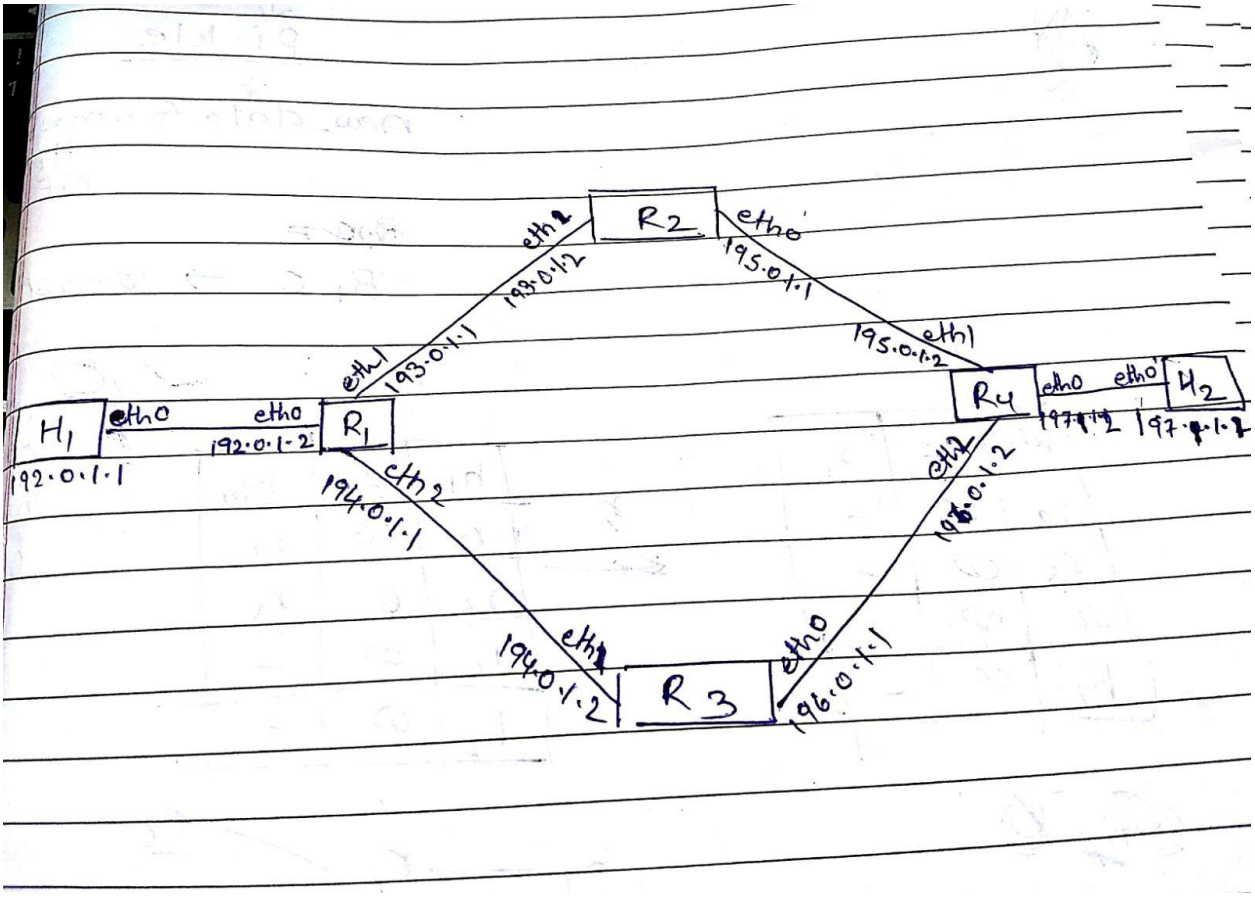
CSE 534 HOMEWORK 3

PART A:

A1

The topo.py is submitted in the folder and the topology figure is as shown below.

```
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)
mininext>
```



A2

The steps to be followed for enabling routing at each node are:

1. Enable the IP routing

   net.get("r1").cmd("sysctl net.ipv4.ip_forward=1")

   The above command enables the ip forward for node R1. Similarly, we need to set this to 1 for every node.

