Energy-Efficient Routing Algorithms for Wireless Sensor Networks

Noamaan Khan
Tory Kennedy
2. Introduction

Objective
To compare and contrast various routing algorithms based on a metric of power utilization and propose concepts to improve these protocols and/or devise our own which utilizes the best feature of each of these.

What is the problem
Current routing protocols are inefficient for Wireless Sensor Networks due to undistributed and excessive power consumption and lack of fault tolerance. Much research is being done to develop a more efficient and optimized protocol utilizing routing, but no conclusive evidence has determined which protocol is best for minimized power utilization.

Why this is a project related to this class
The project involves implementing numerous various routing algorithms, a key feature of computer networks and develops on the growing field of Wireless Sensor Networks (WSN).

Why other approaches are no good
Other approaches may propose new algorithms without an emphasis on comparing to older algorithms. Also they may fail to focus on the particular facet of power consumption, a major area of concern with WSN. Our approach will account for power utilization and distribution in a more readily comparable way for all protocols measured.

Why you think your approach is better
We propose to do a conclusive study of different algorithms out there and try to devise an algorithm that is more efficient and better. We will simulate our algorithm and compare with other algorithms to derive conclusions based tangible data. Our approach will involve emphasizing one of two concepts: considering battery life or alternating between paths, further detail on both can be found below.

Statement of the problem
To derive an algorithm intended to minimize energy consumption in wireless sensor networks.

Area or scope of investigation
This project will focus mostly on the energy consumption aspect of sensor technology, ignoring traffic congestion, economic factors, real-world factors, etc. No actual internet traffic or utilization of protocols in programming will be conducted. Only routing simulations will be developed in C++ code and compared using generic metrics for assumptions on power such as amount of time transmitting a signal and whether or not a node was utilized for data transmission.

3. Theoretical basis and literature review

Definition of the problem
To derive a multipath algorithm to minimize energy consumption in wireless sensor networks.

Theoretical background of the problem
Wireless sensor networks are in use everywhere today. They have many applications ranging from home security to military uses. To implement these networks similar technology has been used which is used in contemporary network systems. As it turns out, present network systems use single path routing, using a single line of communication to transmit data over network. This results in an inefficient use of network resources. Also exhausting few nodes of power. Multipath routing on the other hand could help to distribute data across multiple lines of communication limiting power use on each node.
**Related research to solve the problem**

Research into improving routing algorithms using multiple paths is in its infancy, but fairly extensive, with a broad range of areas of improvement. Below are just a few of the protocols proposed in the past 5 years which emphasize power efficiency.

- **EENDMPR** - Energy Efficient Node Disjoint MPR
- **AOMDV** - Ad Hoc On-Demand Mobile Distance Vector
- **DD** - Direct Diffusion
- **HREEMR** - Highly-resilient, energy-efficient MPR
- **LIEMRO** - Low-Interference Energy-Efficient MPR
- **RFTM** - Reliable Fault-Tolerant Multipath
- **EECA** - Energy Efficient Collision Aware
- **EQSR** - Energy Efficient and QoS Aware
- **REBMR** - Rumor as an Energy-Balancing MPR
- **REER** - Robust and Efficient MPR
- **BP-CMPR** - Bandwidth-power Aware Cooperative MPR
- **EEOR** - Energy-Efficient Optimal MPR
- **RELAX**
- **DSR** - Dynamic Source Routing
- **OSPF** - Open Shortest Path First

**Advantage/disadvantage of those research**

Most of the research is fairly recent and relevant to today’s real-world design applications. They also directly emphasize our interest of energy conservation and apply quite well to this project topic. Unfortunately since most of the research is new and not well established yet, while many protocols have been proposed, very few reference other proposals or compare to more well-known ideas. This makes for difficulties in drawing comparisons between the numerous proposals given that there is no basic foundation from which they are all built off of (Although most do utilize the Dijkstra algorithm for shortest path delay).

**Your solution to solve this problem**

We devise two algorithms that find shortest loop-free paths from a source to a target gateway node, and distributes communication data on different paths based on traffic to minimize power consumption at all nodes on the whole. We simulate some conditions to analyse our algorithm and compare with other algorithms addressing the same issue.

