Efficient Data Retrieval In Faulty Sensor Networks Using A Mobile Mule

Harel Yedidsion*, Aritra Banik†, Paz Carmi*, Matthew J. Katz* and Michael Segal*
* Ben-Gurion University of the Negev, Beer-Sheva, Israel
† Indian Institute of Technology, Jodhpur, India
Email: yedidsio@bgu.ac.il, aritra@iitj.ac.in, carmip@cs.bgu.ac.il, matya@cs.bgu.ac.il, segal@bgu.ac.il

***Abstract*—In this paper, we study the problem of data gathering in ad-hoc sensor networks using a mobile entity called *mule*. The mule traverses the children of failed sensors, to prevent loss of data. Our objective is to define the optimal communication tree and the mule's placement such that the mule's overall traveling distance is minimized. We explore this problem in several network topologies including: unit disc graph on a line (UDL), general unit disc graph (UDG), and a complete graph with failing probabilities on the nodes (CGFP). We provide an optimal solution for the UDL problem and two approximation algorithms for the UDG problem. For the CGFP problem we outline the two possible structures of an optimal solution and provide near optimal approximation algorithms.**

I. Introduction

A wireless ad-hoc sensor network consists of several sensors with communication capabilities. The network's topology is determined by the locations and transmission ranges of the sensors. The ranges determine a directed communication graph, in which the nodes correspond to the sensors and the edges correspond to the communication links. A standard operation for such a network has the sensors monitor their surroundings, and deliver the information to a base station using multi-hop communication. A failure in one node may cause data loss from its neighbors in the graph. A mobile mule may prevent this lose by visiting and collecting data from neighboring nodes. In this paper we investigate the optimal design of the network and the placement of the data mule as to minimize its overall traveling distance under additional constraints.

A. Problem definition

Formally, the problem is defined as follows: Let $G = (V, E)$ be a graph embedded in the Euclidean plane, where V is the set of wireless nodes, $|V| = n$, and E is the set of undirected edges between nodes. Let T be a directed data gathering tree rooted at root r, spanning the nodes in V, where data propagates from leaf nodes to r. Consider a mobile entity with wireless capabilities called *mule*, that visits a subset of nodes $V_m \subseteq V$ and collects the information they sense. The mule can travel anywhere in the plane. Let c_d be the mule's cost of traversing distance d. This cost is proportional to the distance traveled. If a sensor v fails, it is undesirable to lose the data it collected from its children in T, $\delta(v, T)$. Thus, the mule must travel through $\delta(v, T)$ and restore the lost information. The mule's tour through the children of node v is defined as $t(m, \delta(v, T))$ where m represents the placement of the mule. The goal is to find a data gathering tree T and the placement of the mule m which minimize the total traveling distance given that any sensor can fail. The mule can only be positioned on existing nodes. The mule returns to its initial position after every failure tour. Formally, the objective is to have: $\min_{T,r,m} \sum_{v \in V} |t(m, \delta(v, T))|$. When node v's failing probability is given as p_v then the objective becomes $\min_{T,r,m} \sum_{v \in V} |t(m, \delta(v, T))| \cdot p_v$.

II. Related Work

The uses of mobile elements in wireless sensor networks (WSN) include improving connectivity, cost, reliability and energy efficiency [5]. A number of papers have explored the use of mobile elements as a solution to the problem of data gathering in WSN. Crowcroft et al. [2] define the (α, β)-Mule problem, where α is the number of simultaneous node failures and β is the number of traveling mules. They solve this problem under different network topologies such as arbitrary graphs, grids and random linear networks and provide optimal solutions and approximation algorithms. Most relevant to our work is their 4-approximation of the UDL mule problem for which we provide an optimal solution. In [6], Levin et al. study the trade-off between the mules' traveling distance and the amount of information uncertainty caused by not visiting a subset of nodes by the mules. The authors of [8] utilize autonomous mobile base stations (MBSs) to automatically construct new routes to recover disconnected infrastructure. In contrast to our work, it is assumed that MBSs are able to stop at any position in the network area while we only consider discrete node locations. The same assumption is used in [1] where mobile backbone nodes (MBNs) are controlled in order to maintain network connectivity while minimizing the number of MBNs that are actually deployed. In [9] a single mule is scheduled to visit the nodes to collect their data before their buffers are full. Sensor nodes are sampling at different rates, in which case some nodes need to be visited more frequently than others. In [10], the k-TSP approach is used to plan the data collection routes of k mules. TSP with neighborhood was applied for the same purpose in [3].

III. Unit Disc Line

In this section, we solve the mule problem on a one dimensional line topology, common in roadway and waterway monitoring, where n nodes are placed on a line such that the maximum distance between adjacent nodes is 1. The communication model is unit disc, which means that an edge is formed between two nodes u, v if and only if $d(u, v) \leq 1$. Note that this implies that the graph is connected. We call this model of unit disc on a line - UDL. Previous work has found a 4-approximation for this problem [2]. In what follows we outline the characteristics of an optimal solution and present a linear time algorithm that finds such a solution. In general, the optimal solution would have the root at either one of the endpoint nodes or at one of the endpoint backbone nodes (see Definition 1). The mule would be placed at the central interval (see Definition 2). Specifically, the tree built by Algorithm 1 produces an optimal solution for the UDL problem.

Definition 1: A backbone node (BBN) is a node that has children in T.

