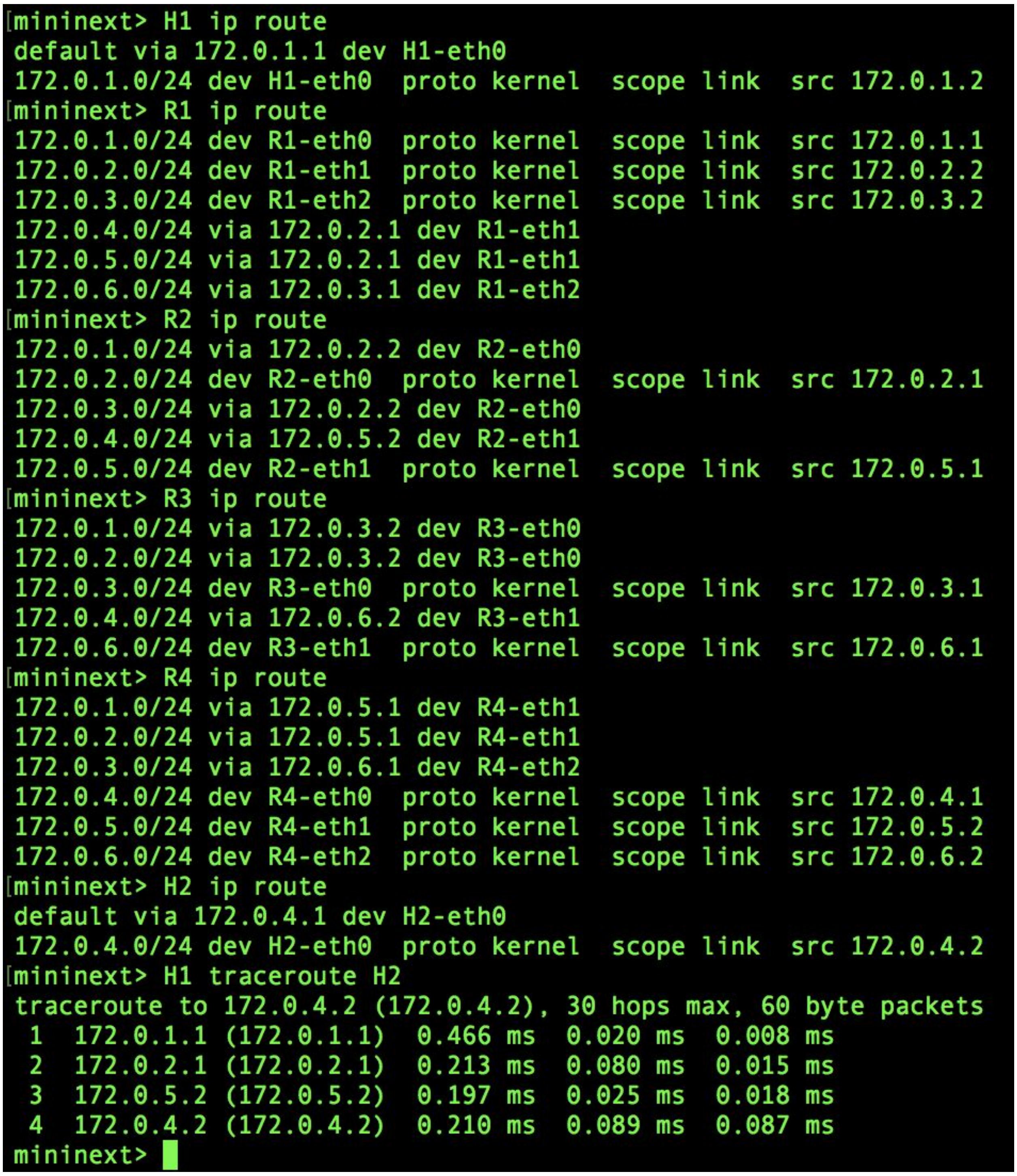
```
$> sysctl -w net.ipv4.ip_forward=1
```

(Alternatively, we could also manually set the forwarding by
"**`$> echo 1 > /proc/sys/net/ipv4/ip_forward`**" At each of the routers)

Now, if we take a look at the routing tables at each of the routers, they should look like **Fig A.1**

As we can see, the traceroute output is also consistent with the routing tables setup for our network.

Note: R1 Here, keeps a path from R2 to the network, in case link R1 R2 goes down, we could flush the iptables using iptables -f and then reload the script to allow R1 to forward all packets directed towards H2 from R3. The route for the same has also been added in the provided script.



```
[mininext> H1 ip route
default via 172.0.1.1 dev H1-eth0
172.0.1.0/24 dev H1-eth0  proto kernel  scope link  src 172.0.1.2
[mininext> R1 ip route
172.0.1.0/24 dev R1-eth0  proto kernel  scope link  src 172.0.1.1
172.0.2.0/24 dev R1-eth1  proto kernel  scope link  src 172.0.2.2
172.0.3.0/24 dev R1-eth2  proto kernel  scope link  src 172.0.3.2
172.0.4.0/24 via 172.0.2.1 dev R1-eth1
172.0.5.0/24 via 172.0.2.1 dev R1-eth1
172.0.6.0/24 via 172.0.3.1 dev R1-eth2
[mininext> R2 ip route
172.0.1.0/24 via 172.0.2.2 dev R2-eth0
172.0.2.0/24 dev R2-eth0  proto kernel  scope link  src 172.0.2.1
172.0.3.0/24 via 172.0.2.2 dev R2-eth0
172.0.4.0/24 via 172.0.5.2 dev R2-eth1
172.0.5.0/24 dev R2-eth1  proto kernel  scope link  src 172.0.5.1
[mininext> R3 ip route
172.0.1.0/24 via 172.0.3.2 dev R3-eth0
172.0.2.0/24 via 172.0.3.2 dev R3-eth0
172.0.3.0/24 dev R3-eth0  proto kernel  scope link  src 172.0.3.1
172.0.4.0/24 via 172.0.6.2 dev R3-eth1
172.0.6.0/24 dev R3-eth1  proto kernel  scope link  src 172.0.6.1
[mininext> R4 ip route
172.0.1.0/24 via 172.0.5.1 dev R4-eth1
172.0.2.0/24 via 172.0.5.1 dev R4-eth1
172.0.3.0/24 via 172.0.6.1 dev R4-eth2
172.0.4.0/24 dev R4-eth0  proto kernel  scope link  src 172.0.4.1
172.0.5.0/24 dev R4-eth1  proto kernel  scope link  src 172.0.5.2
172.0.6.0/24 dev R4-eth2  proto kernel  scope link  src 172.0.6.2
[mininext> H2 ip route
default via 172.0.4.1 dev H2-eth0
172.0.4.0/24 dev H2-eth0  proto kernel  scope link  src 172.0.4.2
[mininext> H1 traceroute H2
traceroute to 172.0.4.2 (172.0.4.2), 30 hops max, 60 byte packets
 1  172.0.1.1 (172.0.1.1)  0.466 ms  0.020 ms  0.008 ms
 2  172.0.2.1 (172.0.2.1)  0.213 ms  0.080 ms  0.015 ms
 3  172.0.5.2 (172.0.5.2)  0.197 ms  0.025 ms  0.018 ms
 4  172.0.4.2 (172.0.4.2)  0.210 ms  0.089 ms  0.087 ms
mininext> 
```

Fig - A.2.1

B1 & B2 *(Combined)*.

From the Quagga documentation, it was identified that the `'ripd'` daemon was responsible for handling RIP protocol. But, to enable dynamic changing of kernel routing table and redistribution of routes we also had to enable zebra daemon (that manages kernel routing table). Since, the zebra config file only contains static routing configuration (*Source: References [2]*) and we need to implement dynamic routing using RIP, we need to maintain network information in the ripd configuration file. For this, we follow the following steps:

Step 1: Enable ripd and zebra daemons in /etc/quagga/daemons.

- **`$>vi /etc/quagga/daemons`**. Edit the following lines to:
- **`zebra=yes`**
- **`ripd=yes`**

Step 2: Copy sample configuration file provided for zebra and rip daemons(We change this later to suit our topology)

- **`$>sudo cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf`**
- **`$>sudo cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf`**

Step 3: Next, we make sure that the quagga user and groups have access to the configuration file

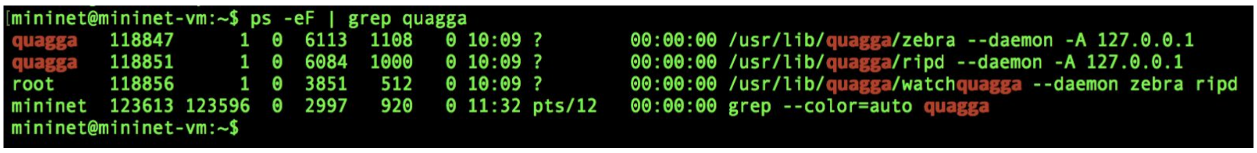
- **`$>chown quagga:quaggavty /etc/quagga/*.conf`**
- **`$>chmod 640 /etc/quagga/*.conf`**

Step 4: Restart the quagga service to trigger ripd and zebra daemons

- **`$>sudo /etc/init.d/quagga restart`**

Step 5: Verify that daemons were initiated if you see the process output similar to the *figure B.1*

