Energy-Efficient and Adaptive Algorithms for Constructing Multipath Routing in Wireless Sensor Networks

Computer Networks Assignment Report

SUBMITTED TO

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAMAIAH INSTITUTE OF TECHNOLOGY

(Autonomous Institute, Affiliated to VTU)

Bangalore – 54

Submitted by

Sashank Agarwal	**1MS16CS090**
Shiv Dutt Tripathi	**1MS16CS096**
Rohit Singh	**1MS16CS080**

SUPERVISED BY

Faculty

Dr. Shilpa Choudhary

Table Of Contents

Section I

Introduction to Wireless Sensor Networks

Wireless sensor network (WSN) refers to a group of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location as shown in Fig 1.1. WSNs measure environmental conditions like temperature, sound, pollution levels, humidity, wind, and so on.

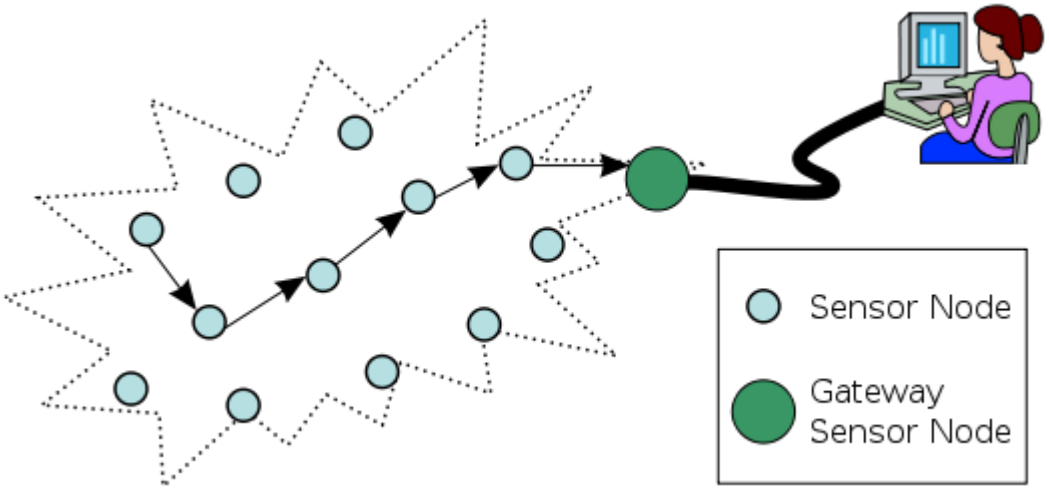
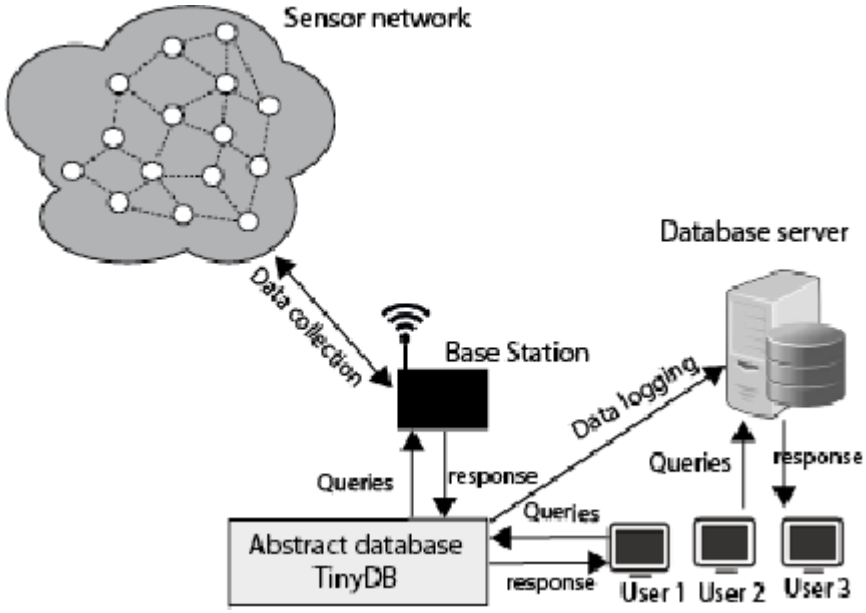


Fig 1.1 Typical multi-hop wireless sensor network architecture

An important aspect of energy-efficiency is performing in-network aggregation while routing data from source sensors through intermediate nodes in the network. Servicing an aggregate query, say Qi, involves disseminating the query from a given sink si, that requested it to all the target sensing nodes relevant for its processing; and sending the results from each of the target nodes back to the sink node [1]. An effective way of disseminating the queries and gathering the query answer is using a tree structure rooted at the sink as shown in Fig 1.2. Once the tree is constructed, each of its nodes has a dual role: forward the answer-sets measured by the children towards the sink; and perform some local aggregation of the data, in order to reduce the communication overhead.

