A Coverage-Aware Distributed k-Connectivity Maintenance Algorithm for Arbitrarily Large k in Mobile Sensor Networks

Vahid Khalilpour Akram[ID], *Member, IEEE*, Orhan Dagdeviren[ID], *Member, IEEE*, and Bulent Tavli[ID], *Senior Member, IEEE*

Abstract—Mobile sensor networks (MSNs) have emerged from the interaction between mobile robotics and wireless sensor networks. MSNs can be deployed in harsh environments, where failures in some nodes can partition MSNs into disconnected network segments or reduce the coverage area. A k-connected network can tolerate at least k-1 arbitrary node failures without losing its connectivity. In this study, we present a coverage-aware distributed k-connectivity maintenance (restoration) algorithm that generates minimum-cost movements of active nodes after a node failure to preserve a persistent k value subject to a coverage conservation criterion. The algorithm accepts a coverage conservation ratio (as a trade-off parameter between coverage and movements) and facilitates coverage with the generated movements according to this value. Extensive simulations and testbed experiments reveal that the proposed algorithm restores k-connectivity more efficiently than the existing restoration algorithms. Furthermore, our algorithm can be utilized to maintain k-connectivity without sacrificing the coverage, significantly.

Index Terms—Mobile sensor networks, k-connectivity, distributed algorithm, connectivity maintenance, restoration, reliability, fault tolerance.

I. Introduction

MOBILE Sensor Networks (MSNs) have a wide range of applications [1]. Robust connectivity is a vital and challenging requirement for MSNs. A network is k-connected if it remains connected after failures in k-1 arbitrary nodes. Preserving the k value after a node failure keeps the fault tolerance of an MSN at the desired level. k-connectivity maintenance (restoration) is a process that is initiated after a node failure to restore the k of a network to its primary value (the k value before the failure). Since MSNs are inherently mobile, moving nodes for k-connectivity restoration is a convenient solution.

Movement-based connectivity restoration can, potentially, lead to loss of coverage in an MSN to some extent (in the absence of backup nodes). However, maintaining connectivity without the loss of coverage can impose higher movement costs. Therefore, there is an inherent trade-off between movements required for connectivity and network coverage. Nevertheless, connectivity and coverage problems are closely related and should be addressed jointly. Although connectivity restoration is a well-known problem for $k = 1$ or $k = 2$ (1 or 2-connectivity restoration), the design, analysis, and implementation of a movement-based coverage-aware distributed k-connectivity restoration approach (for arbitrarily large k) for MSNs is left unstudied to the best of our knowledge.

To address these challenges, we create a movement-based coverage-aware distributed k-connectivity maintenance algorithm for MSNs. Our contributions are as follows:

1) We propose a novel distributed movement-based algorithm for k-connectivity maintenance (for arbitrarily large k) in MSNs.
2) The proposed approach accepts a coverage conservation ratio and adjusts the generated movements to maintain the k-connectivity with minimal coverage loss.
3) We provide the theoretical foundations of the proposed algorithm and its resource consumption analysis in terms of bit, space, time, and computational complexities.
4) Through extensive simulation and testbed experiments by employing IRIS nodes and Kobuki robots, we evaluate the performance of the proposed algorithm in terms of movement costs, coverage loss, wallclock times, and transmitted bits. Our findings clearly demonstrate that the proposed algorithm outperforms its counterparts.

The rest of the paper is organized as follows; Section II provides a concise survey on existing approaches. In Section III, we formulate the problem and explain the network model. The proposed algorithm is described in detail and theoretically analyzed in Section IV. Section V presents the experimental results. The conclusions and future research directions are given in Section VI.

II. Related Work

Studies on k-connectivity can be organized into detection, deployment, and restoration categories. The k-connectivity

detection is the problem of finding the k value of a network [2]–[6]. In the k-connectivity node deployment problem, the aim is to deploy the nodes and arrange their radio ranges in such a way that the resulting topology becomes k-connected [7], [8].

For constant and predefined k values (e.g., $k = 1, 2$), there are many studies on connectivity restoration that attempt to reconnect the nodes in a partitioned network by adding relay nodes [9]–[17], utilizing reserved nodes [3], [18], changing the communication range [19], or moving other active nodes [20]–[26]. However, there is no distributed solution for the k-connectivity restoration problem for arbitrary k.

Another important problem in MSNs is maximizing the area covered by the available sensor nodes considering various constraints such as connectivity, energy, and lifetime [27], [28]. A detailed review of the coverage problem has been provided in [29]. The connected m-coverage problem is defined as ensuring the coverage of each point in the network by, at least, m sensor nodes [30]. In fact, our algorithm is complementary to the m-coverage algorithm in [30] (i.e., the two algorithms can work in cooperation to create a k-connected and m-covered MSN).

A central Minimum-Cost k-Connectivity Restoration (MCCR) algorithm is proposed for the movement-based general k-connectivity restoration problem in [31]. MCCR uses a k-connectivity test and a maximum weighted matching algorithm to find the best possible movements for k-connectivity restoration. After each failure, MCCR creates a set P that includes the positions of all the nodes and a set V that includes the remaining active nodes. It then removes a position from P and if the remaining positions form a k-connected graph, the algorithm creates a bipartite graph over the P and V sets. Next, it applies a maximum matching algorithm to the resulting bipartite graph and repeats this process for all other positions in P selecting a matching with the maximum value as the best movements. The best time complexities of the k-connectivity testing and maximum matching are $O(n^2 \times k^2)$ [2] and $O(n^3)$ [32], respectively. This leads to $O(max\{n^3k^2, n^4\})$ time complexity for MCCR. MCCR is improved by the TAPU algorithm, which replaces the matching algorithm with the shortest path tree algorithm [33]. In this central approach, a shortest path tree rooted by the failed node is established between the nodes and a safe node with the minimum-cost to the root is moved to the position of the failed node. This approach generates exactly the same movements as MCCR, but its running time is lower than that of MCCR. Using central algorithms on large-scale MSNs can be inefficient because it imposes a large amount of message passing and resource consumption to gather the topology information in the sink node.

A restricted version of the k-connectivity restoration problem in heterogeneous mobile networks is presented in [34]. It is assumed that the network has a limited number of mobile nodes, which simplifies the model and reduces the problem complexity by restricting the number of possible solutions.

The main novelty of the proposed distributed algorithm is that nodes can estimate the remote paths in the general topology and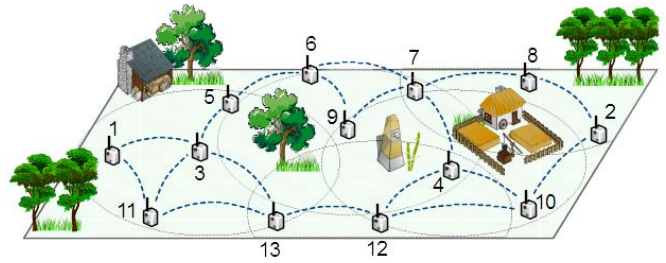 can decide on the required movements based on the local neighborhood information without requring global topology information, in most cases. Indeed, this is the first distributed algorithm that can maintain k-connectivity for any arbitrary k value in MSNs. Furthermore, the proposed algorithm is able to balance the cost of the generated movements for k-connectivity restoration with the overall coverage loss.

Fig. 1. Illustration of an example MSN topology.

III. PROBLEM FORMULATION AND MOTIVATION

We can model an MSN as an undirected graph $G(V, E)$ where V is the set of nodes and E is the set of links (edges). The nodes in MSNs can exchange messages if they are in radio range of each other. Fig. 1 shows a sample 2-connected MSN where the large dashed circles show the radio range of each node. This network remains connected in case of single node failure, yet, failures in two nodes can partition the network.

We assume that the links are bidirectional and stable; consequently, the topology does not change (due to the unstable links) during the algorithm execution. Otherwise, the k value of the network could continuously change making k-connectivity restoration extremely complicated. In a k-connected network, there are at least k disjoint paths between each pair of nodes. Two paths are disjoint if they share no common nodes except the source and target. In Fig. 1, removing any node except nodes 1, 4, and 9 reduces the k value to 1. Thus, we can divide the nodes in an MSN into *Joint* and *Trusted* sets. The failure of any *Joint* node reduces the k value, but failure in a *Trusted* node has no effect on k. For example, in Fig. 1, nodes 1, 4, and 9 are *Trusted* and the other nodes are *Joint*.