- **`$>ps -eF | grep quagga`**



```
[mininet@mininet-vm:~$ ps -eF | grep quagga
quagga   118847      1  0  6113  1108   0 10:09 ?        00:00:00 /usr/lib/quagga/zebra --daemon -A 127.0.0.1
quagga   118851      1  0  6084  1000   0 10:09 ?        00:00:00 /usr/lib/quagga/ripd --daemon -A 127.0.0.1
root     118856      1  0  3851   512   0 10:09 ?        00:00:00 /usr/lib/quagga/watchquagga --daemon zebra ripd
mininet  123613 123596  0  2997   920   0 11:32 pts/12   00:00:00 grep --color=auto quagga
mininet@mininet-vm:~$
```

Fig B.1.1

Step 6: **[Important]** Copy the config files to mininet host local config path. Each host tries to read the quagga config from `configs/<host-label>`. So we copy all the quagga conf files to host local paths.

```
$>cp /etc/quagga/* ~/<path-to-quagga-ixp>/configs/H1
$>cp /etc/quagga/* ~/<path-to-quagga-ixp>/configs/H2
$>cp /etc/quagga/* ~/<path-to-quagga-ixp>/configs/R1
$>cp /etc/quagga/* ~/<path-to-quagga-ixp>/configs/R2
$>cp /etc/quagga/* ~/<path-to-quagga-ixp>/configs/R3
$>cp /etc/quagga/* ~/<path-to-quagga-ixp>/configs/R4
```

At this point, all hosts in mininet topology have sample configuration files and should provide access to the initiated daemons launched in quagga network. Next, we try to configure the daemons to suit our network topology.

Step 1: Initiate mininet using

- **`$>sudo python start.py -d`**

Note: The flag -d has been programmed to disable static routes in the mininet. Failure to provide the flag would automatically set the static routes and establish network connectivity without a need to run RIP daemon.

The code would first try to estimate convergence time by trying to ping from each host to another for the ripd daemon and then give a prompt for mininext shell.

Step 2: Verify the connectivity is not established using

- **`mininext$> pingall`**

NOTE: The nodes should only be able to ping their neighbours in this case. If not, please recheck Step 1.

Step 3: Telnet into router daemons using mininext shell binary (`mx`) included in the code. Here, for each host we try to configure rip by setting the neighbouring subnet network for each of the corresponding daemons.

For each host/router, do the following:

- **`root@mininet-vm$> ./mx <host-label>`** [Eg: H1 for Host1]
  You should get access to the router's CLI. Verify by doing an ifconfig and check if it matches with the <host-label>'s address.

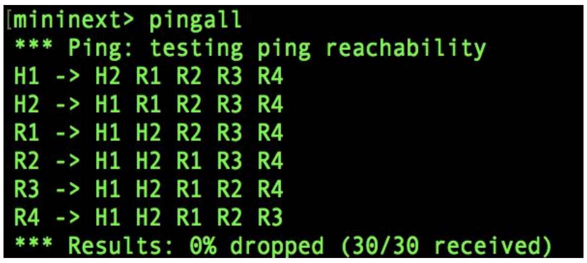
- **`root@mininet-vm$> telnet localhost 2602`**
  (Connect to ripd with `Password: zebra`)

- **`ripd> enable`** (Enable configuration commands)
- **`ripd# configure terminal`** (Go to configuration mode)
- **`ripd(config)# router rip`** (Enable rip and go to rip configuration)
- **`ripd(config-router)# network <Subnet Address>`** [Eg: 172.0.1.0/24 for H1]

  Similarly, add network for other neighbouring subnet for the current host (If Any). There should be one network entry for H1, H2; two for R2, R3; 3 network entries for R1, R4.

- **`ripd(config-router)# write`**
  (Writes the configuration file to `configs/<host-label>/ripd.conf`)
- **`ripd(config-router)# quit`**
- **`ripd(config)# quit`**
- **`ripd# quit`**
- **`root@mininet-vm$> exit`**
  (Exit mx shell and repeat the process for all other hosts)

Step 4: Now go to mininext shell and test RIP using pingall. Full network connectivity should be established

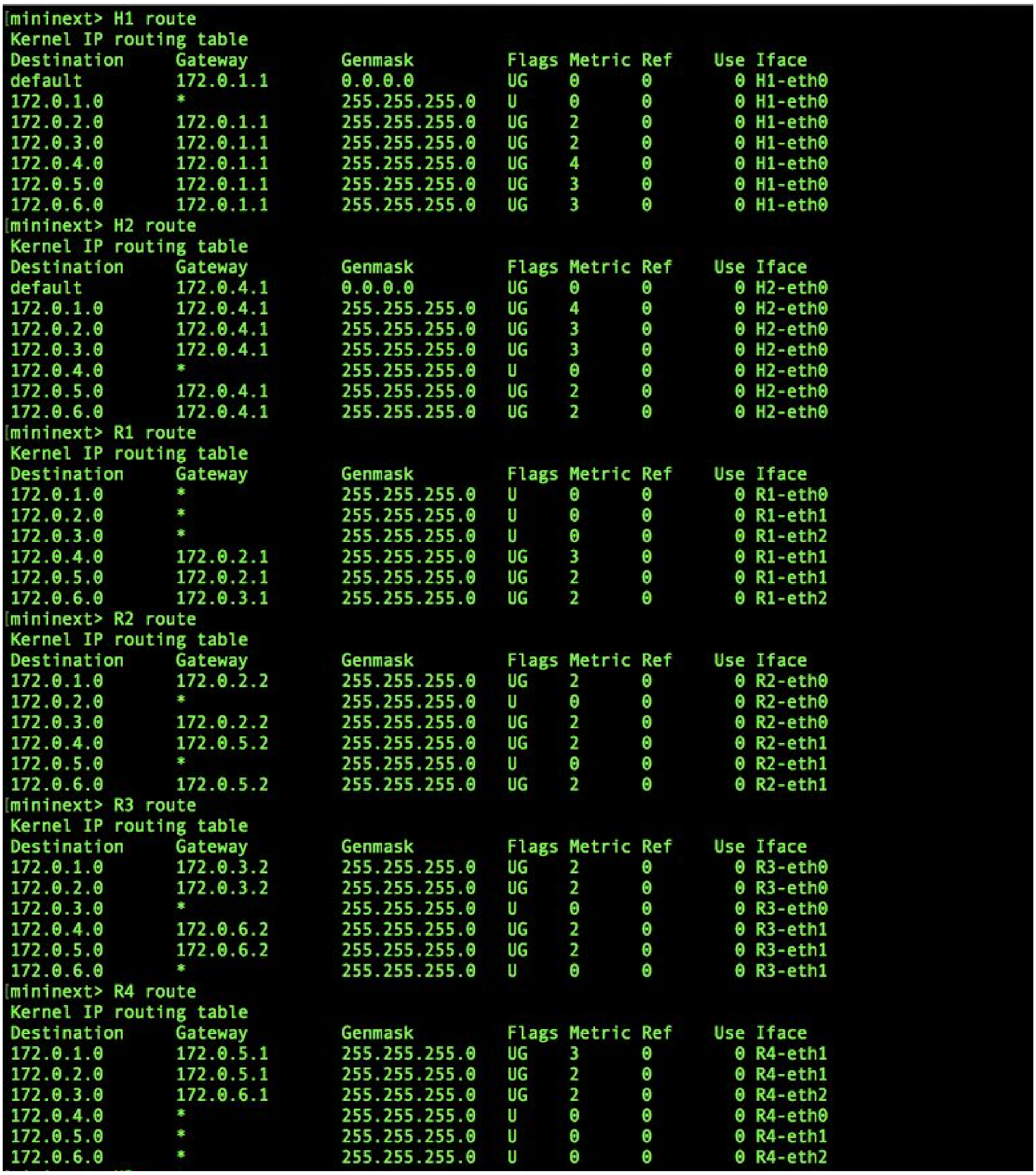


```
mininext> pingall
*** Ping: testing ping reachability
H1 -> H2 R1 R2 R3 R4
H2 -> H1 R1 R2 R3 R4
R1 -> H1 H2 R2 R3 R4
R2 -> H1 H2 R1 R3 R4
R3 -> H1 H2 R1 R2 R4
R4 -> H1 H2 R1 R2 R3
*** Results: 0% dropped (30/30 received)
```

