Bio-inspired Relay Node Placement Heuristics for Repairing Damaged Wireless Sensor Networks

Fatih Senel and Mohamed Younis
Dept. of Computer Science & Elect. Engineering
University of Maryland Baltimore County
Baltimore, MD 21250
{fsenell,younis}@cs.umbc.edu

Kemal Akkaya
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL 62901
kemal@cs.siuc.edu

Abstract—Due to the harsh surroundings and violent nature of the wireless sensor network (WSN) applications, the network sometimes suffers a large scale damage that involves many nodes and would thus create multiple disjoint partitions. This paper investigates a strategy for recovering from such damage through the placement of relay nodes (RNs) and promotes a novel approach. The proposed approach opts to re-establish connectivity (i.e., 1-vertex connectivity) using the least number of relays while ensuring certain quality in the formed topology. Unlike contemporary schemes that often form a minimum spanning tree among the isolated segments, the proposed approach establishes a topology that resembles a spider web, for which the segments are situated at the perimeter. Such a topology not only exhibits stronger connectivity than a minimum spanning tree but also achieves better sensor coverage and enables balanced distribution of traffic load among the employed relays. To further increase the robustness of the formed topology, the spider web approach is further extended so that the final topology is guaranteed to be 2-vertex connected. Both centralized and distributed implementations of the spider web approach are discussed. The simulation results demonstrate the effectiveness of the proposed recovery algorithm.

I. INTRODUCTION

Recent years have witnessed a growing interest in the applications of wireless sensor networks (WSNs) [1][2]. For some of these applications, such as space exploration, coastal and border protection, combat field reconnaissance, and search and rescue, it is envisioned that a set of sensor nodes will be employed to collaboratively monitor an area of interest and track certain events or phenomena. By getting these sensors to operate unattended in harsh environments, it would be possible to avoid the risk to human life and decrease the cost of the application. Sensors in these applications would be battery operated and have limited processing and communication capabilities. Upon their deployment, they are expected to form a network in order to share data and coordinate their action when participating in the execution of a task. For example, in a disaster management application, nodes need to collaborate with each other in order to effectively search for survivors, assess damages and identify safe escape paths. To enable such interaction, nodes need to stay reachable to each other.

Due to the small form factor and the limited onboard energy, sensor nodes are susceptible to failure. To mitigate the risk of reduced degree connectivity and loss of coverage that a faulty node may cause, the network deployment usually engages redundant nodes that can either act as passive spare or pick additional load if some nodes fail. In essence, such a strategy
provisions the tolerance of node failure at the time of network setup [3][4][5]. Other approaches for recovery from a node failure pursue more reactive strategy by changing the position of some nodes in order to restructure the network topology [6][7][8][9]. However, given the inhospitable environment that WSNs operate in, sometimes the network suffers a large scale damage that involves many nodes. For example, some sensors may be buried under snow or sand after a storm or destroyed by enemy explosives. In such scenarios the network may get partitioned and its services become very limited. A provisioned tolerance for occasional failure of individual nodes at the network design level will not be effective in countering a significant structural damage. In addition, coordinated repositioning of nodes will not be feasible since the network connectivity is so severed that the scope of the damage cannot be determined and many nodes will not be able to reach others.

This paper pursues the deployment of RNs in order to restore connectivity among the disjoint partitions of a damaged WSN. Given the cost of RNs and the logistical challenges in deploying them, it is usually desirable to minimize the count of the required nodes. Most of the related schemes found in literature strive to establish a Steiner minimum tree (SMT) for the various segments (i.e., partitions) in order to form a path between every pair of segments[10][11]. However, a SMT based topology usually yields a minimum spanning tree that makes some nodes a hot spot in terms of the traffic load and limits the achievable network throughput, and may thus make the achievable inter-node collaboration insufficient for specific application tasks. Unlike these schemes this paper proposes a novel approach that opts to increase the network connectivity and better spread the load among the employed relays. The main idea of the approach is inspired from the behavior of a spider which establishes a web for spanning gaps between objects. Our heuristic establishes a topology that resembles a spider web and for which the segments are situated at the perimeter. The formed topology not only exhibits stronger connectivity but also achieves better sensor coverage and enables balanced distribution of traffic load on the employed relays. These distinct features are provided with a comparable relay count to that an SMT-based solution will involve.

While the aforementioned features are desirable in conjunction with inter-segment connectivity, sometimes the application may even mandate backup paths for transmitting delay-sensitive data and for increased robustness against potential node failure and traffic bottlenecks. In such cases, the established inter-segment topology needs to boost the degree of connectivity of the network in order to ensure uninterruptable and timely packet delivery. To provide such a service, the proposed SpiderWeb approach is extended to establish 2-vertex disjoint paths between every pair of segments. Unlike, almost all of the related schemes found in the literature which establish 2-connected minimum spanning sub-graph, the SpiderWeb approach forms a polygon whose corners are the disjoint segments and relays are populated inward along the lines from each segment towards the center of mass (CoM). The simulation results confirm the performance advantages of the SpiderWeb algorithm in terms of the required RN count and the quality of the formed topology, assessed by the average node degree, expected path length, and node coverage.
The paper is organized as follows. The next section describes the assumed system model. Section III discusses the related work and highlights the distinct feature of the proposed approach. Section IV details the basic SpiderWeb algorithm, referred to hereafter as 1C-SpiderWeb, for forming a connected inter-segment topology. Section V describes 2C-SpiderWeb, the extended version that provides 2-vertex connectivity. Section VI discusses how the SpiderWeb approach can be implemented in a distributed manner. The validation experiments and simulation results are provided in Section VII. Finally section VIII concludes the paper.

II. SYSTEM MODEL

A WSN in the context of this paper is formed out of a set of sensors that is spread throughout an area of interest. A sensor node is highly energy-constrained and having limited communication and processing capabilities. The mission of the sensors is to serve the need of one or multiple in-situ users. Sensors collaboratively probe their surroundings and forward the findings to the interested user over a multi-hop path. Inter-sensor connectivity is essential for application-level interaction and for data routing. The WSN is operating in a harsh environment where sensors in the proximity of an event are susceptible to failure. Examples include damage by a forest fire, explosion in a combat field, etc. The failure of numerous sensor nodes sever the network connectivity by splitting the topology into isolated segments and may thus hinder proper operation. Therefore, recovering from such major damage is crucial. Figure 1 shows an articulation.

A relay is a more capable node with significantly more energy reserve and longer communication range than sensors. Although they can in principle be equipped with sensor circuitry, relays mainly perform data aggregation and forwarding. Unlike sensors, a relay may be mobile and has some navigation capabilities. Relays are favored in the recovery process since it is easier to accurately place them relative to sensors and their communication range is larger, which facilitate and expedite the connectivity restoration among the disjoint segments. Intuitively, relays are more expensive. Therefore the number of engaged relays is to be minimized. It is assumed that all deployed relays have the same communication range “$R$”. The distance between every pair of segments may be larger than $2R$, and thus multi-relay inter-segment paths would be necessary.

