CHAPTER 3

MULTIPLE SINK POSITIONING AND RELOCATION FOR IMPROVING LIFETIME IN WIRELESS SENSOR NETWORK

In this chapter, K-Partitioned Minimum Depth Tree using the optimal search is proposed in Wireless Sensor Networks (WSN) for improving the network lifetime. Initially the optimal number of sinks is determined using the optimal sink algorithm satisfying the h-hop constraint. Then a K-Partitioned Minimum Depth Tree (k-PMDT) is constructed for positioning multiple sink nodes and setting up the routes. After determining the optimal number of sink positions and routing, best sink reposition is selected by optimum search method. Link movement is done by intelligent movement and it limits the sinks movements while maintaining their direction to the optimal positions.

3.1 Selecting Optimal Number of Sinks

Using this optimal sink algorithm, the optimal numbers of sinks are selected to maximize the network lifetime. "S" is set of sinks and "PS" is set of potential sinks. The set of potential sinks are derived from the set “S”. \( N_n(ps) \) is the set of neighboring sensor nodes of sink \( s \) and \( N_h(ps) \) is the number of the hops from \( v \) to \( s \) is no greater than \( h \).

The collection of sets derived by the set \( S \) of potential sink location is \( C = \{ N_h(ps) / ps \in S \} \). Each sensor node reach one of the chosen sinks with no more than \( h \) hops and that is equivalent to finding a sub-collection \( ps \in S \), the set cover problem is NP complete problem. Instead NP complete, a greedy heuristic is employed then it delivers an approximation solution to the problem with the approximation ratio of \( (\log B) \), where \( B = \max s \in S_\perp \{ |N_h(s)| \} \leq n \).
Consider a sensor CV is referred to be covered by a sink s if the number of hops from CV to S is no more than h; otherwise, CV is uncovered by s. If the sink is uncovered means it is not covered by the sink s.

Algorithm: Optimal Sink Selection

1. Start
2. Define $S = \text{set of sink}$
3. Input $V = \text{set of sinks covered by } s$
4. $S = \text{set of sink}$
5. $ps = 0$  // Initially all sensors in CV are uncovered
6. while $S \neq 0$
7. {
8. If $(S \cap N_h(ps) = \text{max})$
9. Select a set $N_h(ps) \in S$
10. $ps \leftarrow ps \cap S$
11. $S \leftarrow S - N_h(s)$
12. $C \leftarrow C - \{N_h(s)\}$
13. }
14. return ps
15. End

In this algorithm, initially all sensors in CV are uncovered and PS is set to empty. The while loop run iteratively and each time it compare the sensors in the set with the covered sensors in the list.
Figure 3.1 Optimal Sink Selection

In figure 3.1. $S$ is the set and it contains all the nodes (both potential and unpotential nodes). $PS$ is the set contain potential sets. In the optimal sink algorithm initially $PS$ is set to empty. The algorithm runs iteratively and each time it selects the node which has high potential. If the number of hops from CV to $S$ is no more than hop count then sink covered the sensor. If the hop count is same those are nodes covered by the sink, otherwise those set to be uncovered nodes. All covered nodes are comes into the $PS$ set.

3.2 K-Partitioned Minimum Depth Tree

K-Partitioned Minimum Depth Tree (k-PMDT) is designed for a sensor network which has multiple sink nodes and a Minimum Depth Tree (MDT) is a tree constructed, that MDT minimizes the cost from each vertex. In k-PMDT, $k$ means the number of sink nodes and it divide the sensor network into $k$ disjoint partitions.

The k-PMDT algorithm is applied on $k$ sink nodes and for every possible combination of $k$ sink node. The set of sink nodes which maximize $V_{min}$ is chosen.
$V_{\text{min}}$ is the minimum volume produced at a sensor node. The k-PMDT algorithm is given below.