Definition 2: Intervals are the segments of the line that lie between adjacent BBNs or between an endpoint node and an endpoint BBN.

Definition 3: A One-Sided Tree (OST) is a rooted tree connecting nodes on a line where except for the root, BBNs have children only on the side of the BBN which is farther from the root.

Definition 4: A Two-Sided Tree (TST) is a rooted tree connecting nodes on a line where BBNs may have children on both sides of the BBN.

Claim 1: There exists an optimal solution where T is a OST, i.e. the children of every BBN in T are located on the side of the BBN which is farther from the root.

Proof: In proving this claim we apply a constructive method of transforming any tree to OST while maintaining the solution quality. We show that given a rooted tree T which is not an OST and v_k is a BBN in T (other than the root) which has children on both sides, then the tree can be transformed to another tree T, in which a child of v_k on the side of v_k that is closer to the root is made a child of another BBN v_i, and, if v_i is not the root, then its new child is on the side of v_i which is farther from the root. Moreover, the solution cost of T is not more than that of T. This transformation is repeated again and again until the current tree is an OST.

Let nodes v_i and v_k be BBNs in T. We look at two possible configurations for T in Figure 1 where the OST and TST are shown on top and bottom respectively. Node v_j is the closest node to the root in the interval between v_i and v_k. None v_l is the farthest child from the root of v_k. In TST, v_j is connected to v_k and in OST it is connected to v_i. Other nodes may exit in between v_j and v_k and we assume that in OST they are all connected to v_i. Directed edges are drawn as arrows pointing from son to father to reflect the direction of the data flow.

We will show that the OST solution is as good as the TST solution, independent of the location of the mule. Let the distance between v_i and v_j be a, the distance between v_j and v_k be b and the distance between v_k and v_l be c. There can be three cases depending on the location of the mule. The tours to children of nodes other than v_i and v_k are not effected so we only focus on the tours created by failures of v_i and v_k.

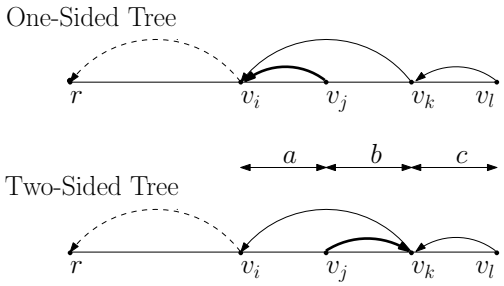

Fig. 1. Illustration of an OST (top) where BBNs only have children on the side of the BBN farther from the root, and TST (bottom) where BBNs have children on both sides i.e., v_j is the son of v_k.

(*i*) $m \in (-\infty, v_j]$. W.l.o.g, assume that the mule is in v_j otherwise we can add the distance $2 \cdot (m, v_j)$ to every tour. In both trees, if v_i fails, the mule must travel to v_k and back and if v_k fails, to v_l and back. In this case, the total cost is $4b + 2c$ for both solutions. However, if v_k does not have any children to the right then if v_k fails, in TST the mule would travel almost to v_k, but in OST no movement is required. Thus, in this case, the cost of OST is smaller or equal to that of TST.

(*ii*) $m \in (v_j, v_k)$. W.l.o.g, assume that the mule is in v_k. In OST, if v_i fails, the mule must travel to v_j and back and if v_k fails then the mule must travel to v_l and back. In TST, if v_i fails, the mule does not move and if v_k fails then the mule must travel to v_j and back and to v_l and back. The cost in both solutions is $2b + 2c$.

(*iii*) $m \in (v_k, v_l)$. W.l.o.g, assume that the mule is in v_l. In OST, if v_i fails, the mule must travel to v_j and back and if v_k fails then the mule must travel almost to v_k and back. In TST, if v_i fails, must travel to v_k and back, and if v_k fails then the mule must travel to v_j and back. Thus, in this case, the cost of OST is smaller than that of TST.

Note that if v_i has children in (v_i, v_j), the analysis does not change. Hence, independently of the location of the mule, the OST solution is as good as that of the TST. The claim holds. ■

Claim 2: There exists an optimal solution for which there is no crossing of edges.

Proof: A crossing occurs when there exist edges (v_i, v_k) and (v_j, v_l), and the nodes are set in the following order $v_i > v_j > v_k > v_l$ or the reversed order. Figure 2 presents the four possible options of edge crossing in the UDL topology as depicted in Cases 1-4. The case on the botom termed NC displays a non-crossing tree on the same node placement. We demonstrate that a tree with no crossing edges provides a solution for the UDL mule problem that is as good as that of any tree with crossing edges regardless of the location of the root and the mule. In the tree with no crossing edges, NC,

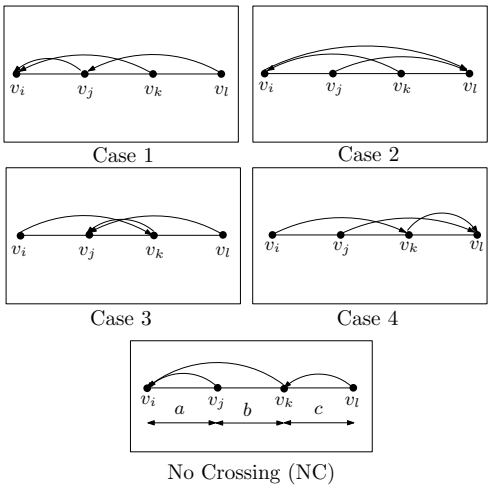

Fig. 2. Four possible crossings of edges. Illustration of Claim 2.

