REND: RELIABLE AND ENERGY-EFFICIENT NODE-DISJOINT PATHS IN WIRELESS SENSOR NETWORKS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Siva Vanteru

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

May 2010

Fargo, North Dakota

North Dakota State University
Graduate School

Title

REND: RELIABLE AND ENERGY-EFFICIENT NODE-DISJOINT PATHS IN WIRELESS SENSOR NETWORKS

By

SIVA VANTERU

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Vanteru, Siva, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, May 2010. REND: Reliable and Energy-Efficient Node-Disjoint Paths in Wireless Sensor Networks. Major Professor: Dr. Weiyi Zhang.

In wireless sensor networks, finding most reliable and most efficient paths between source node and destination node is considered NP hard problem. We can increase reliability by ability to find node-disjoint alternate path when there is any disruption in the path from source to destination. If we can find reliable paths by keeping efficiency in consideration then we can increase both reliability and efficiency of paths between source and destination.

In this paper, we present routing protocol for finding two node-disjoint paths between each pair of nodes in a wireless sensor networks. In proposed protocol, each and every node has the same procedure. In this paper, we compare proposed protocol to traditional shortest path algorithm to find shortest path (diverse shortest path).

A diverse shortest path has been proven most efficient, as it can compute node-disjoint paths connecting a source node to a destination node with minimum total energy. However, computing node-disjoint path connecting source node to destination node in this approach may not be reliable, as it may not find second shortest path all the time. We proposed node-disjoint path algorithm to find reliability of node-disjoint paths in wireless sensor networks. By implementation of node-disjoint path algorithm on wireless sensor networks, reliability has definitely increased. By implementing efficient node-disjoint path algorithm on wireless sensor networks, we improved energy consumption significantly.

ACKNOWLEDGEMENTS

I am heartily thankful to my advisor, Dr. Weiyi Zhang, whose encouragement, guidance and support from the initial level enabled me to develop an understanding of the subject and helped me to complete my master's paper.

I would also like to thank Dr. Jun Kong, Dr. Tariq King and Dr. Jin Li for being a part of my Masters Paper committee.

DEDICATION

I wish to dedicate my work to my advisor Dr. Weiyi Zhang for his incredible support and guidance, and to my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Background

Wireless sensor networks (WSN) are seeing great deal of research attention and increasing usage in sensing applications. WSN has implementations in wide areas from wildlife monitoring to military [1]. These types of networks consist of individual nodes that sense data and transmit data by communicating with each other.

Wireless sensors networks are used in collecting information from remote locations in a cost effective and reliable manner. Examples include Harvard's volcano eruption monitoring project [2] and the SWE project at the University of Vermont to measure the snow-water equivalent in a snow pack [3]. Due to the limited availability of the processing speed and energy, sensor networks need to be very efficient. The goals of wireless sensors are to sense data and transmit it by consuming very less energy [4].

The wireless sensor node in WSNs consists of hardware and software. The hardware basis of WSNs is driven by advances in several technologies. First, System-on-Chip (SoC) technology in which complete system is integrated on single chip. Sensor nodes such as UC Berkeley's motes, UCLA's Medusa and WINS are based on commercial SoC based embedded processors. Second, commercial RF circuits enable short distance wireless communication with extremely low power consumption [5].

Software part of wireless sensor comprised of very small operating system called TinyOS, which is an event-driven operating system specialized for memory constrained embedded systems, specifically wireless sensors. The language that is most commonly used to develop wireless sensor network applications on the TinyOS platform is nesC, which has the low-level control benefits of C, while optimizing the resource limitations of a wireless device. The flexible, component-based design of nesC uses tasks and events to support the particular requirements of this domain. NesC allows several software components to be statically wired together to form node-level applications [6].

A wireless sensor node also contains a radio transceiver. Radio transceiver contains all the necessary circuitry to send and receive data over the wireless media. Similar to microcontrollers, transceivers can operate in different modes. Commercially available transceivers have different characteristics and capabilities.

The sensor node has primarily two functions. First, one is to sense data and second one is to transmit data. Sensing data consumes a lot less energy compared to transmitting data. Sensor nodes has very limited energy source. Due to this, some sensor nodes die over the period. There by decreasing reliability of the complete network.

In order to increase reliability we propose to find energy efficient node-disjoint paths so that if one node fails in a path from source node to destination node then also there will be other path that is free from failed node.

1.2. Notation Used and Network Model

This paper uses following notation:

1. **G(V,E):** Bi-directed bi-connected graph. **V** is the list of n nodes. **E** is the list of m edges and graph **G** is the representation of the real world wireless sensor network. In this paper, words network and graph are used interchangeably.
2. **G'(V',E'):** Directed bi-connected auxiliary graph or generated graph. **V'** is the list of 2n nodes. **E'** is the list of (m+n) edges and graph **G'** is the auxiliary graph of **G**.

3. **S:** Source node in graph **G(V,E)**. **S** $\in$ **V**.
4. **D:** Destination node in graph **G**, **D** $\in$ **V**.
5. **Dense graph:** Graph with high density of nodes.
6. **Sparse graph:** Graph with very low density of nodes.
7. $\mathbf{d_{ij}}$: Distance between node i and node j.
8. **EC(e):** Energy consumption of edge of the graph **G**.
9. **R:** Transmission range of each node.
10. **EC(p):** Energy consumption of edge of the graph **G**.
11. **EC(p):** Energy consumption of path p.
12. $\mathbf{e_{ij}}$: Energy consumption between node i and node j.
13. **R:** Transmission range of each node.
14. **T:** Transmission range of a sensor node.
15. **Field:** Field is the area the wireless sensor nodes are randomly placed, in this paper field size is represented as the area of field where all the nodes of the graph **G** are located and are connected with other nodes within transmission range.
16. **Transmission Range:** (T) Transmission range is the range of the wireless sensor node it can communicate with other wireless sensor nodes in the network. Transmission range is same for all the wireless sensor nodes in wireless sensor network.

The field size of the network is different for different graphs, for a given field size sensor nodes are randomly scattered in the field. For a given graph, transmission ranges of the nodes are constant. Wireless sensor nodes are connected with all the other sensor nodes that are situated within the transmission range of the node. The length of a edge **e** in the graph **G(V,E)** where **e** $\in$ **E** is **d** which is always less than or equals to **T**. i.e., $\mathbf{d \leq T}$.

17. **Bi-directed Graph:** A bi-directed graph **G** is a directed graph with list of nodes V (G) and list of edges E (G) each edge oriented as $\bullet\!\!\leftarrow\!\!\!-\!\!\!-\!\!\!\rightarrow\!\!\bullet$ and this orientation is called out-edge orientation.

18. **Bi-connected Graph:** A graph **G** is called bi-connected if any one node **v∈V** is removed from a graph **G** and graph **G** is still connected or if any two nodes in a graph have two or more node-disjoint paths between them. In a bi-connected graph **G**, between source node S and destination node D there will be two or more node-disjoint paths.

1.3. Problem Statement

Develop a routing algorithm for wireless sensor networks (WSN) to increase the reliability, efficiency and thereby increasing the performance of the network.

INPUT: A bi-directed bi-connected graph G(V,E), source node S, Destination node D.

OUTPUT: For any given source node S and destination node D in graph G(V,E), the problem seeks a pair of node-disjoint paths with optimal energy consumption between given source node S and destination node D.

To increase reliability we propose to find node-disjoint paths. With node-disjoint paths, we can be sure that there will be connectivity and possibility of finding node disjoint paths even after failure of one node on network.

First, we use dijkstra's shortest path algorithm to find diverse shortest paths in graph G(V,E). Second, we propose scheme which uses maximum flow algorithm to find node-disjoint paths. Third, scheme, we improve energy efficiency by finding energy efficient node-disjoint paths.

1.4. Energy Model

The energy model we used to calculate energy consumption for sensor is based on first order radio model presented in [8; 9]. Energy consumption of a sensor node to run the transmitter or receiver circuitry is ε_{elec} = 50 nJ/bit, energy consumption of transmitter amplifier is consumes ε_{amp} = 100 nJ/bit/m^2. The sensor node consumes Rx = ε_{elec} energy for receiving 1-bit data packet.

The energy consumed by a sensor i to transmit data packet to sensor node j is given by Tx = ($\varepsilon_{elec} + \varepsilon_{amp}\, d_{ij}^2$), where d_{ij} is the distance between nodes i and j. The total energy consumption for transmitting packet from sensor node i to sensor node j, is the sum of transmission energy consumption of i and receiving energy consumption of j. Therefore, total energy consumption for transmitting 1-bit data packet is given by

$$Rx + Tx = (2\,\varepsilon_{elec} + \varepsilon_{amp}\, d_{ij}^2)$$

$$Rx + Tx = (2 \cdot 50 + 100\, d_{ij}^2) \qquad (\text{Since } \varepsilon_{elec} = 50 \text{ and } \varepsilon_{amp} = 100)$$

$$Rx + Tx = (100 + 100\, d_{ij}^2)$$

$$Rx + Tx = 100\,(1 + d_{ij}^2)$$

The distance between two nodes in wireless sensor networks is usually very large and square of the distance is much larger. Addition on constant 1 to a very large number is negligible. Therefore, we can remove constant 1 from above equation.

$$Rx + Tx = 100\,(d_{ij}^2)$$

From above equation, we can conclude that total energy consumed to transmit a packet of data is proportional to square of distance between sensor nodes.

$$Rx + Tx \propto (d_{ij}^2)$$

2. DIVERSE SHORTEST PATHS SCHEME

2.1. Description

In this chapter, we will find two diverse shortest paths between source node S and destination node D in graph **G (V,E)** by using **dijkstra's algorithm**. Dijkstra's algorithm solves the single source shortest path problem in bi-directed, bi-connected and weighted graph **G = (V, E)**. Diverse shortest paths between node u and node v are the shortest paths that are node-disjoint i.e., there is no node that is common among two paths $\mathbf{sp_1}$ and $\mathbf{sp_2}$.

After finding the diverse shortest paths between node S and node D, we calculate the energy consumption of the paths by adding all the edge weight of the edge e that belongs to diverse shortest path. We calculate the energy consumption of each link and take the average of two diverse shortest paths.

2.2. Shortest Path Problem

In a **shortest-paths problem**, the input is weighted, bi-directed bi-connected graph G = (V, E), with weight function **w**: E → **R** mapping edges to real-valued-weights i.e., the distance of the edge or length of the edge. The length of path p = v0, v1, ...,vk is the sum of the weights of its constituent edges. [7]

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_k) \quad [7]$$

The shortest-path weight of path from node *u* to node *v* is defined as

$$\delta(u,v) = \begin{cases} \min \{w(p) : u \overset{p}{\rightarrow} v\} & \text{if there is path from } u \text{ to } v \\ \infty, & \text{otherwise.} \end{cases}$$

A shortest path can be defined as any path p from node u to node v where weight of the path w(p) = δ(u.v).

