LINKED STATE ROUTING PROTOCOL
COMP9331 ASSIGNMENT
REPORT

Vandhana Visakamurthy, z5222191

IMPLEMENTATION OF THE LSR PROTOCOL:

The Linked State Routing Protocol is an efficient and robust way to transmit information. This protocol ensures that every router connected within the network is aware of every other router irrespective of whether it's an immediate neighbour or not. This is known as flooding. Flooding is the process of ensuring that every router is aware of the network topology. Based on this knowledge every router calculates the shortest path to every other router using the Dijkstra's algorithm. Dijkstra's is a greedy algorithm that ensures that the shortest path is derived, even if it means it has to compromise the efficacy. The algorithm recalculates the routing path every time a router fails in the network. This ensures that the dependency is maintained throughout.

The main program in the source code does the following:

1. Opening and reading the Config_.txt files.
2. Processing the contents of the files.
3. Initialising sockets for sending and receiving messages.
4. Running and maintaining threads.
5. Managing timers to ensure threads run concurrently without fighting for resources.

After processing the config files of the source router, the details of the neighbouring routers are stored in the list called neighbours and the port numbers of the neighbours are stored in a list named neighbour_ports.
The Linked State Advertisement packet is designed with the format mentioned below:
Sequence_Number, Router_ID, Router_Port, No_of_neighbours, Neigbour_Port_ID, etc.
The sequence number is a randomly generated five digit number between 10000 and 99999.
After the LSA message of the source router is put together by the code, the packets are sent to the other routers using a variety of user defined functions.
The sockets are defined for sending and receiving before using the various elements to implement the protocol.

```
s = socket(AF_INET,SOCK_DGRAM)
s.setsockopt(SOL_SOCKET,SO_BROADCAST,1)
c = socket(AF_INET,SOCK_DGRAM)
c.setsockopt(SOL_SOCKET,SO_BROADCAST,1)
```

The source ports used to send messages are defined using the variable 's' whereas the client port defined to receive messages are defined as c. The port number is bound to only the client port to receive packets as the network interface required to send messages does not require one.

The LSR protocol implemented in this Assignment has the following user defined functions :

1. sendMessages(router_port,neighbour_ports,LSA):
   The sendMessages function takes three arguments which include the port number of the source router, the list of the neighbour ports and the LSA packet.
   The LSA Packet is in the format of a string hence it can be easily encoded and sent to the respective neighbours using s.sendto(message.encode(),('localhost',n)).

2. receiveMessages(router_id,client,neighbours)
   The receiveMessages functions does the following tasks:
   a) Use the client socket to receive packets.
   b) Decodes the packet to determine what kind of packet it is.
   c) The packet can either be a LSA packet, heartbeat message or a node failure packet.
   d) Process the packet using Packet_Processing() function if the packet is a LSA packet.
   e) Update Node failure if it contains information of a node failure.
   f) Keep track of the heartbeats and update the timestamps for all the immediate neighbours.
   g) If the heartbeat fails and the difference between the timestamps is greater than five, the node id declared to be dead and the immediate neighbours are updated using 'DEAD ROUTER <router name>' message.
   h) If the received packet 'DEAD ROUTER <router name>', then it updates its topology and forwards the same to its neighbours to ensure the entire network is aware of the dead node.

3. Dijkstra(start,stop,network)

Dijksytra's algorithm takes in the start router, end router and the topology graph as parameters and returns the shortest path.

4. shortestPath(neighbours,counter,router_id)
   The shortestPath function restricts broadcasting and waits for the topology to get completely updated before it starts to call the Dijkstra's function. Also, this function ensures that Dijkstra waits to calculate the shortest path every thirty seconds.

5. Heartbeats(router_id,router_port,neighbour_ports)
   Heartbeats are messages sent to immediate neighbours alone to update the neighbours that the router is alive.

6. Packet_processing(packet)
   The Packet_Processing function takes a packet as an argument and is called inside the receiveMessages() function. The Packets are processed and split into various arguments. The details of the packet are used to update the topology after processing.

7. Delete(delete_node,graph)
   The Delete() function is called by the receiveMessages() function when node failure is detected. The dead node and the topology are passed as arguments.