nodes v_j and v_k are children of v_i and node v_l is a child of v_k. We only focus on proving Case 1 for each root and mule locations. The analysis for Case 4 is the same as that of Case 1. Regarding Cases 2 and 3, any tree built using these edges would necessarily have a node with children on both sides. Since we have already established in Claim 1 that a one sided tree is as good as a two-sided tree, we can disregard these two cases. Following the same logic we can ignore any root location other than at node v_i (or to the left of it) in Case 1, since it would also cause a two sided tree. Therefore, w.l.o.g, we assume that the root is at node v_i and calculate the solution value for each mule location while partitioning the line to three areas $(i) - (iii)$:

(*i*) $m \in (-\infty, v_j]$. W.l.o.g, assume that the mule is in v_j. In this case in both NC and Case 1, the cost is $2(2b + c)$.

(*ii*) $m \in (v_j, v_k]$. W.l.o.g, assume that the mule is in v_k. In this case in both NC and Case 1, the cost is $2(b + c)$.

(*iii*) $m \in (v_l, +\infty)$. W.l.o.g, assume that the mule is in v_l. In this case in both NC and Case 1, the cost is $2(b + c)$.

Hence, independently of the location of the mule, a tree with no crossing edges provides a solution that is as good as any solution by a tree with crossing edges. Consequently, the claim holds. ■

At this point we have narrowed the possible solution trees only to the trees where children nodes are on the side farther away from the root and have no crossing edges. But, there is still an exponential number of possibilities to build such trees.

A. Algorithm UDL

We now introduce our algorithm for building a tree with these properties and show that this tree provides an optimal solution for the UDL problem given r. The tree is built using an iterative process. In the first iteration, the node farthest from the endpoint node within one unit distance is added to the tree as a BBN. In the next iteration the node farthest from that BBN within one unit distance is added to the tree as the next BBN and so on. The process is performed from both sides alternately until the two sub-trees meet in the middle. The middle point is also added as BBN. The mule is placed on the central BBN as explained in the proof of Claim 4. The location of the root is determined as described in Claim 5 and the nodes between the BBNs are added as children according to Claim 1.

Figure 3 shows a tree (T_1) built by Algorithm 1 over a set of n nodes on a line. Nodes $v_1, v_2 ... v_k$ are BBNs. The intervals between BBNs are marked by the index of the higher BBN. The mule is placed at the $\lfloor \frac{k}{2} \rfloor^{th}$ interval and the root is at the k^{th} BBN.

Definition 5: The distance between the farthest child of BBN_i to BBN_{i-1} is noted by I_i.

The sum of all the mule tours can be summed up by the following Equation:

$$(1) \qquad \sum_{v \in V} |t(m, \delta(v, T))| = 2\Bigg(\sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} Interval_i \cdot i + \sum_{i=\lfloor \frac{k}{2} \rfloor + 1}^{k-1} Interval_i \cdot (k - i) + \sum_{i=\lfloor \frac{k}{2} \rfloor + 1}^{k} I_i \Bigg).$$

The equation is comprised of three parts: the first sum represents the tours to the left of the mule, the second and third sums represent the tours to the right of the mule. The tours are multiplied by 2 to account for the return of the mule to its initial location.

Claim 3: Assuming that the mule is at the $\lfloor \frac{k}{2} \rfloor^{th}$ BBN and the root is at the k^{th} BBN, a tree built by Algorithm 1 provides an optimal solution for the UDL mule problem.

Proof: Let T_1 be a tree built by Algorithm 1. Tree T_1 is depicted by solid lines in Figure 3. In order to prove that T_1 provides as good a solution as any other tree given r and m, we consider tree T_0, whose edges are marked by dashes lines in Figure 3. Tree T_0 represents any other tree over the same set of n nodes. Since Algorithm 1 chooses the farthest node within 1 unit from the left-most node to be the first BBN, and so on, any other tree would have to choose either the same BBNs as in T_1 or other nodes farther from the mule to be its BBNs. As a result, all BBNs in T_0 (see nodes in Figure 3 denoted by $v*$) would be either on the same nodes as T_1 or on nodes which are farther away from the mule than their corresponding BBNs in T_1. This leads to a situation where each tour by the mule to such a BBN on T_1 is shorter or equal to the corresponding tour in T_0. Note that any solution with more BBNs than in T_1 produces a lower quality solution as there are more tours to traverse. Hence, the solution produced by T_1 is as good as any that of any other tree given the root and mule locations. Therefore, the claim holds. ■

Claim 4: The optimal position for the mule is at the $\lfloor \frac{k}{2} \rfloor^{th}$ BBN.