2.3. Dijkstra's Algorithm

The shortest path algorithm used in this paper is **dijkstra's algorithm** [7] as it solves the single pair shortest path problem on bi-directed bi-connected weighted graph **G = (V,E)**.

In Dijkstra's algorithm, final shortest-path weights from the source s have already been determined as they are stored in set S. Vertex u ∈ V – S with the minimum shortest-path estimate are repeatedly selected by algorithm and then adds u to S, after adding u to S all the edges leaving u are relaxed. In the following implementation, we use a min-priority queue Q of vertices, keyed by their d values.

Algorithm 2. Dijktra's algorithm [7]

```
1.  DIJKSTRA(G,w,s)
2.        INITIALIZE-SINGLE-SOURCE(G,s,D)
3.        assign S to ∅
4.        assign Q to V[G]
5.        while Q not equal to ∅
6.        do u ← EXTRACT-MIN(Q)
7.            S ← S union {u}
8.                if u equals D
9.                break
10.           for each vertex v ∈ Adj[u]
11.               do RELAX(u,v,w)
```

In line 1 d and π values are initialized and at line 2 set S is initialized to the empty set. At the start of the each iteration of the while loop of at lines 4-8 algorithm maintains the invariant that Q = V – S. At line 3 min-priority queue is set to contain all the vertices in V, since S = Ø at that time, the invariant is true after line 3. Vertex u is extracted from Q = V – S and added to set S thereby maintaining the invariant. Then, lines 7-8 relax each edge (u, v) leaving u, thus updating the estimate d[v] and the predecessor π[v] if the shortest path to v can be improved by going through u. Vertices are never inserted into Q after line 3 and that each vertex is extracted from Q and added to S exactly once, while loop of lines 4-8 iterates exactly |V| times.

2.4. Finding Diverse Shortest Paths

We use dijkstra's algorithm to find diverse shortest paths between source node S and destination node D in graph G(V,E). First, we will find first diverse shortest path sp_1 between source node S and destination node D. This will give us the first path that is present between source node S and destination node D. After getting the first diverse shortest path sp_1, we remove all the nodes that are present in the first diverse shortest path sp_1 and try finding the second diverse shortest path sp_2 between source node S and destination node D if there exists one.

In algorithm 3, we propose pseudo code to find diverse shortest paths between source node S and destination node D. First, we read graph G, source node S and destination node D. In second step, using algorithm 1 we find first shortest path p_1. In third step, we loop through all the nodes present in first shortest path p_1 and we will remove all corresponding nodes. By this way, we know that when we find next shortest path we will know that second shortest path p_2 doesn't consists of any nodes that were included in first shortest path p_1. In step six, using algorithm 1 we find second shortest path p_2. If there is, a second shortest path then algorithm will return two paths p_1 and p_2. Figure 1, figure 2 and figure 3 explain in detail how to find diverse shortest paths sp_1 and sp_2.

Algorithm 3. Diverse shortest path pseudo code

```
1.    read graph G, source S and destination D
2.    using Algorithm 1 find shortest path $p_1$
3.    **for** each node $n \in p_1$
4.        remove n from graph G
5.    **end for**
6.    using algorithm 1 find second shortest path $p_2$
7.    **if** $\exists p_2$
8.        return $p_1$ and $p_2$
9.    **else**
10.       **drop the request.**
```

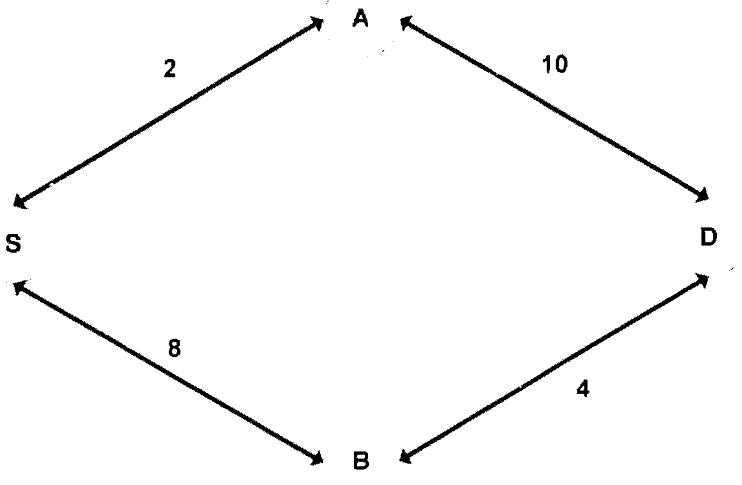

Figure 1. The graph G(V.E) with source node S and Destination node D

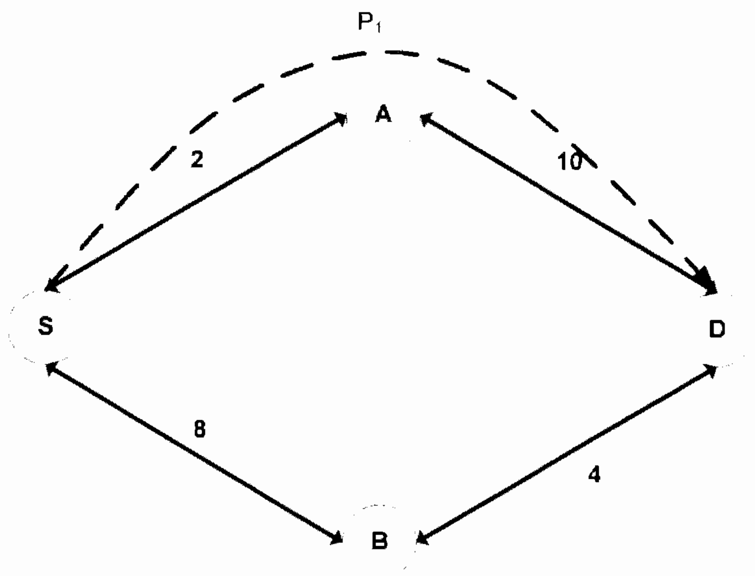

Figure 2. First shortest path sp_1 from source node S to destination node D

We find the first shortest path sp_1 from source node S to destination node D.

$$sp_1 = S \rightarrow A \rightarrow D$$

$$w[p_1] = 2 + 10 = 12$$

Energy consumption of the path is sum of the square of the edge weight of the edges in the path sp_1.

$$Ec(p_1) = (2)^2 + (10)^2 = 4 + 100 = 104$$

$$sp_1 = S \rightarrow B \rightarrow D$$

$$w[p_2] = 8 + 4 = 12$$

Now as we got the first diverse shortest path, we will remove all the nodes that are part of sp_1 except source node and destination node.

Finding diverse shortest paths is not as simple as it seems because using shortest path algorithm to find diverse shortest path may not result in success.

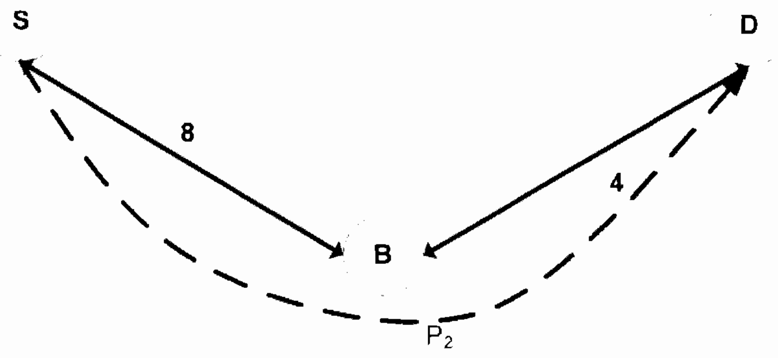

Figure 3. Second shortest path sp_1 from source node S to destination node D

Energy consumption of the path is the summation of the square of the edge weight of the edges in the path sp_1.

$$Ec(p_2) = (8)^2 + (4)^2 = 64 + 16 = 80$$

After finding first diverse shortest path, there is no assurance that there will be another second diverse shortest path between source node S and destination node D.

We will explain in the following figure 4, figure 5 and figure 6 why it is not always possible to find second diverse shortest path between source node S and destination node D.

In this example, we try to find diverse shortest paths sp_1 and sp_2 between source node S and destination node D. The figure 4 shows the graph we choose for this example. The figure 5 shows possible shortest path between source node S and destination node D in graph G2. The figure 6 shows graph after removing nodes that are associated with first shortest path **sp_1**.

The shortest path between source node S and destination node D is

$$sp_1 = S \rightarrow A \rightarrow B \rightarrow D.\ w[sp_1] = 2 + 3 + 4 = 9.$$

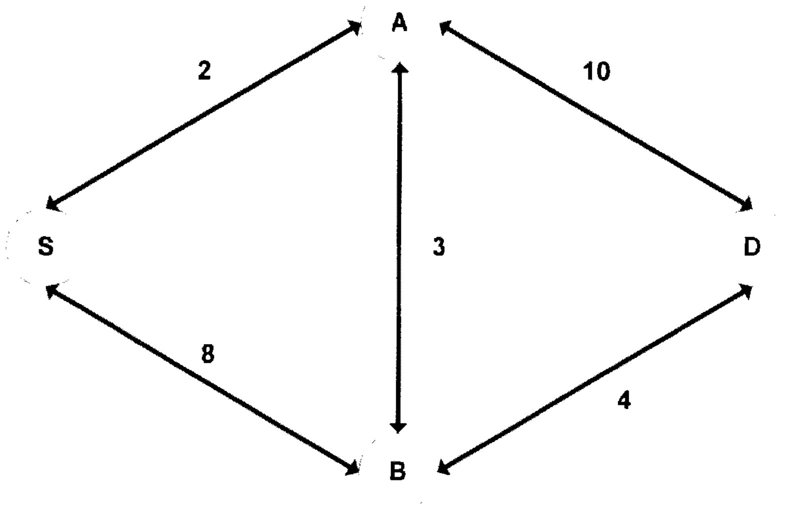

Figure 4. The new graph G2(V,E) with source node S and Destination node D.

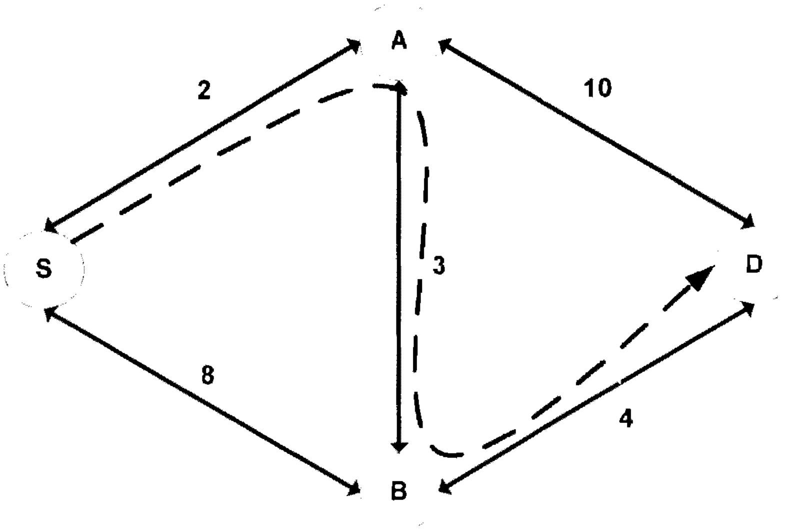

Figure 5. The shortest path between S and D represented by dotted (- ->) line.