Fig 1.2 Query Processing with WSN Abstract Database.



Section II:

Statement Of The Problem

Query Model:

- Users need to be able to interact with the sensor network - by connecting to a sink node and submitting queries of interest for which the network must provide accurate answers in a timely manner.

- Query Language – SQL.

- A wireless sensor network needs to provide mechanisms for query processing that are both energy and bandwidth conscious [2].

- Q: SELECT ALL/MIN/MAX/AVG (measurement)

  FROM Region (R1(x1, y1, ... , xn, yn))

  WHERE Condition (measurement)

  FOR Lifetime

  SAMPLE EVERY Sampling Interval

Problem Formulation:

- Sensor nodes that are outside the sampling region are also important - used in data-relay duties, making the connection between the producer, in the sampling region, and its consumer, the sink node.
- A single shortest path is discovered and used for data transmissions.
- In wireless sensor networks – high density of nodes – shortest path very close to line segment connecting them [8].
- Develop hot-spots – caused due to overuse of same nodes - should be avoided.
- This paper provides a new multipath protocol for mitigating the sensor network hot-spot problem, considering load balancing as well as quality of data.

Section III:

Significance Of the Study

A major research problem, critical to the real world operation of sensor networks, is to design networks that are efficiently and dynamically adaptable to the energy and QoS requirements. The main goal of the proposed research will be to develop energy-efficient and adaptive algorithms for constructing routing trees.

The Results can be implied to technological advances in the fields of:

- Healthcare
- Smart homes
- Iot and secutiry
- Better environment sensoring.

Section IV:

Scope Of The Study

The focus is on efficient processing of a mix of aggregate queries and not on constructing the routing trees themselves in an energy-efficient fashion. In contrast, the objective of our research is to construct such trees considering the evolution of the network as a sequence of generated queries with different semantics and adapting the routing structures both in the sense of constructing a new one and modifying the existing ones.

Section V:

K-Multipath Routing Algorithm

Route Establishment

- Assumption – Each node knows its location and the location of its neighbours.
- Heartbeat algorithm – finds the neighbours for us [7].
- The algorithm for constructing the routing structure - For a given source-sink pair of nodes, the sink will unicast on a shortest path routing (along a straight imaginary line) the query request to the source node.
- K-paths denotes K intermediate destination points.
- Distance between two consecutive lines are equal.
- K-multipath to reduce traffic over network.
- Node disjoint, link disjoint and non-disjoint routes.
- Though node disjoint – transmissions along the routes may interfere if some nodes among the routes are in the same collision domain.
- It is important to establish paths that are as independent as possible to ensure the least interference between the paths.
- Bezier Curves

A Bezier curve of a given set of n+1 points: Pi (i = 0, 1, 2……n), and a parameter tn ∈ [0, 1], is defined as:

$$p(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)$$

Where $B_{i,n}(t)$ Bernstein polynomials, defined as:

$$B_{i,n}(t) = \frac{n!}{i!\,(n-i)!}\, t^i (1-t)^{n-i}$$

Sum to one for any t in [0, 1],

$$\sum_{i=0..n} B_i^n(t) = 1$$

For 3 control points, n = 2,

$$p(t) = (1-t)^2 p_0 + 2t(1-t)p_1 + t^2 p_2$$

For 4 control points, n = 3,

$$p(t) = (1-t)^3 p_0 + 3t(1-t)^2 p_1 + 3t^2(1-t)p_2 + t^3 p_3$$

$$p(t) = (-t^3 + 3t^2 - 3t + 1)\mathbf{p}_0 + (3t^3 - 6t^2 + 3t)\mathbf{p}_1 + (-3t^3 + 3t^2)\mathbf{p}_2 + (t^3)\mathbf{p}_3$$

$$p(t) = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)t^3 + (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)t^2 + (-3\mathbf{p}_0 + 3\mathbf{p}_1)t + (\mathbf{p}_0)1$$

- We can rewrite the equation in matrix form – compact notation – use existing 4x4 Matrix hardware support.

$$p(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\mathbf{a} = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)$$
$$\mathbf{b} = (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)$$
$$\mathbf{c} = (-3\mathbf{p}_0 + 3\mathbf{p}_1)$$
$$\mathbf{d} = (\mathbf{p}_0)$$

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

- The Bezier curve for 3 points P0, P1 and P2 as shown in Fig 5.1.