Fig B.1.2

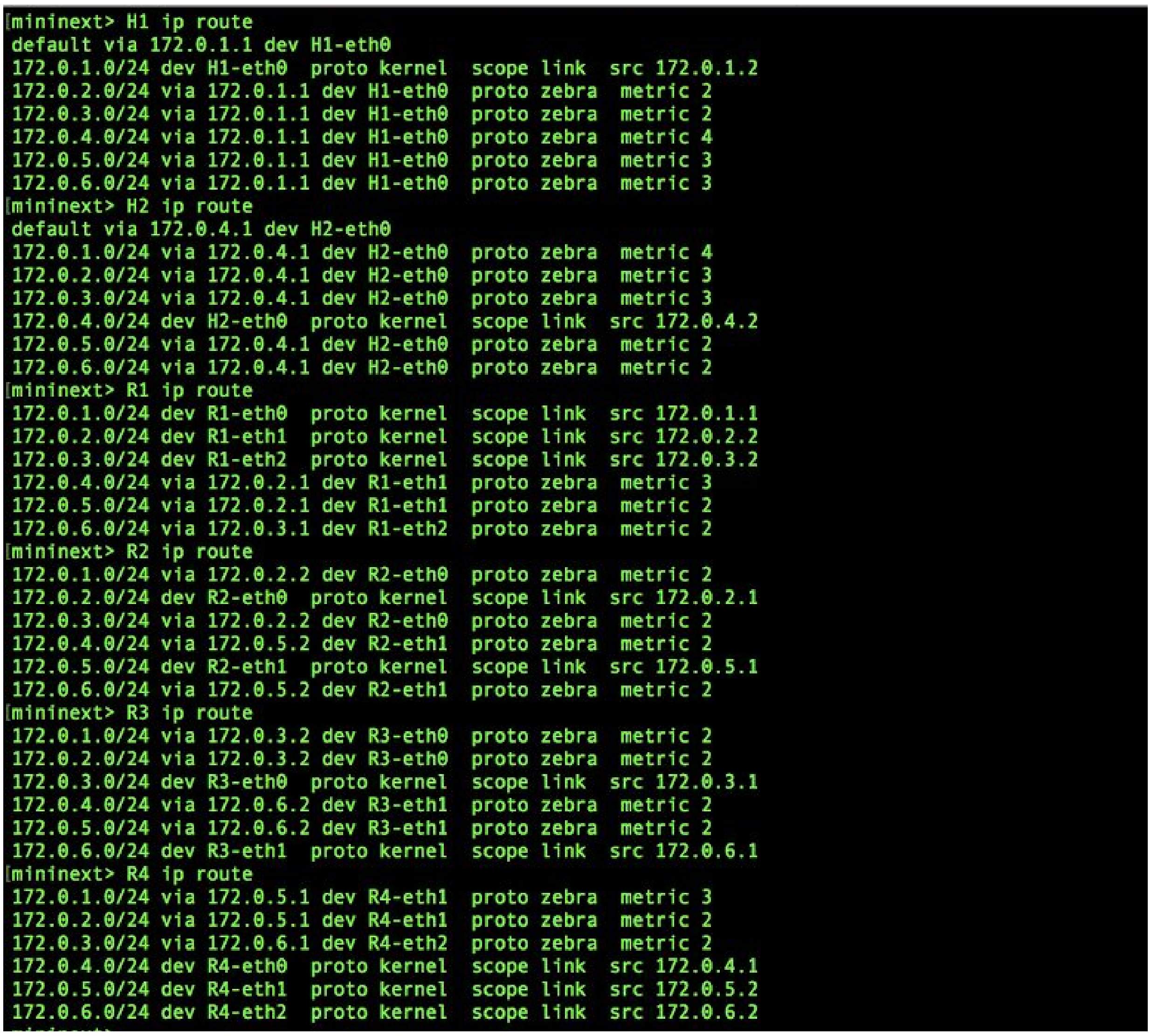
B2 (a), (b):

Kernel Routing Table is shown in *Figure B.1.3*
Quagga Routing Table is shown in *Figure B.1.4*
Traceroute Path between H1 and H2 (Both ways) is shown in *Figure B.1.5*



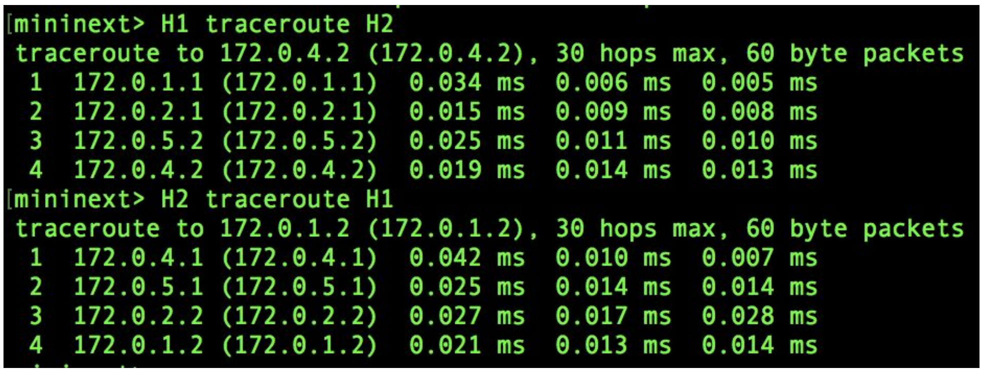
```
[mininext> H1 route
 Kernel IP routing table
 Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 default         172.0.1.1       0.0.0.0         UG    0      0        0 H1-eth0
 172.0.1.0       *               255.255.255.0   U     0      0        0 H1-eth0
 172.0.2.0       172.0.1.1       255.255.255.0   UG    2      0        0 H1-eth0
 172.0.3.0       172.0.1.1       255.255.255.0   UG    2      0        0 H1-eth0
 172.0.4.0       172.0.1.1       255.255.255.0   UG    4      0        0 H1-eth0
 172.0.5.0       172.0.1.1       255.255.255.0   UG    3      0        0 H1-eth0
 172.0.6.0       172.0.1.1       255.255.255.0   UG    3      0        0 H1-eth0
[mininext> H2 route
 Kernel IP routing table
 Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 default         172.0.4.1       0.0.0.0         UG    0      0        0 H2-eth0
 172.0.1.0       172.0.4.1       255.255.255.0   UG    4      0        0 H2-eth0
 172.0.2.0       172.0.4.1       255.255.255.0   UG    3      0        0 H2-eth0
 172.0.3.0       172.0.4.1       255.255.255.0   UG    3      0        0 H2-eth0
 172.0.4.0       *               255.255.255.0   U     0      0        0 H2-eth0
 172.0.5.0       172.0.4.1       255.255.255.0   UG    2      0        0 H2-eth0
 172.0.6.0       172.0.4.1       255.255.255.0   UG    2      0        0 H2-eth0
[mininext> R1 route
 Kernel IP routing table
 Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 172.0.1.0       *               255.255.255.0   U     0      0        0 R1-eth0
 172.0.2.0       *               255.255.255.0   U     0      0        0 R1-eth1
 172.0.3.0       *               255.255.255.0   U     0      0        0 R1-eth2
 172.0.4.0       172.0.2.1       255.255.255.0   UG    3      0        0 R1-eth1
 172.0.5.0       172.0.2.1       255.255.255.0   UG    2      0        0 R1-eth1
 172.0.6.0       172.0.3.1       255.255.255.0   UG    2      0        0 R1-eth2
[mininext> R2 route
 Kernel IP routing table
 Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 172.0.1.0       172.0.2.2       255.255.255.0   UG    2      0        0 R2-eth0
 172.0.2.0       *               255.255.255.0   U     0      0        0 R2-eth0
 172.0.3.0       172.0.2.2       255.255.255.0   UG    2      0        0 R2-eth0
 172.0.4.0       172.0.5.2       255.255.255.0   UG    2      0        0 R2-eth1
 172.0.5.0       *               255.255.255.0   U     0      0        0 R2-eth1
 172.0.6.0       172.0.5.2       255.255.255.0   UG    2      0        0 R2-eth1
[mininext> R3 route
 Kernel IP routing table
 Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 172.0.1.0       172.0.3.2       255.255.255.0   UG    2      0        0 R3-eth0
 172.0.2.0       172.0.3.2       255.255.255.0   UG    2      0        0 R3-eth0
 172.0.3.0       *               255.255.255.0   U     0      0        0 R3-eth0
 172.0.4.0       172.0.6.2       255.255.255.0   UG    2      0        0 R3-eth1
 172.0.5.0       172.0.6.2       255.255.255.0   UG    2      0        0 R3-eth1
 172.0.6.0       *               255.255.255.0   U     0      0        0 R3-eth1
[mininext> R4 route
 Kernel IP routing table
 Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
 172.0.1.0       172.0.5.1       255.255.255.0   UG    3      0        0 R4-eth1
 172.0.2.0       172.0.5.1       255.255.255.0   UG    2      0        0 R4-eth1
 172.0.3.0       172.0.6.1       255.255.255.0   UG    2      0        0 R4-eth2
 172.0.4.0       *               255.255.255.0   U     0      0        0 R4-eth0
 172.0.5.0       *               255.255.255.0   U     0      0        0 R4-eth1
 172.0.6.0       *               255.255.255.0   U     0      0        0 R4-eth2
```