We will now remove all the nodes that are associated with first diverse shortest path sp_1 and try to find second diverse shortest path sp_2. Here we remove node A and node B, there by all the corresponding edges will be removed. The figure 5 shows in detailed graph after removing nodes that are present in first shortest path sp_1.

In figure 6, we can see that there is no connection between source node S and destination node D as the node A and node B are removed the connecting edges are also removed.

Figure 6. Remaining graph G2(V,E) after removing node A and node B

We can say that using shortest path algorithm to find node-disjoint or diverse paths is not always a good thing as we may not find diverse shortest paths every time.

Shortest path algorithm may give us paths with less energy consumption but it may not be reliable even though there is a diverse or node-disjoint path. The reliability of diverse shortest path scheme is not 100%.

In order to improve the reliability, we propose another scheme to find reliable routing. We call it as reliable routing using maximum flow. In the next chapter, we discuses about the reliable routing using maximum flow algorithm and improve network reliability.

3. RELIABLE ROUTING USING MAXIMUM FLOW

3.1. Description

In this chapter, we find all node-disjoint paths between source node S and destination node D in bi-connected bi-directed weighted graph **G (V,E)** by using **maximum flow algorithm**. By using maximum flow algorithm, we can find all node-disjoint paths between any two nodes in bi-connected bi-directed weighted graph G(V,E).

In order to find node-disjoint paths we generate new graph called auxiliary graph **G'(V',E')** from graph **G(V,E)** with 2n nodes and m + n edges. The new graph generated will help in finding the node-disjoint paths when we run maximum flow algorithm on it.

3.2. Maximum Flow

The maximum flow problem [7] is all about computing the greatest rate at which data can be transferred from source node S to destination node D without violating any capacity constraints. This basic technique is used in maximum flow algorithms can be adapted to solve our problem of finding node-disjoint paths.

A flow network **G(V, E)** is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. If $(u, v) \notin E$, we assume that $c(u, v) = 0$. We differentiate two vertices in a flow network: a source S and a destination D. We assume that every vertex lies on some path from source node to sink. That is, for every vertex $v \in V$, there is a path between two nodes in the graph. therefore graph is connected and set of edges is greater than or equal to set of all nodes minus one, $|E| \geq |V| - 1$.

We can now define flows formally. Let **G (V, E)** be a flow network with a capacity function c. Let S be the source of the network, and let D be the destination. A flow in G is a real valued function f : V × V → R that satisfies the following three properties:

1. **Capacity constraint**: For all u, v ∈ V, we require f (u, v) ≤ c(u, v).

   The capacity constraint says that flow from one node to another node must not exceed the given capacity.

2. **Skew symmetry**: For all u, v ∈ V, we require f (u, v) = - f (v, u).

   Skew symmetry says that flow from node u to node v is the negative of the flow in the reverse direction.

3. **Flow conservation**: For all u ∈ V - {s, t}, we require

$$\sum_{v \in V} f(u, v) = 0$$

Flow conversation property says that the total out of a vertex other than source node is 0, i.e., total data flowing into any particular node is always equal to total data flowing out of a node. For all v ∈ V - {S,D}, i.e., the total flow out of a vertex is 0.

The flow from vertex u to vertex v is defined by f(u,v), which can be positive, zero, or negative. The value of flow is defined as

$$|f| = \sum_{v \in V} f(u, v),$$

That is, the total flow value from node u node v.

We can define total positive flow entering node as

$$\sum_{\substack{v \in V \\ f(u,v)}} f(u, v).$$

The total positive flow leaving a node is defined symmetrically. Total net flow at a node can be defined as total positive flow leaving minus total positive flow entering a node. The net flow at a node must equal to zero, it is also referred as "flow in equals flow out".

Algorithm4. Maximum flow algorithm [7]

```
Max-Flow (G, s, t)
1.    assign C to $\max_{(u,v) \in E} c(u, v)$
2.    initialize flow f to 0
3.    assign K to $2^{\lfloor \lg c \rfloor}$
4.    while K greater than or equals 1
5.    do while there exists an argument path p of capacity at least K
6.            do augment flow f along p
7.        assign K to K / 2
8.    return f
```

3.3. Finding Node-Disjoint Paths Using Maximum Flow

There are several steps in finding node-disjoint paths. First step in finding the node-disjoint paths is to generate auxiliary graph G'(V',E') from original graph G(V,E).

3.3.1. Generating Auxiliary Graph G'(V', E')

Generating auxiliary graph is the most important step in finding node-disjoint paths in a network. First step in generating auxiliary graph is to split all the nodes in the graph other than source node S and destination node D.

Splitting nodes is called "node-splitting" and it is done by splitting each node i in G, other than S and D, into two nodes i' and i" and adding a "node-splitting" edge (i' , i") of unit capacity. Figure 7 shows original graph G with source node S and destination node D.

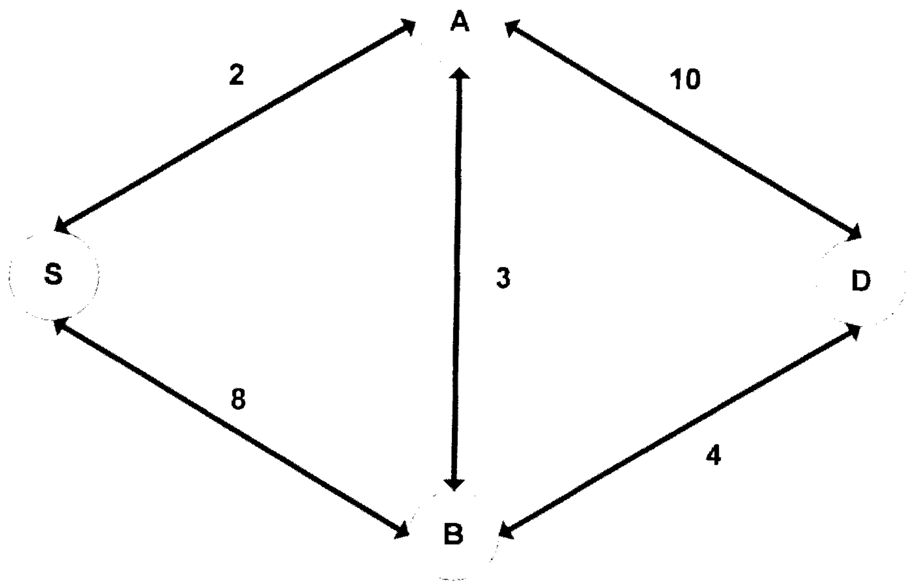

Figure 7. The original graph G with source node S and destination node D.

All the edges in graph G entering node i now enter node i' and all the edges going out from node i, now go out through i'', in auxiliary graph G'. Figure 8 explains how auxiliary graph G' is generated.

After the splitting of the nodes next thing to be done is adding the edges between nodes in graph G' corresponding to the edges in original graph. We assign a capacity of ∞ to each edge in graph G' except the node-splitting edges, which have unit capacity. There will be a one-to-one correspondence between the edge-disjoint paths in G' and node-disjoint paths in G. As a result, the maximum number of arc-disjoint paths in G' is equals the maximum number of node-disjoint paths in graph G. Figure 8 shows auxiliary graph G' that is generated from original graph G.

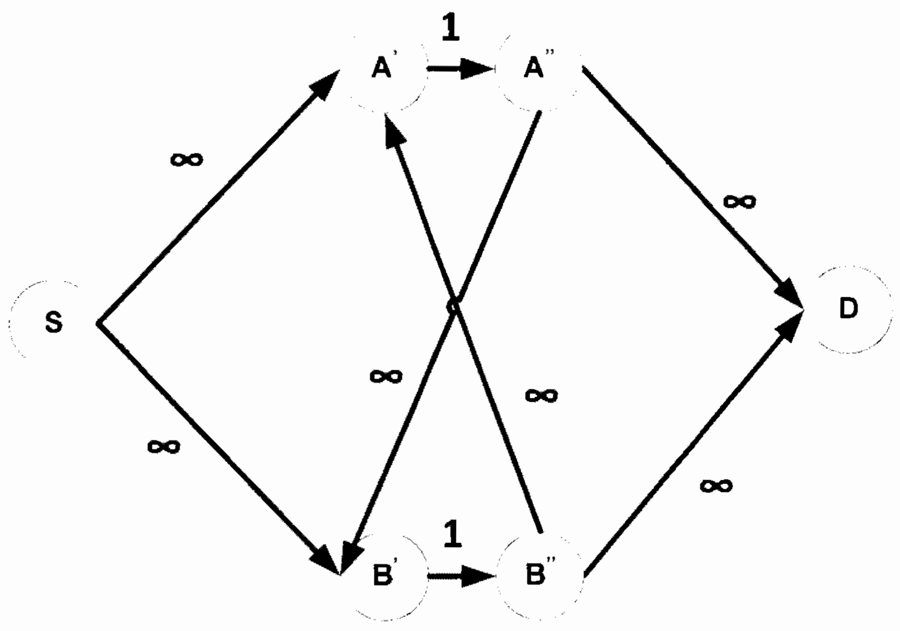

Figure 8. Generation of auxiliary graph G'

Algorithm 5. AUX-GRAPH_GEN(G)

```
1.    read original graph G(V, E)
2.    **repeat**
3.        for each node in original graph G(V, E)
4.            generate two nodes $n_1$ and $n_2$ in auxiliary graph G'(V', E')
5.            add node splitting edge with unit capacity between $n_1$ and $n_2$
6.        n = n + 1
7.        end for
8.    until all nodes in original graph are reached
9.    for all nodes in original graph
10.       an edge entering node n in original graph will enter $n_1$ in auxiliary graph
11.       an edge leaving node n in original graph will leave $n_2$ in auxiliary graph
```

```
12.     end for
13.     write auxiliary graph to file
```

In figure 7, we have original graph for which we have to generate auxiliary graph. In figure 8, we can see that each node other than source node S and destination node D are split. Node A is split into A', A" and node B is split into B', B". After splitting nodes, we add edge with unit capacity between split nodes, i.e., between nodes A', A" and B', B".

We add edges between nodes in auxiliary graph G'(V', E') which are equivalent to original graph G(V,E). But here edge entering node A in original graph will enter node A' in auxiliary graph and edge leaving node A will leave from A", similarly edge entering node B in original graph will enter node B' and node leaving B in original graph will leave B" in auxiliary graph. All the edges in auxiliary graph G' other than node split edges have infinite capacity.

If **n** equals total number of edges in original graph G(V, E) and **e** equals total number of edges in original graph G(V, E), then total number of nodes in auxiliary graph G'(V', E') are **2n-2**, and total number of edges in auxiliary graph G'(V', E') are **e + n**.