The main program contains five threads to handle sending, receiving, heartbeats, shortest path calculation and Dijkstra's. There are no locks used as there is no issues of deadlocks detected.

DATA STRUCTURES :

1. The network topology is maintained is using a dictionary of dictionaries.
   An example of the topology for an arbitrary network is represented as :
   {'a':{'b':6,'d':1},'b':{'a':6,'d':2,'e':2},'c':{'b':5,'e':5},'d':{'a':1,'b':2,'e':1},'e':{'d':1,'b':2,'c':5}}

2. The sequences [] list maintains a list of all the sequence numbers whose packet are received and processed.
3. Status maintains the heartbeats' timestamp and the router ID. This is a dictionary of lists. The list contains the previous timestamp difference and the current timestamp.
4. The LSA is a string containing all details about the source router.

RESTRICTING LINK STATE BROADCAST:
Link State Routing is successful in updating the network's topology to even remote routers because of the concept of flooding. However, if the broadcasts are not restricted after the initial "flood", then the buffer stream contains a lot of redundant packets. The code implemented restricts such broadcasts using two checks:

1. Using sequence numbers to prevent unnecessary packet processing and retransmission.
2. Checking if the packet is from the source router itself or if the retransmitted packet is from the origin router.
   Example : Preventing B from receiving packets from itself that is being retransmitted by A during broadcast. This also prevents B from broadcasting to itself and through its neighbours.

NODE FAILURE:
Node failure is detected and handled using timestamps and a global data structure called status, which is a dictionary of lists.
So every time a heartbeat is detected, a new timestamp depicting the current time is generated. The new timestamp is appended to a list that is mapped to the key, which represents a node.
Example : {'A' : [old timestamp, new timestamp]}
The difference between the old timestamp and the new timestamp should be lesser than 5 seconds. If not then the router is declared dead. If a node is dead, the router immediately broadcasts the "DEAD ROUTER <router name>" to its immediate neighbours. This is retransmitted by the other routers until every router in the network is aware. Delete() function is called to delete the node from the dictionary (topology). When the heartbeat is back, the system uses the cache of packets maintained in a list called Packets [] to re-update the topology.
This ensures that the topology is updated immediately after the heartbeat is detected. The packet stored in Packets[] is processed using Packet_processing() to update the network.

FEATURES IMPLEMENTED:

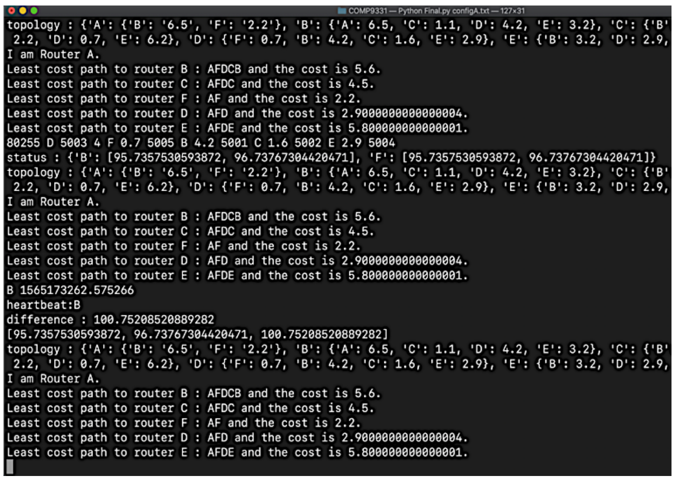
1. Broadcast messages.
2. Flooding the Network.
3. Updating and managing the topology dictionary, heartbeat status and sequence numbers.
4. Restricting Broadcasts to prevent redundant broadcasts and packets.
5. Calculating shortest path using Dijkstra's algorithm.
6. Handling Node failure and node restart.