2. Configure and Assign Ips to all Interfaces.

   We need to closely follow the topo.py file created and assign the ip address to each interface at each node. The interface Ip's should be in proper subnets which depends on the node it is being connected to.

   net.get("r4").cmd("ifconfig r4-eth1 195.0.1.2")

   The above command assigns the ip to r4 eth1 interface.

3. Add Proper routes between nodes

   net.get("r1").cmd("ip route add 197.1.1.0/24 via 193.0.1.2")

   The above command helps us add route to different destination from r1. The first ip addresses from left is the destination Ip address along with its subnet information. The second ip is the address through which the packet is being routed forward. We also set h1 and h2 as default gateway

```
mininext> h1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.0.1.2       0.0.0.0         UG    0      0        0 h1-eth0
192.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 h1-eth0
mininext> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
193.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r1-eth1
194.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r1-eth2
195.0.1.0       193.0.1.2       255.255.255.0   UG    0      0        0 r1-eth1
196.0.1.0       194.0.1.2       255.255.255.0   UG    0      0        0 r1-eth2
197.1.1.0       193.0.1.2       255.255.255.0   UG    0      0        0 r1-eth1
mininext> r2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       193.0.1.1       255.255.255.0   UG    0      0        0 r2-eth0
193.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r2-eth0
194.0.1.0       193.0.1.1       255.255.255.0   UG    0      0        0 r2-eth0
195.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r2-eth1
196.0.1.0       195.0.1.2       255.255.255.0   UG    0      0        0 r2-eth1
197.1.1.0       195.0.1.2       255.255.255.0   UG    0      0        0 r2-eth1
```

```
mininext> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       194.0.1.1       255.255.255.0   UG    0      0        0 r3-eth0
193.0.1.0       194.0.1.1       255.255.255.0   UG    0      0        0 r3-eth0
194.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
195.0.1.0       196.0.1.2       255.255.255.0   UG    0      0        0 r3-eth1
196.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r3-eth1
197.1.1.0       196.0.1.2       255.255.255.0   UG    0      0        0 r3-eth1
mininext> r4 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       195.0.1.1       255.255.255.0   UG    0      0        0 r4-eth1
193.0.1.0       195.0.1.1       255.255.255.0   UG    0      0        0 r4-eth1
194.0.1.0       196.0.1.1       255.255.255.0   UG    0      0        0 r4-eth2
195.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r4-eth1
196.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r4-eth2
197.1.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r4-eth0
mininext> h2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         197.1.1.2       0.0.0.0         UG    0      0        0 h2-eth0
197.1.1.0       0.0.0.0         255.255.255.0   U     0      0        0 h2-eth0
```

```
mininext> h1 traceroute h2
traceroute to 197.1.1.1 (197.1.1.1), 30 hops max, 60 byte packets
 1  192.0.1.2 (192.0.1.2)  0.023 ms  0.004 ms  0.007 ms
 2  193.0.1.2 (193.0.1.2)  0.010 ms  0.005 ms  0.004 ms
 3  195.0.1.2 (195.0.1.2)  0.011 ms  0.006 ms  0.005 ms
 4  197.1.1.1 (197.1.1.1)  0.009 ms  0.006 ms  0.006 ms
mininext>
```

PART B:

B1

The steps followed in this question are as follows:

- We need to modify the files inside the configs folder. In configs folder you will find a folder pertaining to each node. Initially these folders are empty and we need to add 4 files into each folder namely daemons, ripd.conf, debian.conf and zebra.conf.
- **Daemons** – We need to create a file called daemons using command **- sudo vim daemons**. This file you need to create at each node and add the same lines. In the daemon file created we add the following lines