Fig B.1.3



```
[mininext> H1 ip route
 default via 172.0.1.1 dev H1-eth0
 172.0.1.0/24 dev H1-eth0  proto kernel  scope link  src 172.0.1.2
 172.0.2.0/24 via 172.0.1.1 dev H1-eth0  proto zebra  metric 2
 172.0.3.0/24 via 172.0.1.1 dev H1-eth0  proto zebra  metric 2
 172.0.4.0/24 via 172.0.1.1 dev H1-eth0  proto zebra  metric 4
 172.0.5.0/24 via 172.0.1.1 dev H1-eth0  proto zebra  metric 3
 172.0.6.0/24 via 172.0.1.1 dev H1-eth0  proto zebra  metric 3
[mininext> H2 ip route
 default via 172.0.4.1 dev H2-eth0
 172.0.1.0/24 via 172.0.4.1 dev H2-eth0  proto zebra  metric 4
 172.0.2.0/24 via 172.0.4.1 dev H2-eth0  proto zebra  metric 3
 172.0.3.0/24 via 172.0.4.1 dev H2-eth0  proto zebra  metric 3
 172.0.4.0/24 dev H2-eth0  proto kernel  scope link  src 172.0.4.2
 172.0.5.0/24 via 172.0.4.1 dev H2-eth0  proto zebra  metric 2
 172.0.6.0/24 via 172.0.4.1 dev H2-eth0  proto zebra  metric 2
[mininext> R1 ip route
 172.0.1.0/24 dev R1-eth0  proto kernel  scope link  src 172.0.1.1
 172.0.2.0/24 dev R1-eth1  proto kernel  scope link  src 172.0.2.2
 172.0.3.0/24 dev R1-eth2  proto kernel  scope link  src 172.0.3.2
 172.0.4.0/24 via 172.0.2.1 dev R1-eth1  proto zebra  metric 3
 172.0.5.0/24 via 172.0.2.1 dev R1-eth1  proto zebra  metric 2
 172.0.6.0/24 via 172.0.3.1 dev R1-eth2  proto zebra  metric 2
[mininext> R2 ip route
 172.0.1.0/24 via 172.0.2.2 dev R2-eth0  proto zebra  metric 2
 172.0.2.0/24 dev R2-eth0  proto kernel  scope link  src 172.0.2.1
 172.0.3.0/24 via 172.0.2.2 dev R2-eth0  proto zebra  metric 2
 172.0.4.0/24 via 172.0.5.2 dev R2-eth1  proto zebra  metric 2
 172.0.5.0/24 dev R2-eth1  proto kernel  scope link  src 172.0.5.1
 172.0.6.0/24 via 172.0.5.2 dev R2-eth1  proto zebra  metric 2
[mininext> R3 ip route
 172.0.1.0/24 via 172.0.3.2 dev R3-eth0  proto zebra  metric 2
 172.0.2.0/24 via 172.0.3.2 dev R3-eth0  proto zebra  metric 2
 172.0.3.0/24 dev R3-eth0  proto kernel  scope link  src 172.0.3.1
 172.0.4.0/24 via 172.0.6.2 dev R3-eth1  proto zebra  metric 2
 172.0.5.0/24 via 172.0.6.2 dev R3-eth1  proto zebra  metric 2
 172.0.6.0/24 dev R3-eth1  proto kernel  scope link  src 172.0.6.1
[mininext> R4 ip route
 172.0.1.0/24 via 172.0.5.1 dev R4-eth1  proto zebra  metric 3
 172.0.2.0/24 via 172.0.5.1 dev R4-eth1  proto zebra  metric 2
 172.0.3.0/24 via 172.0.6.1 dev R4-eth2  proto zebra  metric 2
 172.0.4.0/24 dev R4-eth0  proto kernel  scope link  src 172.0.4.1
 172.0.5.0/24 dev R4-eth1  proto kernel  scope link  src 172.0.5.2
 172.0.6.0/24 dev R4-eth2  proto kernel  scope link  src 172.0.6.2
```

Fig B.1.4

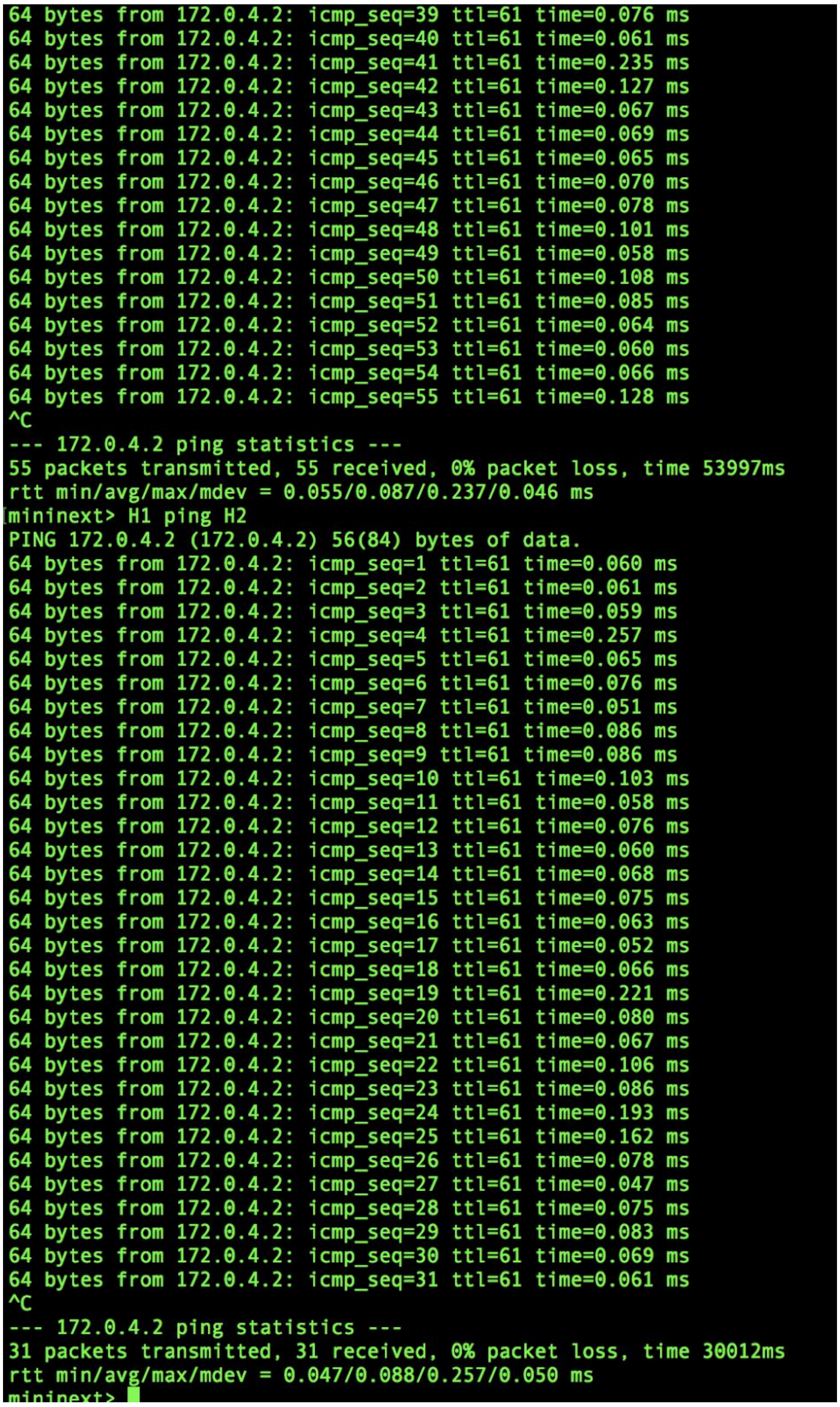


```
[mininext> H1 traceroute H2
 traceroute to 172.0.4.2 (172.0.4.2), 30 hops max, 60 byte packets
  1  172.0.1.1 (172.0.1.1)  0.034 ms  0.006 ms  0.005 ms
  2  172.0.2.1 (172.0.2.1)  0.015 ms  0.009 ms  0.008 ms
  3  172.0.5.2 (172.0.5.2)  0.025 ms  0.011 ms  0.010 ms
  4  172.0.4.2 (172.0.4.2)  0.019 ms  0.014 ms  0.013 ms
[mininext> H2 traceroute H1
 traceroute to 172.0.1.2 (172.0.1.2), 30 hops max, 60 byte packets
  1  172.0.4.1 (172.0.4.1)  0.042 ms  0.010 ms  0.007 ms
  2  172.0.5.1 (172.0.5.1)  0.025 ms  0.014 ms  0.014 ms
  3  172.0.2.2 (172.0.2.2)  0.027 ms  0.017 ms  0.028 ms
  4  172.0.1.2 (172.0.1.2)  0.021 ms  0.013 ms  0.014 ms
```

Fig B.1.5

B2. (c)