Proof: In this proof we show that any placement of the mule other than the central interval, results in an increase the number of times that the central interval is traversed. At

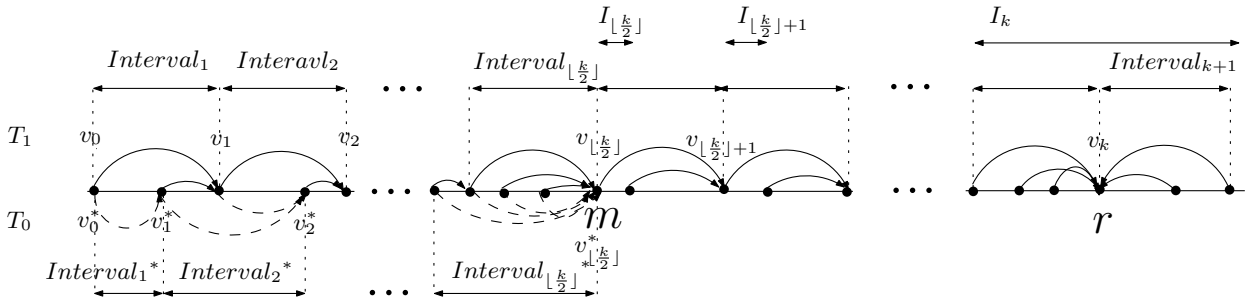

Fig. 3. A comparison of tree T_1 built by Algorithm 1 and marked by solid lines to an arbitrary tree T_0 marked by dashed lines.

Algorithm 1: Algorithm for building an optimal tree for the UDL mule problem

Data: A set of n nodes on a line where the maximal distance between two adjacent nodes is 1 unit. A root node r.

Result: A tree T over the given n nodes rooted at r where all edges are at a length of at most 1 unit.

1 Start from the endpoint nodes on both sides and iteratively add the farthest node within 1 unit distance as BBN until the two opposing BBNs connect;
2 Add the connecting node as BBN;
3 Place m on the central BBN;
4 Connect BBNs and remaining nodes by directed edges according to the location of r in OST fashion;
5 Emit T;

the $\lfloor \frac{k}{2} \rfloor^{th}$ BBN, the mule has $\lfloor \frac{k}{2} \rfloor - 1$ tours to the left and $k - \lfloor \frac{k}{2} \rfloor - 1$ tours to the right. In addition, the mule must travel to $I_{\lfloor \frac{k}{2} \rfloor}$. According to Equation 1, any shift in the placement of the mule may only worsen the solution quality. Assume that the mule is moved from the $\lfloor \frac{k}{2} \rfloor^{th}$ BBN to the BBN to the left of it. In this case, the $\lfloor \frac{k}{2} \rfloor^{th}$ interval would be removed from $\lfloor \frac{k}{2} \rfloor - 1$ tours but added to $k - 1 - \lfloor \frac{k}{2} \rfloor$ tours. In addition, a tour to $I_{\lfloor \frac{k}{2} \rfloor}$ is added. Any further move to the left of distance d would removed this distance from less tours than it is added to. The same thing happens if the mule is moved to the right. In this case, the $\lfloor \frac{k}{2} \rfloor + 1^{th}$ interval would be added to $\lfloor \frac{k}{2} \rfloor$ tours but removed from $k - 1 - \lfloor \frac{k}{2} \rfloor$ and in addition, a distance of the length $Interval_{\lfloor \frac{k}{2} \rfloor} - I_{\lfloor \frac{k}{2} \rfloor}$ is added. Any further move to the right of distance d would removed this distance from less tours than it is added to. Hence, the claim holds.

■

We can run this algorithm for every one of the n nodes as the root, and choose the one that has the best solution. However, we have identified that the optimal locations for the root are either on the endpoint nodes or at one of the nodes which are the farthest from the endpoint nodes within 1 unit.

Observation 1: The length of any two adjacent intervals (not including the central $\lfloor \frac{k}{2} \rfloor^{th}$ interval) is greater than 1 unit distance.

Proof: According to Algorithm 1, each BBN is chosen as the farthest node from the current BBN within 1 unit of distance. If two adjacent intervals are smaller than 1 unit, then the algorithm would have considered them as 1 interval and this is a contradiction. Hence, the claim holds. ■

Claim 5: If there are less than five intervals, then the optimal position for the root is at one of the endpoint nodes or at one of the nodes which are the farthest from the endpoint nodes within 1 unit. Otherwise, if there are five or more intervals, the optimal position for the root is at one of the nodes which are the farthest from the endpoint nodes within 1 unit.

Proof:

(*i*) ($|intervals| < 5$). In this case, there are 4 possible placements for the root. Placing the root at one of the endpoint nodes, saves the distance from it to its closest child. This tour can be as small as $\epsilon \to 0$ or as large as 1 unit. Placing the root at one of the nodes which are the farthest from the endpoint nodes within 1 unit, saves the tour to the closest child of that node. This tour can be as small as ϵ or as large as 1 unit. In addition, this placements adds the tour from the endpoint node to its closest child, which can be as small as ϵ or as large as 1 unit. Any other placement (i.e., the central BBN) provides no additional savings.

(*i*) ($|intervals| \geq 5$). In this case, the placement of the root on one of the nodes which are the farthest from the endpoint nodes within 1 unit, saves the tour to that node. If there are five or more intervals, that means that this tour contains at least two adjacent intervals and according to Observation 1 is greater than 1 unit distance. On the other hand, this placement adds the distance between the endpoint node and the next closest node which is at most 1 unit according to the assumption of the model that the nodes are connected. This means that the movement of the root from the endpoint node to one of the nodes which are the farthest from the endpoint nodes within 1 unit, provides a better solution in any case where there are five or more intervals.

■

Claim 6: A tree built by Algorithm 1 provides an optimal solution for the UDL mule problem.

Proof: Algorithm 1 produces a tree T and mule and root placements which adhere to all the characteristics of an optimal solution to the UDL mule problem as described by Claims 1-6 and therefore, the claim holds. ■

The complexity of running Algorithm 1 is $O(n)$ since all of the operations are linear in the number of nodes.