Figure 1: An articulation of a damaged WSN that got partitioned into multiple disjoint segments.
III. RELATED WORK

A. Connectivity Restoration in WSNs

While a number of schemes have been published recently for repairing network connectivity in partitioned WSNs, almost all of them assumed single node failure that can be fixed locally. The main idea is to identify and relocate some of the nodes. For example, recovering from the failure of cut-vertex is the focus of [12]. Only a special case is considered where the failure of the cut-vertex causes the network to split into 2 disjoint blocks. To re-link them the closest nodes are identified and are moved towards each other. The other nodes in the blocks follow in a cascaded manner. To reduce the travel distance, a connected dominating set (CDS) is determined for the block and only the closest dominatee moves. DARA [13] and PADRA [14][15] also exploit cascading motion in order to restore the connectivity due to failure of a single node. Here the failed node is known and one of its neighbors initiates a recovery process. The goal is to minimize the movement and the number of messages sent. While DARA picks the node with the least node degree, PADRA identify a CDS, as done in [12], to identify a dominatee node. The dominatee does not directly move to the location of the failed node, rather a cascaded motion is pursued to share the burden. Our problem in this paper is different in the sense that we deal with large-scale node failures as opposed to single node failures. PADRA was further extended to support multiple node failures that happen around the same times, named MPADRA [15]. The same connectivity restoration process has been applied to fix failures simultaneously. However, again in MPADRA the failures assumed are individual at different locations and thus the neighbors are aware of the location of the failures. In our case, we consider large scale failures which cannot be fixed by individual node movements.

B. Relay Node Placement

Deploying RNs has been exploited for boosting the performance of WSNs in terms of several metrics other than connectivity. The network lifetime is the metric that has received the most attention. For instance, in [16], RNs are deployed in order to reduce the communication energy consumed by a gateway node in sending the data to the base-station. The RN placement and energy provisioning problem is formulated as a mixed-integer nonlinear programming optimization. As opposed to this work, we do not focus on energy reduction. In addition, there is only one gateway node which needs to be connected with RNs. In our case, we have several partitions which need to be connected.

In [17], the authors consider the placement of RNs that can directly reach the base-station. Given a deployment of sensor nodes, the problem is to find the minimum number of RNs and where they can be placed in order to meet the constraints of network lifetime and sensor-relay node connectivity. The problem is shown to be equivalent to finding the minimum set covering, which is an NP-Hard problem. Therefore, a recursive algorithm is proposed to provide a sub-optimal solution. RNs are placed in
the intersections of the communication range of the largest number of sensors. This work has been further extended in [18][19][20] in order to address the problem of deploying second tier of relay nodes so that the traffic is balanced while using the least number of additional RNs.

Similar problems with different network models and assumptions have been studied in [21][22]. In [21], Pandey et al. have assumed multi-hop paths from SNs to sensors and investigated the effectiveness of several heuristics such as Binary integer linear programming and genetic algorithms. Li et al. [22] have studied the same problem assuming heterogeneous sensor networks with multiple base-stations. There are two main differences between our problem and these works. First, we do not consider energy as a metric in placing the RNs. Second, while connectivity among sensors and relays is considered, these nodes belong to a single network where all the RNs communicate with the same base-station. In our problem, we consider connectivity of several partitions.

C. Establishing k-connected Topologies

Careful placement of RNs has also been used to support connectivity and fault-tolerance. For instance, the objective of the relay placement in [25] is to form a fault-tolerant network topology in which there exist at least 2 distinct paths between every pair of sensor nodes. All RNs are assumed to have the same communication range $R$ that is at least twice the range of a sensor node. A sensor is said to be covered by a relay if it can reach that relay. The authors formulate the placement problem as an optimization model called “2-Connected Relay Node Double Cover (2CRNDC)”. The problem is shown to be NP-Hard and a polynomial time approximation algorithm is proposed. This work is further extended in [26] to cover $k$-connectivity.

Tang et al. have also studied the RN placement problem in a two-tiered network architecture [27] Assuming that $R \geq 4r$, they defined two problems, namely Connected Relay Node Single Cover (CRNSC) and 2-Connected Relay Node Double Cover (2CRNDC). They then proposed 4.5-approximation algorithm for CRNSC and a 6-approximation algorithm for 2CRNDC.

In [28], Kashyap et al., presented O(1)-approximation polynomial time heuristic to achieve 2-connectivity of $n$ nodes with the least RN count. The authors focused on both edge and vertex connectivity separately. Using the algorithm in [31], they first formed $k$-vertex connected spanning sub-graph $G_c$. Then by steinerizing the edges in $G_c$ they established the $k$-vertex connectivity. They also proved that the approximation ratio of this algorithm is 10 for $k=2$. This is the best known heuristic in the literature in terms of RN count.

In [30], Zhang et al. studied both single-tiered and two-tiered relay node placement for achieving 2-connectivity under a condition $R \geq r$ where $R$ and $r$ are communication ranges of RNs and sensor nodes, respectively. A network is said to be single tiered if each sensor node can reach at least one RN and network of RNs and sensors is 2-connected. Similarly, in two tiered networks, a sensor must reach at least two RNs and the network of RNs and sensor is 2-connected. The objective is again to
deploy the fewest RNs such that every SN can reach at least one RN node and the RNs form a connected network. They presented a 14-approximation algorithm and a \((14 + \epsilon)\)-approximation algorithm for single tiered problem and two-tiered problem respectively. These heuristics were further extended for supporting multiple base-stations. The common idea behind all of the algorithms presented in [30] is Steinerization of the edges of minimum 2-connected spanning sub-graph. As in [28], they used the same algorithm presented in [31] in order to find the 2-connected minimum spanning sub-graph.

The main goal in all of these efforts is to provide 1 or \(k\)-connectivity with the least number of RNs. A more detailed survey of relay placement problems can be found in [29]. None of these works considers the quality of the topology in terms of coverage, average node degree, average path length, etc. formed after the RNs are deployed. One or more of these metrics can be crucial in terms of application-level performance and thus they also need to be considered in placing the RNs. In this paper, we jointly consider establishing connectivity and providing efficient topologies which was not studied before.

Finally, we would like to note that all the aforementioned approaches are centralized, and little attention was given to distributed solutions of the connectivity restoration problem whether it is 1-connectivity or \(k\)-connectivity. The only distributed heuristic for RN placement was proposed in [32]. To maintain 2-vertex connectivity, the authors replicate RNs at the same location. This is not practical in our context, since if damage occurs at a certain location, usually it will affect all the RNs sitting at that location, hence all of them will fail at the same time. While our SpiderWeb heuristics are centralized, we also discuss how those heuristics can be extended to work in a distributed manner. A fully distributed placement scheme (i.e., without human intervention) is possible either if the RNs are mobile nodes or there are mobile robots which can carry and deploy the RNs. The latter approach is pursued in [34] to overcome the presence of obstacles. Our approach can be implemented in a distributed manner by following the same idea in this work.

IV. Establishing Inter-Segment Connectivity

The section focuses on establishing a connected inter-segment topology while reducing the number of used RNs. In the balance of this section, we define the problem and describe the proposed 1C-SpiderWeb heuristic.

A. Problem Definition

Our problem can be formally defined as follows: “Given \(m\) disjoint segments of sensors in an area of interest, determine the least count and position of RNs that are needed to connect all segments while maintaining some desirable topology features, such as robustness against relay failure, coverage and balanced traffic load”.

As far as the connectivity metric is concerned, optimal solution for this problem can be found by forming a minimum Steiner tree whose Steiner vertices will be the RNs. However, we need to make sure that the length of each edge in this tree is at most \(R\),
i.e., transmission range. This is referred to as Steiner Minimum Tree with Minimum number of Steiner Points and bounded edge length (SMT-MSP) [11].

**Definition 1 [SMT-MSP]:** Given a set $V$ of nodes and a transmission range of $R$, SMT-MSP is a tree $T$ spanning $V$ with minimum number of Steiner points such that every edge in $T$ has length at most $R$.

