

```

1  /**
2      Robust Energy Efficient Multi-path Routing Protocol for Wireless Sensor Networks
3  */
4
5  #include <iostream>
6  #include <fstream>
7  #include <cmath>
8  #include <cstdlib>
9  using namespace std;
10
11 //Global variables
12 struct Node
13 {
14     int sid;    //Sensor ID
15     float x,y;  //Co-ordinates
16     float R_EN; //Residual Energy
17     float BUF_C; //Remaining Buffer Capacity
18 };
19
20 int n; //No. of sensors
21 float TH; //Threshold Value
22 int MaxSNR; //Max Value of SNR
23 int ReqSNR; //Required value of SNR
24 float T_Range; //Transmission range
25 float dist; //Distance
26 int Source; //Source ID
27 int Sink; //Sink ID
28 int pathcount; //path count
29 int H_Msg; //Hello Message
30 float **N_Table; //Neighbor Table
31 Node *S; //Sensors
32
33 void End()
34 //The only way for program to End
35 {
36     cout<<"No More Paths Found"<<endl;
37     exit(69);
38 }
39
40 int find_index(int id)
41 //Returns array index of node, given the sensor ID
42 {
43     int s_index;
44     for(int i=0;i<n;i++)
45     {
46         if(S[i].sid == id)
47         {
48             s_index = i;
49             break;
50         }
51     }
52     return s_index;
53 }
54
55 void Send_RREQ(int i,int j)
56 //Sends RREQ from node index i to j
57 {
58     if(S[j].sid != Source)
59         N_Table[j][n] = 1;
60     if(i == find_index(Sink))
61         cout<<"\nPath No. "<<pathcount+1<<"\n";
62
63     cout<<i<<"->"<<j<<"\n";
64 }
65
66 int Best_link(int sid)

```

```

67 //returns ID of best quality node out of all available neighbor nodes
68 {
69     int nid,s_index;
70     float val=0;
71     s_index = find_index(sid);
72
73     for(int i=0;i<n;i++)
74     {
75         if( (N_Table[s_index][i] != 0) && (N_Table[i][n] == 0) )
76         {
77             if(val < N_Table[s_index][i])
78             {
79                 val = N_Table[s_index][i];
80                 nid = S[i].sid;
81             }
82         }
83     }
84     if(val == 0)
85     {
86         return -1;
87     }
88     return nid;
89 }
90
91 void Pathselection()
92 //Selects a path for Data Transmission
93 {
94     int sid,s_index,check;
95     cout<<"\nPath Selection Started"<<endl;
96     s_index = find_index(Sink);
97
98     if(N_Table[s_index][n]==0)
99     {
100         while(true)
101         {
102             check = 0;
103             for(int i=0;i<n;i++)
104             {
105                 if(N_Table[s_index][i]!=0)
106                 {
107                     if(S[i].sid == Source)
108                     {
109                         check = 1;
110                         Send_RREQ(s_index,i);
111                         break;
112                     }
113                 }
114             }
115             if(check == 1)
116             {
117                 pathcount++;
118                 break;
119             }
120             else
121             {
122                 N_Table[s_index][n] = 1;
123                 sid = Best_link(S[s_index].sid);
124                 Send_RREQ(s_index,find_index(sid));
125                 s_index = find_index(sid);
126             }
127         }
128     }
129     else
130     {
131         int nid;
132         while(S[s_index].sid != Source)

```

```

133         {
134             nid = Best_link(S[s_index].sid);
135             if(nid == -1)
136                 End();
137             Send_RREQ(s_index, find_index(nid));
138             s_index = find_index(nid);
139         }
140         pathcount++;
141     }
142 }
143
144 void DisplayNTable()
145 //Displays the Neighbor Table
146 {
147     for(int i=0;i<n;i++)
148     {
149         for(int j=0;j<n;j++)
150         {
151             cout<<N_Table[i][j]<<"\t\t";
152         }
153         cout<<"\n";
154     }
155     cout<<"\n";
156
157     while(true)
158         Pathselection();
159 }
160
161 float HELLO(int i, int j, int Hello)
162 //Sends Hello Message from node index i to j
163 {
164     float SNR,Val;
165     int temp = rand();
166     SNR = ((temp%(MaxSNR-ReqSNR)+ReqSNR)/((float)MaxSNR)*100;
167     Val = (SNR + S[j].BUF_C + S[j].R_EN)/3.0;
168     return Val;
169 }
170
171 float Dist(int i, int j)
172 //Returns distance between 2 given nodes
173 {
174     float d;
175     d = sqrt(pow((S[i].x-S[j].x),2)+pow((S[i].y-S[j].y),2));
176     return d;
177 }
178
179 void NeighborSelection()
180 //Finds which nodes are neighbors of each other
181 {
182     cout<<"Neighbor Selection Started\n\n";
183     int visited[n] = {0};
184     int Q[n];
185     int top = -1;
186     int point = 0;
187     int index;
188     int check=1;
189
190     Q[++top] = find_index(Source);
191
192     while(check)
193     {
194         index = Q[point++];
195         visited[index] = 1;
196
197         for(int i=0;i<n;i++)
198         {

```

```

199         if(!visited[i])
200         {
201             dist = Dist(index,i);
202             if(dist < T_Range)
203             {
204                 float val = HELLO(index,i,H_Msg);
205                 if(val > TH)
206                 {
207                     N_Table[index][i] = val;
208                     N_Table[i][index] = val;
209                     Q[++top] = i;
210                 }
211             }
212         }
213     }
214     check = 0;
215     for(int i=0;i<n;i++)
216     {
217         if(visited[i]==0)
218         {
219             check = 1;
220             break;
221         }
222     }
223 }
224 cout<<"Neighbor Selection Ended\n\n";
225 DisplayNTable();
226 }
227
228 void inputfromfile()
229 //Inputs Sensor Information from infile.txt
230 {
231     ifstream infile;
232     infile.open("infile.txt");
233     for(int i=0;i<n;i++)
234     {
235         infile>>S[i].sid;
236         infile>>S[i].x;
237         infile>>S[i].y;
238         infile>>S[i].R_EN;
239         infile>>S[i].BUF_C;
240     }
241     infile>>Source;
242     infile>>Sink;
243
244     cout<<"Input Complete\n\n";
245
246     NeighborSelection();
247 }
248
249 void Initialization()
250 //Inputs Basic Information from initfile.txt and allocates memory to arrays
251 {
252     ifstream infile;
253     infile.open("initfile.txt");
254     infile>>n;
255     infile>>T_Range;
256     infile>>MaxSNR;
257     infile>>ReqSNR;
258     infile>>TH;
259
260     cout<<"Initialization Complete\n\n";
261
262     //Memory Allocation for Arrays:
263     S = new Node[n];
264     N_Table = new float*[n];

```

```

265     for(int i = 0; i < n; ++i)
266         N_Table[i] = new float[n+1];
267
268     //Other Initializations :
269
270     pathcount = 0;
271
272     for(int i=0;i<n;i++)
273         for(int j=0;j<=n;j++)
274             N_Table[i][j] = 0;
275
276     //Calling the input function
277     inputfromfile();
278 }
279
280 int main()
281 //Just Calls Initialization Method
282 {
283     Initialization();
284     return 0;
285 }

```