IV. Unit Disc Graph

In this section, we solve the mule problem on a two dimensional plane. The communication model is unit disc, which means that an edge is formed between two nodes u, v if and only if $d(u, v) \leq 1$. We assume that the graph is connected. We call this model of unit disc graph - UDG. The analysis of our solution to this problem contains two parts. In each part we present a different trade-off between the solution's runtime and its approximation bound.

A. First part of the analysis

In this part we assume that the root and the mule locations are given (alternatively, we can run the algorithm below for each one of the n^2 combinations of root and mule placements, and find the combination providing the best solution).

Algorithm UDG1

Our approach for the tree construction includes finding a special minimal connected dominating set (MCDS) and connecting all remaining nodes as leaves to the closest node of the MCDS (in such a way that the number of nodes in the MCDS in a given area is limited). The connected dominating set problem is defined as follows. Find a subset S of nodes in graph G, such that the subgraph induced by S is connected, and each node is either in S, or adjacent to some node in S. The set has to be minimal is a sense that any removal of a node from S would cause at least one of the previous requirements to be violated. The tree T contains the MCDS and the rest of the nodes as leaves.

In order to bound the performance ratio of Algorithm UDG1, we introduce Algorithm UDG2 for which we can find a constant bound to the cost of the optimal solution, OPT, and show that any solution by Algorithm UDG1 has a lower or equal cost to any solution by Algorithm UDG2, thus establishing a constant approximation for Algorithm UDG1 as well. Note that OPT represents the (unknown) optimal solution given the selected root and mule locations. Since we iterate over all possible mule/root locations, and choose the best one, it necessarily maintains a constant bound from the overall optimal solution.

Definition 6: A backbone node (BBN) is a node that has children in T. Let q be a BBN in the tree built by Algorithm UDG1. We denote by $c(q)$ the disc centered at q with radius 1, and by $OPT(q)$, the subset of nodes which have a child in $c(q)$ according to the optimal tree, thus $OPT(q) = \{p : (p, v) \in OPT \ \wedge \ v \in c(q)\}$.

Let $OPT(q)^*$ be an extended subset of nodes which includes all of the children of the nodes in $OPT(q)$ according to the optimal solution, i.e., $OPT(q)^* = \{x : (x, p) \in OPT \ \wedge \ p \in OPT(q))\}$, where (x, p) is a directed edge from x to p. Nodes p, v and x represent arbitrary nodes in G. Figure 4 demonstrates these concepts. Node q is in the center of the dashed circle $c(q)$. The nodes marked with rings are members of $OPT(q)$, and the nodes marked with Astrix are members of $OPT(q)^*$.

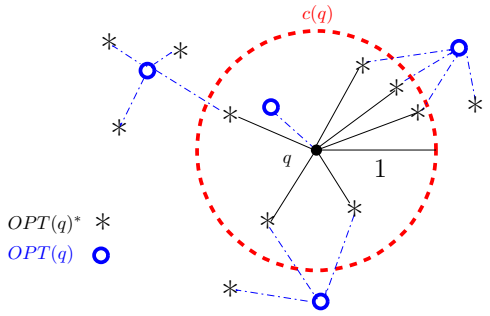

Fig. 4. Graphical visualization of the concepts from Definition 6.

Algorithm UDG2

The tree construction method is detailed below and is identical in both Algorithm UDG1 and UDG2. However, in case of a failure of a node in S, each algorithm uses a different traveling scheme. In Algorithm UDG2, instead of only visiting the children of the failed node q, we visit $OPT(q)^*$ which includes all the children of all the BBNs in OPT that have a child node within unit distance from the failed node. This method aids us in approximating the solution of Algorithm UDG1 compared to the optimal solution. We claim that the number of times a node may be visited by Algorithm UDG2 is constant compared to the number of times it would be visited by the optimal solution. In addition, we show that the solution cost by Algorithm UDG2 is always greater or equal to that of Algorithm UDG1 so this constant bound applies to the solution of Algorithm UDG1 as well.

Tree Construction

Since not every MCDS has the property that the number of its members bounded by a constant within a certain area, we introduce the new method for building an Area Constrained MCDS (ACMCDS). We first construct a dominating set (DS). The procedure is performed iteratively by choosing at each step a node that is not in the current DS and is not within any unit disc of the current members of the DS. The chosen node is added to the DS until no more nodes can be added. At this point we have a dominating set which is not connected. The process of finding the DS takes $O(n^2)$ time. The algorithm continues to connect the DS by adding at most two interconnecting nodes for each DS member. The process of connecting the DS takes $O(n^2)$ time. This can be achieved by first finding for each node its neighbors in the UDG (i.e., within 1 unit) and neighboring DS members in $O(n^2)$ time. The number of neighboring DS members of each node is constant since each node can have at most 5 disjoint neighboring dominating nodes. Each pair of DS members can be connected via one or two other nodes at most according to Observation 3, and no

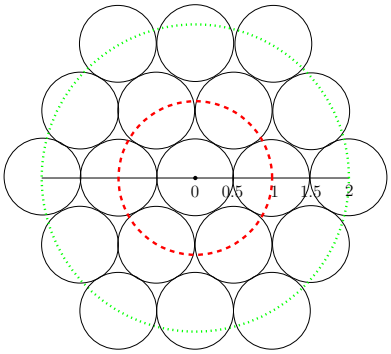

Fig. 5. The maximal number of half radius circles in a double radius circle.

pair can be directly connected. We first go over all the nodes and connect each pair of DS members that can be connected via one node. Then, for each pair of nodes (n^2 pairs), we make a connection if they connect two DS members in $O(n^2)$ time. We then prune the resulting graph to eliminate cycles in $O(n)$ time using BFS. Once the set is connected, the rest of the nodes are connected to the closest member of the ACMCDS to form the data gathering tree T.