**Algorithm: k-PMDT**

1. Start
2. Define
3. For ($\sum_{i=1}^{k} C_i \neq 0$)
4. {
5. For each sensor node in the list
6. {
7. For each sink node
8. {
9. Shortest path is calculated for each sensor node to sink node.
10. }
11. Choose the sink node as a root of the MDT which has shortest path among the all paths to several links
12. }
13. Calculate the $V_{\text{min}}$ for each partitioned MDT using the equation (2)
14. Select the minimal $V_{\text{min}}$ as a K-PMDT $V_{\text{min}}$
15. }
16. Choose the best set of sink nodes which maximizes K-PMDT $V_{\text{min}}$
17. End

In k-PMDT algorithm, shortest path is calculated for each sensor node to sink node in list. For example, if there are $n$ sensor nodes and $k$ sink nodes. The k-PMDT algorithm runs approximately $n\binom{n-1}{k-1}$ times to get the best set of sink nodes.

First calculate the number of children for each sensor node in MDT and then calculate the link cost to parent. The total data volume produced at each sensor node can be calculated from the following [67]
\[ V_{node} = \frac{E_i}{NC \times PR + (NC + 1) \times PT_{child}} \]  

(3.1)

In equation (2),

\[ E_i = \text{Initial energy of sensor node} \]
\[ NC = \text{the number of children} \]
\[ PR = \text{Receiving power consumption per bit} \]
\[ PT = \text{Transmitting power consumption per bit} \]
\[ V_{node} = \text{the total data volume produced at sensor node} \]

The k-PMDT algorithm solves the shortest path problem from each sensor node to a sink node. There are multiple sink nodes in the sensor network, so a sensor node calculates the shortest path to each sink node. Then, the sensor node selects one sink node as a root of the MDT which has the shortest path among the paths to several sink nodes. This process is repeated for every sensor node in the sensor network.

In the given example, MDT is formed using the potential sink nodes. The potential sink nodes are selected using the Optimal Sink Algorithm.

![Figure 3.1 k-PMDT algorithm](image)

3.3 Optimum Search Method

After determining the optimal number of sink positions and routing, the best sink reposition is done by optimum search method.
In the Local Search Approach, $X_0$ is the initial solution and a finite series of solutions $X_i$ is generated with a systematic change of neighborhood. $X_{i+1}$ is derived from $x_i$ such that for all $i$, $f(x_i) > f(x_{i+1})$. $f$ is the evaluation function of the solution. There three levels of transformation to derive the neighborhood of a solution.

3.3.1 One Sink Movement: In this method, only one sink relocated in respect to the initial position. This movement performs in eight directions. North(N), South(S), East(E), West(W), N-E, N-W, SE, S-W.

3.3.2 Two Sinks Movement: In this method, two sinks simultaneous movement to neighborhood of a solution. It avoids the deadlock situation. Using the cardinal points, the transformations are limited. Total 16 movements are possible for the couple of sinks.

3.3.3 Three Sinks Movement: sinks are relocated simultaneously.

Once the optimal sinks positions found, the relocation problem of each sink to its final position big problem. The linear sink movement is costly in terms of energy constraints. Local search does not consider the sink velocity, the distance to travel, and the dynamics of the network. We proposed an approach based on a local search in a constrained space to perform an intelligent movement.

3.3.4 Intelligent Move: In the intelligent move, the sink movements are limited by maintaining their direction to the optimal positions. ld is liberty space and it is based on the current and optimal locations of the sink. “G” is the point located at a distance d from the current position of the sink on the line formed by the current position “cp” and the optimal position of the sink.

Sink new target is defined as the constrained space and a local search is activated in the constrained space using the same principle used to find the optimal position. The constrained optimums are generated by constrained local search to help to perform efficient and power-aware movements towards the optimum position.