A. Problem Formulation

Throughout the paper, $n = |V|$ is the total number of nodes, D is the network diameter, and Γ_v is the set of 1-hop neighbors of node v. We denote the maximum and minimum degrees of nodes by Δ and δ, respectively, the degree of node v by d_v, the k value of graph G by $\kappa(G)$, a path between nodes x and y by $\rho(x, y)$, and the number of disjoint paths between nodes x and y in G by $x \overset{G}{\sim} y$. In Fig. 2a, for example, we have $n = 13$, $d_1 = 2$, $\kappa(G) = 2$, $\Gamma_3 = \{0, 2, 4\}$, $D = 5$, $\Delta = 5$, $\delta = 2$, $\rho(1, 6) = (1, 4, 5, 6)$, and $0 \overset{G}{\sim} 8 = 2$. In this figure, the weights of the edges are the movement costs between the related nodes. We assume that $G_v(V_v, E_v)$ is the 2-hop local subgraph of node v (Fig. 2b shows G_5), and G/v indicates graph G without node v. In this figure, the gray nodes are
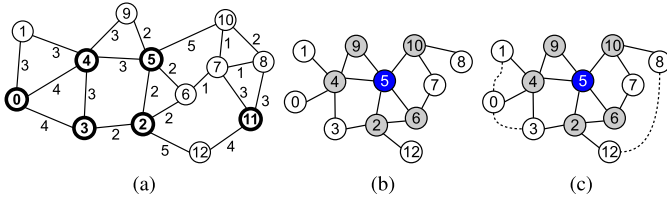
Fig. 2. a) 2-connected MSN, b) G_5, c) G_5^k.

Joint, and the white nodes are *Trusted* nodes. Removing a *Trusted* node has no effect on $\kappa(G)$, but removing a *Joint* node reduces $\kappa(G)$. Formally, we can define the *Trusted* and *Joint* nodes as follows

Definition 1: In a k-connected graph $G(V,E)$, node $v \in V$ is Trusted if $\kappa(G) = \kappa(G/v)$, otherwise v is Joint.

The **k-connectivity restoration problem** can be defined, formally, as follows: Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of nodes where v_i has position i, r is the radio range, $E = \{(v_i, v_j) : v_i, v_j \in V \ and \ ||v_i - v_j|| \leq r\}$ is the set of edges, $f : V \times V \to R^+$ is the movement cost function, and $G(V,E)$ is the topology graph. Given a failed node v_x, the k-connectivity restoration problem is finding a movement cost function $M : V \to V$ such that

$$minimize \sum_{v_i \in V/v_x} f(v_i, M(v_i)) \qquad (1)$$

under the condition that

$$\kappa(G(V_M, E_M)) = \kappa(G(V, E)) \qquad (2)$$

where

$$V_M = \{M(v_i) : \forall v_i \in V/v_x\}.$$
$$E_M = \{(v_i, v_j) : v_i, v_j \in V_M \ and \ ||v_i - v_j|| \leq r\}. \quad (3)$$

The aim of k-connectivity restoration is to find a set of movements with minimal cost to restore the k of the network to its initial value after the failure of a node.

B. Motivation

Maintaining k-connectivity in an MSN brings many direct and indirect benefits. The most important benefit of k-connectivity is the inherent **fault tolerance** built in by design (i.e., a k-connected MSN stays connected even if any $k-1$ node combinations are left inoperational). Indeed, the impact of **bottlenecks and critical nodes** is alleviated in a k-connected network provided that the number of such nodes is less than k. Since the k value is a lower bound for the **edge connectivity** of any network, a k-connected network stays connected even if $k-1$ links stop functioning. Since each node in a k-connected MSN has at least k **disjoint paths** to any other node, the **communication reliability** of such an MSN increases directly with the k value. All the network services in a k-connected MSN (e.g., topology management, routing, and backbone formation) can continue uninterruptedly because unless, at least, k nodes are incapacitated, there is, at least, one path to every operational node from any other operational node. The availability of k **disjoint paths** toward the sink

from each node is an invaluable opportunity for balancing the **energy consumption** throughout the network, which is necessary for **network lifetime maximization**.

The k value of a network can be utilized for estimating the lower or upper bounds of many other properties of the network. The **minimum degree** of a k-connected network is, at least, k (i.e., each node has, at least, k neighbors). The initial knowledge of the degree of a network is a useful input to **clustering and partitioning algorithms**. We can always find a **cycle** of k arbitrary nodes in any k-connected network. If all the links in the network have the same capacity, the k value determines the upper bound of the **network flow** between every pair of nodes. A network with a higher k value, typically, has a higher **node density** than a network with a lower k value. Therefore, the k value can be utilized as an indicator of the node density.

IV. THE PROPOSED APPROACH

The proposed Localized Imaginary Network Aided Restoration (LINAR) algorithm has two main phases. In the first phase, nodes detect their status (*Trusted* or *Joint*) and support degree and share this information with their 2-hop neighbors. The second phase of the algorithm starts when a node detects the failure of a *Joint* neighbor, which triggers the movements of a single node (or multiple nodes). The second phase is initiated by a neighbor of the failed node and finishes when the k value of the network returns to its initial value. In the second phase, the nodes around the failed/moved nodes make local decisions (regarding to move or not to move) based on their 2-hop local subgraphs. Failure in any node can be restored by the network because each node in the network continuously monitors its neighbors and initiates the restoration process when required.

In the first phase, the nodes find their status using an imaginary k-connected local subgraph and broadcast this state to their neighbors. Generally, the 2-hop local subgraph of nodes has a lower k value than the actual k because local subgraphs omit remote paths between the nodes. For example, the presented network in Fig. 2a is 2-connected whereas G_5 is 1-connected. To extract the correct state information from the 2-hop local subgraphs, we first create an imaginary k-connected local subgraph, then we find the number of disjoint paths between the 1-hop neighbors of each node.

A. Imaginary k-Connected Subgraphs

In the proposed algorithm, all nodes estimate some paths existing in the global topology (but they do not exist in their 2-hop local subgraphs) by converting their 2-hop local subgraphs into k-connected graphs. For example, in Fig. 2b, $\kappa(G_5) = 1$ whereas the k value of general topology is 2. Using the provided lemmas and theorems, node $v \in V$ can estimate remote paths between its 2-hop neighbors and converts G_v to a k-connected graph. If $G(V,E)$ is k-connected, then we, definitely, have, at least, k paths between any pair of nodes in G_v even if some of the paths pass through some remote nodes beyond G_v. In other words, if the k value of a local graph, say G_v, is smaller than the k value of the general graph G,

then any pair of nodes in G_v that have fewer than k paths in G_v have some remote paths beyond G_v that connect them by the nodes out of G_v. Formally, if $\kappa(G_v) < \kappa(G)$, then for any $x, y \in V_v / \Gamma_v$ with $i < k$ paths in G_v there are at least $k - i$ remote paths beyond G_v that connect them using remote nodes in G. For example, in Fig. 2b, nodes 0 and 1 have only one path in G_5. However, since the network is 2-connected, node 5 can, safely, conclude that there is at least one more path beyond G_5 between nodes 0 and 1. In the proposed algorithm, each node converts its local subgraph to a k-connected graph that represents the possible existing remote paths between its 2-hop neighbors. To do so, each node adds imaginary edges to its subgraph (in its local memory) to incorporate the possible missing remote paths in G_v. An imaginary edge can be defined as follows

*Definition 2: An **imaginary edge** $e = (x, y)$ is a hypothetical edge that is added to G_v to incorporate the path $\rho(x, y) = (x, \ldots, y) \in G$ where $\rho(x, y) \notin G_v$.*

Our aim is to increase the $\kappa(G_v)$ to the $\kappa(G)$ in such a way that v has the same state in both graphs. Note that no imaginary edge can be added to the 1-hop neighbors of v in G_v because all neighbors of every $u \in \Gamma_v$ are already present in G_v as stated in Remark 1. Therefore, each node can add only imaginary edges between its 2-hop neighbors. In Fig. 2b, for example, node 5 can add only edges between the white nodes.