  zebra =yes

  bgpd=no

  ospfd=yes

  ospf6d=no

  ripd=no

  ripngd=no

- **debian.conf**- We need to copy this file from **/etc/quagga/debian.conf** using the command cp source destination with destination being our config folder having directories for each node.We do not modify this file.
- **zebra.conf**- We need to copy this file from **/usr/share/doc/quagga/examples/zebra.conf.sample** using the command cp source destination with destination being our config folder having directories for each node. We rename this file as zebra.conf. We do not modify this file.
- **ripd.conf -** We need to copy this file from **/usr/share/doc/quagga/examples/ripd.conf.sample** using the command cp source destination with destination being our config folder having directories for each node. We rename this file as ripd.conf. In this file we add every interface ip at each node. For eg in file for R1 we add:

  router rip
  network 192.0.1.2/24
  network 193.0.1.1/24
  network 194.0.1.1/24

  After this we start the quagga services by sudo services quagga start. Also remember that in start.py we need to remove the routes that we added in part 1,RIP protocol will set up the routes. To check type pingall command in mininet CLI

B2

A

1. Daemons at Each Node

```
mininext> h1 ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  12652  1444 tty1     S    23:21   0:00 bash -ms minine
t:h1
quagga      35  0.0  0.0  24452  1100 ?        Ss   23:21   0:00 /usr/lib/quagga
/zebra --daemon -A 127.0.0.1
quagga      39  0.0  0.0  24336  1212 ?        Ss   23:21   0:00 /usr/lib/quagga
/ripd --daemon -A 127.0.0.1
root        44  0.0  0.0  15404   512 ?        Ss   23:21   0:00 /usr/lib/quagga
/watchquagga --daemon zebra ripd
root        58  0.0  0.0  18688  1296 tty1     R    23:48   0:00 ps aux
mininext> h2 ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  12652  1444 tty1     S    23:21   0:00 bash -ms minine
t:h2
quagga      35  0.0  0.0  24452  1100 ?        Ss   23:21   0:00 /usr/lib/quagga
/zebra --daemon -A 127.0.0.1
quagga      39  0.0  0.0  24336  1212 ?        Ss   23:21   0:00 /usr/lib/quagga
/ripd --daemon -A 127.0.0.1
root        44  0.0  0.0  15404   512 ?        Ss   23:21   0:00 /usr/lib/quagga
/watchquagga --daemon zebra ripd
root        57  0.0  0.0  18688  1296 tty1     R    23:48   0:00 ps aux
```

```
mininext> r1 ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  12652  1444 tty1     S    23:21   0:00 bash -ms minine
t:r1
quagga      45  0.0  0.0  24452  1104 ?        Ss   23:21   0:00 /usr/lib/quagga
/zebra --daemon -A 127.0.0.1
quagga      49  0.0  0.0  24336  1216 ?        Ss   23:21   0:00 /usr/lib/quagga
/ripd --daemon -A 127.0.0.1
root        54  0.0  0.0  15404   512 ?        Ss   23:21   0:00 /usr/lib/quagga
/watchquagga --daemon zebra ripd
root        69  0.0  0.0  18688  1296 tty1     R    23:22   0:00 ps aux
mininext> r2 ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  12652  1440 tty1     S    23:21   0:00 bash -ms minine
t:r2
quagga      40  0.0  0.0  24452  1100 ?        Ss   23:21   0:00 /usr/lib/quagga
/zebra --daemon -A 127.0.0.1
quagga      44  0.0  0.0  24336  1216 ?        Ss   23:21   0:00 /usr/lib/quagga
/ripd --daemon -A 127.0.0.1
root        49  0.0  0.0  15404   512 ?        Ss   23:21   0:00 /usr/lib/quagga
/watchquagga --daemon zebra ripd
root        63  0.0  0.0  18688  1296 tty1     R    23:22   0:00 ps aux
```

```
mininext> r3 ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  12652  1444 tty1     S    23:21   0:00 bash -ms minine
t:r3
quagga      40  0.0  0.0  24452  1100 ?        Ss   23:21   0:00 /usr/lib/quagga
/zebra --daemon -A 127.0.0.1
quagga      44  0.0  0.0  24336  1216 ?        Ss   23:21   0:00 /usr/lib/quagga
/ripd --daemon -A 127.0.0.1
root        49  0.0  0.0  15404   512 ?        Ss   23:21   0:00 /usr/lib/quagga
/watchquagga --daemon zebra ripd
root        63  0.0  0.0  18688  1292 tty1     R    23:24   0:00 ps aux
mininext> r4 ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  12652  1444 tty1     S    23:21   0:00 bash -ms minine
t:r4
quagga      45  0.0  0.0  24452  1100 ?        Ss   23:21   0:00 /usr/lib/quagga
/zebra --daemon -A 127.0.0.1
quagga      49  0.0  0.0  24336  1216 ?        Ss   23:21   0:00 /usr/lib/quagga
/ripd --daemon -A 127.0.0.1
root        54  0.0  0.0  15404   512 ?        Ss   23:21   0:00 /usr/lib/quagga
/watchquagga --daemon zebra ripd
root        70  0.0  0.0  18688  1296 tty1     R    23:24   0:00 ps aux
```