Since forming SMT-MSP is an NP-Hard problem [11], several heuristics have been proposed in recent years [10][23]. Since these heuristics focus solely on the minimization of the relay node count, the formed network topologies often lack other important properties in the context of WSNs such as robustness against failure and area coverage. One possible solution for the considered problem would be to try to improve the quality of the formed topology after restoring connectivity. However, this introduces a different problem and may unnecessarily increase the number of RNs. Instead, we propose a new heuristic which opts to support both goals, i.e., restore 1-connectivity and provide the desirable topology features by utilizing the way a spider creates its web.

**B. 1C-SpiderWeb Heuristic**

As illustrated in Figure 1, large scale damage mainly leaves part of the area uncovered with any node. Such a significant node loss affects the connectivity as well as coverage. One of the main drawbacks of pursuing a minimum Steiner tree is that two partitions are usually linked through a single path of RNs. Thus, the deployed relays become cut-vertices in the new network topology and the failure of any of the relays often makes the network partitioned again. The idea behind 1C-SpiderWeb deployment strategy is to place the relays inward in order to yield better network connectivity and coverage. In order to balance the inter-segment path length in terms of number of hops, RNs are placed towards the estimated CoM of the segments. Basically, from each partition to the CoM, we deploy nodes gradually until all the partitions are connected. In this way, we not only increase the total coverage of the network but also reduce the possible number of cut-vertices in the network.

Before the placement of RNs starts, we first need to identify the outer segments in the area of interest. To do this, we randomly pick representative nodes from each partition, and run a convex hull algorithm [24]. The convex hull algorithm returns a subset of representative nodes which sit on the corners of a convex polygon. After finding the convex polygon, we can determine the CoM of the polygon, using the following formula:

$$\text{Area}(A) = \frac{1}{2} \sum_{i=0}^{n-1} (x_{i+1}y_i - x_iy_{i+1})$$

$$c_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_{i+1}y_i - x_iy_{i+1})$$
where \( c_x \) and \( c_y \) are coordinates of the CoM.

Nodes are then deployed along the line between a segment and the CoM. Obviously, the relays around CoM will be in the communication range of each other and the segments thus become connected. To illustrate how 1C-SpiderWeb heuristic works, the example in Figure 2 will be used. The figure shows a network that got split into 7 disjoint segments. Running the Convex-Hull algorithm concludes that \( S_7 \) is an inner segment. Therefore, \( S_7 \) will not be involved in the algorithm. Next, we find the CoM of the polygon whose corners are \( S_1, S_2, S_3, S_4, S_5 \) and \( S_6 \) where \( S_i \) denotes a representative node for segment \( i \). The line between a particular \( S_i \) and CoM will be referred to as \( L_i \).

![Figure 2: An illustrative Example of topology established by employing 1C-SpiderWeb relay node deployment strategy](image)

Depending on the location of the partitions some relays may become connected before reaching the CoM. The 1C-SpiderWeb algorithm exploits this case for optimizing the deployment process. Relays are basically populated starting with the furthest partition in order to increase the probability of reaching an inner partition that may fall in its communication range or another partition on the convex hull, either directly or through one of the relays. In fact, as explained shortly, the direction of relay deployment may be changed according to progress made. The main point is that relays from the various partitions get closer to each other when approaching CoM. Starting with the furthest partition would increase the probability of reaching one of the existing relays and connecting with another partition, as detailed below.

While this case offers an opportunity for minimizing the number of relays, it can prevent all partitions from reaching each other. For example, two segments \( S_i \) and \( S_j \) may be too close to one another and far from the rest. The first relays on the paths \((S_i, \text{CoM})\) and \((S_j, \text{CoM})\), respectively, may be reachable to each other, making \( S_i \) and \( S_j \) connected, without reaching relays from other partitions. To avoid a premature termination of the relays along a line, the 1C-SpiderWeb algorithm applies a connectivity rule. Basically, before terminating the execution of 1C-SpiderWeb algorithm a segment needs to be connected to two neighboring segments one to its right and another to its left. Since we are considering the segments on the convex hull, every segment will have a neighbor to its right and another on its left [24]. The proof that the algorithm yields a connected topology is provided later.
in this section. Assume that there are \( m' \) segments on the convex hull. Let \( L_i \) and \( L_j \) be two neighboring lines where \( i = (j-1) \mod m' \) (i.e., \( L_j \) is the right neighbor of \( L_i \)). If any two nodes on \( L_i \) and \( L_j \) are connected then \( L_i \) will be referred to as RIGHT-CONNECTED and \( L_j \) is considered LEFT-CONNECTED. If \( L_i \) is neither left nor right connected, it will be NOT-CONNECTED. If \( L_i \) is both left and right connected, it will be CONNECTED.

Sometimes one or two partitions may be located very far from the CoM, when compared to other partitions. In those cases, closer partitions will reach each other in early iterations, however, the algorithm will continue deploying RNs along the neighboring lines of the furthest partition. To avoid redundant relay node deployment, we introduce a variable \( d_i \) which is the distance between CoM and closest node on \( L_i \) to CoM to decide where to start the RN placement and sort the lines \( L_1, L_2, \ldots, L_m \), in a descending order according to their \( d_i \) values. Let us assume that the furthest segment is \( S_f \) and the line from CoM to \( S_f \) is denoted as \( L_f \). The relay placement works in rounds. In each round, relays are populated on the lines starting from the line with the

---

**Figure 3**: A detailed example of how IC-SpiderWeb works. Hollow nodes represent the segment representatives and solid nodes represent RNs. Note that \( S_7 \) is not involved in the algorithm since it is an inner partition representative. (a) Resulting topology at the end first iteration, 6 RNs are deployed. (b) In the second iteration, first \( L_1 \) is processed and since it is RIGHT CONNECTED \( R_7 \) is deployed towards \( L_6 \). (c) Next, \( L_3 \) is processed and \( R_8 \) is deployed towards \( L_2 \) since \( L_3 \) is RIGHT CONNECTED. (d) Since \( L_5 \) is NOT CONNECTED \( R_9 \) is deployed towards CoM. (e) \( L_6 \) is processed and \( R_{10} \) is deployed towards \( L_5 \). (f) \( L_4 \) is processed and \( R_{11} \) is deployed towards \( L_5 \) and algorithm terminates since all lines are CONNECTED.
largest \(d_i\). In the first round, a relay is first placed on \(L_f\) at a distance \(R\) away from \(S_f\). Then the \(d_f\) value of \(L_f\) is updated as, \(d_f = d_f - R\). Since \(d_f\) is changed, we update sorted list of lines by inserting \(L_f\) into the correct index of the list. A heap data structure is used in order to maintain the correct ordering of the lines. Then, the connectivity status of \(S_f\) is updated based on whether the relay that has been just placed on \(L_f\) can reach any of the neighboring partitions.

If the relay node cannot reach a neighboring segment (either left of right), the connectivity status of \(S_f\) will not be changed. This process will continue until the connectivity status of \(S_f\) is changed. If the relay on \(L_f\) can reach a segment \(S_r\) to the right of \(S_f\), \(S_f\) becomes RIGHT-CONNECTED and \(S_r\) becomes LEFT-CONNECTED. The direction of relay placement for both \(S_f\) and \(S_r\) will be different from this point forward. Since \(L_r\) is shorter than \(L_f\), when the turn of \(S_r\) comes, the direction for relay placement will be tilted towards the right since \(S_r\) is LEFT-CONNECTED. Also, in the next round the direction of relay placement for \(S_f\) will be tilted to the left since it has already become RIGHT-CONNECTED.