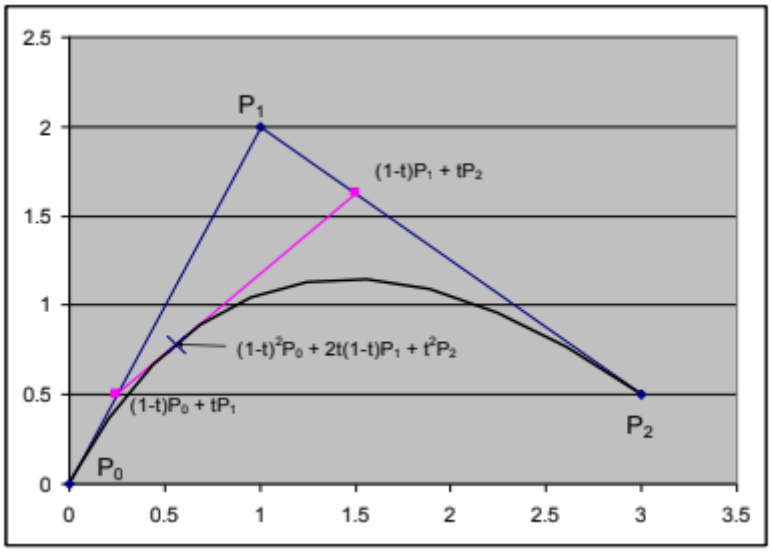


Fig 5.1 Construct a Bezier curve from given three points

- The curve passes through the first, P0 and last vertex points, Pn. The tangent vector at the starting point P0 must be given by P1 – P0 and the tangent Pn given by Pn – Pn-1.

- The properties of the Bernstein polynomials ensure that all Bezier curves lie in the convex hull of their control points as shown in Fig 5.2.

Hence, even though we do not interpolate all the data, we cannot be too far away.

- Using the Bezier curve as the trajectory model, we are able to control the coverage of the sink/source nearby nodes implicated in the routing and attain, in a practical setting, a near 100% utilization of them.

- A node, however, needs only to communicate the coordinates of the control points in order to be able to generate an entire Bezier curve – saves both time and energy – advantage of using parametric curves.

- Affine property of Bezier curve – generates entire family of curves as shown in Fig 5.3.

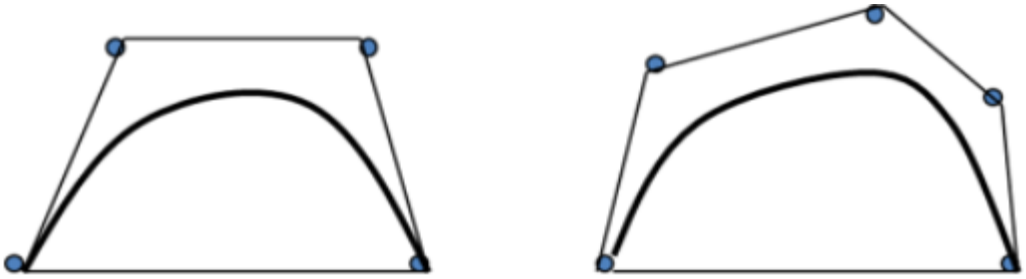


Fig 5.2 Curve resides completely inside its convex hull

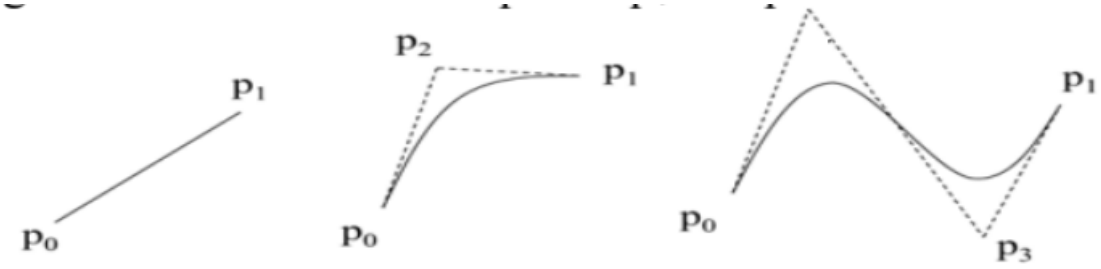


Fig 5.3 Linear, Quadratic and Cubic.

