Part A

A1

(a)topo.py is included in the zip folder
(b)Network topology figure is included in the zipped folder.

Steps:

Sudo python start.py

```
mininext> pingall
*** Ping: testing ping reachability
h1 -> X r1 X X X
h2 -> X X X X r4
r1 -> h1 X X X X
r2 -> X X X X X
r3 -> X X X X X
r4 -> X h2 X X X
*** Results: 86% dropped (4/30 received)

> nodes
available nodes are:
c0 h1 h2 r1 r2 r3 r4


mininext> r1 ip route
50.1.1.0/24 dev r1-eth0  proto kernel  scope link  src 50.1.1.2
mininext> r2 ip
mininext> h2 ip route
55.7.7.0/24 dev h2-eth0  proto kernel  scope link  src 55.7.7.1
mininext> h1 ip route
50.1.1.0/24 dev h1-eth0  proto kernel  scope link  src 50.1.1.1
mininext> r1 ip route
50.1.1.0/24 dev r1-eth0  proto kernel  scope link  src 50.1.1.2
mininext> r2 ip route
51.1.1.0/24 dev r2-eth0  proto kernel  scope link  src 51.1.1.2
```

```
mininext> r3 ip route
52.1.1.0/24 dev r3-eth0  proto kernel  scope link  src 52.1.1.2
mininext> r4 ip route
55.7.7.0/24 dev r4-eth0  proto kernel  scope link  src 55.7.7.2
mininext> h2 ip route
55.7.7.0/24 dev h2-eth0  proto kernel  scope link  src 55.7.7.1



mininext> source part_a1.sh

mininext> net
h1 h1-eth0:r1-eth0
h2 h2-eth0:r4-eth0
r1 r1-eth0:h1-eth0 r1-eth1:r2-eth0 r1-eth2:r3-eth0
r2 r2-eth0:r1-eth1 r2-eth1:r4-eth1
r3 r3-eth0:r1-eth2 r3-eth1:r4-eth2
r4 r4-eth0:h2-eth0 r4-eth1:r2-eth1 r4-eth2:r3-eth1
C0



mininext> source part_a2.sh

mininext> source part_a_f.sh

h1 echo 1 > /proc/sys/net/ipv4/ip_forward
h2 echo 1 > /proc/sys/net/ipv4/ip_forward
r1 echo 1 > /proc/sys/net/ipv4/ip_forward
r2 echo 1 > /proc/sys/net/ipv4/ip_forward
r3 echo 1 > /proc/sys/net/ipv4/ip_forward
r4 echo 1 > /proc/sys/net/ipv4/ip_forward


mininext> pingall
*** Ping: testing ping reachability
h1 -> h2 r1 r2 r3 r4
h2 -> X X X X 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: 13% dropped (26/30 received)


mininext> h1 ping h2
PING 55.7.7.1 (55.7.7.1) 56(84) bytes of data.
64 bytes from 55.7.7.1: icmp_seq=1 ttl=62 time=0.111 ms
64 bytes from 55.7.7.1: icmp_seq=2 ttl=62 time=0.121 ms
64 bytes from 55.7.7.1: icmp_seq=3 ttl=62 time=0.169 ms
64 bytes from 55.7.7.1: icmp_seq=4 ttl=62 time=0.122 ms
64 bytes from 55.7.7.1: icmp_seq=5 ttl=62 time=0.261 ms
```

A2

(a)

***IP route table screenshot file in the zipped folder**

To configure the static route, I included for each start to end node from source to target destination:

```
<Node name>ip route add <destination subnet> via <next hop IP> dev <nodename>-<ethernet interface name>
```

eg:

```
r1 ip route add 55.7.7.0/24 via 52.1.1.2 dev r1-eth2
```

After this I added all the iptable forwarding commands at each intermediate node.

***All the values are included in part_a2.sh included in the zipped folder**

(b)

```
mininext> h1 traceroute h2
traceroute to 55.7.7.1 (55.7.7.1), 30 hops max, 60 byte packets
 1  50.1.1.2 (50.1.1.2)  0.069 ms  0.019 ms  0.024 ms
 2  51.1.1.2 (51.1.1.2)  0.037 ms  0.023 ms  0.062 ms
 3  55.7.7.1 (55.7.7.1)  0.050 ms  0.036 ms  0.036 ms
mininext>
```

Part B

B1

(a)and (b)

To enable zebra daemon and ripd routing protocol

```
vim /etc/quagga/daemons
+zebra=yes
+ripd=yes
```

To include a standard configuration of zebra and ripd

```
sudo cp /usr/share/doc/quagga/examples/zebra.conf.sample
/etc/quagga/zebra.conf
sudo cp /usr/share/doc/quagga/examples/ripd.conf.sample
/etc/quagga/ripd.conf
```