To prove that our solution maintains a constant approximation to the optimal solution, we claim that any arbitrary node in the graph can be within a distance of 2 units from a constant number of nodes from the ACMCDS.

Definition 7: Two nodes x, v are considered neighbors if their unit discs overlap, or in other words, the distance between them is less or equal to 2, i.e., $|(x, v)| \leq 2$.

Observation 2: Any node in graph G can be a neighbor of at most 19 nodes of the dominating set built by Algorithm UDG1. Note that the node itself may by part of the DS.

Proof: Our algorithm for building the DS ensures that there is a constant number of member nodes of the ACMCDS that can be neighbors of any other node in the graph. Since we are considering two virtual trees over the same set of nodes, it is possible for a node in one tree to be a neighbor of the same node in another tree. What we are looking for is how many nodes in the tree we are building can have in their unit disc children of a node in the optimal tree. So we do consider the node itself when counting the number of neighboring nodes. We prove it by considering a circle of radius 2 centered around an arbitrary node v in the graph. Only nodes within the double radius circle are neighbors of v. The DS is built in such a way that no two nodes are contained in each other's unit disc. If we reduce the radius of each unit disc by half, then we can assure that no two discs overlap at all. The area of the double radius disc can accommodate at most 19 centers of non overlapping half radius discs according to [4]. Figure 5 demonstrates this extreme situation where 19 centers of non-overlapping half-radius circles fit into a double radius circle.

■

Observation 3: Any node in the DS built by algorithm UDG1 can be connected to one of its neighboring nodes in the DS by two other nodes.

Proof: The premise of this work is that the graph is connected. Hence, any node v in the DS has at least one other node x such that $d(v, x) \leq 1$. If x is within one unit from one of v's neighbors from the DS, then the connection between v and this node can be made via x. If not, at least one node in v's unit disc must have a connection with at least one node outside of v's unit disc to maintain the connectivity of the graph. The outer node must be connected to another member of the DS. In this case it would take two nodes to connect the two neighboring DS members. ■

Claim 7: The solution of Algorithm UDG2 would visit the nodes in OPT at most 57 times.

Proof: We have established in Observation 2 that there can be at most 19 neighboring DS members to any node in G, and in Observation 3 that at most two nodes are required to connect a node from the DS to one of its neighbors in the DS to form the ACMCDS. Hence, there can be potentially at most 57 nodes in the ACMCDS that neighbor any node in G. According to Algorithm UDG2, in case of a failure of a node in the ACMCDS, all the children of all the neighboring BBNs in OPT are visited. Therefore, any node that is visited by the optimal solution would be visited at most 57 times by Algorithm UDG2. ■

A practical approach for calculating the TSP tours may use a heuristic algorithm for this procedure, for example the $1+\epsilon$ TSP approximation method that runs in $O(n \log n)$ time for each of the n nodes [7].

Since we are using a $1+\epsilon$ approximation algorithm for the traversal route as described in [7], the final approximation ratio is $57+\epsilon$. According to Algorithm UDG1, in case of a failed node in the ACMCDS, only its children are visited. These children are included in the nodes visited by Algorithm UDG2. Therefore, any approximation ratio achieved by Algorithm UDG2 also applies to Algorithm UDG1.

The computational complexity of Algorithm UDG1 is comprised of going over n^2 combinations of root and mule where n is the number of nodes in G. For every root position, the direction of the edges is determined in $O(n)$. In addition, for each combination, the TSP tours must be calculated for at most K tours (where K is the number of BBNs in the ACMCDS). Each tour containing k_i nodes, $i \in \{1...K\}$. Notice that the sum of all nodes in all the tours equals $n + K$ since every tour also includes the mule's node, i.e., $\sum_i k_i = n + K$. The $1 + \epsilon$ approximation for the TSP route of tour k_i is calculated in $O(k_i \log k_i)$ time according to [7]. As a result, calculating K TSP routes for all the tours can be done in $O(\sum_i (k_i \log k_i) \leq O(\sum_i (k_i \log n) \leq O(n \log n)$. The DS is built once in $O(n^2)$ and connected in $O(n^2)$ time and does not effect the overall complexity. So, the final complexity of our first approach is $O(n^3 \log n)$ achieving a $57+\epsilon$ approximation.

B. Second part Of the analysis

In this part of the analysis, in order to speed up the runtime, we run the same tree building method used in the first approach but we relax the assumption that the root and the mule locations are given. Instead of checking all possible n^2

position combinations, we only check n combinations in which the root is located at the position of the mule. We show that by locating the root at the mule's position, the approximation ratio is doubled. However, this approximation only holds for the case that the tree built by algorithm UDG1 has more than two BBNs. For the case that there are two or less BBNs, the solution quality cannot be bounded since in this case the tour could be infinitely small compared to the distance that the root is moved by. Therefore, in applying our approach, we distinguish between two cases. For the case that there are two or less BBNs we develop a near optimal solution based on the optimal solution presented in [2] to the mule problem on a fully connected graph. In order to find the diameter of the tree we simply count the number of BBNs chosen in the process of building the tree.