3.3.2. Finding Node-Disjoint Paths

After generating auxiliary graph G' we run maximum flow on it to find node-disjoint paths. Algorithm 4 explains how the auxiliary graph is generated. In next step, we will give algorithm for finding node-disjoint paths between source node S and destination node D.

Algorithm6. RELIABLE-ROUTING (G, S, D)

```
1.      read graph G, source S and destination D
2.      using algorithm 5 generate auxiliary graph G'
3.      using algorithm 4 compute the maximum flow from source to destination
```

```
4.    for ( i = 1 to f )
5.    generate path $p_i$ by tracing all nodes from destination node D to source node S
      whose edges with flow value 1.
6.    $Ec(p_i) = 0$
7.    end for
8.    for ( i = 1 to f )
9.        for all edges in graph G'
10.           if ( edge e > 1)
11.               if( $e \in p_1$)
12.                   $Ec(p_i) = Ec(p_i) + Ec(e)$
13.               end if
14.           end if
15.       end for
16.   end for
17.   sort ($Ec(p_1)$, $Ec(p_2)$ ...... $Ec(p_f)$)
18.   $Ec_{avg} = Ec(p_1) + Ec(p_2) / 2$
19.   return $Ec_{avg}$
```

After running maximum flow on auxiliary graph $G^{'}$, on each edge we get the flow and capacity values. Here numerator denotes flow value and denominator denotes capacity value.

At node $A^{'}$ "flow in" must equal "flow out". The out flow at node $A^{'}$ must equal 1. the edge between node $A^{'}$ and node $A^{''}$ is 1. At node $A^{''}$ there are two edges leaving node. The flow value on edge node $A^{''}$ and node D is 1 as it results total flow value between source node S and destination node D to be maximum. If the flow value between node A'' and node B' is 1, i.e., There is a flow between them then total flow value between source node S and destination node D is not maximum. We use maximum flow to find maximum amount flow possible between source node S and destination node D

Flow value on edge between source node S and node B' is 1 even though capacity is ∞. This is because the of flow conversation property, that is, the total flow into a vertex has to be 0. The figure 9 shows flow value on edge between source node S and node A' is 1 even though capacity is ∞. This is because the of flow conversation property, that is, the total flow into a vertex has to be 0. In The following figure we will see the flow on the network after running maximum flow on graph G'.

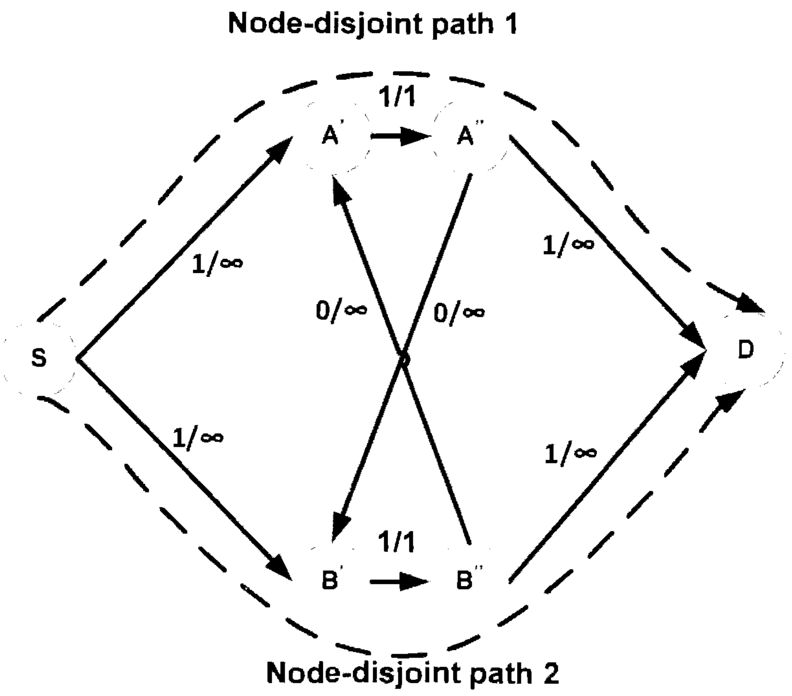

Figure 9. Graph G' with flow values

At node A' "flow in" must equal "flow out". The out flow at node B' must equal 1, the edge between node B' and node B'' is 1. At node B'' there are two edges leaving node. The flow value on edge node B'' and node D is 1 as it results total flow value between source node S and destination node D to be maximum.

Flow value on the edges between node A'' and B' and edge between node B'' and A' is 0. As there is no flow on these edges, they can be removed. We get node-disjoint paths after removing edges that doesn't have any flow.

The figure 10 shows two node-disjoint paths between source node S and destination node D, with flow values are on edges of the paths indicating the node-disjoint paths.

After finding node-disjoint paths on the auxiliary graph, we have to find equivalent paths of node-disjoint paths of auxiliary graph in original graph.

Split nodes in auxiliary graph are unified in the original graph and equivalent node-disjoint paths in original graph doesn't include node split arc but unifies two split nodes into one.

We calculate the energy consumption of the node-disjoint path by adding the energy consumption of each edge (link) in the path. Energy consumption of each edge is the amount of energy it takes a node to forward a packet to next node in the path. Here we find all the node-disjoint paths that are present in the network as shown in figure 11. But finding node-disjoint paths is not final; we have to find the paths with lowest energy consumption and also two paths with minimum total energy consumption. We get two node-disjoint paths P_1 and P_2. Energy consumption of the path is the sum of energy consumption of each and individual edge that is included in the path. i.e., energy consumption of path p = v0, v1, ...,vk is the sum of the energy consumption of its constituent edges.

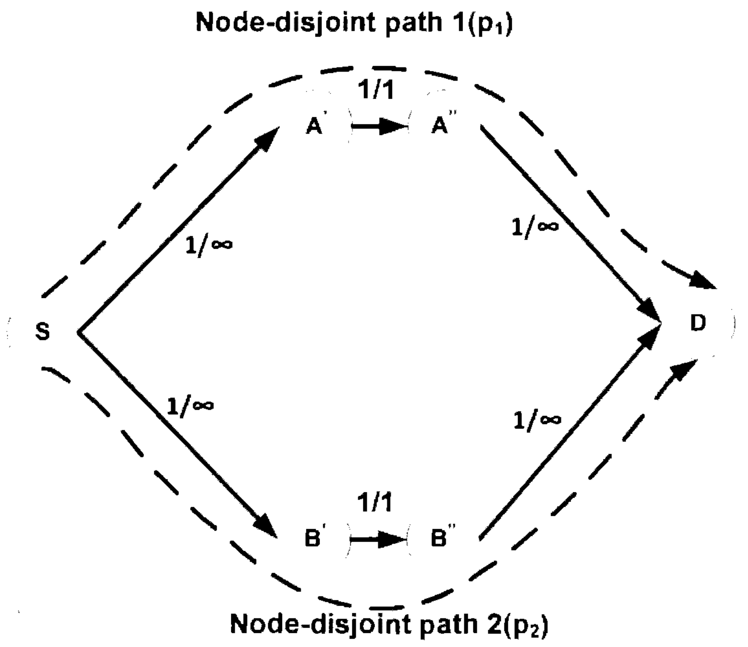

Figure 10. Node-disjoint paths between source node S and destination D

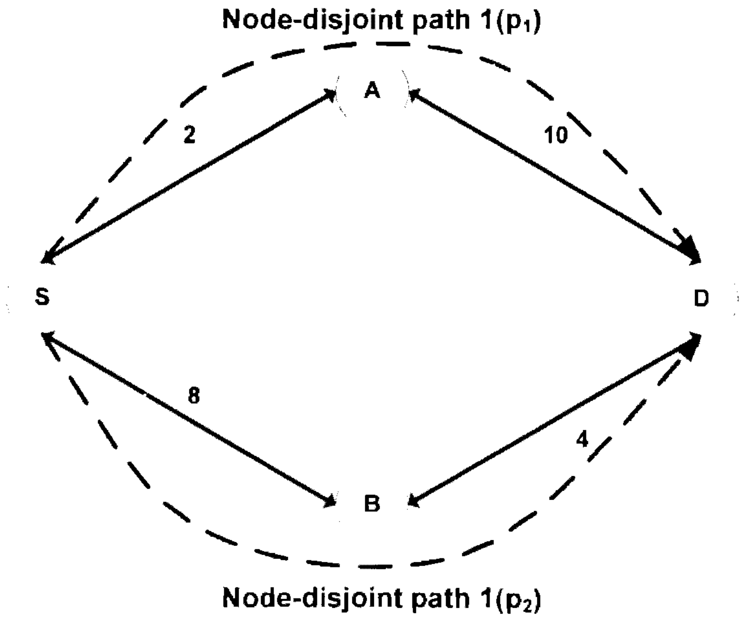

Figure 11. Equivalent graphs in original graph

3.4. Running Maximum Flow on Original Graph

In this section, we tell the reason of generating auxiliary graph G' with spitted nodes which have unit node-splitting edges and why it is necessary to generate auxiliary graph in order to find all the node-disjoint paths.

We use a new graph G for proving that we can't find node-disjoint all the time when we run maximum flow algorithm on graph without node-splitting.

We run maximum flow algorithm on the graph G in figure 12, to see what paths we can get for the maximum flow magnitude between source node S and destination node D.

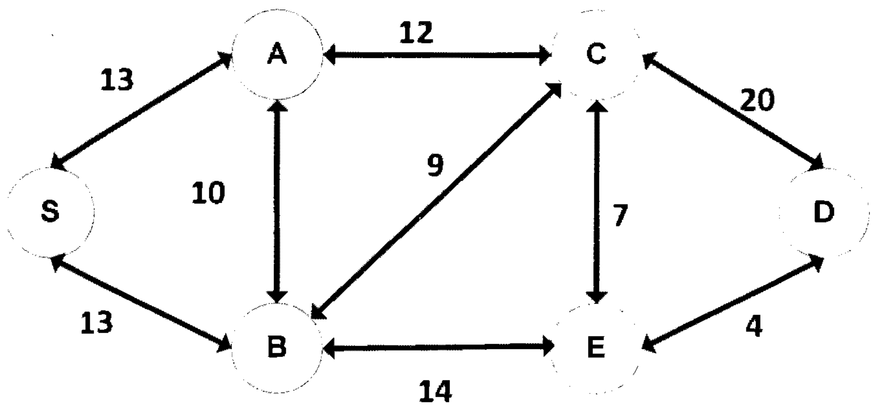

Figure 12. New Graph G with energy consumption as edge weights

The figure 13 shows all possible paths between source node S and destination node D after running maximum flow on the original graph G.

All the nodes in graph other than source node S and destination node D are saturated and there is no way to transmit more flow through the graph. The flow value through the graph is the amount of flow leaving the source node S is equal to the flow value entering destination node D. Here the flow value is 23.