The first algorithm will involve the standard Dijkstra algorithm used in most networking communication algorithm which involve weighted structures (such as ad hoc). This algorithm determines the shortest possible path between 2 points given hops throughout a network. For our purposes, we plan to instead utilize a metric which involves a function of the distance but also incorporates the battery life remaining on the node. This is essential since our aims are not to merely have the shortest time, but, as the node loses battery life, to extend the battery life by reducing the likelihood of choosing that node for a given path.

This metric is evaluated as such:

\[
\text{metric} = \frac{P_t}{f(A)}
\]

Where A is the remaining battery on a given node and Pt is the transmit power for the node to its parent. 

\(f(A)\) is a function of A as yet undetermined. Many variations will be attempted but the aim is to bring about an asymptotic relationship as A approaches 0 (battery drains) and to approach 1 for larger values of A (full battery).
The second algorithm we plan to analyze is more basic, and involves simply alternating parent nodes in the Dijkstra algorithm. In hopes of greater spreading of the routing paths across nodes while still maintaining the simplicity that each node retains little information, we would assign each node 2 parents instead of 1, and allow the node to alternate between the 2 as it transmitted data; first sending info to parent 1, then transmitting to parent 2 next time it sensed data. Ideally this would produce greater distribution (if only slightly) due to greater spreading out of the path routes.

**Where your solution differs from others**
Our solution involves comparing different algorithms that are currently on the cutting edge of WSN routing based on a single metric of power efficiency. We will then propose a new protocol which implements features of each of these such as to improve the overall efficiency. Our solution also emphasizes simplicity and attempts to improve on the basic core functionality of the routing algorithm which could then possibly be applied to any other algorithm in the WSN hierarchy.

**Why your solution is better**
Taking a holistic approach where the key beneficial aspects of each concept is combined often yields a more comprehensive and improved design. Narrowing down the design concept to a single issue (energy efficiency) also simplifies the problem and allows for a streamlined approach to improving the protocol.

4. Hypothesis (or goals)

**Positive/negative hypothesis**
We believe that both algorithms as proposed will contribute to the lifetime of the network and improve long term sustainability at the expense of time and possibly a loss of detection in certain cases (as some nodes may fail and thus certain events will go underdetected in certain area, but the overall network would survive).

5. Methodology

**How to generate/collect input data**
Total number of nodes, maximum area under consideration, and minimum distance between sensor nodes is given as input. A number of coordinates are randomly generated corresponding to each node, which represent their location in the area under consideration. Then distances are calculated between different nodes and nodes falling within minimum distance are connected with an edge. The edges distances (weights) are stored in a table. This process generates an environment for simulation of wireless networks.

The total number of sensor nodes will be a randomly generated number typical for WSN (100 - 1000). Failing nodes and when in the time cycle they fail will also be randomly generated. Which node experiences the notable sensed information will also be random. The placement for nodes will be assumed to be equidistant and optimized for wireless transmission such that little to no overlap occurs. Simulation results will last for an extended duration of “simulation time units” such that a reasonable amount of senses and failures can occur and be compared.

**How to solve the problem**
A comparison of each algorithm will be drawn given identical simulation characteristics based on the energy efficiency metric such that the most optimized protocol can be concluded. We choose to simulate two programs to compare the results of two different approaches. In the first program, we modified Dijkstra’s algorithm to improve the performance of single shortest path algorithm in wireless sensor network. In the second program, we try to implement a multipath routing algorithm which distributes information uniformly over network. The two approaches are discussed below in detail.

**Algorithm design**
DMA Dijkstra-Modified Algorithm
As detailed above, this algorithm involves modifying the general Dijkstra algorithm such that instead of incorporating just the distance of the router, a metric will be used developed as a function of the transmit power and the remaining battery life on the node. The metric can be defined as:

$$\text{metric} = \frac{\text{transmit power}}{\text{battery life}} = c \cdot \frac{d^2}{\sqrt{A/P_i}}$$

Where $c$ is a constant we defined, $d$ is the separation between the nodes, and $A$ is the remaining battery life of the transmitting node and $P_i$ is the initial power of the node. The denominator will vary from 0 to 1 such that for smaller values of $A$, the metric will appear large, and for larger values of $A$ the metric will appear relatively small. This will have the effect of causing the distance as considered in the Dijkstra algorithm as appearing further away for nodes with less battery, thus reducing the likelihood of selecting that node.