In order to illustrate how the algorithm works, let’s consider the example in Figure 3. Assume that the sorted list is \(<L_1, L_3, L_5, L_6, L_2, L_4>\). In the first round, we first deploy relay node \(R_1\) along the line \(L_1\) and update the list as \(<L_2, L_5, L_6, L_2, L_4, L_1>\). Next, we deploy RNs \(R_2, R_3, R_6,\) and \(R_5\) along the lines \(L_3, L_5, L_6,\) and \(L_2,\) respectively and update the list (See Figure 3(a)). None of these relays can reach any segment. However, after deploying \(R_3,\) we update the status of \(L_1\) and \(L_2\) since these two lines are connected via \(R_1\) and \(R_3\) (i.e., \(L_1\) as RIGHT-CONNECTED and \(L_2\) is LEFT-CONNECTED). Therefore, updating the list is seized and the final deployment order of the lines will be \(<L_4, L_1, L_3, L_5, L_6, L_2>,\). Similarly we update the status of \(L_3\) and \(L_4\) after \(R_6\) is placed. When it is the turn of \(L_1,\) we deploy \(R_7\) to its left neighbor \(L_6,\) since \(L_1\) is RIGHT-CONNECTED (Figure 3 (b)). After deploying \(R_7,\) we update the status of \(L_1\) and \(L_6\) as CONNECTED and RIGHT-CONNECTED respectively. Then we deploy \(R_8\) (Figure 3 (c)) and update \(L_2\) and \(L_4.\) Since \(L_3\) is not connected so far, we deploy \(R_9\) along \(L_5\) (Figure 3 (d)). Similarly, we deploy \(R_{10}\) and \(R_{11}\) towards \(L_5\) since \(L_6\) and \(L_4\) are RIGHT-CONNECTED and LEFT-CONNECTED, respectively (Figures 3 (e) and (f)).

We stop when all lines are connected. Note that, unintentionally, \(R_{11}\) gets connected to \(R_5\) as well which increases the average node degree of the network. Finally, we check if there exists a disconnected segment. If so, we connect this segment by filling the gap between that segment and closest node with additional relays.

**C. Detailed Pseudo-Code of 1C-SpiderWeb**

Detailed pseudo-code for the 1C-SpiderWeb heuristic is provided in Algorithm 1. For the set of representative nodes, we find a subset which forms a convex-hull and compute the CoM of the convex hull (line 1 to 2). Then we draw a virtual line (i.e., \(L_i\)) from each corner of the convex polygon to the CoM (line 3). \(d_i\) values of the lines are assumed to be Euclidean distance In line 4 we build a max-heap of lines according to their \(d_i\) values. In lines 5 to 14, we extract the line with largest \(d_i\) from the heap, deploy a
relay node, update $d_i$ and insert the line to heap again. This loop will continue until the connectivity status of the lines are changed. This loop handles the case where one or more segment is very far from the CoM. Once the connectivity status of a line changes we create a sorted array from the heap and iterate on this array (line 15). In lines 16 to 34 of Algorithm 1, we iteratively process each $L_i$. In an iteration, for each $L_i$ (lines 18 to 30) we compute the location of the relay node which will be deployed next, and update the status of the current $L_i$ and neighboring lines. We skip the line if it is already connected.

Updating the line status is performed using the procedure in Algorithm 2. After the node deployment along a particular line, the connectivity of the line may be affected, i.e., becomes right or left-connected. This is checked in line 3 of Algorithm 2. If $L_i$ is left-connected, we will deploy next relay node along $L_i$ towards its right neighbor, rather than towards the CoM. Similarly, in line 6 we check the right connectivity. If a line is right connected, then we will deploy next relay node towards its left neighbor. We stop deploying RNs, if and only if $L_i$ is both left and right connected. Finally, based on this heuristic, the RNs are placed according to Algorithm 3.

D. Analysis of 1C-SpiderWeb Heuristic

In this section, we analyze the time complexity of the 1C-SpiderWeb heuristic and show that it guarantees connectivity of the generated network topology.

The execution time complexity of the approaches of [10][23] is $O(n^3)$ and $O(n^4)$, respectively, where $n$ is the number of segments. Theorem 1 shows that the 1C-SpiderWeb heuristic is superior in that regard.

In this section, we analyze the time complexity of the 1C-SpiderWeb heuristic and show that it guarantees connectivity of the generated network topology.

**Theorem 1:** The execution time complexity of the 1C-SpiderWeb heuristic is $O(n \log n \left\lfloor d/R \right\rfloor)$ where $n$ is the number of segments, $d$ is the length of longest line, and $R$ is the relay node transmission range.

**Proof:** The execution time of Graham Scan is $O(n \log n)$ time [24]. Finding CoM (line 2) takes constant time, and building the heap takes $O(n \log n)$ time. We analyze two while loops separately. The first while loop between line 5 and 14 iterates $n \lfloor d/R \rfloor$ times. Since lines 6, 9 and 10 takes $\log n$ time, the complexity of first while loops is $O(n \log n \lfloor d/R \rfloor)$. The second while loop between line 16 and 34 iterates $\lfloor d/R \rfloor$ times. Inner loop iterates $n$ times since $updateLineStatus()$ and $deploy()$ functions are executed in constant time. Hence, the complexity of second while loop is $O(n \lfloor d/R \rfloor)$. Therefore the overall time complexity of the
We also show that while outperforming the previous heuristics in terms of execution time, our heuristic also guarantees that at the end the resultant network is connected as proved in Theorem 2.

**Theorem 2:** The 1C-SpiderWeb heuristic guarantees connectivity of the resultant network.

**Proof:** Let us assume that the 1C-SpiderWeb heuristic does not yield a connected topology. In that case there exists at least one segment $s$ which is not connected to the rest of the network. There are two possible cases:

1) $s$ is a corner of the convex hull. In this case the algorithm will not terminate until line $L_u$ is connected to its right and left neighbors. This contradicts with the initial assumption.
2) $s$ is not a corner of the convex hull. In this case, the algorithm connects the segments which are corners of convex hull. At the end of the while loop `fillGapBetween UnattachedSegments()` function will handle the situation and eventually $s$ will be connected to the closest relay node. This also contradicts with the assumption.

V. Achieving 2-Connectivity

In this section we will define the problem and describe 2C-SpiderWeb heuristic for 2-vertex connectivity.

A. Motivation and Problem Definition

While ensuring 1-connectivity with robust topology features is crucial for application level requirements, it still does not guarantee a certain level of fault-tolerance. For instance, the resultant topology may still contain cut-vertices, as will be shown in the Section VII. The failure of any of these cut-vertices would partition the network again and would thus disrupt the data delivery. Since this may not be tolerated in application like search and rescue, backup paths are desirable. One of the solutions for this problem is thus to provide $k$-vertex connectivity which can be defined as follows:

**Definition 2 [k-vertex Connectivity]:** A graph is $k$-vertex connected, if there are at least $k$ vertex disjoint paths between every pair of nodes.

Establishing a $k$-connected relay network is shown to be NP-Hard in [28]. Therefore, we pursue heuristics and extend the 1C-SpiderWeb to provide 2-vertex connectivity. We refer to the new heuristic as 2C-SpiderWeb. Nonetheless, we still would like to maintain desirable topology features as we did for 1C-SpiderWeb in the previous section. Thus, the problem can formally be defined as follows: “Given $m$ disjoint segments of sensors in an area of interest, determine the least count and location of RNs such that both the resulting inter-relay topology and that of all nodes will be 2-vertex connected, while maintaining some desirable features, such as minimized average path length, coverage and balanced traffic load.”