Remark 1: Node v cannot attach any imaginary edge to any node $u \in \Gamma_v$.

The purpose of adding imaginary edges is to incorporate missing remote paths between the nodes in G_v until G_v becomes k-connected. Therefore, every added imaginary edge should represent a missing remote path between the nodes. If nodes $a, b \in V_v / \Gamma_v$ already have, at least, k paths in G_v, it is possible that they have no other paths beyond G_v. Therefore, node v can add only imaginary edges between its 2-hop neighbors if fewer than k paths exist between them in G_v as stated in Remark 2.

Remark 2: Node v can add imaginary edges between a, $b \in V_v / \Gamma_v$ if $a \overset{G_v}{\sim} b < k$.

In the proposed algorithm, each node tries to convert its local subgraph to a k-connected graph by creating minimal (ideally zero) extra paths between its 1-hop neighbors. We will prove that this strategy guarantees that the state of *Joint* nodes remains *Joint* in their k-connected local subgraphs. Formally, we define the resulting k-connected local subgraph of node v (after adding imaginary edges) as follows

Definition 3: G_v^k is the k-connected graph obtained after adding the minimum number of imaginary edges to G_v that minimizes $\sum_{x,y \in \Gamma_v} x \overset{G_v}{\sim} y$.

$\sum_{x,y \in \Gamma_v} x \overset{G_v}{\sim} y$ is the total number of disjoint paths between every 1-hop neighbor of node v. Definition 3 implies that node v generates G_v^k such that the number of paths between 1-hop neighbors of node v is minimized. We will prove that if we minimize $\sum_{x,y \in \Gamma_v} x \overset{G_v}{\sim} y$ while adding imaginary edges, a $Trusted$ node v in G_v^k will, definitely, be $Trusted$ in G. Fig. 2c shows G_5^k. Lemma 1 proves an important property about $Trusted$ nodes that helps us identify

them by finding the number of paths between their 1-hop neighbors.

Lemma 1: For any $v \in V$, G/v is k-connected if $\forall (x, y) \in \Gamma_v : x \overset{G/v}{\sim} y \geq k$.

Note that the proofs of all the lemmas and theorems presented in the paper can be found in the online supplement. Lemma 1 implies that if we have $i \overset{G/v}{\sim} j \geq k$ for any pair of nodes $i, j \in \Gamma_v$, then $v \in V$ is a $Trusted$ node in G. In other words, any $Joint$ node $v \in V$ has at least two neighbors $i, j \in \Gamma_v$ such that $i \overset{G/v}{\sim} j < k$. Consequently, to change the state of $Joint$ node $v \in V$ to $Trusted$, we need to create new paths (not passing over v) between i and j. In other words, as long as the number of paths between i and j is not increased the state of a $Joint$ node v does not change to $Trusted$. Lemma 2 states that as long as the number of paths between the 1-hop neighbors of v does not increase, adding any edge to the graph has no effect on the state of v.

Lemma 2: For any $e \notin E$ and $v \in V$, $k(G(V, E)/v) = k(G'(V, E \cup e)/v)$ if $\forall (x, y) \in \Gamma_v : x \overset{G}{\sim} y = x \overset{G'}{\sim} y$.

Lemma 2 indicates that if adding an edge to the graph does not increase the number of paths between the 1-hop neighbors of v, then node v remains $Trusted$ if it was $Trusted$, and remains $Joint$ if it was $Joint$ before adding the edge. In other words, as long as the number of disjoint paths between the 1-hop neighbors of node v does not increase, adding any edge to the graph has no effect on the $Trusted$ or $Joint$ state of node v. Therefore, to maintain the same state for v in both G and G_v^k, we need to keep the number of paths between the 1-hop neighbors of v unchanged. However, this is not always possible. In some cases, we have no choice but to increase the number of paths between the 1-hop neighbors of v. Using Definition 3, we convert G_v to a k-connected graph by creating the minimum number of new paths between the 1-hop neighbors of v. Lemma 3 states that if we generate G_v^k according to Definition 3, the number of paths between 1-hop neighbors of v in G_v^k will be less than or, at most, equal to the number of paths between the same nodes in G.

Lemma 3: For any k-connected graph $G(V, E)$, $\forall v \in V$, and $\forall (x, y) \in \Gamma_v$, we have $x \overset{G_v^\kappa}{\sim} y \leq x \overset{G}{\sim} y$.

Lemma 3 implies that for any added imaginary edge to G_v^k, we, indeed, have an actual path in G. There can be multiple imaginary edge sets to create G_v^κ that minimize $\sum_{x,y \in \Gamma_v} x \overset{G_v}{\sim} y$, yet, all of them satisfy $x \overset{G_v^\kappa}{\sim} y \leq x \overset{G}{\sim} y$. For example, Fig. 3a shows the local subgraph of node 9 and Fig. 3b shows the same graph after adding imaginary edges. Adding the first four imaginary edges (green edges) to G_9 does not increase the number of edges between its 1-hop neighbors. However, to make G_9 2-connected we have no choice but to add one more edge (the orange edge) which increases the number of disjoint paths between the 1-hop neighbors of node 9. Fig. 3c shows the graph obtained after removing node 9 from its 2-connected local subgraph. Since this graph is 2-connected, node 9 is a $Trusted$ node in G_9^2. Node v has more than one choice to generate G_v^k according to Definition 3. In fact, Figs. 3d, 3f, and 3h can be used to create G_9^2. Figs. 3e, 3g, and 3i show that after adding these edges and removing node 9,
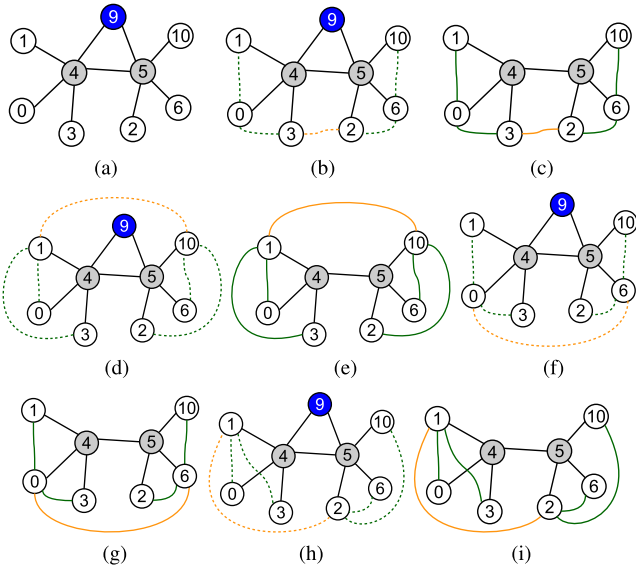
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

AKRAM *et al.*: COVERAGE-AWARE DISTRIBUTED k-CONNECTIVITY MAINTENANCE ALGORITHM

5

Fig. 3. a) G_9: Local subgraph of node 9; b), d), f), and h) G_9^k: 2-connected local subgraphs of node 9; c), e), g), and i) $G_9^k/9$: removing node 9 from G_9^k.
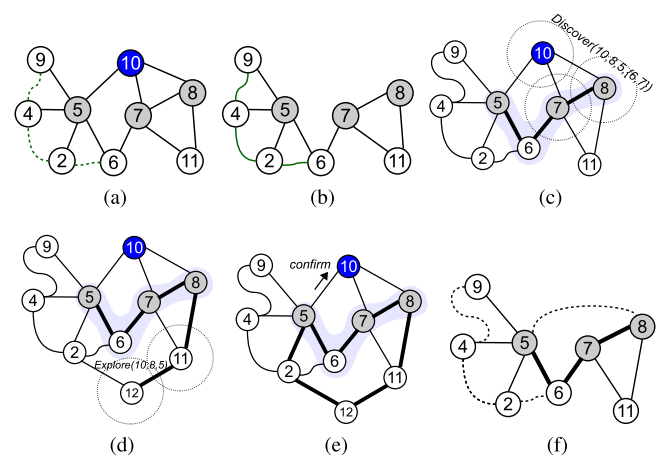

Fig. 4. a) G_{10}^k: 2-connected local subgraph of node 10, c) $G_{10}^k/10$: removing node 10 from G_{10}^k, d) broadcasted $Discover$ messages, e) broadcasted $Explore$ messages, e) sent $Confirm$ message, f) resulting G_{10}^k.

all three G_9^2's remain 2-connected as in the case depicted in Fig. 3c.