LIMITATIONS:

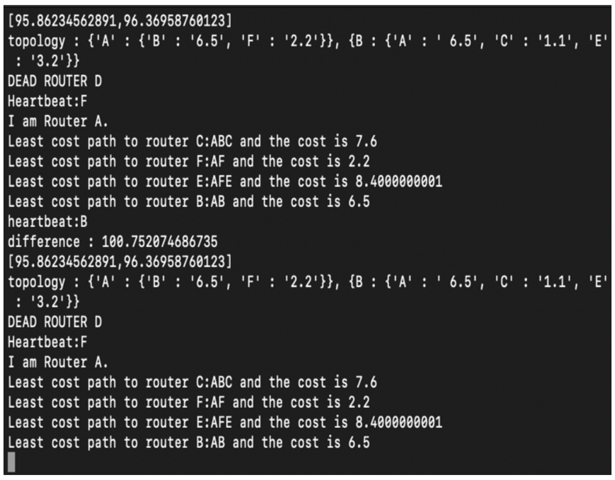
1. The Node failure is detected very late at times.
2. The protocol gets stuck due to threading at times. But is much later during execution.
3. Node re-joining the network takes a while to recalculate the new path.

The main trade-off made while implementing this protocol was efficiency. The node failure and node redetection takes a while to run, but works nonetheless. The concept of Dijkstra for algorithm implementation was taken from Sedgewick and a Youtube lecture video by CodeSchool.

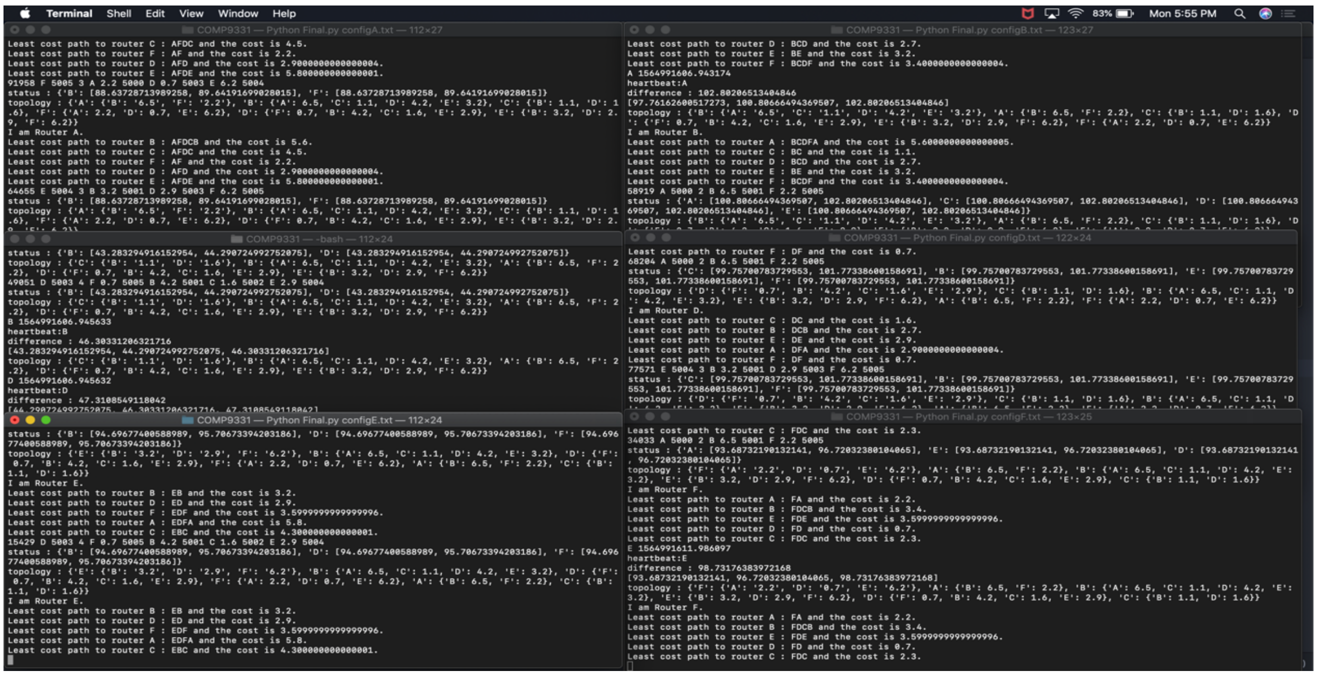
SCREENSHOTS OF OUTPUT:



```
topology : {'A': {'B': '6.5', 'F': '2.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'C': {'B'
 2.2, 'D': 0.7, 'E': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9,
I am Router A.
Least cost path to router B : AFDCB and the cost is 5.6.
Least cost path to router C : AFDC and the cost is 4.5.
Least cost path to router F : AF and the cost is 2.2.
Least cost path to router D : AFD and the cost is 2.9000000000000004.
Least cost path to router E : AFDE and the cost is 5.800000000000001.
80255 D 5003 4 F 0.7 5005 B 4.2 5001 C 1.6 5002 E 2.9 5004
status : {'B': [95.7357530593872, 96.73767304420471], 'F': [95.7357530593872, 96.73767304420471]}
topology : {'A': {'B': '6.5', 'F': '2.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'C': {'B'
 2.2, 'D': 0.7, 'E': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9,
I am Router A.
Least cost path to router B : AFDCB and the cost is 5.6.
Least cost path to router C : AFDC and the cost is 4.5.
Least cost path to router F : AF and the cost is 2.2.