The Analysis of finding multiple node-disjoint paths

- Node-disjoint paths can provide the highest degree fault-tolerance.
- Many algorithms to find node-disjoint paths make use of request/reply cycles.
- A source node initiates a route discovery procedure by broadcasting a Route Request packet and then this ROUTE REQUEST message is flooded to the entire network.
- The k-multipath algorithm to build node-disjoint routes does not generate too much RREQ/RREP packets and then increases the routing overheads.
- Instead, makes use of geometrical knowledge to discover routes.
- It takes more time to deliver packets along the path farther away from the source-sink line segment.
- Multiple paths may present differences in the end-to-end delay of each path.
- Such scenarios require that data coming from different flows needs to be buffered till the flow from the path with the highest delay arrives for reordering the data correctly.
- This solution poses another problem, as high speed memory is extremely expensive, and therefore we should minimize the differential delay. In our simulation, we don't need to consider packet reordering.

Section VI:

Performance Evaluation

The performance of the shortest path routing and multipath routing is compared in different aspects:

- The load distribution: This metric provides the average relayed traffic in packets as function of the distance to the network center, in accordance with the Pham and Perreau's analytical model [3]. We use load distribution as a metric to evaluate theload balancing.

- Average energy consumption: The energy consumption is averaged over all nodes in the network.

- The lifetime extension: The metric studied is the number of hours of communication achieves when 1 percent, 25 percent, 50 percent, and 100 percent of the nodes die using multipath routing and the shortest path routing.

- Query Turn-Around Time: A measure of the initial responsiveness of the query; the time lapsed from the moment the query is submitted to the network by the user until the very first data-packet is received at the user. This will measure and penalize the multipath-construction algorithm with a high set-up time.

These measures are intended to provide insight into the ability of the protocols to route packets to their intended destination, and the energy efficiency of the protocols in accomplishing that task.

The routing overhead is defined as the ratio of the number of routing messages generated by a routing protocol to the number of received data packets at the destinations. This metric is a measure of how many routing messages are needed to receive one data packet.

It captures the efficiency of the routing protocol.

Since the routing overhead is similar and much lower between multipath and the shortest path routing in our simulation, the results of routing overhead are not presented in the paper.

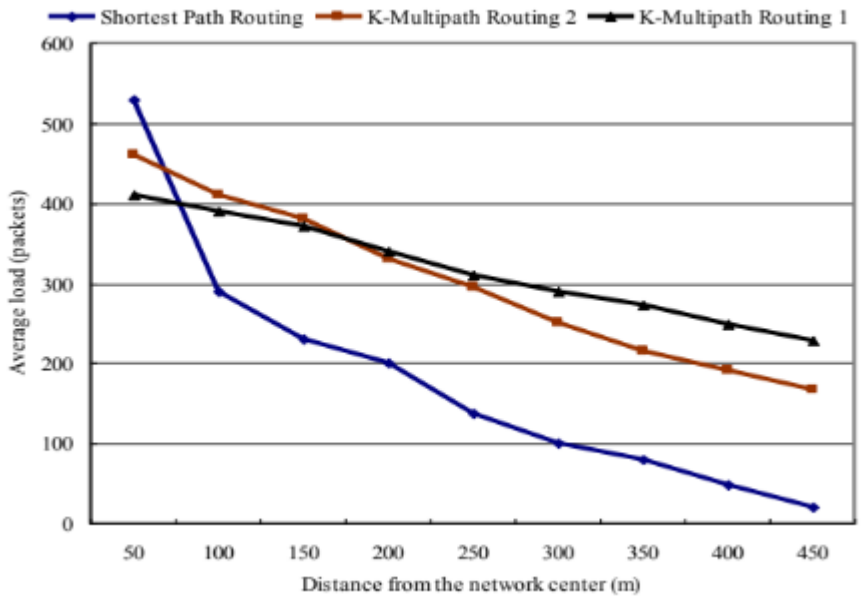


Fig 6.1 The average load distribution as function of distance from the network centre

Conclusion from the as shown in Fig 6.1 is that the shortest path is likely to be overloaded because this route is across or very close to the centre. In addition, though adopting load balancing policy, theoretically, all the nodes should experience approximately the same loads in the multipath routing 1, however, there exists a smooth decrease of the loads as the distance increases. The possible reason is that those packets that travel through longer routes are dropped due to more latency. Still, as the distance from the network center increases, the number of average load for multipath routing 2 drops faster than multipath routing 1. This can be explained that multipath routing 2 is to choose a path with a probability inversely proportional to the length of the path. In other words, the further the distance to the center, the lower the probability that the nodes are used to relay the packets. Moreover, since the load balancing policy is not optimal, those nodes close to the optimal route have to be assigned more traffic in comparison with ones at the rear. Nevertheless, it is important to stress the fact that this multipath routing outperforms the shortest path routing in terms of load balancing.