Multipath Routing Program
In this program we improvised on the above program by routing packets on multiple paths in contrast to single path. During execution of Dijkstra’s algorithm, all the connected nodes to a given node are assigned the given node as parent node. Whenever any node detects an even near it, it transmits the information to the gateway node switching between the parents each time it sends some information. This may cause an equally distributed use of power across network. Below is the program developed to simulate the Multipath Routing.

Language used
C. TCL to automate testing.

Tools used
Linux servers to run code. Eclipse.

How to generate output
Results will consist of graphs comparing energy utilization for varying randomized input parameters. Energy utilization will be determined based on amount of node usage both on a single node as well as distributed across the WSN.

How to test against hypothesis
While current hypothesis is limited in scope, we can verify if the final design proposed by our team improves on overall energy efficiency by running this new protocol through the same simulation and input parameters as the other protocol designs and comparing. To validate the hypothesis, our protocol would need to achieve greater efficiency than all others simulated.

How to prove correctness (required by dissertation only)
The resulting energy efficiency computed for each protocol will be compared to original results from the paper to ensure correctness in implementation.

7. Data analysis and discussion
Output generation
Figure 1: Typical graph of random node placement and gateway given 100 x 100 area
Output analysis
For the DMA, results were inconclusive given the lack of variation between the algorithm and the standard Dijkstra algorithm. Using a metric of distance (as in the Dijkstra algorithm) or just remaining battery life generally results in similar selections for the parent node at each node due to the duality between distance and power consumption. Since transmit power is already a function of the distance, and battery is obviously related to transmit power, the result is a metric very similar to the already used distance metric. For the majority of cases, the same parent node is selected and only a few times is a different node chosen.

Ultimately this difference is insignificant and does not show any trend in either direction. A TCL script was created to run numerous simulations comparing the DMA to the original algorithm for various seed values (such that a new random number sequence is used). After running this script for 100+ simulations, we observed a close match between the two algorithms. We also observed the rate of success for one as compared to the other alternated between the two; when one algorithm performed slightly better one time, the other may perform better another time.
As for Multipath Routing, the results were not helpful in deriving any conclusion. First of all, the multiple paths generated from a node to gateway were not disjoint. Secondly, the assumptions made in the simulation, where all the nodes near an event will report to the gateway node, turned out to cause duplications of same message causing the network to fail soon before valuable number of events could be reported. Also there were many nodes which drained out of battery sooner than others causing the network to fail well before the network was efficiently used.

Hence it could be analysed that a better multipath routing algorithm needs to be developed to improve the efficiency of the network. The multiple paths generated must be as disjoint as they can be. And also, the duplication of transmitted messages should be efficiently handled.

**Compare output against hypothesis**

Unfortunately it seems as if DMA was not as successful as originally hoped at producing a unique and effective improvement on Dijkstra. The variation between remaining battery and distance was not significant enough to have an effect on the general determination of the parent node in most cases. The result was an algorithm slightly different but with little-to-no improvement over the original.

Also in the case of Multipath Routing, the results were not as successful as originally hoped for. Although switching between multiple parents while transmitting information to the gateway node seemed to uniformly distribute power usage across network, to the contrary it depleted few sensor batteries before others. A better conclusion may be derived from more realistic simulation with disjoint multipaths.

**Abnormal case explanation (the most important task)**

**Statistic regression**

As explained, a TCL script was generated (code in Appendices) to expand the simulations for a large number of cases as well as to allow a variation of constants such as number of nodes, area, etc. The script was effective in allowing for extensive testing and greatly growing the data pool, but results proved insignificant for the DMA.