Least cost path to router D : AFD and the cost is 2.9000000000000004.
Least cost path to router E : AFDE and the cost is 5.800000000000001.
B 1565173262.575266
heartbeat:B
difference : 100.75208520889282
[95.7357530593872, 96.73767304420471, 100.75208520889282]
topology : {'A': {'B': '6.5', 'F': '2.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'C': {'B'
 2.2, 'D': 0.7, 'E': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9,
I am Router A.
Least cost path to router B : AFDCB and the cost is 5.6.
Least cost path to router C : AFDC and the cost is 4.5.
Least cost path to router F : AF and the cost is 2.2.
Least cost path to router D : AFD and the cost is 2.9000000000000004.
Least cost path to router E : AFDE and the cost is 5.800000000000001.
```



```
[95.86234562891,96.36958760123]
topology : {'A' : {'B' : '6.5', 'F' : '2.2'}}, {B : {'A' : ' 6.5', 'C' : '1.1', 'E'
 : '3.2'}}
DEAD ROUTER D
Heartbeat:F
I am Router A.
Least cost path to router C:ABC and the cost is 7.6
Least cost path to router F:AF and the cost is 2.2
Least cost path to router E:AFE and the cost is 8.4000000001
Least cost path to router B:AB and the cost is 6.5
heartbeat:B
difference : 100.752074686735
[95.86234562891,96.36958760123]
topology : {'A' : {'B' : '6.5', 'F' : '2.2'}}, {B : {'A' : ' 6.5', 'C' : '1.1', 'E'
 : '3.2'}}
DEAD ROUTER D
Heartbeat:F
I am Router A.
Least cost path to router C:ABC and the cost is 7.6
Least cost path to router F:AF and the cost is 2.2
Least cost path to router E:AFE and the cost is 8.4000000001
Least cost path to router B:AB and the cost is 6.5
```