To generate G_v^k, node v, first, obtains E_v', which is the set of all possible edges between the 2-hop neighbors of node v that do not exist in E_v, by performing a linear search among its 2-hop neighbors. Next, any edge (a, b) from E_v' is removed if $a \overset{G_v}{\sim} b \geq k$ (Remark 2) so that only the possible candidate edges between the 2-hop neighbors of v with fewer than k paths remain in E_v'. Then, one of the subsets $S_v \subseteq E_v'$ that gives the minimum number of total paths between every pair x, $y \in \Gamma_u$ is selected (Definition 3) and G_v^k is obtained by merging S_v and G_v. Theorem 1, directly, implies that if a node is $Trusted$ in G_v^k then it is $Trusted$ in G.

Theorem 1: If G_v^κ/v is k-connected, then G/v is k-connected.

If node v is $Joint$ in G, then it will, definitely, be $Joint$ in G_v^k. However, it is possible for node v to be $Joint$ in G_v^k while it is $Trusted$ in G. For example, Fig. 4a shows the local subgraph of node 10 after adding imaginary edges (G is illustrated in Fig. 2a). Adding three imaginary edges to G_{10} makes it 2-connected without increasing the number of edges between its 1-hop neighbors. Fig. 4b shows the graph obtained after removing node 10 from its 2-connected local subgraph. Since this graph is 1-connected, the algorithm considers node 10 as a $Joint$ node. However, node 10 is a $Trusted$ node in G. Therefore, to avoid such mislabeling, we confirm the status of a detected $Joint$ node (by using only G_v^k) by performing a global path search between $x, y \in \Gamma_v$ with $x \overset{G_v/v}{\sim} y < k$.

Lemma 4: For any $v \in V$, $\kappa(G/v) < \kappa(G)$ if $\exists u \in \Gamma_v$ s.t $d_u = k$.

We can avoid a global path search for some of the $Joint$ nodes based on the degree of their neighbors. Lemma 4 states that if node v has a neighbor $u \in \Gamma_v$ such that $d_u = k$, then v is $Joint$ because removing v reduces the k value of the graph.

All neighbors of u in G already exist in G_v. If $d_u = k$ in G_u, then $d_u = k$ in G, which means that v is a $Joint$ node because in G/v node u has at most k-1 neighbors. Hence, a simple rule for detecting some of the $Joint$ nodes can be expressed as

$$if \ \exists \ u \ \in \ \Gamma_v \ s.t \ d_u \ = \ k \ in \ \ G_v^k \ \to \ v \ is \ \textbf{\textit{Joint}}. \quad (4)$$

For example, in Fig. 2c, the degree of node $9 \in \Gamma_5$ is 2; therefore, node 5 can confirm that it is $Joint$ without initiating any path search. If node v is $Joint$ in G_v^k and does not satisfy Eq. 4, then we should search for a global path between a pair of its 1-hop neighbors. For example, Fig. 4a and Fig. 4b show G_{10}^k and $G_{10}^k/10$, respectively. The degrees of all the gray nodes are higher than 2 and there is only one disjoint path between nodes 2 and 11 in $G_{10}^k/10$. Therefore, node 10 cannot determine its status using Theorem 1 and Eq. 4. The nodes that satisfy Theorem 1 or Eq. 4 find their status and terminate the identification phase. For the other nodes we start a global path search. If node v does not satisfy Theorem 1 and Eq. 4 then we have $x, y \in G_v^k/v$ such that $x \overset{G_v/v}{\sim} y = k-1$. If we can find a path $p(x, y)$ between x and y such that $p(x, y)$ includes none of the nodes from the existing paths in G_v^k, then we can add an imaginary edge between x and y in G_v^k/v. If the resulting graph satisfies Theorem 1, then node v is $Trusted$. Otherwise we repeat the same procedure for the other node pairs that have fewer than k disjoint paths in G_v^k/v. If we cannot find any such paths, then v is a $Joint$ node. For example, in $G_{10}/10$ (Fig. 4b), a local path $p(8, 5) = (8, 7, 6, 5)$ already exists between nodes 8 and 5 and the graph becomes 2-connected if we find another path between them. After adding such an imaginary edge, node 10 can mark itself as $Trusted$ because in the updated $G_{10}^k/10$ there are at least two disjoint paths between every $(x, y) \in \Gamma_{10}$.

Let v be an ambiguous node and $w, u \in \Gamma_v$ be the nodes between which v is interested in finding a disjoint path. The first step is finding disjoint paths between u and w in G_v, which can be achieved by using a k-connectivity detection algorithm [2]. The nodes used in local disjoint paths should

be excluded from the global search. Let A be the set of nodes used in the local disjoint paths between u and w. For example, in Fig. 4b, we have $A = \{7, 6\}$. In the proposed method, node v broadcasts a $Discover(v, u, w, A)$ message to start a global path search in G. In Fig. 4c, for example, node 10 broadcasts a $Discover(10, 8, 5, \{6, 7\})$ message to start a path search between nodes 8 and 5 without the participation of nodes 6 and 7. Except node w, other 1-hop neighbors of v rebroadcast the $Discover(v, u, w, A)$ (Fig. 4c) and in this way all nodes in G_v receive the $Discover$ message. $Discover$ messages are not broadcasted by the 2-hop neighbors of the source node. In fact, only the 1-hop neighbors of the source node rebroadcast $Discover$ messages, which leads to at most Δ $Discover$ messages for each path search initiative. Each node that receives a $Discover(v, u, w, A)$ message from node u broadcasts an $Explore$ message if it is not a member of $A \cup w$. In Fig. 4d, node 11 broadcasts an $Explore$ message after receiving a $Discover$ message from node 8 because $11 \notin \{6, 7, 5\}$. After receiving the $Discover$ message, nodes 6 and 7 ignore all upcoming $Explore$ messages because A covers both nodes. Each $Explore$ message contains the ID of the initiator, source, and target nodes and is rebroadcasted by the other nodes. Nodes 12 and 2 rebroadcast the received $Explore$ message (Figs. 4d) and node 5 sends a *Confirm* message to node 10 to inform that the search was successful (Figs. 4e). In this way, the resulting graph becomes k-connected (Figs. 4f).

The nodes that cannot find their status using their 2-hop local subgraphs start the aforementioned global path search process to determine their status. No synchronization is required for exchanging the *Discover*, *Explore*, and *Confirm* messages, hence, the global path search can be conducted asynchronously between the nodes. The global path search operations are performed in the first (setup) phase of the algorithm when the nodes attempt to detect their status.

Using the outcomes of Lemma 4 and Theorem 1, a subset of the nodes can determine their correct status based on the local information. The other nodes that cannot determine their status using the established theoretical framework start a global path search process and use global information to determine their status. Therefore, all nodes always find their correct status.

B. Trade-Off Between Mobility And Coverage

Maintaining system connectivity while preserving system coverage are the two indispensable essential requirements. Hence, many studies on the connectivity restoration problem emphasize the relationship between the connectivity and coverage problems and usually solve the restoration problem by introducing additional mobile nodes [35]–[41].

To maintain the maximal coverage after a node failure without adding new nodes, a neighbor with a lower individual effect on the covered area can be moved to the position of the failed node. Obviously, this strategy can reduce the k value and can even destroy the connectivity of the network. Another approach is to search the entire network, find a node that has the lowest individual coverage, and move it to the position of the failed node. This approach can generate costly

movements. To obtain a balanced and fair method, we used a $\beta \in [0, 1]$ parameter in the proposed algorithm that defines the desired coverage conservation ratio. $\beta = 0$ means that coverage is not important at all and $\beta = 1$ means that coverage preservation has the highest priority. The proposed approach always restores the k value either by generating minimal movements and ignoring the coverage loss (i.e., lower β values) or by minimizing the coverage loss and sacrificing the minimization of the movements (i.e., higher β values).