8. Conclusions and recommendations

**Summary and conclusions**

Improving upon the typical single path routing algorithm for the growing system of wireless networks is a formidable field with many challenges. The algorithm, though seemingly complex, can have a great deal of variation and alternate factors which may weigh heavily on results. Node position, transmission power, event location, gateway location, node data storage, node data updating; all these points can be very relevant and greatly affect the outcome of the system if not taken into consideration properly.

For our simulations, considering the node as ‘unintelligent’ and knowing only the parent to whom it is assigned to communicate with was a major factor in determining an effective algorithm which adhered to this reality of WSNs. Because of this, the complexity of coding greatly increased and many more considerations emerged.

The result was the DMA and Multipath algorithms; both proposed as tools to improve power distribution given the limited knowledge of the sensor nodes. Unfortunately each algorithm had its issues with performance and neither produced particularly favorable results. The DMA seemed reasonable but was too similar in practice to the actual Dijkstra algorithm. The Multipath algorithm suffered from an inherent assumption made in the Dijkstra algorithm which prevented it from being used for the case of alternate paths properly. Instead the algorithm suffered from long-winded routes and failed connections.
The results tell us, similar to the vast array of research currently being conducted, that there is a great deal of research and understanding still to be performed about these networks. There are many seemingly rationale and mathematically plausible suggestions which may fail when actually implemented or simulated. The high variability and large number of factors can results in an easily unstable system. As such, an optimized algorithm will need to consider all these factors appropriately and will need a great deal of simulations to ensure successful results.

**Recommendations for future studies**

While designing we had many ideas for how to improve the algorithm which, unfortunately, we were unable to attempt due to time or complexity restraints. An obvious consideration would have been to reduce the redundancy we observed. When an event occurred, all nodes within range transmitted the details back, but a single node may have sufficed (depending how a user requires the WSN to function). Multiple transmissions could be reduced by ignoring signals, or piggybacking, or other means of doubling transmission together.

An improvement on the DMA could be to more finely tune the equation for the metric. The distance variable or the battery seem to dominate for most cases, but its possible that a more accurate and well-written equation could better manipulate results such that distance is considered, but battery considerations are made more significant as the battery drains. Its also possible that this ‘sweet spot’ only exists for a certain simulation of certain random placements, and would not be effective for all cases.

While multipath routing does seem to be more energy efficient, better algorithms need to be derived to find disjoint sets of paths. And then, using these multipath routes to efficiently transmit using minimum energy.

9. **Bibliography**


Hongli Xu; Liusheng Huang; Chunming Qiao; Yindong Zhang; Quan Sun, "Bandwidth-Power Aware Cooperative Multipath Routing for Wireless Multimedia Sensor Networks," *Wireless Communications, IEEE Transactions on*, vol.11, no.4, pp.1532,1543, April 2012


10. Appendices

Programming Code
Multipath Program:

//Program to simulate Wireless Sensor Networks using Multipath Routing
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_POWER 1000       //Maximum power
#define C 0.1                 //Constant used in power equation

//Structure used in keeping tracks of parents
typedef struct nn
{
    int id;
    struct nn *next;
}NN; //neighbour node

//Structure representing node
typedef struct node
{
    int id;       //Node ID
    int x;        //X- coordinate
    int y;        //Y-Coordinate
    NN *neighbours; //Parents it can send information to
    int next_node_to_send; //Next parent to send received information
    int reachable_flag; //Flag representing nodes reachability from gateway node
    int parent;   //Parent from dijsktra's algorithm
    int battery;  //Battery level of node
    struct node *next; //Pointer to next node
}NODE;

//Structure to represent event
typedef struct event
{
    int id;
    int x;
    int y;
    int received;
    struct event *next;
}EVENT;
//Structure storing edge information
typedef struct table
{
    int from;  //Node-1
    int to;   //Node-2
    int dist; //Distance between nodes
    struct table *next; //Pointer to next table
} TABLE;

int total_edges, no_of_nodes, max_dist, nhbr_dist; //Total number of edges, Maximum distance of plain, Maximum neighbouring distance
NODE *start = NULL;
TABLE *start_table = NULL;