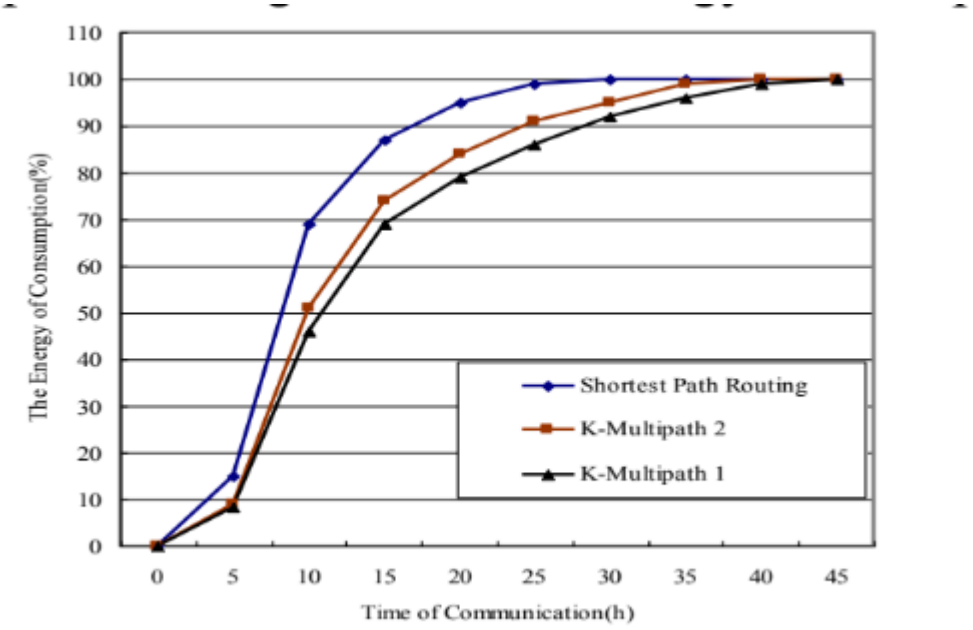


Fig 6.2 The average energy consumption as function of the number of hours of communication

Clearly, both two multipath routing have smaller energy consumption than that of the shortest path routing as shown in Fig 6.2. This result can be explained by the fact that the traffic of the network is evenly regulated to the different paths while the single path always chooses the geographic-based shortest path, which will unfairly distribute more loads to the nodes along this optimal route than their neighboring nodes [6]. Moreover,

we also notice that there exists a slightly improvement of energy consumption between multipath routing 1 and multipath routing 2, even compared to the shortest path routing. This is because the nodes, even with no routing tasks, have to passively listen to neighboring nodes' radio transmission, which inevitably consumes battery energy. Even though the used multipath routing is not optimal, it is important to stress the fact that it still outperforms the shortest path routing in terms of energy consumption.