We use a local-neighborhood-based heuristic to estimate the coverage loss after any event that leads to a decreased k. We define a support degree value for each node that determines the number of supporting neighbors that have intersecting coverage areas with the node's coverage area. Formally, $sup(v)$ is a value that is used to estimate the likelihood of the coverage loss due to the departure or failure of node v. The support degree is a relative value that allows us to compare the nodes based on their individual covered area. A higher support degree of a node means that we lose a smaller covered area after moving or losing that node in comparison to the other nodes. We can model an MSN as a unit disk graph and our heuristic for calculating the support degree is based on the following two facts about unit disk graphs

1) The nodes with a higher number of neighbors have a lower individual covered area and can have a higher support degree,
2) The nodes that have a higher number of independent neighbors (the neighbors that have no direct link between themselves) have a lower individual covered area and can have a higher support degree.

For example, in Fig. 5a, node 1 has six neighbors and the area it covers is completely covered by other nodes. In Fig. 5f, node 1 has only two connected neighbors and about half of the area it covers is not covered by other nodes. Based on the above facts, we propose the following equation to calculate a support degree for each node

$$Sup(u, G(V, E)) = \frac{1}{k} \times \sum_{v \in \Gamma_u} d_u - |\{(v, w) \in E : w \in \Gamma_u\}|$$
$$if \quad k(G/u) = k(G) \quad (5)$$

In Eq. 5, for each node u and one of its neighbors $v \in \Gamma_u$, we find the difference between the degree of u and the number of common neighbors of u and v, which indicates the number of neighbors of u that are not directly connected to v. We calculate the sum of these values for all the neighbors of v and divide it by k to have the same base line for all nodes. Using Eq. 5, nodes with lower individual covered areas obtain higher support degrees. Fig. 5 illustrates the relationship between the neighbors and the covered area of node 1 where we assume that the general network is 2-connected (the nodes are a part of a 2-connected network). In Fig. 5a, for example, node 1 has six neighbors and $d_1 = 6$. Therefore, for node 1, we have $sup_1 = 1/2 \times ((6 - 1) + (6 - 2) + (6 - 1) + (6 - 1) + (6 - 2) + (6 - 1)) = 14$. Similarly, in Figs. 5b, 5c, 5d, 5e, and 5f, the support values of node 1 are 8, 6, 4, 2, and 1, respectively.

Note that, only the support degree of $Trusted$ nodes are used in the restoration phase. Since the $Joint$ nodes are not
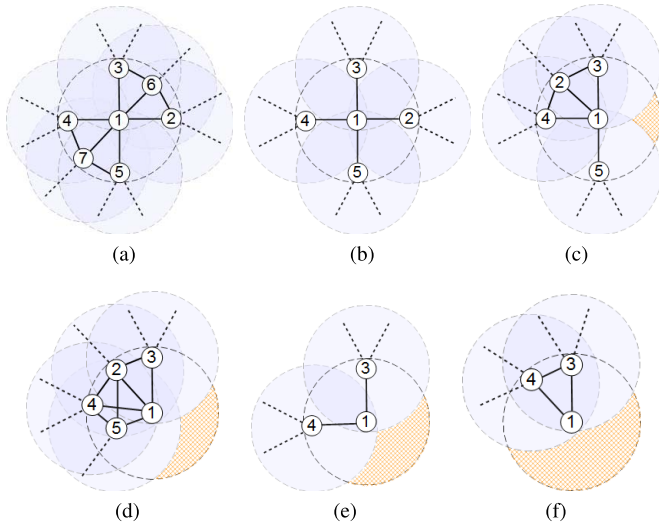
Fig. 5. The bluish texture indicates the intersecting coverage areas of different nodes and the yellowish texture indicates the coverage loss if node 1 moves away with a) $sup_1 = 14$, b) $sup_1=8$, c) $sup_1=6$, d) $sup_1=4$, e) $sup_1=2$, and f) $sup_1=1$.

selected for moving, their support degree has no effect on the moving strategy.

As illustrated in Fig. 5, it is more likely that for a larger support value, the individual covered area is lower, hence, the coverage loss after moving is also lower. Note that the support value of a node provides only an estimation about the coverage loss after failure. In the proposed algorithm, node v moves to the position of node u under one of the following conditions

1) u is a failed *Joint* node and v has the lowest coverage-aware moving cost to u among the other neighbors,
2) u moved to the location of another node, $(sup(u) - sup(v)) \times \beta > 1$ and v has the lowest coverage-aware moving cost to u among the other neighbors.

$(sup(u) - sup(v)) \times \beta > 1$ implies that moving v to the position of u increases the coverage ratio with respect to the β value. To calculate the coverage-aware moving cost of node v to the position of node u, we divide the movement cost between nodes u and v by $1 + (sup(v) \times \beta)$. Thus, higher β and sup values, effectively, reduce the moving costs and lead to the selection of the nodes with lower individual covered areas even if they have relatively higher moving costs. The desired β value can be determined by the user based on the requirements of the application.

C. The Steps of the Proposed Algorithm

The steps of the proposed algorithm (LINAR) are presented in Algorithm 1.The algorithm accepts the k value and the coverage conservation ratio β as an input parameter. Each node maintains the following set of local variables

- I_u: Available information about other nodes.
- Γ_u: The 1-hop neighbor list of node u.
- V_u: The nodes in the 2-hop local subgraph of node u.
- E_u: The edges in the 2-hop local subgraph of node u.

- $stat_u$: Status (*Trusted* or *Joint*) of node u.
- T_u: Target nodes of u in the path search.
- R_u: The received messages in node u.

Each node starts the algorithm by broadcasting a *Start* message (line 2) that includes its coordinates (position). When node u receives a *Start* message from node w (line 3), it adds w to Γ_u and the position of w to $I_u[w].pos$ (line 4). Each node calls the *CreateGraph* procedure after ts time units from the first *Start* message reception (line 5). The ts delay should be long enough to allow sending all the *Start* messages. In the *CreateGraph* procedure, each node creates its 1-hop local subgraph and broadcasts its 1-hop neighbor list and their position information in an Ngb message (lines 7-8). When node u receives an Ngb message from node w (line 9), it updates I_u, V_u, and E_u with the received lists and calls the *DetectState*(u) procedure if it receives Ngb messages from all $v \in \Gamma_u$ (lines 9-12).

In the *DetectState* procedure, node u creates its 2-hop local subgraph G_u using the sets V_u and E_u. Node u then calculates its support degree using G_u and the provided method in Eq. 5 (line 14). Node u ignores the remaining commands in the *DetectState* procedure (line 15) and broadcasts its default status value (which is *Joint*) if it has at least one neighbor v with $d_v = k$ (based on Lemma 4). Otherwise, node u stores the set of all possible imaginary edges based on rules 1 and 2 in E'_u (line 16). Node u then selects one of the subsets of E'_u that minimizes $\sum_{x,y \in \Gamma_v} x \overset{G_v}{\sim} y$ (Definition 3) and stores this subset in S_u (line 17).

After creating the k-connected local graph G^k_u using S_u, node u stores the G^k_u/u in G'_u (line 18) and sets its state to *Trusted* if G'_u is k-connected (line 19). Otherwise, u selects two nodes $\{v, z\} \in \Gamma_u$ with the minimum degree such that $v \overset{G'_u/u}{\sim} z < k$, detects the nodes that are already used in $k-1$ disjoint paths between v and z in G'_u, and stores them in set A (line 21). Afterward, u adds an edge between v and z, and adds z to the search target list T_u (line 22). Then, u starts a global path search process by broadcasting a $Discover(u, v, z, A)$ message (line 23). Node u sends a new $Discover$ message for each pair of selected nodes from G'_u until G'_u becomes k-connected. Finally, node u broadcasts its default (*Joint*) or updated (*Trusted*) state and its support degree (line 24).