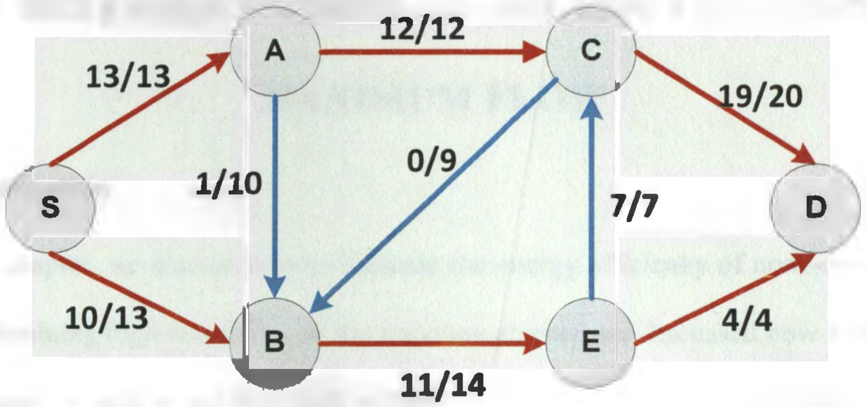

Figure 13. Edges with flow value and possible paths with high flow magnitude

In this figure 13, we have different edges with two different colors "red" color and other in "blue" color. All the edges with flow value at least 1 will constitute paths between source and destination. In order to get node-disjoint paths we need to have edges with red color present in paths. If we can get edges with red color all the time then we can get node-disjoint paths all the time. But maximum flow doesn't guarantee that it will lead us to node-disjoint paths all the time. Paths from source node to destination can be combination of red edges and blue edges.

We may get different values on the edges that are not on minimum cut, when choose paths in different order. But the total flow value will be same i.e., the flow magnitude will be same.

By this method, we can't conclude that we can get node-disjoint paths on the network. But by generating auxiliary graph, we can guarantee that every time we can get node-disjoint paths. And we can find all the possible node-disjoint paths that are present in the network.

4. RELIABLE ENERGY-AWARE ROUTING USING MAXIMUM FLOW

4.1. Description

In this chapter, we discuss how to increase the energy efficiency of node-disjoint paths while maintaining high reliability. In the previous chapter, we discussed how to find node-disjoint paths. A way to achieve high reliable node-disjoint paths is by using maximum flow algorithm and generating auxiliary graph.

4.2. Finding Energy Efficient Node-Disjoint Paths

Here we propose new scheme to improve energy efficiency of the reliable node-disjoint path routing using maximum flow algorithm. In a network, some links between nodes can have high energy consumption. These edges will drain all energy of nodes it corresponds and thereby decreasing the life time of the network.

If we can find a way to eliminate edges or links between nodes that are not necessary in communication between nodes, by this we can avoid unnecessary wastage of energy and thereby decreasing overall consumption of the network. This intern increases overall reliability of the network there by increasing total performance of network.

We know increase reliability by combining high energy consuming link removal with maximum flow algorithm. This combination of our scheme with maximum flow algorithm increases the efficiency of the network. The figure 14 shows the graph we consider for this scheme. We have a network (graph G) for which we have to find node-disjoint paths from source S to destination D. We have discussed finding node-disjoint paths in previous chapter. Here also we are going to use same scheme. We will be generating auxiliary graph

G' from original graph G to find node-disjoint paths.

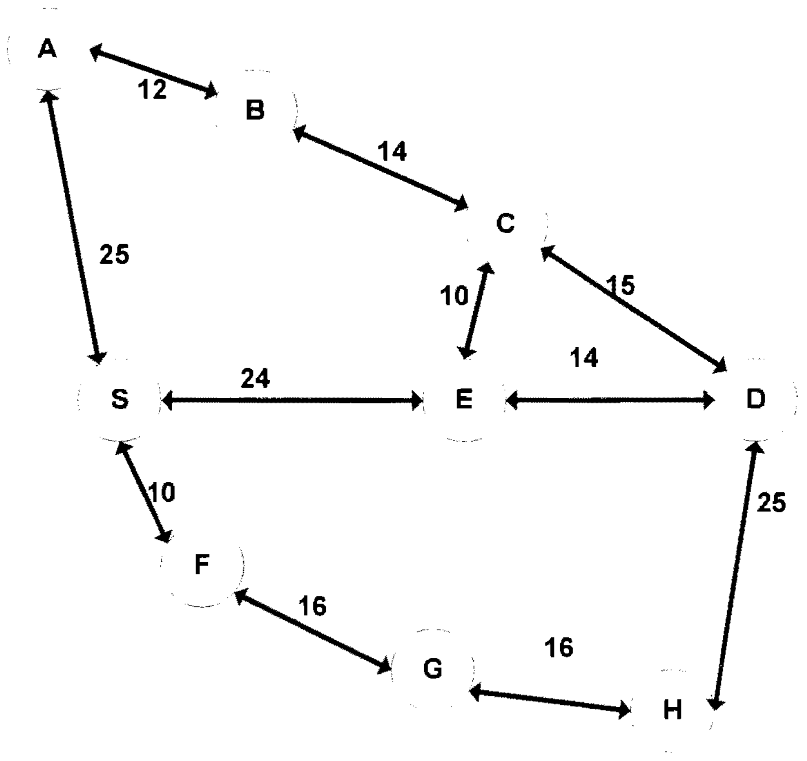

Figure 14. Graph G **(V,E)** for DSP algorithm

In order to generate auxiliary graph G' we split all nodes in graph G' and add unit edges between them and call them as "node splitting edges". In auxiliary graph G', we use capacity in contrast to energy consumption used in original graph G. Splitting nodes is called "node-splitting" and it is done by splitting each node i in G, other than S and D, into two nodes i' and i" and adding a "node-splitting" edge (i',i") of unit capacity. All the edges in graph G entering node i now enter node i' and all the edges going out from node i now go out through i" in auxiliary graph G'.

After the splitting of the nodes next thing to do is to add the edges between nodes in graph G' corresponding to the edges in original graph. We assign a capacity of ∞ to each edge in graph G' except the node-splitting edges, which have unit capacity. There will be a one-to-one correspondence between the edge-disjoint paths in G' and node-disjoint paths in G. As a result, the maximum number of arc-disjoint paths in G' is equals the maximum number of node-disjoint paths in graph G.

Generating auxiliary graph G' is the first part of the process, next part would be to find node-disjoint paths between source node S and destination node D. We find node-disjoint paths similar way as we find in chapter 3. i.e., by running maximum flow algorithm on graph G, to find all node-disjoint paths that are present between source node S and destination node D.

Running maximum flow on auxiliary network gives us node-disjoint paths. It is similar to what we have done in previous chapter. Algorithm 6 explains working of the scheme we use in this chapter to find reliable node-disjoint paths.

Algorithm6. Energy Efficient Reliable Node-disjoint Path Algorithm

REL-NODE-DIS(G, S, D)

```
1.    read graph G, source S and destination D
2.    using algorithm 5 generate auxiliary graph G'
3.    using algorithm 4 compute the maximum flow from source to destination
4.    for ( i = 1 to f )
5.    generate path $p_i$ by tracing all nodes from destination node D to source node
      S whose edges with flow value 1.
6.    $Ec(p_i) = 0$
7.    end for
8.    if (f > 1)
```

```
9.        then
10.            for all edges in graph G
11.                    search for edge e with highest energy consumption in Graph G
12.                        if ( $Ec(e) > Ec(G - e)$ )
13.                            then
14.                                $G = G-e$
15.                                    goto step 1
16.       else
17.            $G = G + e$
18.            goto step 1
19.    end if
20.    for ( $i = 1$ to f )
21.                for all edges in graph G'
22.                        if ( edge $e > 1$)
23.                            if( $e \in p_1$)
24.                                    $Ec(p_i) = Ec(p_i) + Ec(e)$
25.                            end if
26.                        end if
27.                end for
28.            end for
29.            sort ($Ec(p_1)$, $Ec(p_2)$ ...... $Ec(p_f)$)
30.            $Ec_{avg} = Ec(p_1) + Ec(p_2) / 2$
31.            return $Ec_{avg}$
```

The figure 15 shows one possibility of pair of node-disjoint paths that are present between source node S and destination node D. Both paths are distantly marked with different colors. The one with red color is first path p_1 and the one with green is second path p_2.

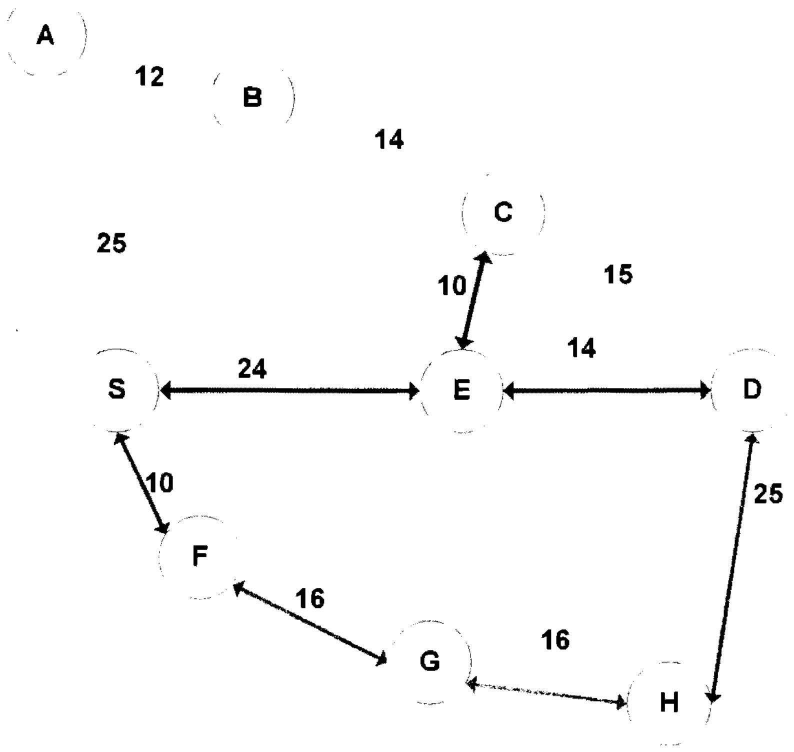

Figure 15. Possible node-disjoint paths

When we run maximum flow on auxiliary graph G' we get all possible paths present in the network. Among all paths, we consider only two paths which have minimum sum of energy consumption of these two paths. Paths p_1 and p_2 have very less combined energy consumption. This is the main reason for considering these two paths for this particular network.

Now we calculate the energy consumption of each path. According to energy model defined previously. Energy consumption of each edge is approximately equal to the square of the distance. Energy consumption of total path is sum of all edges present in the path.

$Ec(p_1) = (24)^2 + (14)^2$

$Ec(p_1) = 772.$

Similarly, we calculate energy consumption of second path p_2.

$Ec(p_2) = (10)^2 + (16)^2 + (16)^2 + (25)^2$

$Ec(p_1) = 100 + 256 + 256 + 625$

$Ec(p_1) = 1237.$

We got energy consumption of each path. The total energy consumption of two paths is the summation of energy consumption of paths p_1 and p_2.