2. Routing Table at Each Node

```
mininext> h1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 h1-eth0
193.0.1.0       192.0.1.2       255.255.255.0   UG    2      0        0 h1-eth0
194.0.1.0       192.0.1.2       255.255.255.0   UG    2      0        0 h1-eth0
195.0.1.0       192.0.1.2       255.255.255.0   UG    3      0        0 h1-eth0
196.0.1.0       192.0.1.2       255.255.255.0   UG    3      0        0 h1-eth0
197.1.1.0       192.0.1.2       255.255.255.0   UG    4      0        0 h1-eth0
mininext> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
193.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r1-eth1
194.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r1-eth2
195.0.1.0       193.0.1.2       255.255.255.0   UG    2      0        0 r1-eth1
196.0.1.0       194.0.1.2       255.255.255.0   UG    2      0        0 r1-eth2
197.1.1.0       193.0.1.2       255.255.255.0   UG    3      0        0 r1-eth1
mininext> r2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       193.0.1.1       255.255.255.0   UG    2      0        0 r2-eth1
193.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r2-eth1
194.0.1.0       193.0.1.1       255.255.255.0   UG    2      0        0 r2-eth1
195.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r2-eth0
196.0.1.0       195.0.1.2       255.255.255.0   UG    2      0        0 r2-eth0
197.1.1.0       195.0.1.2       255.255.255.0   UG    2      0        0 r2-eth0
```

```
mininext> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       194.0.1.1       255.255.255.0   UG    2      0        0 r3-eth1
193.0.1.0       194.0.1.1       255.255.255.0   UG    2      0        0 r3-eth1
194.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r3-eth1
195.0.1.0       196.0.1.2       255.255.255.0   UG    2      0        0 r3-eth0
196.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
197.1.1.0       196.0.1.2       255.255.255.0   UG    2      0        0 r3-eth0
mininext> r4 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       195.0.1.1       255.255.255.0   UG    3      0        0 r4-eth1
193.0.1.0       195.0.1.1       255.255.255.0   UG    2      0        0 r4-eth1
194.0.1.0       196.0.1.1       255.255.255.0   UG    2      0        0 r4-eth2
195.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r4-eth1
196.0.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r4-eth2
197.1.1.0       0.0.0.0         255.255.255.0   U     0      0        0 r4-eth0
mininext> h2 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.0.1.0       197.1.1.2       255.255.255.0   UG    4      0        0 h2-eth0
193.0.1.0       197.1.1.2       255.255.255.0   UG    3      0        0 h2-eth0
194.0.1.0       197.1.1.2       255.255.255.0   UG    3      0        0 h2-eth0
195.0.1.0       197.1.1.2       255.255.255.0   UG    2      0        0 h2-eth0
196.0.1.0       197.1.1.2       255.255.255.0   UG    2      0        0 h2-eth0
197.1.1.0       0.0.0.0         255.255.255.0   U     0      0        0 h2-eth0
mininext>
```

B

```
mininext> h1 traceroute h2
traceroute to 197.1.1.1 (197.1.1.1), 30 hops max, 60 byte packets
 1  192.0.1.2 (192.0.1.2)  0.020 ms  0.004 ms  0.003 ms
 2  193.0.1.2 (193.0.1.2)  0.010 ms  0.004 ms  0.004 ms
 3  195.0.1.2 (195.0.1.2)  0.012 ms  0.007 ms  0.006 ms
 4  197.1.1.1 (197.1.1.1)  0.010 ms  0.007 ms  0.008 ms
mininext>
```