When node u receives an unvisited $Discover(x, v, z, A)$ message from node w, it adds the message to the received messages set R_u and rebroadcasts the $Discover$ if the sender is the initiator of the search process (lines 25-27). If $u \in A$, then it adds an $Explore(x, y, z)$ to R_u to ignore the upcoming $Explore$ messages (line 29). If $v \in \Gamma_u$, then node u broadcasts an $Explore$ message after ts time units (line 30). The ts delay is to ensure that all nodes receive $Discover$ messages before receiving $Explore$ messages. The first piece of data in each $Explore$ message is the ID of the initiator node. The second and third data pieces are the IDs of the source and target nodes in the path. When node u receives an unvisited $Explore(x, v, z)$ from node w, it adds the message to R_u (lines 31-33). If u is the target of the search, it sends a $Confirm$ message to the initiator node (line 34). Otherwise, node u rebroadcasts the received $Explore$ message (line 35).

Algorithm 1 LINAR (k, β)

1: **Initially:**
   $I_u \leftarrow []$. //information of 2-hop neighbors in u
   $\Gamma_u \leftarrow .$ //1-hop neighbor list of u
   $V_u \leftarrow \{u\}, E_u \leftarrow .$ //nodes and edges sets of local subgraph
   $stat_u \leftarrow joint, sup_u \leftarrow 0$. //status and num. of independent neighbors
   $T_u \leftarrow , R_u \leftarrow .$ //target nodes and Received messages set

   // Initialization Phase

2: Node u starts the algorithm by **broadcasting** *Start*(pos_u).

3: *When u receives Start(pos_w) from w:*
4:   $\Gamma_u \leftarrow \Gamma_u \cup \{w\}, I_u[w].pos \leftarrow p_w$.
5:   **if** $|\Gamma_u| = 1$ **then call** *CreateGraph* after ts time units.

6: *procedure CreateGraph:*
7:   $\forall w \in \Gamma_u : V_u \leftarrow V_v \cup \{w\}$ *and* $E_u \leftarrow E_u \cup \{(w, u)\}$.
8:   **broadcast** *Ngb*(Γ_u, I_u).

9: *When u receives Ngb(Γ_w, I_w) from w:*
10:   $\forall v \in I_w : I_u[v] \leftarrow I_w[v]$.
11:   $\forall v \in \Gamma_w : V_u \leftarrow V_u \cup \{v\}$ *and* $E_u \leftarrow E_u \cup \{(v, w)\}$.
12:   **if** $\forall v \in \Gamma_u : v$ has sent Ngb **then call** *DetectState(u)*.

13: *procedure DetectState(u):*
14:   $G_u \leftarrow (V_v, E_v), sup_u \leftarrow Sup(u, G_u)$.
15:   **if** $\forall v \in \Gamma_u : d_v > k$ **then**
16:     $E'_u \leftarrow \{e = (i, j) \notin E_u : i, j \in V_u/\Gamma_u \text{ and } i \overset{G_u}{\sim} j < k\}$.
17:     $S_u \leftarrow \{S \subseteq E' : \kappa(G'(V_u, S \cup E_u)) = k \text{ and } \sum\limits_{x,y \in \Gamma_u} x \overset{G'}{\sim} y \text{ is min.}\}$.
18:     $G'_u \leftarrow (V_u, E_u \cup S_u)/u$.
19:     **if** $\kappa(G'_u) = k$ **then** $stat_u \leftarrow trusted$.
20:     **else while** $\kappa(G'_u) < k$ **do**

21:       $A \leftarrow dpath(G'_u, v, z)$ **where** $(v, z) \in \Gamma_u$ *and* $v \overset{G'_u/u}{\sim} z < k$.
22:       $G'_u \leftarrow (V_u, E_u \cup (v, z)), T_u \leftarrow T_u \cup \{z\}$.
23:       **broadcast** *Discover*(u, v, z, A).
24:   **broadcast** *Stat*$(stat_u, sup_u)$.

25: *When u receives Discover(x, v, z, A) from w:*
26:   **if** $u \neq z$ **and** $Discover(x, y, z) \notin R_u$ **then**
27:     $R_u \leftarrow R_u \cup Discover(x, y, z)$.
28:     **if** $x = w$ **then broadcast** *Discover*(x, v, z, A).
29:     **if** $u \in A$ **then** $R_u \leftarrow R_u \cup Explore(x, y, z)$.
30:     **else if** $v \in \Gamma_u$ **then broadcast** *Explore*(x, v, z) after ts time unit.

31: *When u receives Explore(x, v, z) from w:*
32:   **if** $Explore(x, y, z) \notin R_u$ **then**
33:     $R_u \leftarrow R_u \cup Explor(x, y, z)$.
34:     **if** $z = u$ **then send** *Confirm* to x.
35:     **else broadcast** *Explore*(x, v, z).

36: *When u receives Confirm from w:*
37:   $T_u \leftarrow T_u/w$.
38:   **if** $T_u = $ **then** $stat_u \leftarrow trusted$, **broadcast** *Stat*$(state_u, supp_u)$.

   // Recovery Phase

39: *When u receives Stat(s, i) from w:* $I_u[w].stat \leftarrow s, I_u[w].sup \leftarrow i$.
40: *When u detects failure in w,* $I_u[w].stat$ =*joint:* call *update(w)*.
41: *When u detects move of w,* $(I_u[w].sup - sup_u) \times \beta > 1$: call *update(w)*.

42: *procedure update(w):*
43:   **if** $stat_u$=*Trusted* **then**
44:     **if** $\nexists (w, v) \in E_u$ s.t $(I_u[v].stat= trusted$ **and** $LowerCost(v,w) = true)$
45:       **then move** to position $I_u[w].pos$.
46:   **else if** $I_u[w].stat$=*joint* **then**
47:     **if** $\nexists (w, v)$ **s.t** $(I_u[v].stat= Trusted$ **or** $LowerCost(v,w)=true)$
48:       **then move** to position $I_u[w].pos$.

49: *procedure LowerCost(v,w):*
50:   $c_v \leftarrow cost(I_u[v].pos, I_u[w].pos)/(1 + (I_u[v].sup * \beta))$.
51:   $c_u \leftarrow cost(pos_u, I_u[w].pos)/(1 + (sup_u * \beta))$.
52:   **if** $c_v < c_u$ **or** $(c_v = c_u$ **and** $v < u)$ **then return** *true*.
53:   **else return** *false*.

When node u receives a $Confirm$ message from w, it removes w from the target list and changes its state to *Trusted* if the target list becomes empty (lines 36-38). The nodes that change their status broadcast a $Stat$ message including their new states and support values (line 38). When node u receives a $Stat(s, i)$ message from its neighbor w, it updates the state and support values of w in its local set I_u (line 39). We assume that the nodes periodically broadcast a beacon packet (e.g., one packet per 2 s) to indicate that they are still alive. Therefore, when a node does not receive a beacon from a neighbor for a certain number of periods (e.g., five periods), it assumes that the neighbor is unavailable (e.g., it stopped working or moved away). When node u detects the failure of a *Joint* node w, it calls the *update* procedure to move to the failed node's position if there is no better candidate than itself (line 40). Similarly, node u calls the *update* procedure if it detects the movement of a neighbor node w with $(I_u[w].sup - sup_u) \times \beta > 1$ (line 41).

The *update* procedure (line 42) of node u accepts the ID of a target node w and moves node u to the position of w if there is no better candidate than u. If u is a trusted node and w has no trusted neighbors with a lower movement cost than u, then u moves to the position of w (lines 44-45). Otherwise, if u is a joint node and w has no trusted neighbors or joint neighbors with a lower moving cost, than u then u moves to the position of w (lines 46-48).

The $LowerCost$ procedure of node u accepts the IDs of two other nodes (v and w) and returns true if the coverage-aware movement cost of u to w is lower than the coverage-aware movement cost of v to w or if their movement costs are equal but the ID of u is smaller than the ID of v (as elaborated in Section IV-B). Otherwise, this procedure returns false (lines 49-53). The movement cost of u to w is calculated in a similar manner. The procedure returns true if $c_v < c_u$ or ($c_v = c_u$ and $v < u$).

D. Complexity Analysis

In this subsection, we provide theorems on the bit, time, space, and computational complexities of LINAR.