Note that unlike previous work [28][32], we guarantee not only the 2-connectivity of the whole network (i.e., network of RNs and segments) but also the 2-connectivity of the inter-relay topology. We hereafter will call this property **dual 2-vertex connectivity**.

**Definition 2 [Dual 2-vertex Connectivity]:** Let $G = (V,E)$ be the network of partitions and RNs. $G$ is said to be dual 2-vertex connected if and only if both $G$ and its inter-relay node network are 2-vertex connected.

This property is very crucial because a 2-connected inter-relay network will serve as a backbone among disjoint segments and can help in ensuring reliable data routing among the segments and in providing certain QoS (e.g., real-time data delivery and reliability) for the applications.
B. 2C-SpiderWeb Heuristic

The proposed heuristic for forming a 2-vertex connected inter-segment topology operates in two stages by first employing 1C-SpiderWeb to establish connectivity and then achieving 2-connectivity by carefully placing additional relays. As presented in the previous section, 1C-SpiderWeb establishes the connectivity by forming a ring of RNs around the CoM of the segments. By connecting the segments to the ring with a path, we form a connected network, as seen in Figure 4. From a segment point of view, this path is the only path to the ring. Let \( p_i \) be the shortest path from segment \( S_i \) to the ring, e.g., \( p_1 = R_1 \rightarrow R_7 \rightarrow R_{13} \) in Figure 4a. Since a ring is already 2-vertex connected network, the cut-vertices must belong to \( p_i \). Let \( C \) be the set of cut vertices.

\[
C \subseteq \bigcup_{i=1}^{n} p_i \forall i \text{ s.t. } 1 \leq i \leq n
\]

where \( n \) is the number of segments.

![Figure 4](image_url)

**Figure 4:** (a) Sample topology formed by 1C-SpiderWeb (b) Topology after running 2C-SpiderWeb, i.e., each segment is connected to the second closest RN to establish an additional vertex independent path to the ring.

Therefore, 2C-SpiderWeb first invokes 1C-SpiderWeb, then forms an alternative path \( p_i' \) from segment \( S_i \) to the ring such that \( p_i' \cap p_i = \emptyset \). In addition to that, the first elements of \( p_i \) and \( p_i' \) must be connected in order to support dual 2-vertex connectivity, as shown in Figure 4b.

To explain the details of 2C-SpiderWeb, we use the following notation assuming that \( G_R = (V_r, E_r) \) is the relay network which is generated by 1C-SpiderWeb. We classify the vertices in \( G_R \) by labeling them with labels *first*, *cut* and *ring*.
**first:** the first node of every $p_i$.

**cut:** all the nodes of $p_i$ except the corresponding **first** and **ring-entrance** nodes.

**ring:** all the nodes of the ring

The classification is performed while running 1C-SpiderWeb. As we have discussed in 1C-SpiderWeb, RNs are deployed iteratively along the line connecting a segment to the CoM until the segments get connected to their left and right neighbors. We mark all the RNs which are deployed in the first iteration with ‘**first**’ label. The nodes which are deployed in later iterations will be marked with ‘**cut**’ label. In 1C-SpiderWeb, we invoke `updateLineStatus` function after deploying each relay node in order to update the connectivity status of the line. If the connectivity status of line $L_i$ changes upon deploying $R_i$, we mark $R_i$ with ‘**ring**’ label. The classification of nodes is illustrated in Figure 5.

Figure 5: Sample labeling of RNs (labels are color encoded)

As mentioned above, 2C-SpiderWeb forms an alternative path $p'_i$ for each $S_i$. In essence, for each segment $S_i$ the algorithm finds the second closest **ring** node, and fills gap between them. If the first nodes of $p'_i$ and $p_i$ are not connected to each other, then we deploy additional nodes between these two nodes. At the end of each iteration we label newly added nodes with **ring** label, and update the labels of the nodes of $p_i$ with **ring** label.

**C. Pseudo-Code of 2C-SpiderWeb**

Detailed pseudo-code for 2C-SpiderWeb is provided in Algorithm 4. Given a set of segment representative, we apply 1C-SpiderWeb to establish 1-vertex connectivity (line 1). As described above, the classification and labeling of RNs would be performed during the execution of 1C-SpiderWeb. Lines 2 - 13 establish an alternative path from every segment to the ring. Let $p_i$ be the array of RNs from segment $S_i$ to the ring. The first and last of elements of $p_i$ are labeled with ‘**first**’ and ‘**ring**’ labels respectively. In lines 6 - 12, we look for the closest relay node $c$ which is labeled with ring and is not an element of $p_i$. In line 11, we fill the gap between the $S_i$ and $c$. In order to maintain dual 2-vertex connectivity, we need to check if the first elements of $p_i$ and $p'_i$ are connected. If they are not connected, we need to fill the gap between them.

**D. Analysis of 2C-SpiderWeb Heuristic**

In this section, we analyze the time complexity of 2C-SpiderWeb heuristic and prove that it guarantees dual 2-vertex connectivity.
**Theorem 3:** The execution time complexity of 2C-SpiderWeb is

\[ O(nm + n \log \left\lfloor \frac{d}{R} \right\rfloor) \]

where \( n \) is the number of partitions, \( d \) is the length of the longest line considered while executing 1C-SpiderWeb, \( R \) is the relay node transmission range and \( m \) is the number of RNs placed by 1C-SpiderWeb.

**Proof:** We have proven in Theorem 1 that 1C-SpiderWeb runs in \( O(n \log \left\lfloor \frac{d}{R} \right\rfloor) \) time. The outer “for” loop in Algorithm 4 (Lines 2 – 13) iterates \( n \) times. The inner “for” loop (Lines 6 – 12) iterates \( m \) times. Therefore, the overall time complexity of 2C-SpiderWeb is \( O(nm + n \log \left\lfloor \frac{d}{R} \right\rfloor) \). 

**Lemma 1:** Let \( G = (V, E) \) be a 2-vertex connected graph. \( \forall u, v_1, v_2 \in V \) and \( v_1 \neq v_2 \), there is at least one path \( p_1 \) from \( v_1 \) to \( u \) and at least one path \( p_2 \) from \( v_2 \) to \( u \) such that \( p_1 \) and \( p_2 \) are vertex disjoint.

**Proof:** From the definition of 2-vertex connectivity, it is obvious that there are at least two vertex disjoint paths \( p_1 \) and \( p_1' \) from \( v_1 \) to \( u \), similarly \( p_2 \) and \( p_2' \) from \( v_2 \) to \( u \). Let

\[
p_1 = \{w_1, ..., w_i, w_{i+1}, ..., w_n\}
\]

\[
p_1' = \{w_1', ..., w_{i'}, w_{i'+1}, ..., w_n'\}.
\]

There are 3 possible cases.

1) \( p_2 \) does not intersect either with \( p_1 \) or with \( p_1' \).

The proof is trivial.

2) \( p_2 \) intersects with \( p_1 \) but not with \( p_1' \).

This case implies that \( p_2 \) and \( p_1' \) is vertex disjoint.

3) \( p_2 \) intersects both with \( p_1 \) but not with \( p_1' \).