Observation 4: If the diameter of the graph is greater than 2, by moving the root from its original position to the position of the mule, the distance that the mule travels would increase by at most a factor of 2.

Proof: By moving the root r from its original position r_o to the position of the mule m, the only mule's tours that are effected are the ones on the direct path between the mule and r_o. We denote this path $P_{r_o,m}$. All the other tours remain the same and are not effected by the move since their direction of the edges and their tree structure remains the same. On $P_{r_o,m}$ however, the direction of the edges changes as does the tree structure. When r is moved from r_o to a neighboring child x, the directed edge from x to r_o reverses its direction and r_o becomes a child of x. This implies that when x fails, the mule must travel to r_o in addition to all of x's original children. Thus the tour is increased by the length of $d(x, r_o)$. This is true for every edge in $P_{r_o,m}$ and as a consequence, the path $P_{r_o,m}$ is added to the original mule's tours. In case there are more than two BBNs, this path can be as long as the entire original mule's tour in the worst case, so we can approximate this addition as doubling the original tour. ■

Claim 8: If the diameter of T is less than or equal to 2, the optimal solution for the UDG mule problem consists of a tree with a star formation. In this case we get a $(1+\epsilon)$ approximation of the optimal solution.

Proof: If the diameter of T is less than or equal to 2, we create a star formation. This formation has been proven to be optimal for a fully connected graph in [2]. As in the fully connected graph, the optimal place for the root, is the node whose distance to its closest neighbor is the largest. This location minimizes the mules tour since this root's node is never visited by the mule. Unlike the fully connected graph where all the other nodes can be connected to the root, in UDG the center of the star is a node which can be within 1 unit distance at most from all the other nodes. Our assumption on the connectivity of the graph implies that such a node exists. The mule is located at the center of the star so that if the root fails no movement is required and if the central node fails, a traversal of all the other nodes except the root is required. The solution we get is a $(1+\epsilon)$ approximation of the optimal solution due to the TSP heuristic. ■

Claim 9: If the diameter of T is greater than 2, the solution that our second approach achieves has an approximation ratio of $(114+\epsilon)\cdot OPT$.

Proof: Following Observation 4 we can now argue that the approximation ratio for the second approach is double that of the first approach and equals $(114+\epsilon)\cdot OPT$. However, the runtime of this approach is improved to $O(n^2 \log n)$. ■

V. Complete Graph with Failing Probabilities

In this section we assume that each node has a weight associated with its probability of failing. Here too the mule can be placed on one of the nodes and it must visit the children of any failing node. Unlike the previous sections, here the underlying topology of the graph from which we choose the data gathering tree is a complete graph with different failing probabilities for each node (CGFP). Our goal is to find the optimal topology for the directed data gathering tree and the root and mule placements as to minimize the sum of all possible tours. We note by p_v the probability (weight) of node v failing. Formally, the goal function is:

$\min_{T,r,m} \sum_{v \in V} |t(m, \delta(v, T))| \cdot p_v$.

Previous work on a similar problem with a complete graph in which all the nodes have the same probability of failing showed that the optimal topology of T is a star [2]. We show that this is not always the case in CGFP by providing a counter example. We further illustrate the two possible optimal configurations for the mule problem in CGFP.

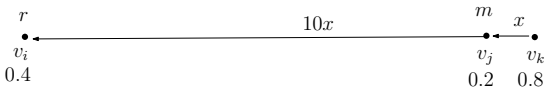

Fig. 6. An example of an optimal data gathering tree which is not a star in a complete graph with failing probabilities.

Figure 6 presents an example of a data gathering tree created from a CGFP with three nodes v_i, v_j and v_k,. The nodes weights are the numbers displayed near the nodes and the distances between nodes are displayed on the edges. The root is located at node v_i and the mule is located at node v_j. If node v_i fails, the mule must visit node v_j, and if node v_j fails, the mule must visit node v_k. Node v_k has no children so there is no need to visit any other nodes in case it fails. Since the mule is located at v_j, the objective function equals $0.4 \cdot 0 + 0.2 \cdot x + 0.8 \cdot 0 = 0.2x$. In this case, if we move to a star-formation, where v_k is directly linked to v_i, the objective function would increase to $0.4 \cdot x + 0.2 \cdot 0 + 0.8 \cdot 0 = 0.4x$. Note that in a complete graph with equal probability of failing both solutions would yield the value of x. Any other root-mule configuration would yield a higher cost solution.

Definition 8: Denote by $t(V \backslash v_i)$ the tour of all the nodes in V except for node v_i.

Denote by $v_{T_{min}}$ the node for which $t(V \backslash v_i)$ is the minimal length tour of all nodes $v_i \in V, i \in 1...n$.

Denote by $v_{P_{min}}$ the node with the minimal weight.

Claim 10: A tree that has the following properties along with the stated mule and root placements presents an optimal solution for the CGFP mule problem:

1) The root should be the node $v_{T_{min}}$.
2) The tree should have a star configuration. The central node would be $v_{P_{min}}$ having the smallest probability $\min_{v \in V} P_v$.
3) If the optimal node for the root $v_{T_{min}}$ also happens to be the node with the smallest weight, $v_{P_{min}}$ then the root would be placed in the center of the star and the mule would be placed on one of the other nodes. Otherwise, if $v_{T_{min}}$ does not have the smallest weight, the mule would be placed at $v_{P_{min}}$ in the center of the star.