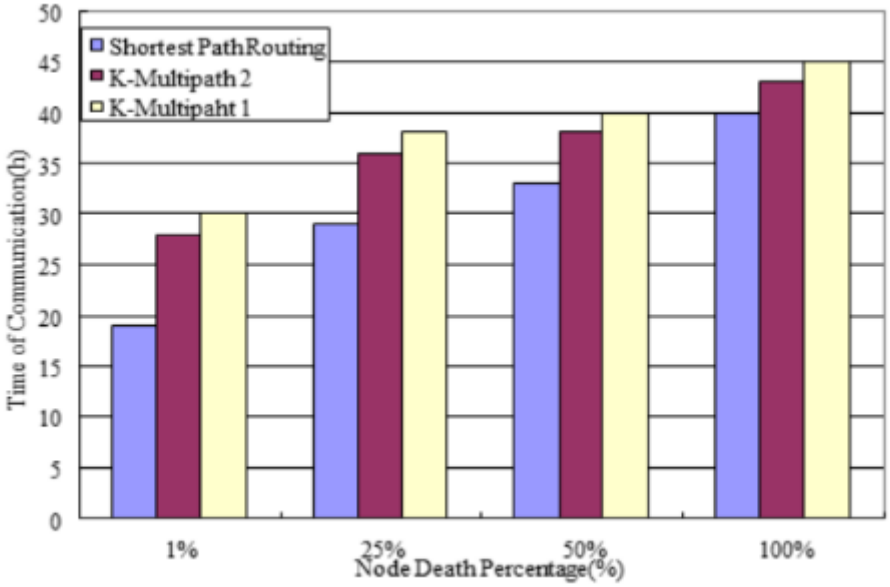


Fig 6.3 The lifetime of the network as function of the percentage of node death

Both of the two multipath routing can yield improvements over the shortest path routing in all cases while the lifetime extension of the multipath routing 1 is trivial compared to the multipath routing 2 as shown in Fig 6.3. According to the research, the battery energy of a network node is mainly consumed on forwarding control and data packets. Multipath routing usually increases the energy consumption on the transmission of data messages because some data packets traverse sub-optimal paths [4]. On the other hand, it will decrease the energy consumption on the transmission of control messages. This reveals that k-multipath routing 1 can achieve the best balanced energy dissipation among the sensor nodes to have full use of the complete sensor network.

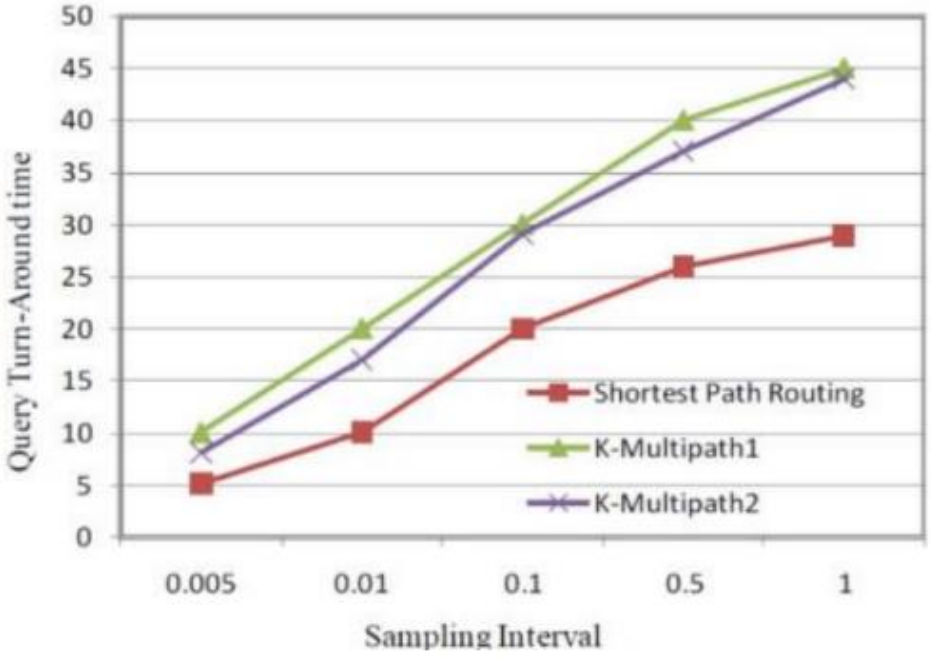


Fig 6.4 Query turn-around time as function of sampling interval

Section VII:

Limitations

- Large Complexity and overhead when compared to Single shortest path protocol.
- Large routing tables at intermediate nodes are a problem.
- Allocation of packets to their multiple routes.
- Latency in the arrival of segmented packets to the destination.

Section VIII:

Conclusions

A novel load-balancing mechanism for wireless sensor networks is presented in the paper. The new scheme is simple but very effective to achieve load balancing and congestion alleviation. The paper has explored an exiperimental comparison btween k-multipath routing and the shortest path routing. The performance study shows that the network traffic can be distributed more evenly onto multipath routing. Load balancing is important to fairly distribute the routing task among the nodes of the network. It can also

protect a node from failure considering that a node with heavy duty is likely to deplete its energy quickly. Still it takes longer time for packet delivery than the shortest path method. The use of the k-multipath routing to balance the energy dissipation to maximize the lifetime of the nodes in a sensor network. However, minimizing energy in isolation has drawbacks because energy efficiency often brings additional latency and practical applications set limits on acceptable latency as specified by QoS requirements. The paper explores an experimental comparison between k-multipath routing and the shortest path routing. So through distributing the energy load among the nodes, the lifetime and quality of data in a sensor network can be increased [5].

Section IX:

References

[1] X.Chen, M.Chamania, A. Jukan, A. C. Drummond, N. L. S. da, Fonseca, "QoS-Constrained Multi-path Routing for High-End Network Applications", IEEE INFOCOM 2009 High-Speed Networks Workshop, Vol. 12, No. 2, pp. 435-489, April 2009.