Consider Figure 3.3, the sinks are there and each sink contain two nodes. The nodes between the two sinks calculate the ld distance and based on the ld distance it moves the corresponding node. The movement is based on the nearest node and optimum position with less number of moves.
Figure 3.3 Sink Movements
3.4 Total Work Flow

Start

Select the optimal number of sinks using the Optimal Sink Algorithm

K-Partitioned Minimum Depth Tree is constructed using the k-PMDT algorithm

Local Search Approach

Intelligent Move

End

Figure 3.4 Flow chart

3.5 Network Simulator (NS-2)

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks.

NS is an event driven network simulator developed at UC Berkeley that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. The NS project is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats. Currently, NS (version 2) written in C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT) is available.
As shown in Figure 3.5, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object).

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object.

Figure 3.6 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.
3.5.1 Network components in a mobile node

The network stack for a mobile node consists of a link layer(LL), an Address Resolution Protocol (ARP) module connected to LL an interface priority queue(IFq), a Media Access Control (MAC) layer, a network interface(netIF), all connected to the channel. These network components are created and plumbed together in OTcl.

**Link Layer**: The LL used for mobile node has an ARP module connected to it which resolves all IP to hardware (Mac) address conversions. Normally for all outgoing (into the channel) packets, the packets are handed down to the LL by the Routing Agent. The LL hands down packets to the interface queue. For all incoming packets (out of the channel), the mac layer hands up packets to the LL which is then handed off at the node_entry_point.

**ARP**: The Address Resolution Protocol module receives queries from Link layer. If ARP has the hardware address for destination, it writes it into the mac header of the packet. Otherwise it broadcasts an ARP query, and caches the packet temporarily. For each unknown destination hardware address, there is a buffer for a single packet. In case additional packets to the same destination are sent to ARP, the earlier buffered packet is dropped. Once the hardware address of a packet’s next hop is known, the packet is inserted into the interface queue.

**Interface Queue**: The class PriQueue is implemented as a priority queue which gives priority to routing roTOCOL packets, inserting them at the head of the queue. It supports
running a filter over all packets in the queue and removes those with a specified destination address.

**MAC Layer:** The IEEE 802.11 distributed coordination function (DCF) Mac protocol has been implemented by CMU. It uses a RTS/CTS/DATA/ACK pattern for all unicast packets and simply sends out DATA for all broadcast packets. The implementation uses both physical and virtual carrier sense.

**Network Interfaces:** The Network Interface layer serves as a hardware interface which is used by mobile node to access the channel. The wireless shared media interface is implemented as class Phy/WirelessPhy. This interface subject to collisions and the radio propagation model receives packets transmitted by other node interfaces to the channel. The interface stamps each transmitted packet with the meta-data related to the transmitting interface like the transmission power, wavelength etc. This meta-data in pkt header is used by the propagation model in receiving network interface to determine if the packet has minimum power to be received and/or captured and/or detected (carrier sense) by the receiving node. The model approximates the DSSS radio interface (LucentWaveLan direct-sequence spread-spectrum).

**Radio propagation model:** It uses Friss-space attenuation (1/r^2) at near distances and an approximation to Two ray Ground (1/r^4) at far distances. The approximation assumes specular reflection off a flat ground plane.

**Antenna:** An Omni-directional antenna having unity gain is used by mobile nodes.

### 3.6 Simulation results

### 3.6.1 Simulation Model and Parameters

The Network Simulator (NS2) is used to simulate the proposed architecture. In the simulation, the mobile nodes move in a 500 meter x 500 meter region for 50 seconds of simulation time. All nodes have the same transmission range of 250 meters. The simulated traffic is Constant Bit Rate (CBR).
The simulation settings and parameters are summarized in Table 3.1

<table>
<thead>
<tr>
<th>Table 3.1 Simulation Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
</tr>
<tr>
<td>Area Size</td>
</tr>
<tr>
<td>MAC Protocol</td>
</tr>
<tr>
<td>Transmission Range</td>
</tr>
<tr>
<td>Simulation Time</td>
</tr>
<tr>
<td>Traffic Source</td>
</tr>
<tr>
<td>Packet Size</td>
</tr>
<tr>
<td>Initial Energy</td>
</tr>
<tr>
<td>Transmission Power</td>
</tr>
<tr>
<td>Receiving Power</td>
</tr>
<tr>
<td>Rate</td>
</tr>
</tbody>
</table>

### 3.6.2 Performance Metrics

The proposed Multiple Sink Positioning and Relocation (MSPR) is compared with the k-PMDT technique. The performance is evaluated mainly, according to the following metrics.