Proof: Regarding the root location, in CGFP, as in the equal probability case, each node except for the root must be visited at least once. Therefore, the root must be located at the node that minimizes the mules traversal to all other nodes except that node and that is $v_{T_{min}}$.

Regarding the shape of the tree, every tour starts from the mule's initial location and the mule must return to this location. In order to minimize the number of travels from there and back, we can minimize the number of tours to only one by shaping the tree in a form of a star. It has been shown in the fully connected graph with equal probabilities that, to minimize the mule's tour, T should have the shape of a star. Furthermore, in CGFP we also have to make sure that the probability of the tour being taken is also minimized. This is done by choosing the node with the smallest weight as the center of the star.

To prove this notion, for contradiction, consider a tree $\hat{T}$ where not all the nodes are children of $v_{P_{min}}$. For simplicity, assume there is a node n_i, that is a child of $v_{P_{min}}$ that also has children in $\hat{T}$. We show that tree T in which all the nodes are children of $v_{P_{min}}$ has a better solution than $\hat{T}$. The solution to $\hat{T}$ includes two tours $p_i(|t(m, \delta(n_i, T)|) + P_{min}(|t(m, \delta(v_{P_{min}}, T)|)$ while in T the solution only includes one tour $P_{min}(|t(m, \delta(v_{P_{min}}, T)|)$. Since the two TSP tours of $\hat{T}$ both include m, these tours can be merged into one as in T, by eliminating one movement from and back to m and thus shortening the total traveled distance. In addition, the probability of taking the tour does not increase since $P_{min} \leq p_i$. Thus, the solution of T is smaller than $\hat{T}$.

Finally, We distinguish between two cases:

(a) $v_{T_{min}} = v_{P_{min}}$ In this case, the root is also the center of the star so we would choose the mule's placement at one of the leaf nodes.

(b) $v_{T_{min}} \neq v_{P_{min}}$ In this case the root is not at the center of the star so we would choose the mule's placement at the center of the star to prevent any additional traversal to that node if the root fails.

■

Figure 7 presents an example of the two possible optimal configurations in a CGFP. Note that finding the optimal $v_{T_{min}}$

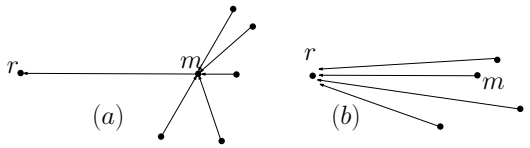

Fig. 7. An example of two possible optimal configurations in CGFP.

demands calculating the optimal TSP tour of $t(V \backslash v_i)$ for all the nodes in V. Using the $1+\epsilon$ TSP approximation method that runs in $O(n \log n)$ time for each of the n nodes [7], we get a near optimal solution for the CGFP mule problem may be obtained in $O(n^2 \log n)$ time and provide a $1+\epsilon$ approximation of the optimal solution.

VI. Conclusion

In this paper we advance the research on the mule approach for increasing network resiliency to communication failures. Our contributions include:

1) Improving the previously found 4-approximation bound for the UDL mule problem to optimal. The proposed algorithm runs in $O(n \log n)$.
2) Providing two solution methods for the UDG mule problem:
   a) $(57+\epsilon)$-approximation in $O(n^3 \log n)$.
   b) $(114+\epsilon)$-approximation in $O(n^2 \log n)$.
3) Finding two possible optimal solution structures for the CGFP mule problem, and providing a $(1+\epsilon)$ approximation algorithm for constructing them in $O(n^2 \log n)$.

VII. Acknowledgments

The research was been supported by the following sources:

1) Israel Science Foundation (grant No. 317/15).
2) IBM Corporation.
3) The Israeli Ministry of Economy and Industry.

References

[1] S. Anand, G. Zusseman, and E. Modiano. Construction and maintenance of wireless mobile backbone networks. *IEEE/ACM Transactions on Networking*, 17.1:239–252, 2009.

[2] J. Crowcroft, L. Levin, and M. Segal. Using data mules for sensor network data recovery. *Ad Hoc Networks*, 40:26–36, 2016.

[3] D. Kim et al. Minimizing data collection latency in wireless sensor network with multiple mobile elements. *INFOCOM, 2012 Proceedings IEEE. IEEE*, 2012.

[4] F. Fodor. The densest packing of 19 congruent circles in a circle. *Geometriae Dedicata*, 74.2:139–145, 1999.

[5] M. Di Francesco, S. K. Das, and A. Giuseppe. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Transactions on Sensor Networks (TOSN)*, 8.1:7–38, 2011.

[6] L. Levin, A. Efrat, and M. Segal. Collecting data in ad-hoc networks with reduced uncertainty. *Ad Hoc Networks*, 17:71–81, 2014.

[7] Satish B Rao and Warren D Smith. Approximating geometrical graphs via spanners and banyans. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 540–550. ACM, 1998.

[8] T. Rui, H. Li, and R. Miura. Dynamic recovery of wireless multi-hop infrastructure with the autonomous mobile base station. *IEEE Access*, 4:627–638, 2016.

[9] A. A. Somasundara, A. Ramamoorthy, and M. B. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. 2004.

[10] O. Tedas, V. Isler, J. h. Lim, and A. Terzis. Using mobile robots to harvest data from sensor fields. *IEEE Wireless Communications*, 16.1:22, 2009.