Terminal
Shell
Edit
View
Window
Help
83%
Mon 5:55 PM
COMP9331 — Python Final.py configA.txt — 112×27
COMP9331 — Python Final.py configB.txt — 123×27
COMP9331 — -bash — 112×24
COMP9331 — Python Final.py configD.txt — 122×24
COMP9331 — Python Final.py configE.txt — 112×24
COMP9331 — Python Final.py configF.txt — 123×25

```
Least cost path to router C : AFDC and the cost is 4.5.
Least cost path to router F : AF and the cost is 2.2.
Least cost path to router D : AFD and the cost is 2.9000000000000004.
Least cost path to router E : AFDE and the cost is 5.800000000000001.
91958 F 5005 3 A 2.2 5000 D 0.7 5003 E 6.2 5004
status : {'B': [88.63728713989258, 89.64191699028015], 'F': [88.63728713989258, 89.64191699028015]}
topology : {'A': {'B': '6.5', 'F': '2.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'C': {'B': 1.1, 'D': 1
.6}, 'F': {'A': 2.2, 'D': 0.7, 'E': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.
9, 'F': 6.2}}
I am Router A.
Least cost path to router B : AFDCB and the cost is 5.6.
Least cost path to router C : AFDC and the cost is 4.5.
Least cost path to router F : AF and the cost is 2.2.
Least cost path to router D : AFD and the cost is 2.9000000000000004.
Least cost path to router E : AFDE and the cost is 5.800000000000001.
64655 E 5004 3 B 3.2 5001 D 2.9 5003 F 6.2 5005
status : {'B': [88.63728713989258, 89.64191699028015], 'F': [88.63728713989258, 89.64191699028015]}
topology : {'A': {'B': '6.5', 'F': '2.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'C': {'B': 1.1, 'D': 1
.6}, 'F': {'A': 2.2, 'D': 0.7, 'E': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.
```

```
Least cost path to router D : BCD and the cost is 2.7.
Least cost path to router E : BE and the cost is 3.2.
Least cost path to router F : BCDF and the cost is 3.4000000000000004.
A 1564991606.943174
heartbeat:A
difference : 102.80206513404846
[97.76162600517273, 100.80666494369507, 102.80206513404846]
topology : {'B': {'A': '6.5', 'C': '1.1', 'D': '4.2', 'E': '3.2'}, 'A': {'B': 6.5, 'F': 2.2}, 'C': {'B': 1.1, 'D': 1.6}, 'D
': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}, 'F': {'A': 2.2, 'D': 0.7, 'E': 6.2}}
I am Router B.
Least cost path to router A : BCDFA and the cost is 5.6000000000000005.
Least cost path to router C : BC and the cost is 1.1.
Least cost path to router D : BCD and the cost is 2.7.
Least cost path to router E : BE and the cost is 3.2.
Least cost path to router F : BCDF and the cost is 3.4000000000000004.
58919 A 5000 2 B 6.5 5001 F 2.2 5005
status : {'A': [100.80666494369507, 102.80206513404846], 'C': [100.80666494369507, 102.80206513404846], 'D': [100.806664943
69507, 102.80206513404846], 'E': [100.80666494369507, 102.80206513404846]}
topology : {'B': {'A': '6.5', 'C': '1.1', 'D': '4.2', 'E': '3.2'}, 'A': {'B': 6.5, 'F': 2.2}, 'C': {'B': 1.1, 'D': 1.6}, 'D
```

```
status : {'B': [43.283294916152954, 44.290724992752075], 'D': [43.283294916152954, 44.290724992752075]}
topology : {'C': {'B': '1.1', 'D': '1.6'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'A': {'B': 6.5, 'F': 2
.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}}
49051 D 5003 4 F 0.7 5005 B 4.2 5001 C 1.6 5002 E 2.9 5004
status : {'B': [43.283294916152954, 44.290724992752075], 'D': [43.283294916152954, 44.290724992752075]}
topology : {'C': {'B': '1.1', 'D': '1.6'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'A': {'B': 6.5, 'F': 2
.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}}
B 1564991606.945633
heartbeat:B
difference : 46.30331206321716
[43.283294916152954, 44.290724992752075, 46.30331206321716]
topology : {'C': {'B': '1.1', 'D': '1.6'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'A': {'B': 6.5, 'F': 2
.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}}
D 1564991606.945632
heartbeat:D
difference : 47.3108549118042
```

```
Least cost path to router F : DF and the cost is 0.7.
68204 A 5000 2 B 6.5 5001 F 2.2 5005
status : {'C': [99.75700783729553, 101.77338600158691], 'B': [99.75700783729553, 101.77338600158691], 'E': [99.75700783729