- **Packet Delivery Ratio**: It is the ratio between the number of packets received and the number of packets sent.
- **Packet Drop**: It refers the average number of packets dropped during the transmission.
- **Residual Energy**: It is the amount of energy in the nodes after data transmission.
- **Delay**: It is the amount of time taken by the nodes to transmit the data packets.

### 3.6.3 Results and Analysis

The number of nodes is varied as 20, 40, 60, 80 and 100. Table 3.2 shows the results of MSPR and KPMĐT algorithms, when the number of nodes is varied.
Table 3.2 Results for Varying the Nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Delay (Sec)</th>
<th>Delivery Ratio</th>
<th>Drop (Packets)</th>
<th>Residual Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSPR</td>
<td>KPMDT</td>
<td>MSPR</td>
<td>KPMDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MSPR</td>
<td>KPMDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MSPR</td>
<td>KPMDT</td>
</tr>
<tr>
<td>20</td>
<td>1.475</td>
<td>3.054</td>
<td>0.9893</td>
<td>2498</td>
</tr>
<tr>
<td>40</td>
<td>1.640</td>
<td>5.291</td>
<td>0.8762</td>
<td>2346</td>
</tr>
<tr>
<td>60</td>
<td>2.629</td>
<td>6.557</td>
<td>0.8285</td>
<td>2627</td>
</tr>
<tr>
<td>80</td>
<td>5.687</td>
<td>6.623</td>
<td>0.5841</td>
<td>4662</td>
</tr>
<tr>
<td>100</td>
<td>5.765</td>
<td>7.262</td>
<td>0.5352</td>
<td>5271</td>
</tr>
</tbody>
</table>

Figure 3.7 Nodes Vs Delay

Figure 3.7 shows the delay occurred for MSPR and KPMDT when the nodes are varied. When the number of nodes is increased from 20 to 100, it will increase the positioning and repositioning delay. As we can see from the figure, the delay of MSPR linearly increases from 1.4 to 5.7 seconds and delay of KPMDT increases from 3.0 to 7.2 seconds. Since MSPR selects the best sink position, the delay is less when compared to KPMDT algorithm.

We can see that, for 20 nodes, the delay of MSPR is 51% lower, for 40 nodes, the delay of MSPR is 68% lower, for 60 nodes, delay of MSPR is 60% lower, for 80 nodes, delay of MSPR is 14% lower and for 100 nodes, delay of MSPR is 20% lower.
than KPMĐT algorithm. Hence the average decrease in delay of MSPR is 42% when compared to KPMĐT approach.

![Nodes Vs Delivery Ratio](image)

**Figure 3.8 Nodes Vs Delivery Ratio**

Figure 3.8 shows the packet delivery ratio measured for MSPR and KPMĐT when the nodes are varied. When the number of nodes is increased from 20 to 100, the coverage and connectivity problem of sink increases, leading to decrease in delivery ratio. As we can see from the figure, the delivery ratio of MSPR linearly decreases from 0.98 to 0.53 and delivery ratio of KPMĐT decreases from 0.81 to 0.44. Since MSPR selects the best sink position, the delivery ratio is high when compared to KPMĐT algorithm.