/**********************************************
Function : plot_nodes
Purpose  : To plot nodes on a graph
***********************************************/
void plot_nodes()
{
    int x,y,i;
    NODE *temp;
    printf ("\nWSN plot\n");
    for (y = max_dist-1; y >= 0; y--)
    {
        if (y%10==0) printf ("%d", y/10);
        else printf (" ");
        printf ("%d", y%10);
        for (x = 0; x < max_dist; x++)
        {
            temp = start;
            for (i=0; i<no_of_nodes; i++)
            {
                if ((x==temp->x) & (y==temp->y) & (temp->battery>0)) break;
                temp = temp->next;
            }
            if (i==0) printf ("0");
            else if (i<no_of_nodes) printf ("X");
            else printf ("_");
        }
        printf ("\n");
    }
    printf (" ");
    for (x = 0; x < max_dist; x++)
    if (x%10==0) printf ("%d", x/10);
}
else printf(" ");
printf("\n ");
for (x = 0; x < max_dist; x++) printf("%d", x%10);
printf("\n");
}

/**************************************************************/
Function : write_nodes
Purpose : To write nodes to output
/**************************************************************/
void write_nodes(char *filename)
{
    FILE *file;
    NODE *temp;
    int i;

    file = fopen(filename, "w");
    temp = start;
    for (i=0; i<no_of_nodes; i++)
    {
        if (temp->battery>0) fprintf(file, "%d,%d,\'k\'\n", temp->x,temp->y);
        else fprintf(file, "%d,%d,\'r\'\n", temp->x,temp->y);
        temp=temp->next;
    }
    fclose (file);
}

/**************************************************************/
Function : edge_exists
Purpose : To check if edge exists
/**************************************************************/
int edge_exists(int dest_node)
{
    int i;
    TABLE *temp_table;

    temp_table = start_table;
    for(i=0; i<total_edges; i++)
    {
        if(temp_table->from == 0)
        {
            if(temp_table->to == dest_node)
            {
               return temp_table->dist;
            }
        }
        temp_table = temp_table->next;
    }
return 0;
}

/*************************************************************/
Function: gateway_access
Purpose: To check if gateway is accessible from at least one node
*************************************************************/
int gateway_access()
{
    int i, dist, remaining_battery;
    NODE *temp;

    temp = start->next;
    for(i=1; i<no_of_nodes; i++)
    {
        dist = ((int)sqrt(((temp->x)-start->x)*((temp->x)-start->x)) +
        (((temp->y)-start->y)*((temp->y)-start->y)));
        remaining_battery = temp->battery - C*dist*dist;
        if ((dist < nhbr_dist) && (remaining_battery>0))
        {
            if(temp->id == 83)
                continue;
            return 1;
        }
        temp = temp->next;
    }

    return 0;
}

/*************************************************************/
Function: find_next_neighbour
Purpose: To find next parent to send info to, for a given node
Input: Node pointer
Return: Nothing
*************************************************************/
void find_next_neighbour(NODE *temp)
{
    int nhbr;
    NN *temp_nn;

    nhbr = temp->next_node_to_send;
    temp_nn = temp->neighbours;

while((temp_nn->id != nhbr) && (temp_nn->next != NULL))
    temp_nn = temp_nn->next;
if((temp_nn->next == NULL) && (temp->neighbours != NULL))
    temp->next_node_to_send = temp->neighbours->id;
else
    temp->next_node_to_send = temp_nn->next->id;
}

/************************************************************
 Function : send_transmission
 Purpose : To transmit information from a given node
 Input : Index of node to send info from
 Return : Success(1) or Failure(0) to send
*************************************************************/

int send_transmission(index)
{
    int i, dist, flag, repeat_flag;
    NODE *temp, *parent;
    int record_node;//used to see if same node is used to send again in the do/while loop below
    while (1)
    {
        temp = start;
        //printf("index: %d", index);
        for (i=0; i<index; i++) temp = temp->next;
        if (temp->id==0) return 1;
        else if (temp->battery<=0 || temp->reachable_flag==0) return 0;
        parent = start;
        record_node = temp->next_node_to_send;
        if(record_node == -1)
            return 0;
        flag = 1;
        repeat_flag = 0;
        do{
            parent = start;
            for (i=0; i<temp->next_node_to_send; i++) parent = parent->next;
            if((parent->battery < 0 || (parent->reachable_flag == 0))
                find_next_neighbour(temp);
            if(temp->next_node_to_send == record_node)
            {
                flag = 0;
                repeat_flag = 1;
                break;
            }
        }
        else
        {
            //printf("flag: %d", flag);
            //printf("repeat_flag: %d", repeat_flag);
            flag = 0;
            repeat_flag = 0;
            break;
        }
    }
}