Assume that \( p_2 \) first intersects with \( p_1' \) at \( w_1' \) and then intersect with \( p_1 \) at \( w_j \). Let

\[
p_2 = \{t_1, t_2, ..., w_1', ..., w_j, t_k, ..., t_n\}
\]
We update $p_2$ as follows so that $p_1$ and $p_2$ will be vertex disjoint:

$$p_2 = \{t_1, t_2, ..., w'_i\} \cup \{w'_{i+1}, w'_{i+2}, ..., w'_n\} \blacksquare$$

**Lemma 2:** Let $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ be two graphs where $|V_i \cap V_j| \geq 2$. If $G_i$ and $G_j$ are 2-vertex connected, then the union of $G_i$ and $G_j$ is also 2-vertex connected.

**Proof:** (Proof by construction) Let $u, v \in V_i \cup V_j$ and $u \neq v$.

There are 3 possible cases:

1) If $u \in V_i$ and $v \in V_i$ then there are two vertex disjoint paths between $u$ and $v$ since $G_i$ is 2-vertex connected.

2) If $u \in V_j$ and $v \in V_j$ then there are two vertex disjoint paths between $u$ and $v$ since $G_j$ is 2-vertex connected.

3) We will be done, if we can find two vertex disjoint paths from $u$ to $v$ where $u \in V_i$ and $v \in V_j$.

Let $w_1, w_2 \in V_i \cap V_j$ and $w_1 \neq w_2$.

We can find two paths $p_1$ and $p_2$ from $w_1$ to $u$ and from $w_2$ to $u$ such that $p_1$ and $p_2$ are vertex disjoint (from Lemma 1).

Similarly, we can find two paths $p'_1$ and $p'_2$ from $w_1$ to $v$ and from $w_2$ to $v$ such that $p'_1$ and $p'_2$ are vertex disjoint.

So we can construct two vertex disjoint path from $u$ to $v$ as follows:

$$P_1 = (p_1)^\text{reverse} \cup (p'_1)^\text{reverse}$$

$$P_2 = (p_2)^\text{reverse} \cup (p'_2)^\text{reverse} \blacksquare$$

**Theorem 4:** 2C-SpiderWeb guarantees dual 2-vertex connectivity.

**Proof:** 1C-SpiderWeb stops deploying RNs if each line is both left and right connected as explained before in Section IV. In other words, it forms an inner ring. In 2C-SpiderWeb, each terminal is connected to the ring at two different vertices via two vertex disjoint paths. We will prove two cases:

Case 1) 2C-SpiderWeb guarantees the 2-connectivity of whole network.

The nodes that are part of the ring and the connections form a segment to the ring is a 2-vertex connected sub-graph. The whole network can be expressed as the union of $n$ of such 2-vertex connected sub-graphs, where $n$ is the number of segments. Based on Lemma 2, the formed network is also 2-vertex connected. Figure 6 illustrates the idea through an example, where,

$$G_i = (V_i, E_i) \text{ where } V_i = \{S_1, R_1, R_2, R_7, R_8\}$$
and

\[ G_2 = (V_2, E_2) \text{ where } V_2 = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7\} \]

It is obvious that \( G_1 \) and \( G_2 \) are both 2-vertex connected (they are both rings) and \( |V_1 \cap V_2| = 2 \). By Lemma 2, \( G_1 \cup G_2 \) is also 2-vertex connected.

Case 2) 2C-SpiderWeb guarantees the 2-vertex connectivity of inter-relay network.

Since the first nodes along the path from segment to the ring (in this example \( R_7 \) and \( R_8 \)) are also connected to each other, the relay node network can be expressed as the union of two 2-vertex connected networks.

VI. DISTRIBUTED SPIDERWEB

It is challenging to develop a distributed algorithm for optimal relay node placement since the problem is NP-Hard. As a result, almost all of the heuristics proposed for this problem in the literature need global knowledge to establish the connectivity. However, sometimes this is impractical, since the exact locations of partitions are not always globally available. In general, a distributed implementation of node placement heuristics is possible when the relays are mobile nodes or there are means such as robots to deploy the relays without any human intervention.

In this section, we discuss how the SpiderWeb heuristic can be adapted to perform the damage recovery in a distributed manner. First, an approximate calculation of CoM would be pursued since segments will not have information about the network state after damage has been inflicted. The CoM can be estimated using the initial network state, e.g. based on a uniform random distribution of sensors in the area of interest. Moreover, unlike the centralized SpiderWeb algorithms, the distributed versions require messaging in order to detect whether the connectivity is established and the termination condition is met.

The distributed version of SpiderWeb may work as follows. First, each partition elects the closest node to CoM as the partition representative. By looking at all the neighbors’ location, each node determines whether it is candidate representative. All candidate representative information is flooded to the partition, therefore the partition representative is elected. Each partition representative uses its ID to distinguish the segment from others and draws a line from itself to the approximate CoM without knowing where the other partitions are. Every representative iteratively deploys RNs along the line with a distance \( R \) between them. The relay can simply come from the segment and would thus be aware of the segment ID. If a relay encounters a neighbor node with different segment ID, it sends this information to its segment representative. In a distributed version of the algorithm,
however, segments cannot update their deployment direction after being LEFT-CONNECTED or RIGHT-CONNECTED, since they do not know where the next neighbor line is. Therefore, the position of the relay that is reached from another segment is queried and shared with the segment representative. Using the position of the reached segment, the segment representative updates its connectivity status as either LEFT-CONNECTED or RIGHT-CONNECTED, and it continues deploying relays until it receives second connectivity update message from a different direction. Since two or more consecutive same-direction (left or right) connectivity update message does not yield full connectivity, the partitions continue deploying RNs until they receive connectivity update message from different direction or the distance between the last deployed relay node and the CoM becomes less than $R/2$ units away. After receiving the second message, the representative updates its connectivity status as 1-CONNECTED and stops deploying RNs towards CoM. After neighboring representatives reach 1-CONNECTED status, the representative calculates the position of closest ring relay node belonging to a different segment, from two connectivity status update messages. To establish 2-vertex connectivity, the representative deploys RNs towards this relay node. Once it gets connected to it, the representative stops deploying RNs and updates its connectivity status as 2-CONNECTED.

Running the SpiderWeb algorithm in distributed manner also triggers another special case. When the scope of the damage spans a large area, the partitioned segments may not be evenly distributed in the deployment field. One or more segments may be located very far from the other segments. In such a scenario the estimated CoM will not be accurate. Also the order of connectivity messages can get messed up and two consecutive connectivity update messages from same direction can be received. An example is illustrated in Figure 7. In the first round the RNs $R_{11}, R_{21}, R_{31}, R_{41}$ are deployed. After deployment, RNs $R_{11}$ and $R_{31}$ send left and right connectivity update messages to their segments respectively. Note that since the node $R_{21}$ is both left and right connected, segment $S_2$ stops node deployment. In the second round, after deploying RNs $R_{12}$ and $R_{32}$ the segment $S_1$ received left connectivity update message and segment $S_3$ received left connectivity update message again, but this time with a different segment ID.

![Figure 7. Distributed RN deployment until all representatives become 1-CONNECTED](image-url)
VII. PERFORMANCE EVALUATION

The performance of the SpiderWeb approach is validated through simulation. This section describes the experiment setup, performance metrics and performance results.

A. Experiment Setup and Performance Metrics

Two parameters are varied in the experiments, namely the number of segments and the communication range of a relay node ($R$). We have created varying number of partitions (3 to 15) that are randomly located in an area of interest (1200m x 1000m). When studying the impact of $R$ on the performance, it is varied between 50m to 200m while the number of partitions is fixed at 9. The results of the individual experiments are averaged over 100 runs of different topologies. We observed that with 99% confidence level, the simulation results stay with 6%-10% of the sample mean. The figures in this section thus show such errors with the error bars. The following metrics are considered in evaluating the performance:

- **Number of RNs**: This metric reports the total number of RNs required for restoring connectivity. As mentioned earlier, RNs are usually more expensive than sensor nodes. Thus, this metric reflects the cost of repairing the network.

