**Design Document**

Pavan Kumar Lakote

Email ID: pavan.lakotessm@gmail.com

**Problem Description:**

You are given a set of interacting processes that exchange neighbours adjacency data. The process works like this:

1. Each process reads neighbour information and cost from a file
2. Each process builds a network graph from all nodes using this neighbour information

Given this information answer the following questions:

1. Find the shortest path from any to any node given.
2. Find the minimum spanning tree.

Also, reconfigure the network connectivity graph on a node when a node (process) dies or an edge is lost

Issues to be handled:

1. Nodes (process) can start-up and die at any point of time
2. No central manager for the set of nodes (process)

**1. Understanding:**

1. There will be multiple node, each node running as a process.
2. Each node will have information of the all the node in that topology.
3. Considering all the information with the different node, it will create a graph.
4. This graph will the information of how to reach to other node, either having a direct route or the indirect route from other node.
5. Based on this graph we need to create a shortest path from any node to any node in that graph. And also create a bath without cycles (Minimum Spanning Tree).
6. Considering any issue with node (process) i.e. if it dies, shortest path has to be calculated again.
7. And if the node has come back to life, the calculation has to be done again to find the shortest path.
8. Other case, where the node alive but the edge is removed the above case has to be repeated.

**2. Approach:**

1. Process should start with a unique node ID from command line argument.
2. It also takes unique Port ID as a command line argument on which it will be running.
3. Each process will act as a server and as a client.
4. And once the program is up and running, user has to give <= 3 or port number to which it is to be connected.
5. It also takes 3 different port ID with which it is connect to. Limiting that each node is connected to 3 different nodes.

**2.1. Connection Establishment and Management:**

1. When the port number is entered, it will send a ConnectReq type on that port and waits for the response once the ConnectResp is received both will updated there graph table.
2. To check whether the port is alive a message ALIVE will sent to check the connection (edge) is still present and wait for YESALIVE message.
3. Whenever there is not response from the peer node, we will update the graph table.

**2.2. Timer Management:**

1. Need to timer to continuously trigger to check if the connect peers are ALIVE.
2. For every defined time in sec, will trigger a check.

**2.3. Information learning of all nodes:**

1. When all the nodes are up or the adjacency nodes are updated there graph table with the neighbors information. This same information or graph will be with all the nodes.
2. Using Dijkstra’s Algorithm we will calculate the short path having the updated graph and the source.
3. Once this is done we will calculate the spanning tree of that graph.
4. Using Kruskal’s Algorithm we will find the minimum spanning tree of that graph.
5. Whenever the process is down or edge is removed then graph has to be updated.
6. After update the same information has to be shared with all the other peer nodes as well. And step 2 and 4 applied again.

**2.4. Shortest Path Calculation (Dijkstra’s):**

1. First thing is distance from source node is 0.
2. And the distance to the all the other node in the graph is infinity.
3. For the initial step all the other node need to set as unvisited.
4. For every visited node has to be pushed to the queue, until the queue is empty and all the node as visited.
5. While visiting the nodes, path count is maintained. While the search if the shortest path is found path value will be updated.

**Pseudocode for above:**

function Dijkstra(Graph, source):

dist[source] := 0

for each vertex v in Graph:

if v ≠ source

dist[v] := infinity

previous[v] := undefined

end if

add v to Q

end for

while Q is not empty:

u := vertex in Q with min dist[u]

remove u from Q

for each neighbor v of u:

alt := dist[u] + length(u, v)

if alt < dist[v]:

dist[v] := alt

end if

end for

end while

return dist[], previous[]

end function

**2.5 Minimum Spanning Tree (Kruskal’s):**

1. We need a graph with all the node connected with edges.
2. We have to create a weighted graph such that it doesn’t have any cycle within that graph.
3. We need to create a partition of the nodes of that graph, where each node forms a separate set.
4. We also need a queue to maintain the priority of edges and there weights.

**2.6. Data Structures:**

1. Structure to send the packet out:

typedef struct Packet{

int RequestType;

char Data[1024];

}PACKET;

1. Structure that hold the initial setup info:

Typedef struct Connection{

Int NodeID;

Int PortID;

}CONNECTION;

1. Graph data structure:

#define GRAPHSIZE 8

Int iGraph [GRAPHSIZE][GRAPHSIZE];

1. Different Events:

enum{

CONNECTREQ = 0,

CONNECTRESP,

ISALIVE,

YESALIVE,

PEERUPDATE

};