The ping time from H1 to H2 was measured as an average of two ping sessions measured over a time period of 30 - 50 seconds(Till the avg ping time seems to be narrowing down in a observed window). The Average RTT was seen to be around 0.088ms for a ping request from H1 to receive reply from H2. The screenshot is attached along in *Figure B.1.6*.



```
64 bytes from 172.0.4.2: icmp_seq=39 ttl=61 time=0.076 ms
64 bytes from 172.0.4.2: icmp_seq=40 ttl=61 time=0.061 ms
64 bytes from 172.0.4.2: icmp_seq=41 ttl=61 time=0.235 ms
64 bytes from 172.0.4.2: icmp_seq=42 ttl=61 time=0.127 ms
64 bytes from 172.0.4.2: icmp_seq=43 ttl=61 time=0.067 ms
64 bytes from 172.0.4.2: icmp_seq=44 ttl=61 time=0.069 ms
64 bytes from 172.0.4.2: icmp_seq=45 ttl=61 time=0.065 ms
64 bytes from 172.0.4.2: icmp_seq=46 ttl=61 time=0.070 ms
64 bytes from 172.0.4.2: icmp_seq=47 ttl=61 time=0.078 ms
64 bytes from 172.0.4.2: icmp_seq=48 ttl=61 time=0.101 ms
64 bytes from 172.0.4.2: icmp_seq=49 ttl=61 time=0.058 ms
64 bytes from 172.0.4.2: icmp_seq=50 ttl=61 time=0.108 ms
64 bytes from 172.0.4.2: icmp_seq=51 ttl=61 time=0.085 ms
64 bytes from 172.0.4.2: icmp_seq=52 ttl=61 time=0.064 ms
64 bytes from 172.0.4.2: icmp_seq=53 ttl=61 time=0.060 ms
64 bytes from 172.0.4.2: icmp_seq=54 ttl=61 time=0.066 ms
64 bytes from 172.0.4.2: icmp_seq=55 ttl=61 time=0.128 ms
^C
--- 172.0.4.2 ping statistics ---
55 packets transmitted, 55 received, 0% packet loss, time 53997ms
rtt min/avg/max/mdev = 0.055/0.087/0.237/0.046 ms
mininext> H1 ping H2
PING 172.0.4.2 (172.0.4.2) 56(84) bytes of data.
64 bytes from 172.0.4.2: icmp_seq=1 ttl=61 time=0.060 ms
64 bytes from 172.0.4.2: icmp_seq=2 ttl=61 time=0.061 ms
64 bytes from 172.0.4.2: icmp_seq=3 ttl=61 time=0.059 ms
64 bytes from 172.0.4.2: icmp_seq=4 ttl=61 time=0.257 ms
64 bytes from 172.0.4.2: icmp_seq=5 ttl=61 time=0.065 ms
64 bytes from 172.0.4.2: icmp_seq=6 ttl=61 time=0.076 ms
64 bytes from 172.0.4.2: icmp_seq=7 ttl=61 time=0.051 ms
64 bytes from 172.0.4.2: icmp_seq=8 ttl=61 time=0.086 ms
64 bytes from 172.0.4.2: icmp_seq=9 ttl=61 time=0.086 ms
64 bytes from 172.0.4.2: icmp_seq=10 ttl=61 time=0.103 ms
64 bytes from 172.0.4.2: icmp_seq=11 ttl=61 time=0.058 ms
64 bytes from 172.0.4.2: icmp_seq=12 ttl=61 time=0.076 ms
64 bytes from 172.0.4.2: icmp_seq=13 ttl=61 time=0.060 ms
64 bytes from 172.0.4.2: icmp_seq=14 ttl=61 time=0.068 ms
64 bytes from 172.0.4.2: icmp_seq=15 ttl=61 time=0.075 ms
64 bytes from 172.0.4.2: icmp_seq=16 ttl=61 time=0.063 ms
64 bytes from 172.0.4.2: icmp_seq=17 ttl=61 time=0.052 ms
64 bytes from 172.0.4.2: icmp_seq=18 ttl=61 time=0.066 ms
64 bytes from 172.0.4.2: icmp_seq=19 ttl=61 time=0.221 ms
64 bytes from 172.0.4.2: icmp_seq=20 ttl=61 time=0.080 ms
64 bytes from 172.0.4.2: icmp_seq=21 ttl=61 time=0.067 ms
64 bytes from 172.0.4.2: icmp_seq=22 ttl=61 time=0.106 ms
64 bytes from 172.0.4.2: icmp_seq=23 ttl=61 time=0.086 ms
64 bytes from 172.0.4.2: icmp_seq=24 ttl=61 time=0.193 ms
64 bytes from 172.0.4.2: icmp_seq=25 ttl=61 time=0.162 ms
64 bytes from 172.0.4.2: icmp_seq=26 ttl=61 time=0.078 ms
64 bytes from 172.0.4.2: icmp_seq=27 ttl=61 time=0.047 ms
64 bytes from 172.0.4.2: icmp_seq=28 ttl=61 time=0.075 ms
64 bytes from 172.0.4.2: icmp_seq=29 ttl=61 time=0.083 ms
64 bytes from 172.0.4.2: icmp_seq=30 ttl=61 time=0.069 ms
64 bytes from 172.0.4.2: icmp_seq=31 ttl=61 time=0.061 ms
^C
--- 172.0.4.2 ping statistics ---
31 packets transmitted, 31 received, 0% packet loss, time 30012ms
rtt min/avg/max/mdev = 0.047/0.088/0.257/0.050 ms
mininext>
```

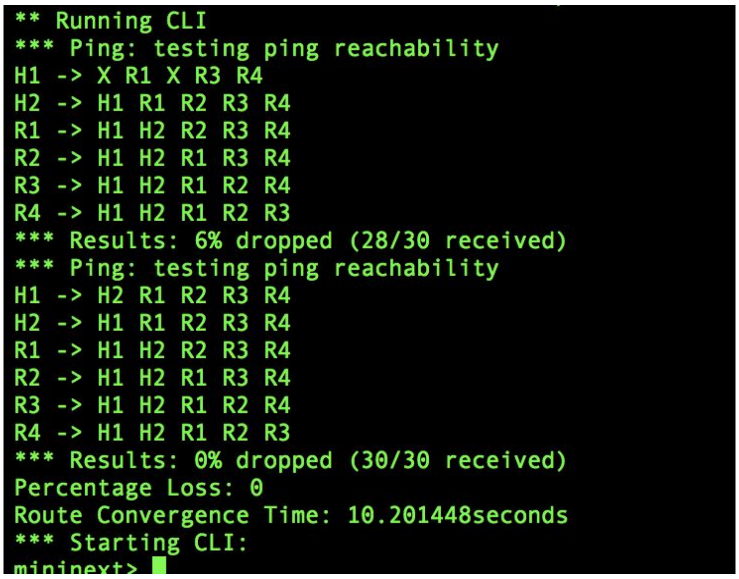
Fig B.1.6

B2. (d)

The convergence time for RIP was measured through mininet python API. First, timestamp measurement was taken when the Quagga controller is started and then after setting-up the interface ips and configuring NAT firewalls. The script keeps pinging all the hosts(every host to another)in a loop till the ping loss reaches to zero. As soon as it does, another timestamp measurement is taken and the difference is the time taken for the network to converge and the hosts to be able to ping each other.

Note: In some mininet versions, pingAll seems to have been blocked from H1 tries to ping other nodes, while in others, the pinging was seen to be asynchronous, thus we see an extra blocking skew in the convergence time. (This blocking might be attributed to older mininext versions). When tested the same with other versions, the convergence time turned out to be order of 2 - 3 seconds.

Time taken for convergence comes out to be: 10.2 seconds as seen in *Figure B.1.7*.

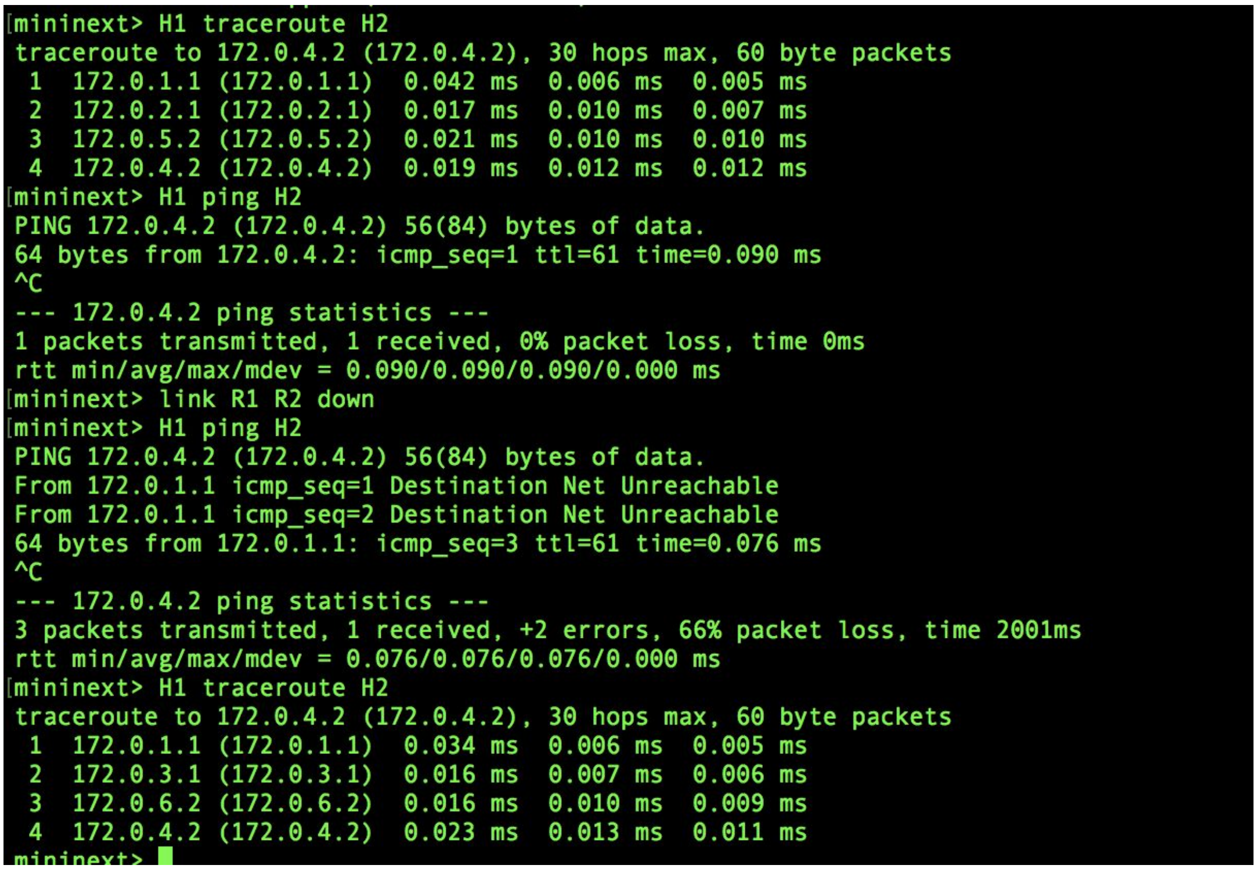


```
** Running CLI
*** Ping: testing ping reachability
H1 -> X R1 X R3 R4
H2 -> H1 R1 R2 R3 R4
R1 -> H1 H2 R2 R3 R4
R2 -> H1 H2 R1 R3 R4
R3 -> H1 H2 R1 R2 R4
R4 -> H1 H2 R1 R2 R3
*** Results: 6% dropped (28/30 received)
*** Ping: testing ping reachability
H1 -> H2 R1 R2 R3 R4
H2 -> H1 R1 R2 R3 R4
R1 -> H1 H2 R2 R3 R4
R2 -> H1 H2 R1 R3 R4
R3 -> H1 H2 R1 R2 R4
R4 -> H1 H2 R1 R2 R3
*** Results: 0% dropped (30/30 received)
Percentage Loss: 0
Route Convergence Time: 10.201448seconds
*** Starting CLI:
mininext>
```