$Ec(p_1 + p_2) = Ec(p_1) + Ec(p_1).$

$Ec(p_1 + p_2) = 772 + 1237.$

$Ec(p_1 + p_2) = 2009.$

We got the energy consumption of two node-disjoint paths. We propose scheme in order to improve energy consumption.

If we have two or more than two node-disjoint paths present between source node and destination node, then we take out the edges with highest energy consumption in the network and run maximum flow algorithm on the network to find node-disjoint paths. If we get two or more than two node-disjoint paths between source node S and destination node D even after removing edges with highest magnitude, then we will remove edges with next highest magnitude and run maximum flow on the network. We do this until we get flow value of maximum flow as one i.e., there is one node-disjoint path present between source node S and destination node D. But we need a pair of node-disjoint paths to improve reliability of the network. So we go back one step, we add last removed edges back to network. This way of adding all the edges back to network will give us two node-disjoint paths in network between source S and destination D.

The figure 16 shows after removing the edges with highest magnitude we are able to get a pair of node-disjoint paths with improved energy consumption.

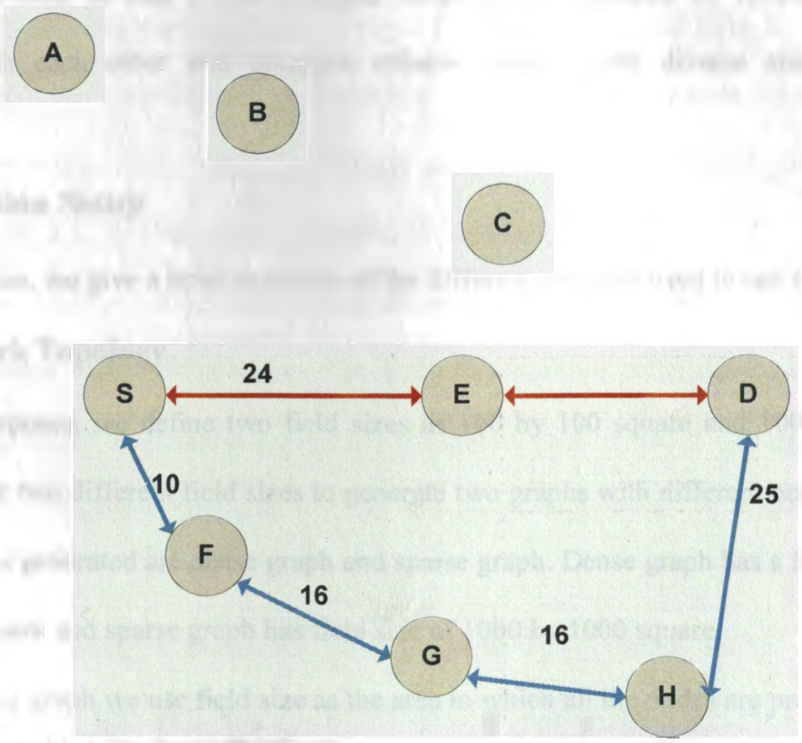

Figure 16. Reliable and energy efficient node-disjoint paths

5. PERFORMANCE EVALUATION

5.1. Description

This chapter, we present simulation results to evaluate the performance of proposed routing algorithms. We have two goals. First, determine how our reliable routing algorithms perform in real world. Second, compare performance of reliable routing algorithm with each other and compare reliable routing with diverse shortest path algorithm.

5.2. Simulation Setup

In this section, we give a brief overview of the different modules used in our simulator.

5.2.1. Network Topology

For test purposes, we define two field sizes as 100 by 100 square and 1000 by 1000 square. We use two different field sizes to generate two graphs with different node density. The two graphs generated are dense graph and sparse graph. Dense graph has a field size of 100 by 100 square and sparse graph has field size of 1000 by 1000 square.

In generating graph we use field size as the area in which all the nodes are present, these nodes are randomly generated which is similar to random deployment of sensor nodes in the field. Each node has transmission range of 25 in dense network and transmission range of 150 in sparse network, every node can transmit data within its transmission range.

We assume that geographical topology is same throughout our simulation for all networks.

5.2.2. Performance Evaluation

In performance evaluation, we considered to run all three algorithms on different graphs with different number of nodes in both dense and sparse networks. In order to get accurate results we have taken average of ten Node-Disjoint paths for each graph. Taking the average value is significant as it helped in finding accurate value.

5.2.2.1. Dense Network

First, we considered to run all three algorithms diverse shortest path (DSP), node-disjoint path (NDP), and energy efficient shortest path (ENDP) on dense graph.

The field size of dense network is considered 10,000 square and each sensor network has transmission range of 25. We run all three algorithms on different graphs with different number of nodes in it. The density is different for different graphs.

Networks with nodes 10, 25, 50, 75,100,200 are considered as dense networks for field size 10,000 square. Here DSP is diverse shortest path algorithm, NDP is node-disjoint path algorithm and ENDP is efficient node-diverse shortest path. Table 1 shows the energy consumption of different networks for all the three algorithms DSP, NDP, ENDP respectively.

Table 1. Shows energy consumption for dense graph with different density

Density	DSP	NDP	ENDP
0.001	1277.9	1529.1	1354.8
0.0025	1036.3	1158.3	1135.7
0.005	1157.9	1650.1	1599
0.0075	859.2	1302.7	1277.6
0.0085	561.1	805.2	718.199
0.01	761.799	1128.6	1039.3
0.02	507.5	1205.5	883.8

.The figure 17 is the graphical representation of energy consumption for dense networks. Here the densities of networks are 0.001, 0.0025, 0.005, 0.0075, 0.0085, 0.01, and 0.02.

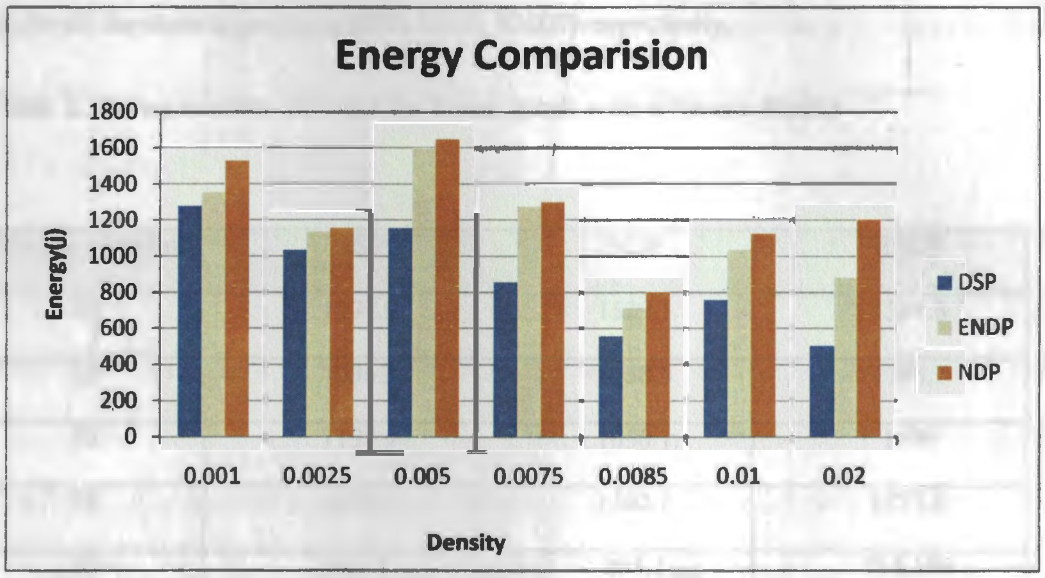

Figure 17. Comparison of energy consumption to density in dense graphs

As the shortest path algorithm is the most efficient algorithm for finding shortest path accurately. Hence diverse shortest path (DSP) algorithm can find routes with least energy consumption. It can be seen that node-disjoint paths found through DSP consume least amount of energy.

Node-disjoint path (NDP) algorithm can find reliable disjoint paths between source and destination nodes. We can see from figure 16 that disjoint routes by NDP consume more energy as it tries to find reliable routes between source and destination nodes. NDP algorithm compromises efficiency to reliability. Here energy consumption is significantly more than other two algorithms.

Efficient node-disjoint path (ENDP) algorithm uses same principle in finding reliable node-disjoint paths as NDP algorithm but tries to minimize energy consumption.

Table 2 shows the energy consumption of different networks with different number of nodes for all the three algorithms DSP, NDP, ENDP respectively.

Table 2. Shows number of nodes for dense graph with different density

Number of nodes	DSP	NDP	ENDP
10	1277.9	1529.1	1354.8
25	1036.3	1158.3	1135.7
50	1157.9	1650.1	1599
75	859.2	1302.7	1277.6
85	561.1	805.2	718.199
100	761.799	1128.6	1039.3
200	507.5	1205.5	883.8

We can see from figure 18, that node-disjoint paths by ENDP algorithm consume significantly less energy than NDP algorithm without compromising reliability. We can calculate gain in energy consumption by ENDP algorithm. The energy consumption gain can be calculated by

$$\boldsymbol{Energy\,gain} = \frac{\mathbf{EC(\sum NDP\ paths) - EC(\sum ENDP\ paths)}}{\mathbf{EC(\sum NDP\ paths)}} * \mathbf{100}$$

$$\boldsymbol{Energy\,gain} = \frac{\mathbf{9277-8008}}{\mathbf{9277}} * \mathbf{100}$$

$$\boldsymbol{Energy\,gain} = \mathbf{13.67\%}$$

The gain in energy consumption is 13.67 %; this is significant gain in energy consumption without compromising reliability.

Shortest path algorithm is efficient algorithm to find shortest paths between source and destination nodes. Diverse shortest path algorithm to find node-disjoint paths is also efficient but not reliable. There is possibility that diverse shortest path algorithm may not be able to find second shortest path.