dist = ((int)sqrt(((temp->x)-parent->x)*((temp->x)-parent->x)) +
(((temp->y)-parent->y)*((temp->y)-parent->y))));

temp->battery = temp->battery - C*dist*dist;
flag = 0;
find_next_neighbour(temp);

} while(flag);

if(repeat_flag || (temp->battery < 0)) return 0;

printf ("Power on node %d updated to %d, parent is %d\n", temp->id, temp->battery, temp->parent);

index=parent->id;

/**********************
Function : add_neighbour
Purpose : To add parent to a given node
Input : Node ID and its parents ID
Return : Nothing
**********************/

void add_neighbour(int sensor, int nhbr)
{
    int i, cnt, dist;
    NODE *temp, *from, *to;
    NN *temp_nn;

    /*from = start;
    while(from->id != sensor)
        from = from->next;
    
    to = start;
    while(to->id != nhbr)
        to = to->next;
    */

    dist = ((int)sqrt(((from->x)-(to->x))*((from->x)-(to->x)) + ((from->y)-(to->y))*((from->y)-(to->y))));

    temp = start;
    for(i=0; i<no_of_nodes; i++)
    {
        if(temp->id == sensor)
        {
            if(temp->neighbours == NULL)
            {
...
temp->neighbours = (NN*)malloc(sizeof(NN));
temp->neighbours->id = nhbr;
temp->neighbours->next = NULL;
}

} else {

temp_nn = temp->neighbours;
if(temp_nn->id == nhbr)
   return;
while(temp_nn->next != NULL)
{
   if(temp_nn->next->id == nhbr)
      return;
   temp_nn = temp_nn->next;
}

while(temp_nn->next != NULL)
   temp_nn = temp_nn->next;

temp_nn->next = (NN*)malloc(sizeof(NN));
temp_nn->next->id = nhbr;
temp_nn->next->next = NULL;
}

temp->next_node_to_send = nhbr;
}

temp = temp->next;

}/**
 *************************************************************
 Function : dijkstras_algo
 Purpose : Implementing dijkstra's algorithm to find shortest paths
 Input : None
 Return : Nothing
 *************************************************************/
void dijkstras_algo()
{
    TABLE *temp_table;
    NODE *temp, *temp2;
    NN *temp_nn;
    int S[no_of_nodes], dist[no_of_nodes], edge;
    int i, j, u, min_dist, transmit_p;

    S[0] = 0;
    dist[0] = 0;
    temp = start->next;
for(i=1; i<no_of_nodes; i++)
{
    S[i] = 0;
    if (temp->battery<=0) S[i] = 1;
    temp->parent = -1;
    dist[i] = max_dist*2;
    edge = edge_exists(i);
    if (edge)
    {
        dist[i] = edge;
        temp->parent=0;
        temp->neighbours = (NN*)malloc(sizeof(NN));
        temp->neighbours->id = 0;
        temp->neighbours->next = NULL;
    }
    temp=temp->next;
}

S[0] = 1;
start->parent = 0;
for(i=1; i<no_of_nodes; i++)
{
    min_dist = max_dist*2;
    for(j=0; j<no_of_nodes; j++)
    {
        if((min_dist > dist[j]) && (S[j] == 0))
        {
            min_dist = dist[j];
            u = j;
        }
    }
}

S[u] = 1;
temp2 = start;
for(j=0; j<no_of_nodes; j++)
{
    if (temp2->battery>0)
    {
        if(S[j] == 0)
        {
            temp_table = start_table;
            while(temp_table != NULL)
            {
                if (((temp_table->from == u) && (temp_table->to == j))
                {
                    add_neighbour(j, u);
                    if (dist[j] > (dist[u]+(temp_table->dist)))
                    {
                        