[2] Y. Ganjali, A.Keshavarzian, "Load balancing in ad hoc networks: single-path routing vs. multipath routing", Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004), pp. 155-176, March 2004.

[3] P.P. Pham and S. Perrau, "Performance Analysis of Reactive Shortest Path and Multipath Routing Mechanism with Load Balance", IEEE INFOCOM Conference, pp. 251-259, April 2003.

[4] Sungoh Kwon and Ness B. Shroff, "Analysis of Shortest Path Routing for Large Multi-Hop Wireless Networks", IEEE/ACM Transactions on Networking, Vol. 17, No. 3, pp. 857-869, June 2009.

[5] Shrivastava N., Buragohain C., Agrawaid., "Medians and beyond: new aggregation techniques for sensor networks", Second International Conference on Embedded Networked Sensor Systems (SenSys'04), pp. 239-249, 2004.

[6] Prasenjit Chanak, Tuhina Samanta, Indrajit Banerjee, "Fault-tolerant multipath routing scheme for energy efficient wireless sensor networks", International Journal of Wireless & Mobile Networks (IJWMN), Vol. 5, No. 2, April 2013.

[7] Sudarshan Vasudevan, Micah Adler, Dennis Goeckel, Don Towsley, “Efficient Algorithms for Neighbour Discovery in Wireless Networks”, IEEE/ACM Transaction Networking, pp. 69-83, 2013.

[8] I.F. Akyildiz, T. Melodia, and K.R. Chowdhury, “A Survey on wireless multimedia sensor networks”, Computer Networks (Elsevier) J., 2007.

Code Description

- To implement the data structure for nodes, the code uses a vector list of objects of class named node. Vectors are used because they are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.
- The code starts at the main function and asks the user to give the number of nodes. When the user enters the no. of nodes they are stored as integer and we use a for loop to go through the process of taking the entered number of nodes from the user again.
- The next part is to assign the source and destination nodes. This is done by asking the user about which two nodes he/she wants to assign as the source and the destination. To make things simpler we have incorporated a SWAP macro. What SWAP does is that it takes two nodes and changes their respective positions. To make the source node as the first node and the destination as the last node is a work done using the SWAP macro. This step helps us in identifying the source and destination nodes in all parts of the code.
- We then print the nodes, the code has taken from the user.
- To insert nodes we use insertNode() method and we take the id of the node, x co-ordinate and the y co-ordinate of the node. Then we use the push_back call to store them in the vector.
- To delete a node deleteNode() method is called and if the node id entered is of the source or the destination it cannot be deleted message is popped or otherwise the node is deleted.
- The function called next is the findPaths(). This assigns the source node to ‘s’ and destination node to ‘d’ and then asks the user to enter a control point. The source node, destination node and the control point are then sent to Bezier function, which returns with the number of nodes which lie on the Bezier curve.
- To solve for the points on the Bezier curve, the Bezier function uses the equation:

  ```
  double x = (1-i)*(1-i)*s.v1+2*c.v1*i*(1-i)+pow(i,2)*d.v1;
  double y = (1-i)*(1-i)*s.v2+2*c.v2*i*(1-i)+pow(i,2)*d.v2;
  ```

  If the node lies on the same point it is chosen otherwise rejected.
- The code then uses a radius variable of integer type which is given as the range of each node. This variable tells us the maximum range of that particular node and that is it's transmission range. We then use this to calculate which next node on the Bezier set of nodes lie within this distance and the node is added to the path.
- A set of nodes so obtained gives us one of the k-multipaths.
- The code uses a for-loop to inculcate the multipaths and display them.