In figure 19, we can see graph for all the networks with no second shortest path when we used diverse shortest path (DSP) to find node-disjoint paths between source and destination. We can see in figure 18, as density of network increases performance of DSP increases as the connectivity of network increases. In sparse graph network is more likely less connected compared to dense graph. Percentage gain is calculated by.

$$= \% \text{ of no second shortest paths} = \frac{\text{Number of no second shortest path}}{\text{Total number of paths}} * 100$$

$$\text{i.e. } \% \text{ of no second shortest paths} = \frac{8}{7*10} * 100 = 11.43\%$$

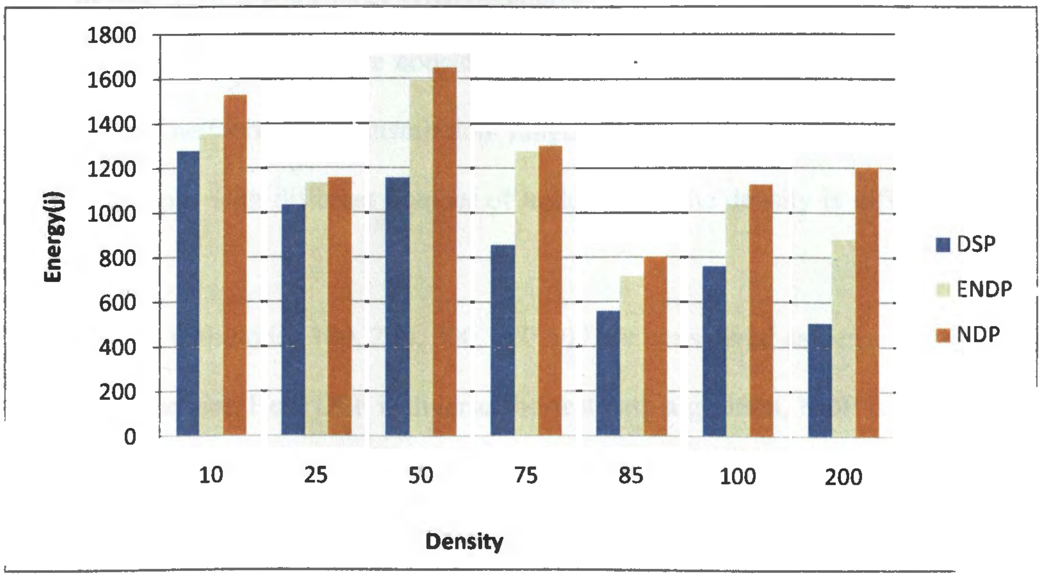

Figure 18. Energy consumption vs number of nodes in dense graphs

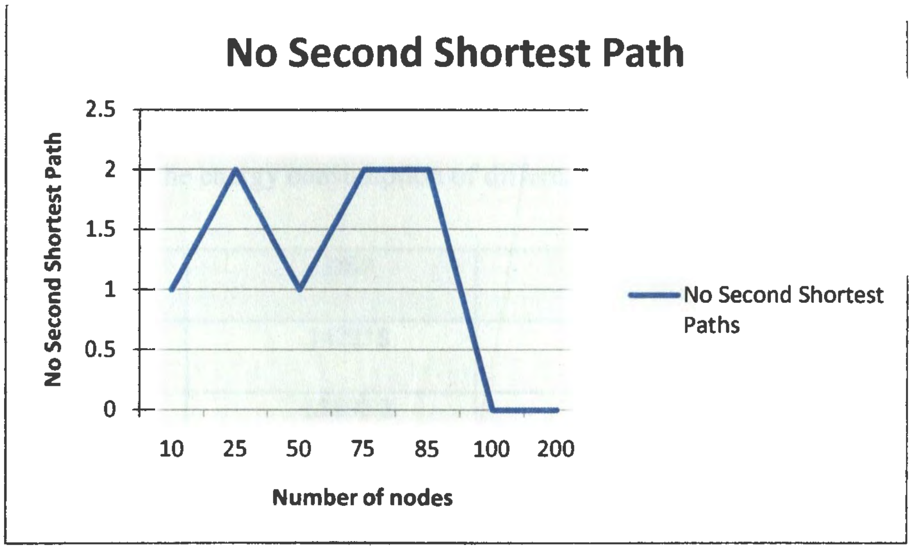

Figure 19. Diverse shortest paths without second shortest path

We can calculate percentage of no second shortest path for networks considered.

5.2.2.2. Sparse Network

Here we considered to run all three algorithms diverse shortest path (DSP), node-disjoint path (NDP), and energy efficient shortest path (ENDP) on sparse graph.

The sparse network we are considering here has field of size 1000*1000 square and each sensor network has transmission range of 150. We run all three algorithms on different graphs with different number of nodes in it. The density is different for different graphs.

Networks with nodes 100, 200, 300, 400,500 are considered as dense networks for field size 10,000 square. Here DSP is diverse shortest path algorithm, NDP is node-disjoint path algorithm and ENDP is efficient node-diverse shortest path.

Figure 20 and figure 23 are the graphical representation of energy consumption for sparse networks and dense networks respectively. Here the densities of networks are

0.0001, 0.0002, 0.0003, 0.0004, and 0.0005. Table 3 shows energy consumption of different networks with different network density.

Table 3. Shows the energy consumption of different networks for all three algorithms

Density	DSP	NDP	ENDP
0.0001	142218	195950	185917
0.0002	53832.2	84743.9	82845.1
0.0003	48773.7	89439.4	87756.9
0.0004	37008.4	75227.8	74394.8
0.0005	28814.9	68765.5	67932.1

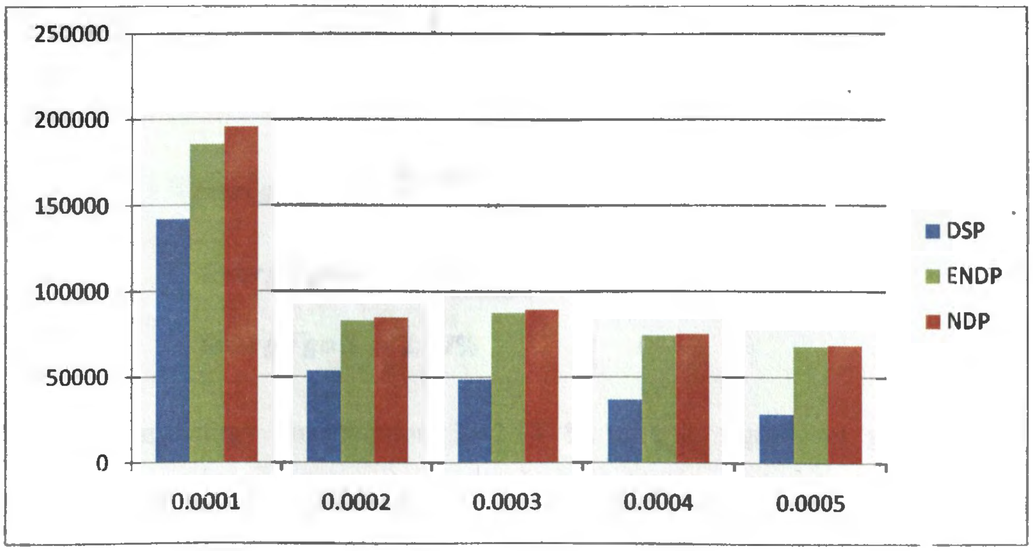

Figure 20. Energy consumption of sparse networks with different number of nodes

As the shortest path algorithm is the most efficient algorithm for finding shortest path accurately. Hence diverse shortest path (DSP) algorithm can find routes with least energy consumption. It can be seen in above figure that node-disjoint paths found through DSP consume least amount of energy.

Node-disjoint path (NDP) algorithm can find reliable disjoint paths between source and destination nodes. We can see from figure 18, that disjoint routes by NDP consume more energy as it tries to find reliable routes between source and destination nodes. NDP algorithm compromises efficiency to reliability. Here energy consumption is significantly more than other two algorithms

Efficient node-disjoint path (ENDP) algorithm uses same principle in finding reliable node-disjoint paths as NDP algorithm but tries to minimize energy consumption.

We can see from figure 18, that node-disjoint paths by ENDP algorithm consume significantly less energy than NDP algorithm without compromising reliability. We can calculate gain in energy consumption by ENDP algorithm. The energy consumption gain can be calculated by

$$\textbf{\textit{Energy gain}} = \frac{\mathbf{EC(\sum NDP\ paths) - EC(\sum ENDP\ paths)}}{\mathbf{EC(\sum NDP\ paths)}} * \mathbf{100}$$

$$\textbf{\textit{Energy gain}} = \frac{\mathbf{514126.6 - 498845.9}}{\mathbf{514126.6}} * \mathbf{100}$$

$$\textbf{\textit{Energy gain}} = \mathbf{2.97\%}$$

The gain in energy consumption is 2.97 %, this is significant gain in energy consumption without compromising reliability. When we compare energy gain with dense networks, actual energy gain is much less, due to the fact that there are much less alternative routes from source to destination. Table 4 shows energy consumption for different number of nodes for sparse graph with constant density. Figure 21 shows energy consumption for different number of nodes on sparse network.

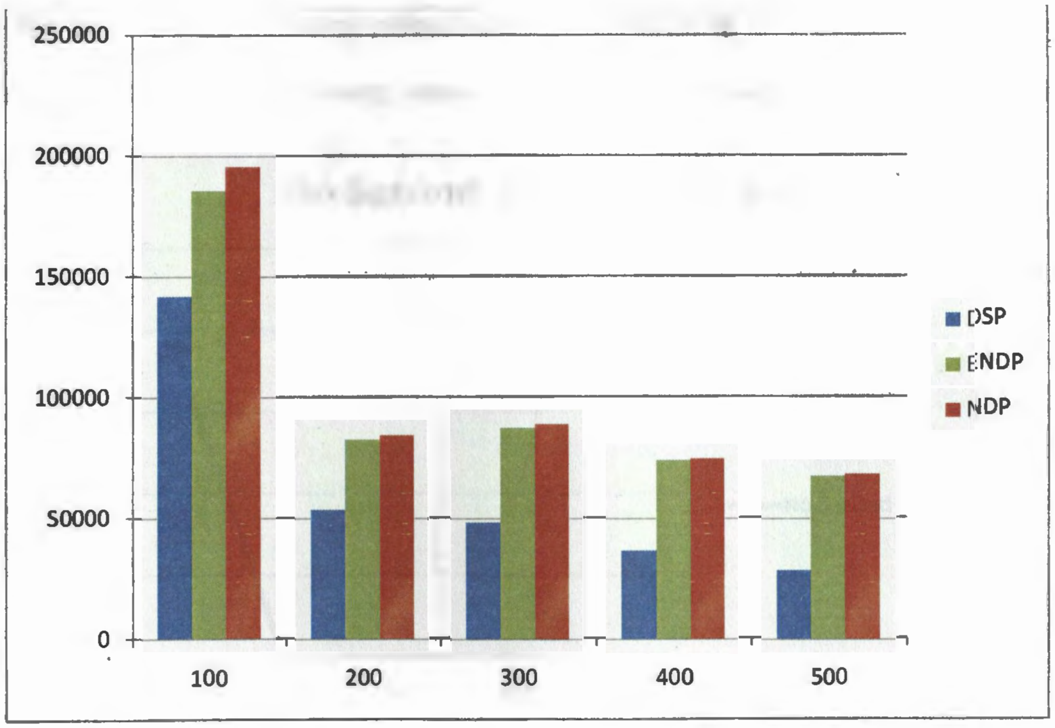

Figure 21. Energy consumption vs. number of nodes in sparse graphs

Table 4. Shows number of nodes for sparse graph with different density

Number of nodes	DSP	NDP	ENDP
100	142218	195950	185917
200	53832.2	84743.9	82845.1
300	48773.7	89439.4	87756.9
400	37008.4	75227.8	74394.8
500	28814.9	68765.5	67932.1

Shortest path algorithm is efficient algorithm to find shortest paths between source and destination nodes. Diverse shortest path algorithm to find node-disjoint paths is also efficient but not reliable. There is possibility that diverse shortest path algorithm may not be able to find second shortest paths. Figure 22 shortcoming of DSP in finding second shortest path. Table 5 shows energy consumption for dense network with constant density.