- **Percentage of cut-vertices**: This metric reflects the total number of articulation points in the resultant network topology relative to the number of deployed relays. The presence of fewer cut-vertices makes the topology more robust to node failures. This metric is only used in 1C-SpiderWeb, since 2-vertex connectivity yields no cut vertices.

- **Average node degree**: This metric shows the average number of neighbors for each node in the resultant topology. Higher node degree not only indicates stronger connectivity but also helps in spreading the traffic to balance the load among nodes and reduce data latency.

- **Average Path Length**: This metric indicates the expected path length between two partitions that are chosen. A small path length is desirable since this will reduce the data transmission latency between partitions.

- **Coverage**: This metric captures the total area that the connected network can probe. It is calculated as the union of the coverage of all the individual nodes. The sensing coverage of a single node is assumed to be a circle with radius $R_{sns}$ where $R_{sns}$ is the sensing range of the node. In our simulations, $R_{sns}$ is set to 40m. Better node spreading increases the coverage since it minimizes the overlapping.

- **Recovery Time**: This metric represents the time needed to restore the connectivity when a distributed implementation is pursued. In such a case, the RNs are assumed to be mobile or carried by a mobile robot to be placed at their designated locations. Minimizing the recovery time can be important for some of the critical monitoring applications such as border control.
B. Performance Evaluation of 1C-SpiderWeb

1) Baseline for Comparison

We compare 1C-SpiderWeb with two baseline approaches; the Steinerized Minimum spanning tree (SMST) and the Steiner Minimum Tree with Minimum number of Steiner Points and bounded edge length (SMT-MSP) algorithms. We used a 3-approximation version of the latter [10].

As in the 1C-SpiderWeb approach, the SMST algorithm starts with identifying the partitions and picking random representative nodes from each partition. Then it constructs a complete graph $G(V, E)$ where $V$ is the set of representatives and $E$ is the set of edges created with all possible $(u, v)$ where $u, v \in V$ and $u \neq v$. Once $G$ is created, Kruskal’s MST algorithm [23] is executed to find the tree edges. On each edge RNs are deployed to establish connectivity between the vertices.

Meanwhile in the first step of the SMT-MSP, all $e_i \in E$ are sorted in the ascending order. For each set of three representative nodes $a$, $b$, and $c$, which belong to different connected components, the algorithm looks for a point $s$, that is at a most distance $R$ away from $a$, $b$ and $c$. If such a point exists, the algorithm form a 3-star which includes the edges $(s,a)$ $(s,b)$ $(s,c)$. After this step, for each edge $e_i$ that is longer than $R$ the algorithm fills the gap between the endpoints of $e_i$ using relays.

2) Simulation Results

Number of Relay Nodes: Figure 8(a) shows that 1C-SpiderWeb requires more RNs to reconnect the partitions compared to SMST and SMT-MSP. This is expected because our goal is not only to find the least relay count but rather focus on providing better topologies while still striving to keep the relay count close to minimal. In all three algorithms, the number of RNs increases almost linearly with the number of partitions. In addition, with the increase in the number of partitions the cost grows at a higher rate for SpiderWeb than the baseline approaches. The reason for the widened performance gap between 1C-SpiderWeb and the baseline approaches is that in addition to establishing connectivity we are striving to create topologies with more features such as

![Figure 8](image_url)

**Figure 8:** Number of RNs required (a) with varying partition count, $R = 100m$, (b) for different $R$, # of Partitions = 9.
node degree, path length, and coverage. For instance, the degree of the nodes solely depends on the number of neighbors they have and with the increased number of partitions, more neighbors will be needed to maintain the same average degree in the network.

Figure 8(b) indicates that the performance under varying $R$. While the SMST and SMT-MSP approaches outperform the 1C-SpiderWeb heuristic for small transmission ranges, the 1C-SpiderWeb approach closes the gap with $R$ larger than 150m. This can be explained as follows. The 1C-SpiderWeb approach attempts to connect three partitions in a single round. After each relay node is deployed, we update the connectivity status of the current and neighboring lines. For bigger transmission ranges, the algorithm terminates at earlier rounds, since all the lines get connected before getting close to the CoM.

**Percentage of cut-vertices:** Percentage of cut vertices is an assessment of how tolerant the resulting topology to a relay node failure. The results in Figure 9(a) reveal that the SpiderWeb approach always performs better than the baseline approaches as the partition count increases. However, under varying $R$, we observe that SMT-MSP approach performs very well for bigger transmission ranges, as seen in Figure 9(b). Both SMST and SMT-MST steinerize the edges in minimum spanning tree. Cut-vertices are usually located on the MST edges. For bigger radio ranges, the SMT-MSP approach puts more 3-stars, as explained above, and minimizes the number of MST edges that has to be steinerized. Therefore, percentage of cut-vertices decreases significantly. The performance of our approach also enhances when $R$ grows mostly because of the increased connectivity that results from extending the communication range.

![Figure 9](image)

Figure 9: Comparison of 1C-SpiderWeb and the baseline approaches with respect to the percentage of cut-vertices among the deployed relays, (a) for different number of partitions, and (b) under varying $R$.

**Average Node Degree:** Figure 10(a) and (b) illustrate that 1C-SpiderWeb always provides higher average node degrees than the baseline approaches. The performance advantage for 1C-SpiderWeb is quite significant. Therefore, 1C-SpiderWeb produces topologies having balanced traffic and reduced latency. This can be attributed to the fact that 1C-SpiderWeb requires more RNs than SMST and SMT-MSP, yielding higher relay density and stronger connectivity. In addition to that the performance of 1C-SpiderWeb grows in significance as $R$ increases. The same reasoning applies when $R$ is increased in Figure 10(b).
Average Path Length: The results in Figure 11(a) illustrate that 1C-SpiderWeb significantly outperforms the baselines in terms of average path length as the number of partitions increases. This is because in SMST the height of the formed tree will affect the average path length between any pair of partitions. Similarly, in SMT-MSP the average path length will be affected by how many 3-stars are found in the second step of the algorithm [10]. However, in 1C-SpiderWeb, the convex hull diameter determines the average path length.

Coverage: As far as the network coverage is concerned, the 1C-SpiderWeb approach again outperforms both baseline approaches particularly when the partitions count increases, indicating the scalability of our approach (See Figure 12(a)). When $R$ (i.e., communication range) grows, Figure 12(b), fewer relays are deployed and the coverage diminishes. Recall that in our simulations, $R_{\text{ms}}$ is set to 40m. However, for larger $R$ the coverage performance of SMST is not as affected since the number of RNs required to connect the partitions is almost same as shown in Figure 8(b).
**Recovery Time:** To simulate distributed recovery, we have assumed that one of the segments will be the RN provider. Specifically, all deployed RNs for restoring connectivity will move to their final locations from this segment. Recovery time is directly proportional to the distance between the segment which provides RNs and farthest RN from this segment as well as the speed of the RNs. The RN provider is randomly picked among the partitions. We have assumed that the speed of a RN is equal to 6 meters/min [35].