Theorem 2: The bit complexities of LINAR are $O(n^2 \times log_2 n)$ and $\Omega(n \times k \times log_2 n)$ for the worst and best cases, respectively.

Theorem 3: The space complexity of LINAR is $O(\Delta^4 + n \times \Delta)$.

Theorem 4: The time complexity of LINAR is $O(D)$.

Theorem 5: The computational complexity of LINAR is $O(n^2 \times \Delta)$.

V. PERFORMANCE EVALUATION

We implemented LINAR, MCCR, and TAPU on testbed environments along with a *Localized*, a *Greedy*, and a *Basic* central algorithm to compare their performances with LINAR. In the *Localized* approach, the sink node selects a neighbor of the failed node with the smallest degree and moves it to the position of the failed node and repeats this process until the resulting topology becomes k-connected. The sink node sends a message including the target position to each node that should move to a new position. In the *Greedy* algorithm, the sink node selects the nearest neighbor (the neighbor with the minimum moving cost) of the failed node and moves it to the position of the failed node (by sending a message
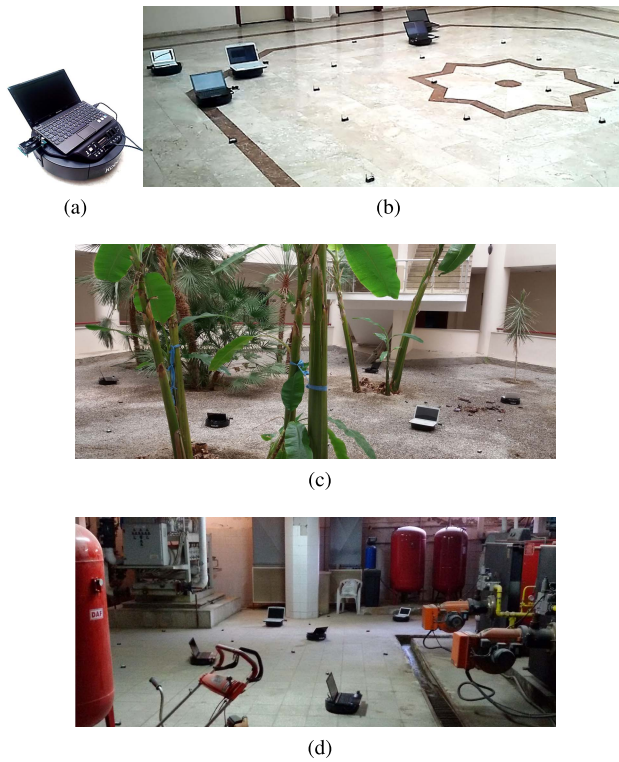
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

AKRAM *et al.*: COVERAGE-AWARE DISTRIBUTED k-CONNECTIVITY MAINTENANCE ALGORITHM

9

(a)   (b)

(c)

(d)

Fig. 6.   a) A mobile node utilized in experiments b) hall testbed, c) garden testbed, d) powerhouse testbed.

Fig. 7.   a) Wallclock times b) sent bytes c) movement costs.

similar to the *Localized* approach) and repeats this process until the resulting network becomes k-connected. In the *Basic* algorithm, we assume that there are a sufficient number of redundant nodes and after each failure, one of these nodes is sent to the position of the failed node.

All nodes broadcast a beacon packet every 2 s to inform that they are alive. A node is marked as failed by its neighbors if it does not broadcast a beacon for 10 s. In central algorithms, all nodes send their neighbor lists to the sink node and the sink node creates a graph for the entire network topology, furthermore, the nodes that detect a failure send the failed node ID to the sink and wait for the movement messages from the sink. In the experiments, we counted the total sent bytes of all messages except the beacon packets. We used Crossbow IRIS motes [42] and Kobuki [43] robots to create MSNs. We integrated an IRIS mote with a Kobuki robot using a small laptop to create a mobile node (Fig. 6a). We established 15 random networks in three different environments (five networks for each type of terrain) and set the communication range of all nodes to 1.5 m to establish the desired topologies. We established MSNs with 5 mobile and 15 static nodes in a hall with marble flooring (Fig. 6b), a garden with a soil surface (Fig. 6c), and a powerhouse with mosaic flooring (Fig. 6d). Because of the limited number of available Kobuki robots and IRIS nodes, we established networks with $k = 1$, 2, and 3 with $\beta = 0$ in all experiments. However, we performed simulations with different β and k values. In all experiments and simulations, we set the moving cost function of nodes
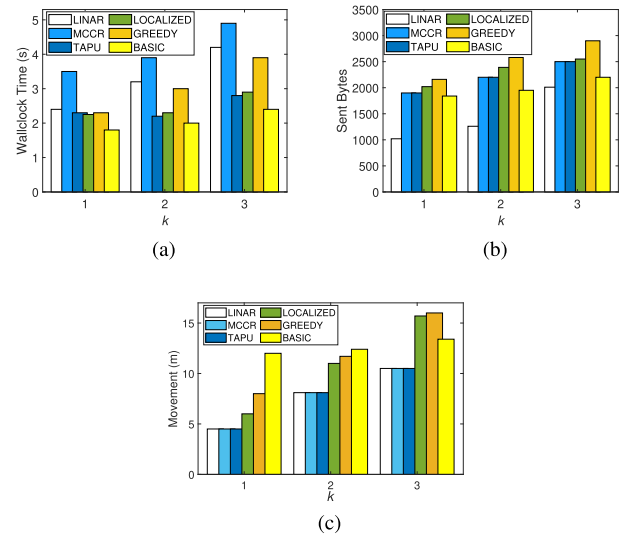
to the Euclidean distance between their source and target locations.

Fig. 7a shows the average wallclock times of the algorithms in the testbed experiments. The moving times of nodes from the source to the target position are not included in the wallclock times. The wallclock times of MCCR are higher than those of all other algorithms and the *Basic* algorithm has the best performance. The wallclock times of LINAR are lower than those of MCCR. Fig. 7b shows the sent bytes of algorithms after failures in two nodes in different topologies. The sent bytes of LINAR are significantly lower than those of the other algorithms because in this algorithm the topology information is not sent to a single node. For $k = 1$, the total sent bytes of LINAR are at least 47.1% lower than those of the other algorithms. For $k = 3$, this ratio is approximately 8.3%. Fig. 7c shows the average movement costs of the implemented algorithms after failures in two *Joint* nodes. LINAR, MCCR, and TAPU generate movements with lower costs than the *Greedy*, *Basic*, and *Localized* algorithms in all k values.

To measure the performances of the algorithms in large-scale networks, we simulated the algorithms using Java on geometric bidirectional weighted graphs in a field of 1000 m × 1000 m area. In the generated random networks, the weight of each edge is the movement cost between the associated nodes. The transmission range of each node is set to 20 m and five classes of topologies with 50, 100, 150, 200, and 250 nodes are generated. For each class, we generated random topologies with different k values from 1 to 7. For each specific node count and k value, we created 10 random topologies.