Change the ownership and access permissions so that we can make required changes

```
sudo chown quagga:quaggavty /etc/quagga/*.conf
sudo chmod 640 /etc/quagga/*.conf
```

To start quagga:

```
sudo /etc/init.d/quagga restart
```

To view all the quagga processs

```
ps -ef | grep quagga

ls /etc/quagga

daemons  debian.conf  ripd.conf  zebra.conf
```

Copy the above configuration files to all the nodes:

```
cp /etc/quagga/* /home/myfolder/quagga-ixp/configs/h1
cp /etc/quagga/* /home/myfolder/quagga-ixp/configs/h2
cp /etc/quagga/* /home/myfolder/quagga-ixp/configs/r1
cp /etc/quagga/* /home/myfolder/quagga-ixp/configs/r2
cp /etc/quagga/* /home/myfolder/quagga-ixp/configs/r3
cp /etc/quagga/* /home/myfolder/quagga-ixp/configs/r4
```

B2:

Steps
(Run in other terminal)

```
sudo python start.py
source part_a1.sh
```

Check for ip addresses on all nodes , they all should be proper.

H1

```
./mx h1
telnet localhost 2602
ripd> en
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 50.1.1.0/24
There is a same network configuration 50.1.1.0/24
ripd(config-router)# write
Configuration saved to /etc/quagga/ripd.conf
ripd(config-router)# exit
ripd(config)#
```

R1

```
./mx r1
telnet localhost 2602
ripd> en
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 50.1.1.0/24
There is a same network configuration 50.1.1.0/24
ripd(config-router)# network 52.1.1.0/24
There is a same network configuration 52.1.1.0/24
ripd(config-router)# network 51.1.1.0/24
There is a same network configuration 51.1.1.0/24
ripd(config-router)# exit
ripd(config)# exit
```

R2

```
./mx r2
telnet localhost 2602
ripd> en
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 51.1.10/24
% [RIP] Unknown command: network 51.1.10/24
ripd(config-router)# network 51.1.1.0/24
There is a same network configuration 51.1.1.0/24
```

```
ripd(config-router)# network 53.1.1.0/24
There is a same network configuration 53.1.1.0/24
ripd(config-router)# write
Configuration saved to /etc/quagga/ripd.conf
ripd(config-router)# exit
ripd(config)# exit
ripd#
```

R3

```
./mx r3
telnet localhost 2602
ripd> en
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 52.1.1.0/24
There is a same network configuration 52.1.1.0/24
ripd(config-router)# network 54.1.1.0/24
There is a same network configuration 54.1.1.0/24
ripd(config-router)# write
Configuration saved to /etc/quagga/ripd.conf
ripd(config-router)# exit
ripd(config)# exit
ripd# exit
```

R4

```
./mx r4
telnet localhost 2602
ripd> en
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 53.1.1.0/24
There is a same network configuration 53.1.1.0/24
ripd(config-router)# network 54.1.1.0/24
There is a same network configuration 54.1.1.0/24
ripd(config-router)# network 55.7.7.0/24
There is a same network configuration 55.7.7.0/24
```

```
ripd(config-router)# write
Configuration saved to /etc/quagga/ripd.conf
ripd(config-router)# exit

H2
./mx h2
telnet localhost 2602
ripd> en
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# network 55.7.7.0/24
There is a same network configuration 55.7.7.0/24
ripd(config-router)# write
Configuration saved to /etc/quagga/ripd.conf
ripd(config-router)# exit


mininext> source part_a_f.sh
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)
```

(a)