Complete Code

```
#include <iostream>
#include <algorithm>
#include <list>
#include <vector>
#include <cmath>
using namespace std;
int visited[100] = {0};
class node{
    public:
        int v1;
        int v2;
        int id;
        node(int id, int v1, int v2)
        {
            this->id = id;
            this->v1 = v1;
            this->v2 = v2;
        }
        node(){};
};
vector<node> nodes;
list<int>::iterator it;
list<int> paths[100];
#define SWAP(a, b, t) t = a, a = b, b = t
#define dist(a, b, x, y, d) d = sqrt((a-x) * (a-x) + (b-y) * (b-y))
void insertNode();
void findPaths();
int bezier(node, node, node, int[], int);
void deleteNode(int, int);
int menu();
int k = 0;
bool compare(node n1, node n2)
{
    if(n1.v1 < n2.v1)
        return true;
    else if(n1.v1 == n2.v1)
        if(n1.v2 < n2.v2)
            return true;
    return false;
}
int main()
{
    int n;
    int s, d;
    cout << "Enter the number of nodes:";
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cout << "Enter the id, and (x,y):";
        int a, b, sId;
```

```
        cin >> sId >> a >> b;
        node n1 = node(sId, a, b);
        nodes.push_back(n1);
    }
    cout << "Enter the source id and destination id: ";
    cin >> s >> d;
    node n1;
    int cS, cD;
    for(int i = 0; i < nodes.size(); i++)
        if(s == nodes[i].id)
        {
            cS = i;
            break;
        }
    for(int i = 0; i < nodes.size(); i++)
        if(d == nodes[i].id)
        {
            cD = i;
            break;
        }
    SWAP(nodes[cS], nodes[0], n1);
    SWAP(nodes[cD], nodes[n-1], n1);
    for(int i = 0; i < n; i++)
        cout << nodes[i].id << ": " << nodes[i].v1 << " " << nodes[i].v2 <<
endl;

    int m = 1;
    while(m)
    {
        m = menu();
        if(m == 1) insertNode();
        else if(m == 2) deleteNode(s, d);
        else if(m == 3) findPaths();
    }
}
void insertNode()
{
    int id, v1, v2;
    cout << "Enter the id, (x, y):";
    cin >> id >> v1 >> v2;
    node n(id, v1, v2), n1;
    nodes.push_back(n);
    SWAP(nodes[nodes.size()-1], nodes[nodes.size()-2], n1);
}
void deleteNode(int s, int d)
{
    int id;
    cout << "Enter the id of node to delete:";
    cin >> id;
    if(id == s || id == d)
    {
        cout << "The node cannot be deleted\n";
        return;
    }
    vector<node>::iterator itr;
    for(itr = nodes.begin(); itr != nodes.end(); itr++)
        if((*itr).id == id)
            break;
    if(itr == nodes.end())
        cout << "Invalid node Id\n";
    else
```

```
    {
        cout << "Node deleted --> " << (*itr).id << ": " << (*itr).v1 << " "
<< (*itr).v2 << endl;
        nodes.erase(itr);
    }
}
void findPaths()
{
    for(int i = 0; i < 100; i++) paths[i].clear();
    node s, d;
    s = nodes[0];
    d = nodes[nodes.size()-1];
    int id, v1, v2;
    cout << "enter a control point(id,x,y):";
    cin >>id >>v1 >>v2;
    node c(id,v1,v2);
    int path[100];
    k = 0;
    cout<<"possible paths are:"<<endl;
    bezier(s, d, c, path, 0);
    cout<<endl;
}

int bezier(node s, node d, node c, int path[], int ind)
{
    path[ind] = s.id;
    vector<node> nodes2;
    int id=1;
    for(double i=0.0;i<=1.0;i+=0.1)
    {
       double x = (1-i)*(1-i)*s.v1+2*c.v1*i*(1-i)+pow(i,2)*d.v1;
       double y = (1-i)*(1-i)*s.v2+2*c.v2*i*(1-i)+pow(i,2)*d.v2;
       node n(id,x,y);
       id++;
       nodes2.push_back(n);
       cout<< x <<" "<< y << "           "<<endl;
    }
}
int menu()
{
    int n;
    cout << "1.Insert a node:\n";
    cout << "2.Delete a node:\n";
    cout << "3.Find multipath:\n";
    cin >> n;
    return n;
}
```

Snapshots

```
shivdutt@shivdutt-HP-Notebook:~/Documents/back-up-5th-sem/CN/code_for_cn$ ./a.out
Enter the number of nodes:4
Enter the id, and (x,y):1
0
0
Enter the id, and (x,y):2
2
0
Enter the id, and (x,y):3
4
0
Enter the id, and (x,y):4
6
0
Enter the source id and destination id: 4
1
1: 0 0
2: 2 0
3: 4 0
4: 6 0
1.Insert a node:
2.Delete a node:
3.Find multipath:
2
Enter the id of node to delete:4
The node cannot be deleted
1.Insert a node:
2.Delete a node:
3.Find multipath:
2
Enter the id of node to delete:1
The node cannot be deleted
1.Insert a node:
2.Delete a node:
3.Find multipath:
2
Enter the id of node to delete:2
```

```
1.Insert a node:
2.Delete a node:
3.Find multipath:
3
enter a control point(id,x,y):5
3
0
possible paths are:
0 0
0.6 0
1.2 0
1.8 0
2.4 0
3 0
3.6 0
4.2 0
4.8 0
5.4 0
6 0
```