Main Funtion/Diver:

```
#include<iostream>


#include"network/nodes.hpp"
#include "network/graph.hpp"
#include "network/Packates.hpp"
#include "routing/topology.hpp"
#include "routing/Dijikstra.hpp"
#include "routing/Transmission.hpp"
//using namespace std;

int main()
{
    int vertex=10;
    Graph g(vertex);
    Topology topology;
    topology.create_Network(g);
    topology.view_Network(g);

    Dijikstra d;
//    d.shortest_path(g,2,8);

    Packet packet;
    packet.setMessage();
    packet.setHeaderInfo();

    g.genrateTable(packet.getSource(),packet.getDesti());
    g.setTable();

    Transmission t;
    t.startTransmission(g,packet);

    //Print routing tabele
   /* for(auto x:g.Routing_Table)
    {
        cout<<x.first<<" "<<x.second<<endl;

    }
    */
    g.individual_Nodes[packet.getSource()].nodePacket=packet;
   // while(packet.)



}
```

Header file of Node:

```
#include "routingTable.hpp"
#include "Packates.hpp"
#ifndef NODE
#define NODE

class Node
{
    public:
    int Node_id;
   // int data; //char queue  p p2 p
   // int weight;
     map<int,int> Routing_Table;
    int range=1;
    Packet nodePacket;

    //means it can access node inside the range of 10unit
    //range
    //nearby node

};


#endif // NODE
```

Header File of Graph:

```
#include "nodes.hpp"
#include<iostream>
#include <algorithm>
#include <map>
#include <list>
#include<stack>
#include "../routing/Dijikstra.hpp"
#ifndef GRAPH
#define GRAPH
```

```
using namespace std;


class Graph
{
   public:
   int vertex;
   map<int ,map<int,int>> adjList; //use extern
   map<int,int> Routing_Table; //int->current node, int ->nextHope

   Node *individual_Nodes; //It contains all the enformation about each nodes


   Graph(int v) :vertex {v}
   {
      //this->vertex=v;
      individual_Nodes=new Node[v]; //Here all individual nodes created
      for(int i=0;i<v;i++)
      {
         individual_Nodes[i].Node_id=i;
      }

   }

   void addEdge(int src,int des,int wt) //Let graph is bidirectional
   {
      adjList[src][des]=wt;
      adjList[des][src]=wt;

     // info[src].Node_id=src;

   }

   void getInfo(int node)
   {



   }
   void printList()
   {

      for(auto x:adjList)

      {
         cout<<x.first<<" ->: ";
         for(auto y:adjList[x.first])
         {
            cout<<"( "<<y.first<<", "<<y.second<<")";
         }
         cout<<endl;
      }
```

```
    }

    //here routing table is genrated
    void genrateTable(int src,int des)
    {
        Dijikstra d;
        stack<int> s=d.shortest_path(this->adjList,src,des,vertex);

        while(!s.empty())
        {
            int i=s.top();
            s.pop();
            if(des!=i)
            {
                Routing_Table [i]=s.top();
            }
        }


    }

    //Set table to each node
    void setTable()
    {
        for(int i=0;i<vertex;i++)
        {
            individual_Nodes[i].Routing_Table=Routing_Table;
        }

    }

};


#endif
```

Header File of PACKATES:

```
#include<iostream>
#ifndef PACKETS
#define PACKETS
using namespace std;
class Header
```

```
{
   public:
   int length=10,id,src,des,TYP; //id =packet Id
   //next hop
   //if next hop== dst stop
   // routing table calculate by dijik
   //for range- use euclirian distance, if dis<20 then only node is wih in range
   //check medium relaibility
   //src and des ask fro usr and return a path wih in the range
   //if does not have action in routing table then only send t the routing table

};

class Packet
{
   Header header;
   string payload; //Actual data
   //dijikstra for
   public:
   void setMessage()
   {
      std::cout<<"Enter Message :";
      getline(cin,payload);
      //this->payload=msg;
   }
   void setHeaderInfo() //will set information of header
   {
      cout<<"\nEnter Source of message :";
      cin>>header.src;
      if(header.src<0 || header.src>9)
      {
         cout<<"Invalid Source: please check network\n";
         header.src=0;
         exit(1);

      }

      cout<<"\nEnter Desti of message :";
      cin>> header.des;

       if(header.des<0 || header.des>9)
      {
         cout<<"Invalid Desination : please check network\n";
         header.des=0;
         exit(1);

      }
      header.id=1;
      header.TYP=0; //TYP 0 means msg container


   }
```

```
    int getSource()
    {
        return header.src;
    }
    int getDesti()
    {
        return header.des;
    }
    string getMessage()
    {
        return payload;
    }


};


#endif // PACKETS
```