Fig B.1.7

B3.

Next, we take down the link R1 - R2 which can be seen above is the path taken from H1 to H2 using the commands as shown in *Figure B.1.8*



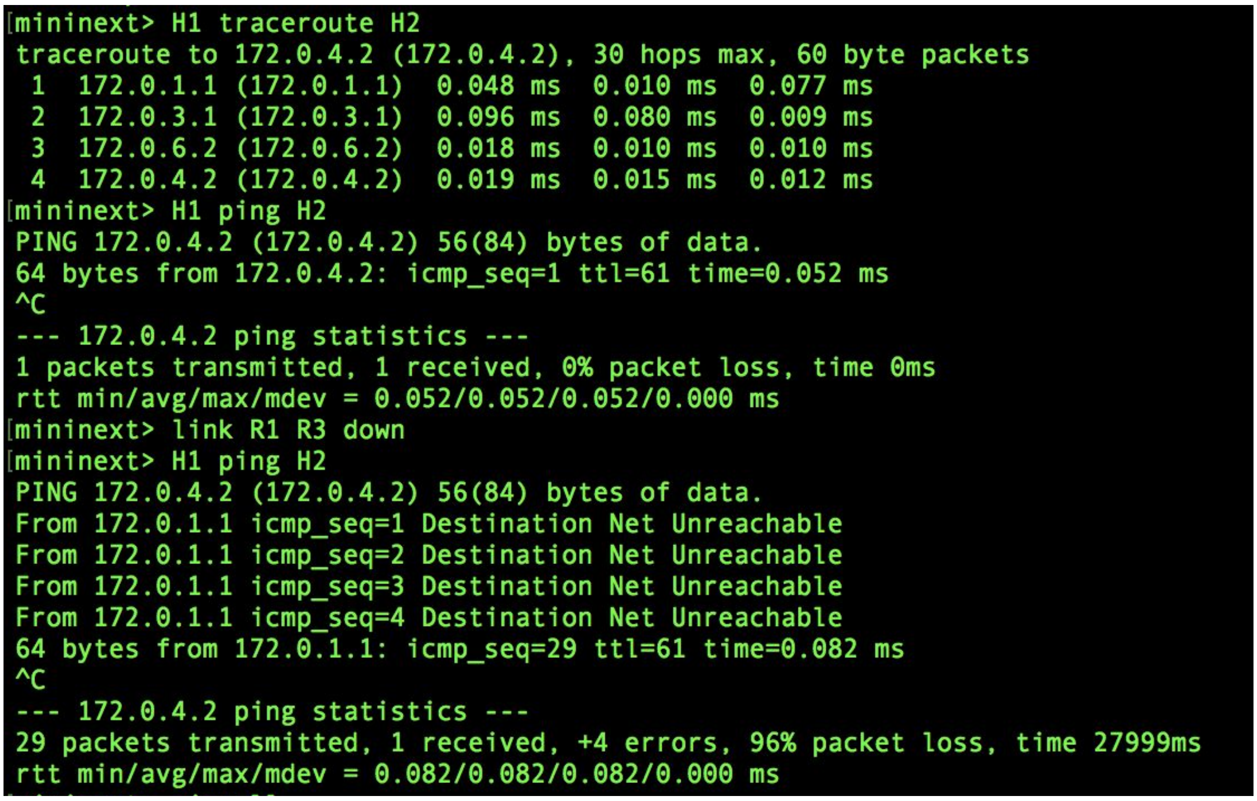
```
mininext> H1 traceroute H2
traceroute to 172.0.4.2 (172.0.4.2), 30 hops max, 60 byte packets
 1  172.0.1.1 (172.0.1.1)  0.042 ms  0.006 ms  0.005 ms
 2  172.0.2.1 (172.0.2.1)  0.017 ms  0.010 ms  0.007 ms
 3  172.0.5.2 (172.0.5.2)  0.021 ms  0.010 ms  0.010 ms
 4  172.0.4.2 (172.0.4.2)  0.019 ms  0.012 ms  0.012 ms
mininext> H1 ping H2
PING 172.0.4.2 (172.0.4.2) 56(84) bytes of data.
64 bytes from 172.0.4.2: icmp_seq=1 ttl=61 time=0.090 ms
^C
--- 172.0.4.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.090/0.090/0.090/0.000 ms
mininext> link R1 R2 down
mininext> H1 ping H2
PING 172.0.4.2 (172.0.4.2) 56(84) bytes of data.
From 172.0.1.1 icmp_seq=1 Destination Net Unreachable
From 172.0.1.1 icmp_seq=2 Destination Net Unreachable
64 bytes from 172.0.1.1: icmp_seq=3 ttl=61 time=0.076 ms
^C
--- 172.0.4.2 ping statistics ---
3 packets transmitted, 1 received, +2 errors, 66% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.076/0.076/0.076/0.000 ms
mininext> H1 traceroute H2
traceroute to 172.0.4.2 (172.0.4.2), 30 hops max, 60 byte packets
 1  172.0.1.1 (172.0.1.1)  0.034 ms  0.006 ms  0.005 ms
 2  172.0.3.1 (172.0.3.1)  0.016 ms  0.007 ms  0.006 ms
 3  172.0.6.2 (172.0.6.2)  0.016 ms  0.010 ms  0.009 ms
 4  172.0.4.2 (172.0.4.2)  0.023 ms  0.013 ms  0.011 ms
mininext>
```

Fig B.1.8

As we can see from the above figure that the initial path from H1 to H2 followed through R2. Next we take R1 - R2 down using mininext's link command.
[Source: http://mininet.org/walkthrough/#link-updown]

As soon as we take the link down, we initiate a ping from H1 to H2 and measure the time taken to receive the first successful ping reply. This turns out to be 2001ms as seen above. Now, since this time also includes the ping RTT, the time taken for connectivity to get established would be PingRespnoseTime - (avgRTT/2). [Subtract the time for ICMP ping response since connectivity was already established by the time H1's ping request reached H2]. This would roughly give us the time of 2000.9ms of connectivity establishment time.
Note: This test was performed on a freshly established connectivity. It was also observed that in some cases, and for the networks in which RIP has been running for longer times, connectivity establishment time was in the orders of 14 - 25 seconds. (Probably due to stale floating RIP packets in the networks).

One such output is shown in the following scenario in *Figure B.1.9* where R1 - R3 was taken down from the H1 - H2 path and the time taken to establish connectivity from R1 - R2 took approx 28 seconds.



```
[mininext> H1 traceroute H2
traceroute to 172.0.4.2 (172.0.4.2), 30 hops max, 60 byte packets
 1  172.0.1.1 (172.0.1.1)  0.048 ms  0.010 ms  0.077 ms
 2  172.0.3.1 (172.0.3.1)  0.096 ms  0.080 ms  0.009 ms
 3  172.0.6.2 (172.0.6.2)  0.018 ms  0.010 ms  0.010 ms
 4  172.0.4.2 (172.0.4.2)  0.019 ms  0.015 ms  0.012 ms
[mininext> H1 ping H2
PING 172.0.4.2 (172.0.4.2) 56(84) bytes of data.
64 bytes from 172.0.4.2: icmp_seq=1 ttl=61 time=0.052 ms
^C
--- 172.0.4.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.052/0.052/0.052/0.000 ms
[mininext> link R1 R3 down
[mininext> H1 ping H2
PING 172.0.4.2 (172.0.4.2) 56(84) bytes of data.
From 172.0.1.1 icmp_seq=1 Destination Net Unreachable
From 172.0.1.1 icmp_seq=2 Destination Net Unreachable
From 172.0.1.1 icmp_seq=3 Destination Net Unreachable
From 172.0.1.1 icmp_seq=4 Destination Net Unreachable
64 bytes from 172.0.1.1: icmp_seq=29 ttl=61 time=0.082 ms
^C
--- 172.0.4.2 ping statistics ---
29 packets transmitted, 1 received, +4 errors, 96% packet loss, time 27999ms
rtt min/avg/max/mdev = 0.082/0.082/0.082/0.000 ms
```