        }
transmit_p =
\[ C^* \left( \text{dist}[u] + (\text{temp_table->dist}) \right) \left( \text{dist}[u] + (\text{temp_table->dist}) \right) \]

if
((temp2->battery-transmit_p)>(3*transmit_p))
{
    dist[j] = dist[u] + (temp_table->dist);
    temp2->parent = u;
}

}

temp_table = temp_table->next;

}

temp2 = temp2->next;

}

// Clean up separated nodes
temp = start;
for(i=0; i<no_of_nodes; i++)
{
    // For disconnected nodes, effective battery is 0
    if (temp->parent==1) temp->battery=0;
    temp=temp->next;
}
temp = start;
for(i=0; i<no_of_nodes; i++)
{
    printf("%d - (x:%d, y:%d)\tparent:%d\tbattery:%d", i, temp->x, temp->y, temp->parent, temp->battery);
    temp_nn = temp->neighbours;
    printf("\n");
    while(temp_nn != NULL)
    {
        printf(" %d ", temp_nn->id);
        temp_nn = temp_nn->next;
    }
    temp = temp->next;
}
//exit(0);
printf("\n");

//Main Program
int main() {

int i, j, dist, seed, max_nodes, event_total, rx_good;
NODE *curr, *temp;
TABLE *curr_table, *temp_table;
EVENT *start_event, *event, *next_event;

//scanf("%d", &max_dist);
//scanf("%d", &nhbr_dist);
//scanf("%d", &max_nodes);
//scanf("%d", &seed);
//Assigning input values
max_dist = 100;
hnbr_dist = 25;
max_nodes = 100;
seed = 200;
no_of_nodes = max_nodes;

//Creating nodes
srand(seed);
start = (NODE*)malloc(sizeof(NODE));
start->id = 0;
start->battery = MAX_POWER;
start->x = rand()%max_dist;
start->y = rand()%max_dist;
start->neighbours = NULL;
start->reachable_flag = 0;
start->next_node_to_send = -1;
start->next = NULL;
curr = start;
for(i=1; i<no_of_nodes; i++)
{
    temp = (NODE*)malloc(sizeof(NODE));
    temp->id = i;
    temp->battery = MAX_POWER;
    //if(i == 83 || i==98 ) temp->battery = 0;
    temp->x = rand()%max_dist; //max_dist;
    temp->y = rand()%max_dist; //max_dist;
    temp->neighbours = NULL;
    temp->reachable_flag = 0;
    temp->next_node_to_send = -1;
    temp->next = NULL;
    curr->next = temp;
    curr = temp;
}

plot_nodes();
write_nodes("nodes.txt");
curr = temp = start;
for(i=0; i<no_of_nodes; i++)
{
    for(j=0; j<no_of_nodes; j++)
    {
        dist = (int)sqrt(((temp->x)-(curr->x))*((temp->x)-(curr->x)) +
        ((temp->y)-(curr->y))*((temp->y)-(curr->y)));
        if(dist <= nhbr_dist)
        {
            if(start_table == NULL)
            {
                start_table = (TABLE*)malloc(sizeof(TABLE));
                start_table->from = curr->id;
                start_table->to = temp->id;
                start_table->dist = sqrt(((temp->x) - (curr->x))^2 + ((temp->y) -
                (curr->y))^2);
                start_table->next = NULL;
                curr_table = start_table;
            }
            else
            {
                temp_table = (TABLE*)malloc(sizeof(TABLE));
                temp_table->from = curr->id;
                temp_table->to = temp->id;
                temp_table->dist = dist; //sqrt(((temp->x) - (curr->x))^2 + ((temp->y) -
                (curr->y))^2);
                temp_table->next = NULL; //sqrt(((temp->x) - (curr->x))^2 + ((temp->y) -
                (curr->y))^2);
                curr_table->next = temp_table;
                curr_table = temp_table;
            }
            total_edges++;
            temp->reachable_flag = 1;
        }
        temp = temp->next;
    }
    curr->reachable_flag = 1;
    curr = curr->next;
    temp = start;
}