Header file Dijikstra

```
#include<iostream>
#include<map>
#include<stack>
#ifndef DIJIKSTRA
#define DIJIKSTRA
#define inf 1e9
using namespace std;

class Dijikstra
{
  public:



int findMin( map<int,bool> &visited, map<int,int> &weigth,int v)
{
    int minWtNode=-1;
    for(int i=0;i<v;i++) //I represents the nodes
    {
        if(!visited[i] && (minWtNode==-1 || weigth[i]<weigth[minWtNode]))
        {
            minWtNode=i;
        }

    }
```

```
    return minWtNode;

}
stack<int> shortest_path(map<int ,map<int,int>> adjList,int src,int des,int v)
{
  // int v=g.vertex;
   map<int,bool> visited;
   map<int, int> parent;
   map<int,int> weigth;
   for(int i=0;i<v;i++)
   {
      visited[i]=false;
      weigth[i]=inf;
   }
  // parent[0]=-1;
   weigth[src]=0;
   for(int i=0;i<v;i++)
   {
      int minWtNode=findMin(visited,weigth,v);
      visited[minWtNode]=true;
      for(auto neigh: adjList[minWtNode])
      {
         if(!visited[neigh.first]) //negh.first represents destination, 0->1 1 is neigh.first
         {
            if(weigth[neigh.first]>neigh.second+weigth[minWtNode])
            {
               weigth[neigh.first]=neigh.second+weigth[minWtNode];
               parent[neigh.first]=minWtNode;
            } //neigh.second reprsnt wt fro adjList
         }
      }

   }
   stack<int> s;
   s.push(des);
   int i=des;
   while(s.top()!=src)
   {
     // cout<<"->"<<parent[s.top()];
      s.push(parent[s.top()]);
   }


   //cout<<weigth[des];

   cout<<"Total Cost Required For this Transmission :"<<weigth[des]<<endl;
   return s;

}

};
#endif // DIJIKSTRA
```

Header File TopoLogy:

```
#include "../network/graph.hpp"
#ifndef TOPO
#define TOPO
class Topology
{
   int vertex=10;
   public:
   void create_Network(Graph & g) //It will create a graph
   {
      int i=1;
   while(i<(vertex*4)-vertex)
   {
      int u=(rand()%vertex);
      int v=(rand()%vertex);
      if(u!=v)
      {
         i++;
         //cout<<u<<", "<<v<<endl;
         int wt=(u+v)%20;
         g.addEdge(u,v,wt);
      }

   }
   }



   void view_Network(Graph g)

   {
      g.printList();
   }
   /*
   void create_Network(Graph & g)
   {
      g.addEdge(0,1,8);
      g.addEdge(0,4,5);
      g.addEdge(0,7,7);
      g.addEdge(1,2,8);
      g.addEdge(1,6,10);
      g.addEdge(1,3,15);
      g.addEdge(2,3,6);
      g.addEdge(2,5,7);
      g.addEdge(3,4,6);
      g.addEdge(3,10,9);
```

```
        g.addEdge(4,5,10);
        g.addEdge(4,9,9);
        g.addEdge(5,6,5);
        g.addEdge(5,8,6);

        g.addEdge(6,7,9);
        g.addEdge(6,10,13);
        g.addEdge(7,8,6);
        g.addEdge(8,9,8);
        g.addEdge(9,10,11);

    }
     */

};


#endif // TOPO
```

Header File Transmission: //It is Responsible for how packet will transmit

```
#include "../network/graph.hpp"
#include "../network/Packates.hpp"
#include "../network/nodes.hpp"
#ifndef TRANSMIT
#define TRANSMIT


class Transmission
{
          public:
    void startTransmission(Graph &g, Packet &packet)
    {
        int src=packet.getSource();
        int des=packet.getDesti();
        Node current=g.individual_Nodes[src];
        do
        {
```

```
            current.nodePacket=packet;
            cout<<"\nStatus: \nPacket reached at"<<current.Node_id<<endl;

            int nextHopeId=current.Routing_Table[current.Node_id];
            Node nextHope=g.individual_Nodes[nextHopeId];

            cout<<"Packet Sent to"<<nextHopeId<<endl;
            if(current.range < g.adjList[current.Node_id][nextHopeId])
            {
               cout<<"\nNext Hope is out of range!!!"<<endl;
               cout<<"Packet lost at "<<current.Node_id<<endl;
               cout<<"Head Info:"<<endl;
               cout<<"Source :"<<packet.getSource()<<endl;
               cout<<"Desti: "<<packet.getDesti()<<endl;
               cout<<"Msg: "<<packet.getMessage()<<endl;

               cout<<"Transmission end!!"<<endl;

               exit(1);


            }
            else
            {
               current=nextHope;
            }



         }
         while(current.Node_id!=des);

         cout<<"**Transmission Success**"<<endl;
         cout<<"path: " ;

         for(auto x:current.Routing_Table)
         {
            cout<<x.first<<" ->";
         }

         cout<<des;

      }

};
#endif // TRANSMIT
```