553, 101.77338600158691], 'F': [99.75700783729553, 101.77338600158691]}
topology : {'D': {'F': '0.7', 'B': '4.2', 'C': '1.6', 'E': '2.9'}, 'C': {'B': 1.1, 'D': 1.6}, 'B': {'A': 6.5, 'C': 1.1, 'D
': 4.2, 'E': 3.2}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}, 'A': {'B': 6.5, 'F': 2.2}, 'F': {'A': 2.2, 'D': 0.7, 'E': 6.2}}
I am Router D.
Least cost path to router C : DC and the cost is 1.6.
Least cost path to router B : DCB and the cost is 2.7.
Least cost path to router E : DE and the cost is 2.9.
Least cost path to router A : DFA and the cost is 2.9000000000000004.
Least cost path to router F : DF and the cost is 0.7.
77571 E 5004 3 B 3.2 5001 D 2.9 5003 F 6.2 5005
status : {'C': [99.75700783729553, 101.77338600158691], 'B': [99.75700783729553, 101.77338600158691], 'E': [99.75700783729
553, 101.77338600158691], 'F': [99.75700783729553, 101.77338600158691]}
topology : {'D': {'F': '0.7', 'B': '4.2', 'C': '1.6', 'E': '2.9'}, 'C': {'B': 1.1, 'D': 1.6}, 'B': {'A': 6.5, 'C': 1.1, 'D
```

```
status : {'B': [94.69677400588989, 95.70673394203186], 'D': [94.69677400588989, 95.70673394203186], 'F': [94.696
77400588989, 95.70673394203186]}
topology : {'E': {'B': '3.2', 'D': '2.9', 'F': '6.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'D': {'F':
 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'F': {'A': 2.2, 'D': 0.7, 'E': 6.2}, 'A': {'B': 6.5, 'F': 2.2}, 'C': {'B':
1.1, 'D': 1.6}}
I am Router E.
Least cost path to router B : EB and the cost is 3.2.
Least cost path to router D : ED and the cost is 2.9.
Least cost path to router F : EDF and the cost is 3.5999999999999996.
Least cost path to router A : EDFA and the cost is 5.8.
Least cost path to router C : EBC and the cost is 4.300000000000001.
15429 D 5003 4 F 0.7 5005 B 4.2 5001 C 1.6 5002 E 2.9 5004
status : {'B': [94.69677400588989, 95.70673394203186], 'D': [94.69677400588989, 95.70673394203186], 'F': [94.696
77400588989, 95.70673394203186]}
topology : {'E': {'B': '3.2', 'D': '2.9', 'F': '6.2'}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E': 3.2}, 'D': {'F':
 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'F': {'A': 2.2, 'D': 0.7, 'E': 6.2}, 'A': {'B': 6.5, 'F': 2.2}, 'C': {'B':
1.1, 'D': 1.6}}
I am Router E.
Least cost path to router B : EB and the cost is 3.2.
Least cost path to router D : ED and the cost is 2.9.
Least cost path to router F : EDF and the cost is 3.5999999999999996.
Least cost path to router A : EDFA and the cost is 5.8.
Least cost path to router C : EBC and the cost is 4.300000000000001.
```

```
Least cost path to router C : FDC and the cost is 2.3.
34033 A 5000 2 B 6.5 5001 F 2.2 5005
status : {'A': [93.68732190132141, 96.72032380104065], 'E': [93.68732190132141, 96.72032380104065], 'D': [93.68732190132141
, 96.72032380104065]}
topology : {'F': {'A': '2.2', 'D': '0.7', 'E': '6.2'}, 'A': {'B': 6.5, 'F': 2.2}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E':
3.2}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'C': {'B': 1.1, 'D': 1.6}}
I am Router F.
Least cost path to router A : FA and the cost is 2.2.
Least cost path to router B : FDCB and the cost is 3.4.
Least cost path to router E : FDE and the cost is 3.5999999999999996.
Least cost path to router D : FD and the cost is 0.7.
Least cost path to router C : FDC and the cost is 2.3.
E 1564991611.986097
heartbeat:E
difference : 98.73176383972168
[93.68732190132141, 96.72032380104065, 98.73176383972168]
topology : {'F': {'A': '2.2', 'D': '0.7', 'E': '6.2'}, 'A': {'B': 6.5, 'F': 2.2}, 'B': {'A': 6.5, 'C': 1.1, 'D': 4.2, 'E':
3.2}, 'E': {'B': 3.2, 'D': 2.9, 'F': 6.2}, 'D': {'F': 0.7, 'B': 4.2, 'C': 1.6, 'E': 2.9}, 'C': {'B': 1.1, 'D': 1.6}}
I am Router F.
Least cost path to router A : FA and the cost is 2.2.
Least cost path to router B : FDCB and the cost is 3.4.
Least cost path to router E : FDE and the cost is 3.5999999999999996.
Least cost path to router D : FD and the cost is 0.7.
Least cost path to router C : FDC and the cost is 2.3.
```