We can see that, for 20 nodes, the delivery ratio of MSPR is 17% higher, for 40 nodes, the delivery ratio of MSPR is 12% higher, for 60 nodes, delivery ratio of MSPR is 23% higher, for 80 nodes, delivery ratio of MSPR is 20% higher and for 100 nodes, delivery ratio of MSPR is 15% higher than KPMĐT algorithm. Hence the average increase in delivery ratio of MSPR is 17% when compared to KPMĐT approach.
Figure 3.9 Nodes Vs Packet Drop

Figure 3.9 shows the packet drop measured for MSPR and KPMDT when the nodes are varied. When the number of nodes is increased from 20 to 100, the coverage and connectivity problem of sink increases, leading to increase in packet drop. As we can see from the figure, the packet drop of MSPR linearly increases from 1128 to 5271 packets and packet drop of KPMDT increases from 2498 to 11273 packets. Since MSPR selects the best sink position, the packet drop is less when compared to KPMDT algorithm.

We can see that, for 20 nodes, the packet drop of MSPR is 54% lower, for 40 nodes, the packet drop of MSPR is 52% lower, for 60 nodes, packet drop of MSPR is 65% lower, for 80 nodes, packet drop of MSPR is 45% lower and for 100 nodes, packet drop of MSPR is 53% lower than KPMDT algorithm. Hence the average decrease in packet drop of MSPR is 54% when compared to KPMDT approach.
Figure 3.10 Nodes Vs Residual Energy

Figure 3.10 shows the average residual energy measured for MSPR and KPMDT algorithms when the nodes are varied. When the number of nodes is increased from 20 to 100 the residual energy of the node increases. As we can see from the figure, the residual energy of MSPR linearly increases from 14.21 to 16.17 joules and residual energy of KPMDT increases from 13.48 to 15.16 joules. Since MSPR selects the best sink position, the residual energy is higher when compared to KPMDT algorithm.

We can see that, for 20 nodes, the residual energy of MSPR is 5% higher, for 40 node, the residual energy of MSPR is 4% higher, for 60 nodes, the residual energy of MSPR is 7% higher, for 80 nodes, the residual energy of MSPR is 3% higher and for 100 nodes, the residual energy of MSPR is 6% higher than KPMDT. Hence the average increase in residual energy of MSPR is 5% of higher when compared to KPMDT.
Table 3.3 summarizes the percentage wise Improvement of MSPR over KPM DT, when the number of nodes is increased.

Table 3.3 Percentage wise Improvement of MSPR over KPM DT

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Delay (%)</th>
<th>Delivery Ratio (%)</th>
<th>Drop (%)</th>
<th>Residual Energy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>51.6</td>
<td>17.2</td>
<td>54.4</td>
<td>5.1</td>
</tr>
<tr>
<td>40</td>
<td>68.9</td>
<td>12.7</td>
<td>52.5</td>
<td>3.6</td>
</tr>
<tr>
<td>60</td>
<td>59.9</td>
<td>23.4</td>
<td>65.0</td>
<td>6.7</td>
</tr>
<tr>
<td>80</td>
<td>14.1</td>
<td>20.3</td>
<td>49.0</td>
<td>3.1</td>
</tr>
<tr>
<td>100</td>
<td>20.6</td>
<td>15.9</td>
<td>53.2</td>
<td>6.2</td>
</tr>
</tbody>
</table>

K -Partitioned Minimum Depth Tree using optimal search is proposed for wireless sensor networks for optimal sink positioning. Initially the optimal number of sinks is determined using the optimal sink algorithm satisfying the h-hop constraint. Then a K-Partitioned Minimum Depth Tree (k-PMDT) is constructed for positioning multiple sink nodes and setting up the routes. After determining the optimal number of sink positions and routing, best sink reposition is selected by optimum search method. Sink movement is done by using the intelligent movement and it limit the sinks movements while maintaining their direction to the optimal positions.

The main advantage of this method is, using of node life time in the construction of tree the tree lifetime will be improved and the optimal numbers of sinks are placed in sensor network for improving the network lifetime. By using NS2 simulation results, it is shown that MPSR outperforms KPM DT by 43% in terms of delay, by 18% in terms of delivery ratio, by 55% in terms of packet drop and by 5% in terms of residual energy when the number of nodes is increased.