C

```
mininext> h1 ping -c 10 h2
PING 197.1.1.1 (197.1.1.1) 56(84) bytes of data.
64 bytes from 197.1.1.1: icmp_seq=1 ttl=61 time=0.032 ms
64 bytes from 197.1.1.1: icmp_seq=2 ttl=61 time=0.061 ms
64 bytes from 197.1.1.1: icmp_seq=3 ttl=61 time=0.055 ms
64 bytes from 197.1.1.1: icmp_seq=4 ttl=61 time=0.066 ms
64 bytes from 197.1.1.1: icmp_seq=5 ttl=61 time=0.082 ms
64 bytes from 197.1.1.1: icmp_seq=6 ttl=61 time=0.086 ms
64 bytes from 197.1.1.1: icmp_seq=7 ttl=61 time=0.070 ms
64 bytes from 197.1.1.1: icmp_seq=8 ttl=61 time=0.103 ms
64 bytes from 197.1.1.1: icmp_seq=9 ttl=61 time=0.066 ms
64 bytes from 197.1.1.1: icmp_seq=10 ttl=61 time=0.080 ms

--- 197.1.1.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.032/0.070/0.103/0.018 ms
```

The ping time is 0.07 ms

D.

I took the convergence time manually(using stop watch) by calculating the time between the typing of start command to getting the first ping response.

Convergence time= 14.60 ms

B3

1.

Initial Route R1- R2

```
traceroute to 197.1.1.1 (197.1.1.1), 30 hops max, 60 byte packets
 1  192.0.1.2 (192.0.1.2)  0.019 ms  0.003 ms  0.003 ms
 2  193.0.1.2 (193.0.1.2)  0.011 ms  0.004 ms  0.004 ms
 3  195.0.1.2 (195.0.1.2)  0.012 ms  0.007 ms  0.005 ms
 4  197.1.1.1 (197.1.1.1)  0.009 ms  0.007 ms  0.007 ms
mininext>
```

Bring route down by command **link r1 r2 down**

2.

The time taken for connecting the link is again is 29.34 ms . This time path R1- R3 is used.

3.

```
mininext> h1 traceroute h2
traceroute to 197.1.1.1 (197.1.1.1), 30 hops max, 60 byte packets
 1  192.0.1.2 (192.0.1.2)  0.021 ms  0.004 ms  0.003 ms
 2  194.0.1.2 (194.0.1.2)  0.011 ms  0.005 ms  0.005 ms
 3  196.0.1.2 (196.0.1.2)  0.010 ms  0.006 ms  0.006 ms
 4  197.1.1.1 (197.1.1.1)  0.012 ms  0.007 ms  0.007 ms
mininext>
```

PART C

C1

Initially I have maintained a dictionary of neighbors and dictionary of addresses of nodes present in topology. At each node I have a csv file which has information such nodes, its distance to other nodes and next hop to that node. For implementing topology, I have implemented a client Server model where I call class Server with parameters. This class creates a Server per node by spawning a Server Thread which the interacts with the clients at neighboring nodes. I also have a client thread at each node whose job is to accept the distance data frame sent by server thread and run Bell man ford on it and write the updated values to a file. The Server thread is continuously checking this file and if there is update in its distance vector it propagates this data frame to its neighbors. Also, for convergence I maintain a single file which I use to read the value of converge variable and when all variables are converged I say that the algorithm is converged. I have implemented bellman ford logic on the pandas data frame and when I get updated data frame I write to the common file at each node. For sending of data between the noes I have used pickle.

C3

The Bellman ford algorithm can be used to detect the negative edge in the graph. We can used this to modifying the algorithm to changing the way it works. If in the Nth iteration where n is number of vertices the algorithm finds a loop we can make it backtrack until we reach edge through which we reach a node still not explored. Another way of thinking this is modifying the weight of negative link to average of the sum of weight of other links. Otherwise replacing it with 0 or median of weights of other links.