Fig B.1.9

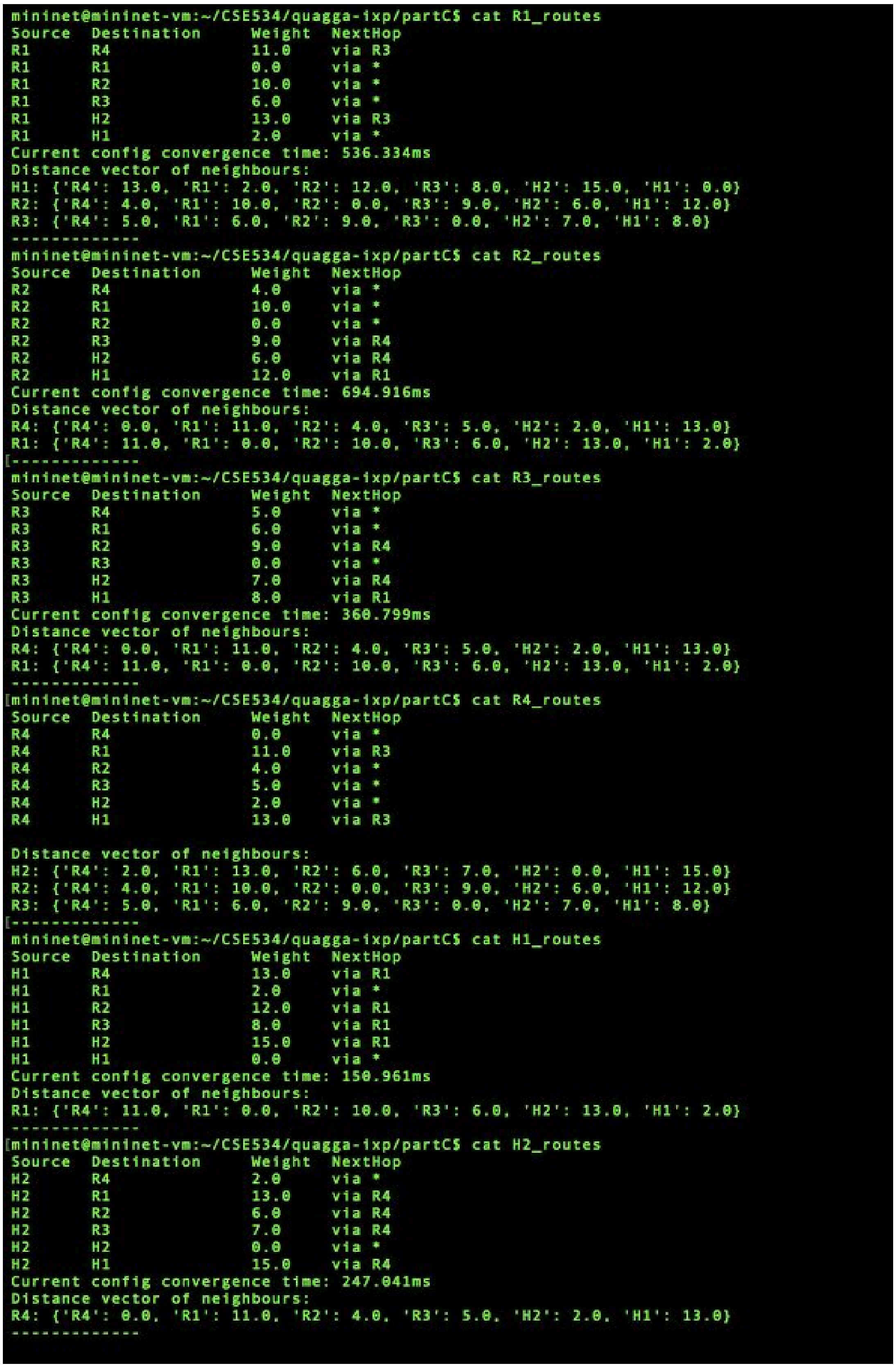
C1.

The shortest path between H1 and H2 comes through R1 - R3 link for the given topology and weights. The time taken for convergence at each node was precisely calculated from the moment it triggers the protocol on itself and till the time the node receives the last routing update from any of its neighbour including the time taken to update its own distance vector.

The final routing table at each node is written to `<nodelabel>_routes` by the code and also includes the convergence time for each of themselves. The same can be seen from the *Figure C.1.1.*

We also include the next hop for each of the nodes in the routing table and observe that if for instance the R1 - R2 weight changes from 1 to 10, the protocol reroutes and converges to the next hop path from R1 - R4 through R2.

Note: '*' means either direct neighbour or the node itself. The convergence time also contains a skew introduced as a result of the trigger script provided. The absolute convergence has an error rate of about +- (50 - 100) milliseconds. The error rate falls on the higher side of interval for nodes with more neighbours(R1, R4 for instance) and is near to 50ms or lower for nodes with only direct connections. Thus the convergence time for H1 to be able to ping H2 turns out to be around 150 milliseconds (on average over multiple testing sessions).



```
mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R1_routes
Source  Destination     Weight  NextHop
R1      R4              11.0    via R3
R1      R1              0.0     via *
R1      R2              10.0    via *
R1      R3              6.0     via *
R1      H2              13.0    via R3
R1      H1              2.0     via *
Current config convergence time: 536.334ms
Distance vector of neighbours:
H1: {'R4': 13.0, 'R1': 2.0, 'R2': 12.0, 'R3': 8.0, 'H2': 15.0, 'H1': 0.0}
R2: {'R4': 4.0, 'R1': 10.0, 'R2': 0.0, 'R3': 9.0, 'H2': 6.0, 'H1': 12.0}
R3: {'R4': 5.0, 'R1': 6.0, 'R2': 9.0, 'R3': 0.0, 'H2': 7.0, 'H1': 8.0}
-------------
mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R2_routes
Source  Destination     Weight  NextHop
R2      R4              4.0     via *
R2      R1              10.0    via *
R2      R2              0.0     via *
R2      R3              9.0     via R4
R2      H2              6.0     via R4
R2      H1              12.0    via R1
Current config convergence time: 694.916ms
Distance vector of neighbours:
R4: {'R4': 0.0, 'R1': 11.0, 'R2': 4.0, 'R3': 5.0, 'H2': 2.0, 'H1': 13.0}
R1: {'R4': 11.0, 'R1': 0.0, 'R2': 10.0, 'R3': 6.0, 'H2': 13.0, 'H1': 2.0}
[-------------
mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R3_routes
Source  Destination     Weight  NextHop
R3      R4              5.0     via *
R3      R1              6.0     via *
R3      R2              9.0     via R4
R3      R3              0.0     via *
R3      H2              7.0     via R4
R3      H1              8.0     via R1
Current config convergence time: 360.799ms
Distance vector of neighbours:
R4: {'R4': 0.0, 'R1': 11.0, 'R2': 4.0, 'R3': 5.0, 'H2': 2.0, 'H1': 13.0}
R1: {'R4': 11.0, 'R1': 0.0, 'R2': 10.0, 'R3': 6.0, 'H2': 13.0, 'H1': 2.0}
-------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R4_routes
Source  Destination     Weight  NextHop
R4      R4              0.0     via *
R4      R1              11.0    via R3
R4      R2              4.0     via *
R4      R3              5.0     via *
R4      H2              2.0     via *
R4      H1              13.0    via R3

Distance vector of neighbours:
H2: {'R4': 2.0, 'R1': 13.0, 'R2': 6.0, 'R3': 7.0, 'H2': 0.0, 'H1': 15.0}
R2: {'R4': 4.0, 'R1': 10.0, 'R2': 0.0, 'R3': 9.0, 'H2': 6.0, 'H1': 12.0}
R3: {'R4': 5.0, 'R1': 6.0, 'R2': 9.0, 'R3': 0.0, 'H2': 7.0, 'H1': 8.0}
[-------------
mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat H1_routes
Source  Destination     Weight  NextHop
H1      R4              13.0    via R1
H1      R1              2.0     via *
H1      R2              12.0    via R1
H1      R3              8.0     via R1
H1      H2              15.0    via R1
H1      H1              0.0     via *
Current config convergence time: 150.961ms
Distance vector of neighbours:
R1: {'R4': 11.0, 'R1': 0.0, 'R2': 10.0, 'R3': 6.0, 'H2': 13.0, 'H1': 2.0}
-------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat H2_routes
Source  Destination     Weight  NextHop
H2      R4              2.0     via *
H2      R1              13.0    via R4
H2      R2              6.0     via R4
H2      R3              7.0     via R4
H2      H2              0.0     via *
H2      H1              15.0    via R4
Current config convergence time: 247.041ms
Distance vector of neighbours:
R4: {'R4': 0.0, 'R1': 11.0, 'R2': 4.0, 'R3': 5.0, 'H2': 2.0, 'H1': 13.0}
-------------
```

Fig C.1.1

C2.