//Finding shortest paths, and also parents
dijkstras_algo();

if (!gateway_access())
{
    printf("Coverage not initially available. System has failed\n");
    exit(1);
}
//Creating events
    event_total=1;
    rx_good=0;
    start_event = (EVENT*)malloc(sizeof(EVENT));
    start_event->id = 0;
    start_event->x = rand()%max_dist;
    start_event->y = rand()%max_dist;
    start_event->next = NULL;
    
    event = start_event;

Simulating Wireless Sensor Networks, transmitting event information to gateway node
while(1)
{
    printf("Event-total: %d   Successful events: %d\n", event_total, rx_good);
    printf("Event at x:%d y:%d\n", event->x, event->y);
    temp = start->next;
    for (i=1; i<no_of_nodes; i++)
    {
        if (temp->battery>0)
        {
            dist = (int)sqrt(((temp->x)-event->x)*((temp->x)-event->x)) +
            (((temp->y)-event->y)*((temp->y)-event->y)));
            if (dist <= nhbr_dist)
            {
                if (send_transmission(temp->id))
                {
                    event->received=1;
                    printf("Transmission sent from node %d\n", temp->id);
                } else printf("Failed to transmit from %d\n", temp->id);
            }
        }
        temp = temp->next;
    }
    if (event->received)
    {
        rx_good++;
        printf("Event (%d,%d) successfully received\n",event->x,event->y);
    } else
    {
        printf("Event (%d,%d) failed to be detected\n",event->x,event->y);
    }
    dijkstras_algo();
    if (!gateway_access())
    {
        printf("All nodes depleated or inaccessible. Successfully detected %d out of %d events.\n", rx_good, event_total);
    }
}
exit(0);

}
event_total++;
next_event = (EVENT*)malloc(sizeof(EVENT));
next_event->id = event_total-1;
next_event->received = 0;
next_event->x = rand()%max_dist;
next_event->y = rand()%max_dist;
next_event->next = NULL;
event->next=next_event;
event = next_event;
}
return 0;
}

// TCL script
#!/usr/bin/tclsh
#exec make

set run_test_total 100
set max_dist 100
set nhbr_dist 15
set max_nodes 100
set seed [expr {int(rand()*4294967296)}]

proc writeInput {max_dist nhbr_dist max_nodes seed} {
    if [catch {open input.txt w} fileID] {
        puts stdout "ERROR: Cannot open file: $filename"
        exit
    }
    puts $fileID "$max_dist
$nhbr_dist
$max_nodes
$seed"
    close $fileID
}

proc writeOutput {line} {
    if [catch {open test_results.txt a} fileID] {
        puts stdout "ERROR: Cannot open file: $filename"
        exit
    }
    puts $fileID "$line"
    close $fileID
}

proc readFile {} {
    if [catch {open output.txt r} fileID] {
        puts stdout "ERROR: Cannot open file: $filename"
        exit
    }
}

set fileData [read $fileID]
close $fileID
return $fileData
}

#for {set nhbr_dist 5} {$nhbr_dist < 75} {incr nhbr_dist} {
#for {set max_nodes 100} {$max_nodes < 1000} {incr max_nodes} {
    for {set i 0} {$i < $run_test_total} {incr i} {
        set seed [expr {int(rand()*4294967296)}]
        writeInput $max_dist $nhbr_dist $max_nodes $seed
        if {[catch {exec ./dijks.o < input.txt > output.txt} result] == 0} {
            set output_data [readFile]
            set output_lines [split $output_data 
]
            foreach line $output_lines {
                if {[regexp {Successfully detected} $line]} {
                    puts "$i:$line"
                    writeOutput $line
                }
            }
        } else {
            puts stderr "Failed to execute program properly"
        }
    }
    if {[catch {exec ./dijks_mod.o < input.txt > output.txt} result] == 0} {
        set output_data [readFile]
        set output_lines [split $output_data 
]
        foreach line $output_lines {
            if {[regexp {Successfully detected} $line]} {
                puts "$i:$line"
            }
        }
    } else {
        puts stderr "Failed to execute program properly"
    }
}
//End