the routing tables at each node (both the kernel and the Quagga routing table)

```
mininext> h1 ip route
50.1.1.0/24 dev h1-eth0  proto kernel  scope link  src 50.1.1.1
51.1.1.0/24 via 50.1.1.2 dev h1-eth0  proto zebra  metric 2
52.1.1.0/24 via 50.1.1.2 dev h1-eth0  proto zebra  metric 2
```

```
53.1.1.0/24 via 50.1.1.2 dev h1-eth0  proto zebra  metric 3
54.1.1.0/24 via 50.1.1.2 dev h1-eth0  proto zebra  metric 3
55.7.7.0/24 via 50.1.1.2 dev h1-eth0  proto zebra  metric 4


mininext> r1 ip route
50.1.1.0/24 dev r1-eth0  proto kernel  scope link  src 50.1.1.2
51.1.1.0/24 dev r1-eth1  proto kernel  scope link  src 51.1.1.1
52.1.1.0/24 dev r1-eth2  proto kernel  scope link  src 52.1.1.1
53.1.1.0/24 via 51.1.1.2 dev r1-eth1  proto zebra  metric 2
54.1.1.0/24 via 52.1.1.2 dev r1-eth2  proto zebra  metric 2
55.7.7.0/24 via 52.1.1.2 dev r1-eth2  proto zebra  metric 3


mininext> r2 ip route
50.1.1.0/24 via 51.1.1.1 dev r2-eth0  proto zebra  metric 2
51.1.1.0/24 dev r2-eth0  proto kernel  scope link  src 51.1.1.2
52.1.1.0/24 via 51.1.1.1 dev r2-eth0  proto zebra  metric 2
53.1.1.0/24 dev r2-eth1  proto kernel  scope link  src 53.1.1.2
54.1.1.0/24 via 53.1.1.1 dev r2-eth1  proto zebra  metric 2
55.7.7.0/24 via 53.1.1.1 dev r2-eth1  proto zebra  metric 2



mininext> r3 ip route
50.1.1.0/24 via 52.1.1.1 dev r3-eth0  proto zebra  metric 2
51.1.1.0/24 via 52.1.1.1 dev r3-eth0  proto zebra  metric 2
52.1.1.0/24 dev r3-eth0  proto kernel  scope link  src 52.1.1.2
53.1.1.0/24 via 54.1.1.1 dev r3-eth1  proto zebra  metric 2
54.1.1.0/24 dev r3-eth1  proto kernel  scope link  src 54.1.1.2
55.7.7.0/24 via 54.1.1.1 dev r3-eth1  proto zebra  metric 2
```

```
mininext> r4 ip route
50.1.1.0/24 via 53.1.1.2 dev r4-eth1  proto zebra  metric 3
51.1.1.0/24 via 53.1.1.2 dev r4-eth1  proto zebra  metric 2
52.1.1.0/24 via 54.1.1.2 dev r4-eth2  proto zebra  metric 2
53.1.1.0/24 dev r4-eth1  proto kernel  scope link  src 53.1.1.1
54.1.1.0/24 dev r4-eth2  proto kernel  scope link  src 54.1.1.1
55.7.7.0/24 dev r4-eth0  proto kernel  scope link  src 55.7.7.2

mininext> h2 ip route
55.7.7.0/24 dev h2-eth0  proto kernel  scope link  src 55.7.7.1
```

(b)

```
mininext> h1 traceroute h2
traceroute to 55.7.7.1 (55.7.7.1), 30 hops max, 60 byte packets
 1  50.1.1.2 (50.1.1.2)  0.036 ms  0.006 ms  0.005 ms
 2  52.1.1.2 (52.1.1.2)  0.015 ms  0.009 ms  0.007 ms
 3  55.7.7.2 (55.7.7.2)  0.043 ms  0.018 ms  0.028 ms
```

(c)

```
mininext> h1 ping h2
PING 55.7.7.1 (55.7.7.1) 56(84) bytes of data.
64 bytes from 55.7.7.1: icmp_seq=1 ttl=63 time=0.118 ms
64 bytes from 55.7.7.1: icmp_seq=2 ttl=63 time=0.101 ms
64 bytes from 55.7.7.1: icmp_seq=3 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=4 ttl=63 time=0.097 ms
64 bytes from 55.7.7.1: icmp_seq=5 ttl=63 time=0.102 ms
64 bytes from 55.7.7.1: icmp_seq=6 ttl=63 time=0.098 ms
64 bytes from 55.7.7.1: icmp_seq=7 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=8 ttl=63 time=0.098 ms
64 bytes from 55.7.7.1: icmp_seq=9 ttl=63 time=0.043 ms
64 bytes from 55.7.7.1: icmp_seq=10 ttl=63 time=0.093 ms
64 bytes from 55.7.7.1: icmp_seq=11 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=12 ttl=63 time=0.098 ms
```

```
64 bytes from 55.7.7.1: icmp_seq=13 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=14 ttl=63 time=0.098 ms
64 bytes from 55.7.7.1: icmp_seq=15 ttl=63 time=0.101 ms
64 bytes from 55.7.7.1: icmp_seq=16 ttl=63 time=0.093 ms
64 bytes from 55.7.7.1: icmp_seq=17 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=18 ttl=63 time=0.109 ms
64 bytes from 55.7.7.1: icmp_seq=19 ttl=63 time=0.102 ms
64 bytes from 55.7.7.1: icmp_seq=20 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=21 ttl=63 time=0.109 ms
64 bytes from 55.7.7.1: icmp_seq=22 ttl=63 time=0.136 ms
64 bytes from 55.7.7.1: icmp_seq=23 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=24 ttl=63 time=0.059 ms
64 bytes from 55.7.7.1: icmp_seq=25 ttl=63 time=0.076 ms
64 bytes from 55.7.7.1: icmp_seq=26 ttl=63 time=0.110 ms
64 bytes from 55.7.7.1: icmp_seq=27 ttl=63 time=0.116 ms
64 bytes from 55.7.7.1: icmp_seq=28 ttl=63 time=0.109 ms
64 bytes from 55.7.7.1: icmp_seq=29 ttl=63 time=0.096 ms
64 bytes from 55.7.7.1: icmp_seq=30 ttl=63 time=0.099 ms
64 bytes from 55.7.7.1: icmp_seq=31 ttl=63 time=0.102 ms
64 bytes from 55.7.7.1: icmp_seq=32 ttl=63 time=0.103 ms
64 bytes from 55.7.7.1: icmp_seq=33 ttl=63 time=0.222 ms
64 bytes from 55.7.7.1: icmp_seq=34 ttl=63 time=0.111 ms
64 bytes from 55.7.7.1: icmp_seq=35 ttl=63 time=0.112 ms
64 bytes from 55.7.7.1: icmp_seq=36 ttl=63 time=0.109 ms
^c
```