The results in Figure 13(a) confirm that the recovery time needed for 1C-SpiderWeb is always shorter than the baseline approaches as the number of partitions increases. This can be attributed to the fact that, 1C-SpiderWeb deploys RNs from the convex hull towards the center. Therefore the farthest RN from the provider segment is somewhere inside the convex hull. However in SMST and SMT-MSP 3-app algorithms, the farthest RN can be at one end of the network while the provider is located at the other end. We have also assessed the recovery time under varying R. As shown in Figure 13 (b) 1C-SpiderWeb always outperforms the baseline approaches. According to the results in Figure 13 (b) we can conclude that recovery time is not much affected by the communication range. This is expected since the recovery time is calculated based on the Euclidean distance between the provider segment and the farthest RN, which depends on the distribution of partitions in the area.

![Figure 13](image1.png)

**Figure 13:** Comparison 1C-SpiderWeb and the baseline approaches with respect to the recovery time, assuming that the speed of a RN is 6 meters/mins (a) under varying partition count, and (b) under varying R.

### C. Performance Results for 2C-SpiderWeb

1) **Baseline for Comparison**

We compare 2C-SpiderWeb with the relay placement heuristic for k-vertex connectivity proposed by Kashyap et al.,[28]. We refer to that algorithm as 2C-SSG, since it is based on 2-connected minimum spanning sub-graph. The algorithm constructs a weighted complete graph \( G_c = (V, E_c) \) of representative nodes of the segments, where \( V \) is the set of representatives and \( E_c \) is the set of all possible edges. The weights of edges are assigned as follows
$$c_e = ||e|| - 1$$ where $|e|$ is the length of edge $e$.

After assigning the edges, the algorithm in [31] is executed to compute an approximate minimum 2-vertex connected spanning sub-graph $G'$. By steinerizing the edges in $G'$, they establish 2-vertex connectivity. The last step of the algorithm is pruning. If the absence of a node $u$ does not violate 2-vertex connectivity, the algorithm removes it. In [28], it is proven that the approximation ratio of this algorithm is 10.

2) Simulation Results

Number of Relay Nodes: Figure 14(a) illustrates that 2C-SpiderWeb requires more RNs than 2C-SSG in order to achieve 2-vertex connectivity as the number of segments increases. As we discussed above, this is expected since providing better topologies is a prime objective of 2C-SpiderWeb. In addition, unlike 2C-SSG, we are providing dual 2-vertex connectivity. It is intuitive that more RNs are required for ensuring dual 2-vertex connectivity. Even though 2C-SpiderWeb employs more RNs, it is a linear function of the number of partitions. Figure 14(b) shows that the performance of 2C-SpiderWeb gets closer to the performance of 2C-SSG for large transmission ranges. The reason is that, 2C-SpiderWeb forms an inner ring in earlier iterations of the deployment, such a ring becomes a backbone and thus the required number of RNs drops quickly.

![Figure 14: Number of RNs required (a) with varying partition count, $R = 100$m, (b) for different $R$, # of Partitions = 9.](image)

Average Node Degree: Figure 15(a) and (b) indicates that 2C-SpiderWeb outperforms 2C-SSG approach in terms of average node degree. As the number of partitions increase, the gap between 2C-SpiderWeb and 2C-SSG increases. This indicates that 2C-SpiderWeb provides better topologies in terms of load balancing since it yields better connectivity. Similarly, in Figure 15(b), 2C-SpiderWeb performs significantly better than 2C-SSG as the communication range increases. It is important to note that although the required number of RNs in 2C-SpiderWeb gets closer to 2C-SSG as the communication range increases as shown in Figure 14(b), the performance gap between 2C-SpiderWeb and 2C-SSG increases in Figure 15(b). Thus, it can be concluded that 2C-SpiderWeb performs significantly better for large communications ranges.
**Average Path Length:** Figure 16(a) shows that the average path length in 2C-SpiderWeb is always superior to that of 2C-SSG. The performance advantage even grows as the number of partitions increases. Since a ring is 2-vertex connected network, 2C-SSG tries to form the smallest ring of all segments. The expected path length is equal to the diameter of this ring which is $\frac{n}{2}$ where $n$ is the number of nodes in the network. 2C-SpiderWeb, however, deploys RNs inward towards the CoM. The algorithm terminates, if all the lines reach their neighboring lines. In essence, 2C-SpiderWeb forms a ring around the CoM, inside the convex hull of the segments. This ring is intuitively smaller than the one that formed by 2C-SSG. When number of partitions is equal to three, 2C-SSG performs better than 2C-SpiderWeb. The reason is that the 2-connected spanning subgraph of three segments will obviously be a triangle which spans all the three segments. Therefore every segment pair will be able to reach each other through one edge of the spanning triangle, which is expectedly shorter than reaching through a point inside of the triangle.

Figure 16(b) illustrates that 2C-SpiderWeb performs better than 2C-SSG as the communication range increases. Same reasoning can be applied in this case.

**Coverage:** Figures 17(a) and (b) compare the performance of 2C-SpiderWeb to 2C-SSG in terms of network coverage. The results in Figure 17(a) indicate that 2C-SpiderWeb yields more coverage than 2C-SSG. This can be attributed to the fact that 2C-SpiderWeb provides better node distribution and thus minimizes the coverage overlaps as the number of partitions increases.
Figure 17(a) also shows that the performance of 2C-SSG is better than 2C-SpiderWeb when there are few partitions. The reason for this is the size of the ring formed by different algorithms. As mentioned earlier if two segments are close to each other, they will most likely be LEFT or RIGHT-CONNECTED in early iterations and the two segments will change their deployment direction towards other neighboring segments. This early change of direction will enable us to form a bigger ring. However for small number of partitions, (e.g., \# of partitions = 3), the segments will most probably far from each other and the size of the ring formed by 2C-SpiderWeb algorithm will be small and Hence it will increase the coverage overlap among the individual RNs. When communication range increases, the network coverage drops in both approaches as seen in Figure 15(b). However, for 2C-SpiderWeb the drop is at a much lower rate and the performance gap between 2C-SpiderWeb and 2C-SSG widen for large communication ranges.

Recovery Time: Figures 18 (a) and (b) show that 2C-SpiderWeb always restores the connectivity in a shorter time than 2C-SSG both under varying partition count and varying \( R \). We can apply same reasoning which we have discussed in 1C-SpiderWeb. If we compare the Figures 13 (b) and 18 (b), it can be concluded that the SpiderWeb heuristic requires almost same amount of time to achieve both 1-connectivity and 2-connectivity.

VIII. CONCLUSION

In this paper we have presented a relay node placement approach which will not only guarantee connectivity among a set of disjoint segments of a partitioned WSN but also provide topologies with several desirable features such as robustness, extended coverage and balanced traffic load. The idea of our approach is to connect the segments by deploying the RNs like a spider weaving its web. Our aim is to connect multiple segments at once. To do so we identify the boundary segments by running convex hull algorithm and then we compute the CoM of these segments. The locations of the RNs are computed iteratively with respect to the location of a corresponding segment and CoM. We have further extended this heuristics to ensure 2-vertex connectivity. We
first applied, 1C-SpiderWeb, the 1-connected version of the algorithm to establish the connectivity. 1C-SpiderWeb forms a smaller ring of RNs around the CoM. In order to provide 2-vertex connectivity we connect the partition and the ring via second node independent path.

The simulation results have confirmed that the topologies created as a result of running the 1C-SpiderWeb and 2C-SpiderWeb schemes decreases the average inter-segment path-length while increasing the network coverage and average node degree. As a future work we plan to extend our approach to consider QoS requirements such as bandwidth and inter-segment delay constraint.

Acknowledgement: This work is supported by the National Science Foundation, contracts # 0000002270, 00005844 and 1018404.

REFERENCES