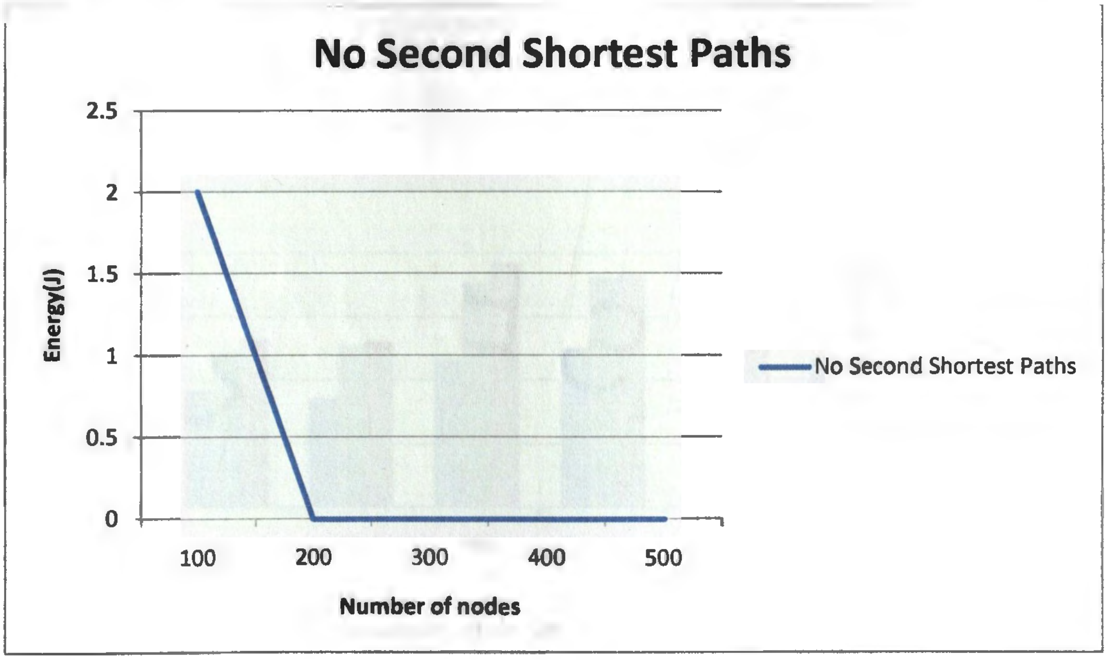

Figure 22. Diverse shortest paths with no second shortest path

Table 5. Sparse network on constant density

Density	Number Of Nodes	DSP	NDP	ENDP
0.0075	50	925	1326.7	1225.1
0.0075	75	859.2	1302.7	1277.6
0.0075	100	1162.8	1924	1767.4
0.0075	125	1254.9	1839.6	1812.4
0.0075	150	1699	2941.8	2899.5
0.0075	175	1182.9	1873.7	1805.6

The figure 23 shows graphical representation of energy consumption for different number of nodes with density of nodes constant.

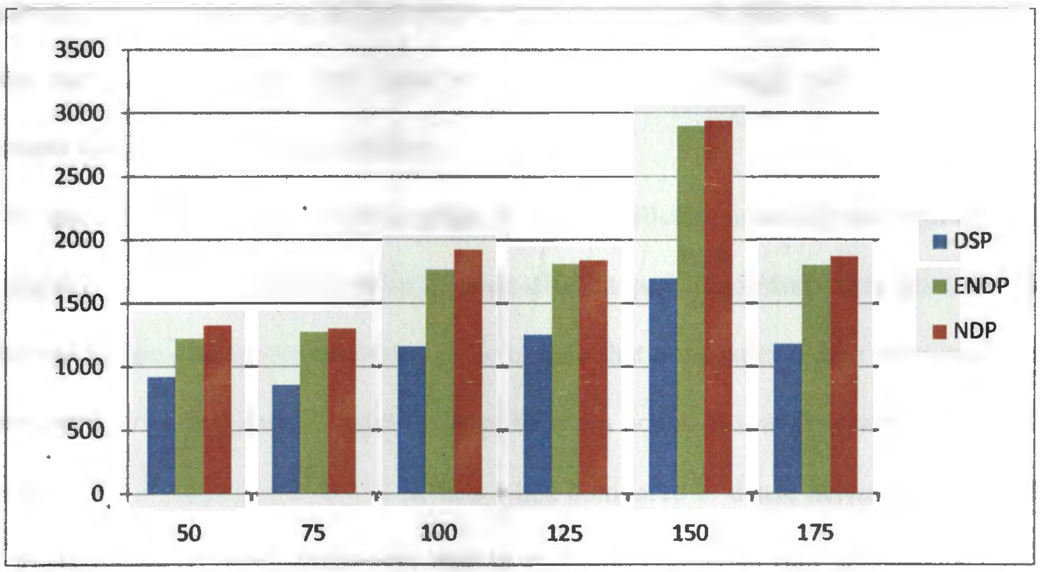

Figure 23. Energy consumption of dense networks with different number of nodes

5.3. Observations

The experimental results had lot of variation from dense network to sparse network. This is because of the density of the network. Denser the network is, more edges or links are present between nodes. Denser the network is stronger the connectivity between node within the network. Less dense the network or the sparser network is lesser is the connectivity between nodes within the network as there are few edges or links between nodes.

Energy efficient node-disjoint algorithm behaved differently on dense network and sparse network. In dense network, the results of energy efficient node-disjoint path algorithm were expected. The energy consumption is less than reliable node-disjoint algorithm.

As density is more, there are more nodes connected with each other. This in turn increases the possibility of more node-disjoint paths between nodes. Due to high connectivity between nodes, we have more alternate paths with less energy consumption in dense networks, nodes are very near to each other. Less energy will be consumed to transmit data from one node to another.

In sparse network energy, consumption of energy efficient node-disjoint path algorithm is slightly greater or equal to reliable node-disjoint path algorithm. This phenomena is observed because in sparse network we have node that are scattered in a very large field. There will be less number of edges or links between nodes in sparse network. Due to this connectivity of entire network will be less when compared to dense networks. Here nodes are scattered, some nodes can be very near to each other might be very far from each other. We generally observe edges with high energy consumption and edges with very low energy consumption.

In sparse network when we are implementing energy efficient node-disjoint path algorithm, we observe paths having many edges have less energy consumption. This is because in energy efficient node-disjoint path algorithm, we avoid the links or edges with largest energy and choose edges with less energy consumption. In sparse networks as nodes are scattered some nodes are close to each other and some far, when we try to avoid a edge with high energy consumption then we will end up with more number of edges with less individual energy consumption. Total energy consumption of all these small edges will be high.

6. DISCUSSION

6.1. Description

In this chapter, we discuss lessons learned, challenges and limitations of using maximum flow algorithm to find reliable and energy-efficient node-disjoint paths in wireless sensor networks.

6.2. Lessons Learned

There are many ways to find node-disjoint paths in wireless sensor networks. By using shortest path algorithm is the easiest way due to limitations of shortest path algorithm we couldn't find node-disjoint path always. By using maximum flow algorithm, we can find reliable node-disjoint paths in WSNs. Complexity of finding node-disjoint paths increases as we include parameters like reliability and energy efficiency.

6.3. Challenges

Due to limitations of shortest path algorithm to find node-disjoint paths in WSNs, we are forced to use maximum flow algorithm. The creation of auxiliary graph was important step in finding node-disjoint paths.

In finding reliable and efficient node-disjoint paths, we had to use auxiliary graph in conjunction with maximum flow algorithm.

6.4. Limitations

Shortest path algorithm isn't the best solution for finding node-disjoint paths between wireless sensor networks. Using maximum flow algorithm solves the problem to some degree but also creates some new problems. In finding reliable and efficient node-disjoint paths, we hide edge with highest edge weight, this forces algorithm to find edges with less weight. But this solution doesn't work every time, especially in sparse network where single large edge might consume less energy than group of small edges. This is one of the limitations of this algorithm on sparse network.

Using maximum flow algorithm and generating auxiliary graph consumes more processing power; this in turn depletes battery by consuming more power.

7. RELATED WORK

In this chapter, we discuss related work done in finding node-disjoint path in wireless sensor networks

Other approaches finding node-disjoint multi paths is node-disjoint parallel multi-path routing algorithm (DPMR) [10], it take full advantage of known geographic information and finds the node-disjoint multi-paths. Idea used in this approach is Local Minimum Phenomenon. Through improving the packet delivery performance and evenly distributing the energy load among the sensors, DPMR [10] can prolong the networks system lifetime. This approach is good as far as we know geographic information of the area in with wireless sensor nodes is deployed.

Alternative approach with dealing with failed nodes in the network is by using relay nodes. By using relay nodes, it takes strain out of each node as the most of the transmission of sensed data is done by relay nodes, which have relatively high power transceivers and more power. Idea in using relay nodes is formulation and approximation

7.1. Future Work

Future works include finding maximum flow algorithm with idea like local minimum phenomenon and fine node-disjoint reliable and energy efficient paths and compare them with the approach I used to in this paper.

To find alternative to shortcoming of shortest path algorithm, by combing shortest path with idea like local minimum phenomenon and compare with my results in this paper.

REFERENCES

[1] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, "Next Centuary Challenges: Scalable Coordination in Sensor Networks," MobiCOM, August 1999, pp. 263-270.

[2] Volcano monitoring. Harvard Sensor Networks Lab: http:// fiji.eecs.harvard.edu/.

[3] Snow Water Equivalent Monitoring with Wireless Sensor Networks. Sensor Networks and Wireless Workgroup: http://www.cems.uvm.edu/research/cems/snow/swe.php.

[4] H. Karl and A. Willig, Protocols and Architectures for Wireless Sensor Networks. West Sussex, England: Wiley 2005.

[5] Information Processing And Routing In Wireless Sensor Networks by Yang Yu (motorola labs, usa), Viktor K Prasanna (university of southern california, usa), &Bhaskar Krishnamachari (university of southern california, usa).

[6] P. Levis, TinyOS Programming. http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf. June 28, 2006.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford SteinIEEE "Introduction to Algorithms, Second Edition" The MIT Press Cambridge, Massachusetts London, England.

[8] Channel Wireless Mesh Networks, ACM MobiHoc'2005, and pp. 68-77.

[9] M.A Labrador, P.M Wightman "Topology Control in Wireless Sensor Networks: With a Companion Simulation Tool for Teaching and Research".

[10] Shanping Li, Zhendong Wu, "Node-Disjoint Parallel Multi-Path Routing in Wireless Sensor Networks" ICESS 2005.

[11] BinHao,JianTang and GuoliangXue "Fault-Tolerant Relay Node Placementin Wireless Sensor Networks : Formulation and Approximation"