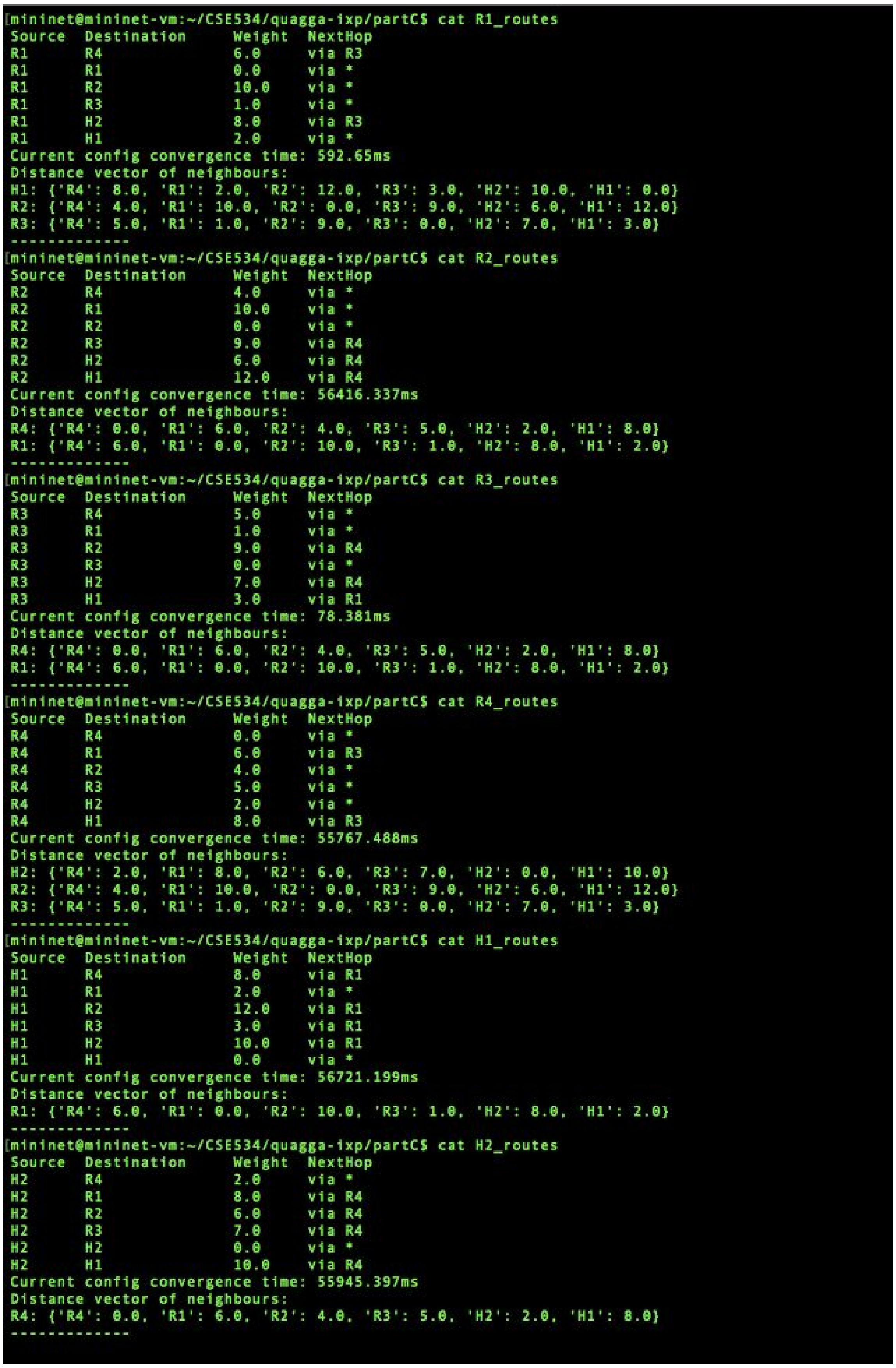
If we change the link cost of R1 - R3 from 6 to 1, we see that the path taken from H1 to H2 remains the same but the path costs changes as per the updated topology. As can be seen from the screenshot in *Figure C.1.2*, The new convergence time for R1 turns out to be approx 592ms. Note that this time doesn't contain any skew since the scripts at any of the servers were not interrupted and the link costs were updated dynamically as soon as the changes were detected. Also, the updated convergence is only calculated for the link whose edge costs have changed(In this case, R1 and R3). Since, any of the other nodes didn't see any changes in neighbouring links, their convergence time is seen as the difference between the time the protocol was initiated on them and the time till the last update.

C3.

As discussed on Piazza post@140, we treat the link costs as bi-directional. Thus, if we encounter any negative link weight, we automatically have a negative edge weight cycle in any path across connected network. We detect any such cycles by the proof of optimality of Bellman-Ford algorithm. The correctness of Bellman Ford claims that at the end of O(|V|*|E|) iterations, we get the shortest path from the source to any other vertex. Thus, if we see that at another iteration of Bellman Ford algorithm, we are able to find a path shorter than the one we already have, that means we have a negative edge weight cycle in our graph. At that moment, we abort the algorithm and notify the user.

References:

- Distance vector Bellman-Ford Theory Reference (Doesn't contain any code): https://www.geeksforgeeks.org/computer-network-routing-protocols-set-1-distance-vector-routing/
- https://buildbot.quagga.net/docs/nightly/quagga/quagga.html



```
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R1_routes
 Source  Destination      Weight  NextHop
 R1      R4               6.0     via R3
 R1      R1               0.0     via *
 R1      R2               10.0    via *
 R1      R3               1.0     via *
 R1      H2               8.0     via R3
 R1      H1               2.0     via *
 Current config convergence time: 592.65ms
 Distance vector of neighbours:
 H1: {'R4': 8.0, 'R1': 2.0, 'R2': 12.0, 'R3': 3.0, 'H2': 10.0, 'H1': 0.0}
 R2: {'R4': 4.0, 'R1': 10.0, 'R2': 0.0, 'R3': 9.0, 'H2': 6.0, 'H1': 12.0}
 R3: {'R4': 5.0, 'R1': 1.0, 'R2': 9.0, 'R3': 0.0, 'H2': 7.0, 'H1': 3.0}
 -------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R2_routes
 Source  Destination      Weight  NextHop
 R2      R4               4.0     via *
 R2      R1               10.0    via *
 R2      R2               0.0     via *
 R2      R3               9.0     via R4
 R2      H2               6.0     via R4
 R2      H1               12.0    via R4
 Current config convergence time: 56416.337ms
 Distance vector of neighbours:
 R4: {'R4': 0.0, 'R1': 6.0, 'R2': 4.0, 'R3': 5.0, 'H2': 2.0, 'H1': 8.0}
 R1: {'R4': 6.0, 'R1': 0.0, 'R2': 10.0, 'R3': 1.0, 'H2': 8.0, 'H1': 2.0}
 -------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R3_routes
 Source  Destination      Weight  NextHop
 R3      R4               5.0     via *
 R3      R1               1.0     via *
 R3      R2               9.0     via R4
 R3      R3               0.0     via *
 R3      H2               7.0     via R4
 R3      H1               3.0     via R1
 Current config convergence time: 78.381ms
 Distance vector of neighbours:
 R4: {'R4': 0.0, 'R1': 6.0, 'R2': 4.0, 'R3': 5.0, 'H2': 2.0, 'H1': 8.0}
 R1: {'R4': 6.0, 'R1': 0.0, 'R2': 10.0, 'R3': 1.0, 'H2': 8.0, 'H1': 2.0}
 -------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat R4_routes
 Source  Destination      Weight  NextHop
 R4      R4               0.0     via *
 R4      R1               6.0     via R3
 R4      R2               4.0     via *
 R4      R3               5.0     via *
 R4      H2               2.0     via *
 R4      H1               8.0     via R3
 Current config convergence time: 55767.488ms
 Distance vector of neighbours:
 H2: {'R4': 2.0, 'R1': 8.0, 'R2': 6.0, 'R3': 7.0, 'H2': 0.0, 'H1': 10.0}
 R2: {'R4': 4.0, 'R1': 10.0, 'R2': 0.0, 'R3': 9.0, 'H2': 6.0, 'H1': 12.0}
 R3: {'R4': 5.0, 'R1': 1.0, 'R2': 9.0, 'R3': 0.0, 'H2': 7.0, 'H1': 3.0}
 -------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat H1_routes
 Source  Destination      Weight  NextHop
 H1      R4               8.0     via R1
 H1      R1               2.0     via *
 H1      R2               12.0    via R1
 H1      R3               3.0     via R1
 H1      H2               10.0    via R1
 H1      H1               0.0     via *
 Current config convergence time: 56721.199ms
 Distance vector of neighbours:
 R1: {'R4': 6.0, 'R1': 0.0, 'R2': 10.0, 'R3': 1.0, 'H2': 8.0, 'H1': 2.0}
 -------------
[mininet@mininet-vm:~/CSE534/quagga-ixp/partC$ cat H2_routes
 Source  Destination      Weight  NextHop
 H2      R4               2.0     via *
 H2      R1               8.0     via R4
 H2      R2               6.0     via R4
 H2      R3               7.0     via R4
 H2      H2               0.0     via *
 H2      H1               10.0    via R4
 Current config convergence time: 55945.397ms
 Distance vector of neighbours:
 R4: {'R4': 0.0, 'R1': 6.0, 'R2': 4.0, 'R3': 5.0, 'H2': 2.0, 'H1': 8.0}
 -------------
```

Fig C.1.2

-x-x-x-x-*FIN*-x-x-x-x-