Time taken for the ping ~ 100ms

(d)

Convergence time= 0.118 ms

B3

(a)

Old path : R1-R3

To break the path :

```
./mx R1
netstat -na
telnet localhost 2602
en
configure terminal
router rip
no network 52.1.1.0/24
write
Exit
```

(b)

The time it takes for connectivity to be established(This time path R1-R2 is used)

Time taken to reestablish the connection : 22 seconds

(c)

```
mininext> h1 traceroute h2
traceroute to 55.7.7.1 (55.7.7.1), 30 hops max, 60 byte packets
 1  50.1.1.2 (50.1.1.2)  0.036 ms  0.006 ms  0.005 ms
 2  51.1.1.2 (51.1.1.2)  0.032 ms  0.012 ms  0.017 ms
 3  55.7.7.2 (55.7.7.2)  0.043 ms  0.018 ms  0.028 ms
```

Part C:

I developed both a tcp client and tcp server service at each node.

The tcp server service has to be run over each of the node by calling the main method with command line arguments in the order
args[0] <localhost port number>
args[1] <filename>
args[2] <next hop neighbour ip address>
args[3] <next hop neighbour port>
For each incoming request, the server creates a new thread. Inside the thread the tcp server listens for the changed weight for the path. With the changed path it calls bellman ford algorithm which updates the routing table stored as a file by running Bellman-Ford algorithm, only if the changed weight is of value smaller than the old value in the file. The server then binds to the socket of the next hop server connecting to it as a client and sends it the updated file contents over the socket as a string.

*I have not included option for a client to send information to more than one next hop neighbour in the program, to do this separate instance of the client has to be started
*All the servers have to be started in order from last hop to the first hop before any client request has been initiated.

The Tcp client has been designed to be invoked when a node has to communicate any change in weight to its next hop(s). The information of IP and port number of the next hop have to be entered as the command line arguments:
args[0] <next hop neighbour ip address>
args[1] <next hop neighbour port>
args[2] <weight change info>
args[3] <filename>
Before sending the weight change information to the server the client updates its own routing table file by calling the

Bellman-Ford algorithm. After that it writes over the socket the changed weight info meant for the next hop (server) side.

Bellman-Ford Algorithm:
The program takes as input the weight change string in the format <source> <destination> <weight>. It then checks if the changed weight is smaller than the previous weight value between the same source in the destination already present in the routing table file. If the weight is smaller it is written to a data structure along with previous weight values on the file. Bellman ford algorithm is called with this data which checks for the shortest distance of all nodes from the node by running a loop V*E number of times. Where V is the total number of nodes and E is the total number of hop to hop paths between nodes. If a negative cycle is found after these many steps, that means the presence of a negative loop, In such case the algorithm prints error.
In other words, It works with negative edge but prints error if it finds a negative cycle.
The algorithm idea has been taken from geeks for geeks : http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/

For running the program, First of all on each node the server has to be started with the appropriate command line arguments as mentioned above and all need to run in the background.
For any weight change the client service has to be started with the appropriate command line arguments as mentioned above.

C1.

Run your protocol and get the time taken for the protocol to find the shortest path between H1 and H2. Submit: (a) the routing protocol code. (b) The time taken for your protocol to find the shortest path with explanation of how you estimated this (c) the application layer routing table at each node

- Approx 90 seconds
- a> included in the zipped folder
- b> approx 90 seconds, I manually checked in on a counter.
- c> Included in the zipped folder

C2.

- a> Approx 110 seconds
- b> the tables have been included in the zipped folder