After starting the algorithms, we stopped 20% of the nodes randomly and measured the total sent bytes, movement costs, wallclock times, and coverage area before and after the restoration for $\beta = 0$, 0.3, 0.6, and 0.9. We saved each topology, before and after the restoration, in different text files and used the Shapely and Matplotlib libraries of Python to calculate the coverage area. Since the other k-connectivity restoration algorithms do not consider the coverage constraint, for the sake
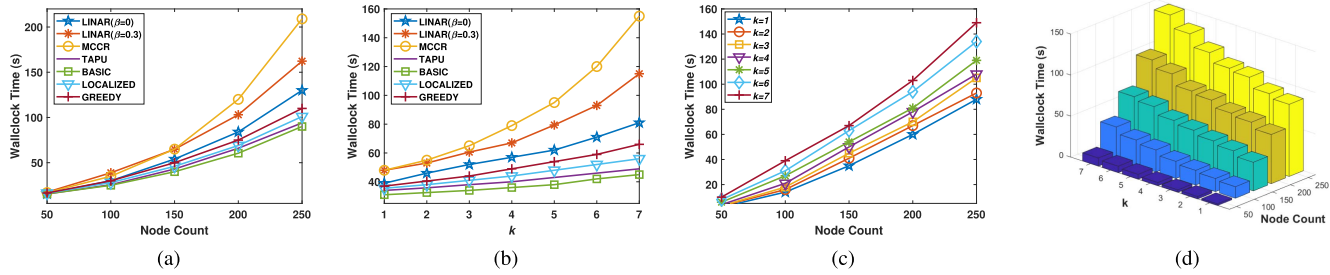
Fig. 8. Wallclock times of: a) algorithms against node count, b) algorithms against k, c) LINAR ($\beta = 0$) against node count, d) LINAR ($\beta = 0$) against node count and k.
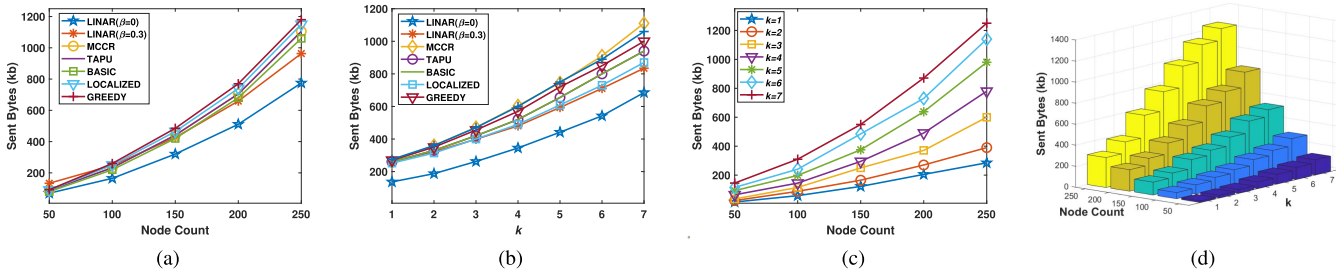

Fig. 9. Average sent bytes of: a) algorithms against node count, b) algorithms against k, c) LINAR ($\beta = 0$) against node count, d) LINAR ($\beta = 0$) against node count and k.
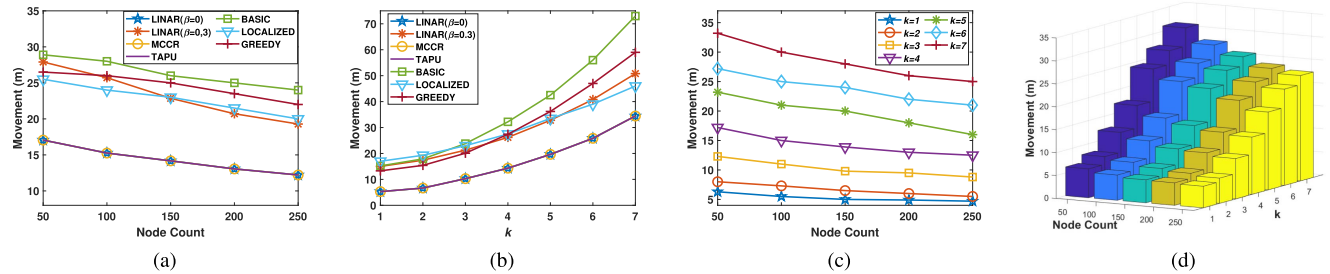

Fig. 10. Movement costs of: a) algorithms against node count, b) algorithms against k, c) LINAR ($\beta = 0$) against node count, d) LINAR ($\beta = 0$) against node count and k.

of fairness, we first compared the performances of LINAR with $\beta = 0$ (no coverage constraint) and $\beta = 0.3$ with the existing algorithms and then evaluated the performance of LINAR for different β values, in detail.

MCCR has the highest wallclock time taking approximately 214 s in the networks with 250 nodes (Fig. 8a). After MCCR, LINAR with $\beta = 0.3$ and $\beta = 0$ exhibits the next highest wallclock times. Note that none of the algorithms have any coverage constraint except LINAR with $\beta = 0.3$. In the networks with 50 nodes, the wallclock times of LINAR with $\beta = 0$ are less than 10 s and increase up to 131 s in the networks with 250 nodes, which are approximately 38.7% lower than MCCR and 19.1% higher than the *Greedy* algorithm. Increasing the k value increases the wallclock time of MCCR higher than the other algorithms (Fig. 8b). The *Basic* algorithm is the fastest algorithm taking less than 45 s for all k values. The wallclock time of LINAR with $\beta = 0$ is up to 47.3% lower than that of MCCR and grows almost linearly with increasing k values. In the worst case, for $k = 7$,
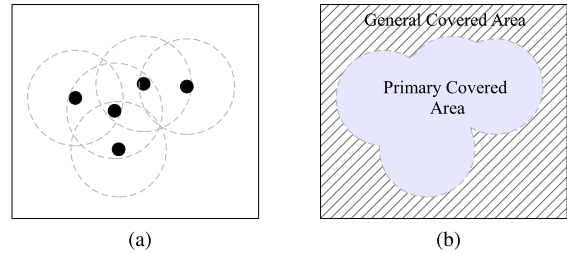

Fig. 11. Primary versus general coverage.

the wallclock time of LINAR with $\beta = 0$ is approximately 43.8% higher than that of the fastest implemented algorithm. LINAR runs faster for small k values, but the gaps between different k values are limited (Fig. 8c). In networks with 250 nodes, LINAR with $\beta = 0$ takes approximately 89 s for $k = 1$, whereas this value for $k = 7$ is approximately 150 s, which indicates that LINAR exhibits a stable behavior in all topologies. The wallclock time of LINAR moderately
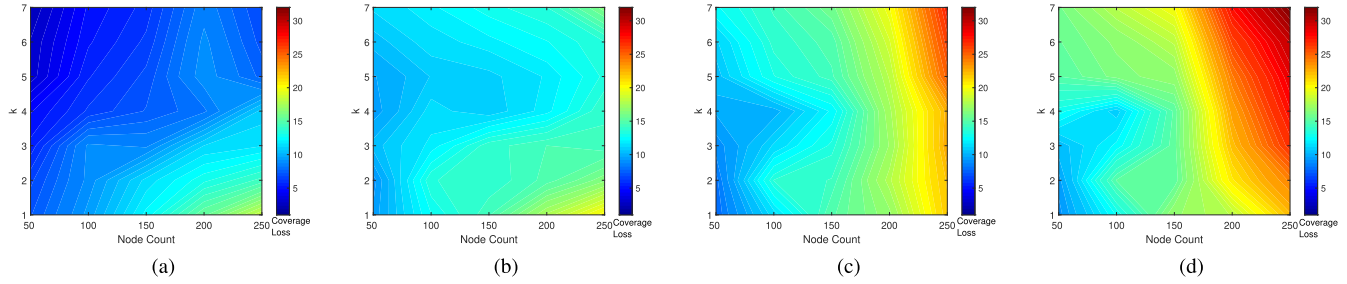
Fig. 12.   Primary coverage loss after node failures: a) without restoration, b) with $\beta = 0.6$, c) with $\beta = 0.3$, d) with $\beta = 0$.

increases when we increase the node count, whereas wallclock times grow slowly with increasing k values (Fig. 8d).

Sending neighborhood information of all nodes to a single node increases the total sent bytes of the central approaches up to 34.3% more than the sent bytes of LINAR with $\beta = 0$ (Fig. 9a). In the *Basic* algorithm, the sink node sends a 1-hop message after each failure, hence, its average sent bytes are lower than those of the other central algorithms. In the *Localized*, *Greedy*, MCCR, and TAPU algorithms, the sink can send more than one multi-hop messages after each node failure. The *Basic*, MCCR, and TAPU algorithms send less bytes than the *Greedy* and *Localized* algorithms, but their sent bytes are at least 19.8% higher than LINAR for all k values (Fig. 9b). LINAR with $\beta = 0.3$ always sends more bytes than LINAR with $\beta = 0$ because LINAR with $\beta = 0.3$ is the only algorithm that has a coverage constraint, unlike all the others. The sent bytes of LINAR with $\beta = 0$ for small k values are lower than the sent bytes for higher k values because for smaller k values more nodes can find their status with the proposed local relations (Fig. 9c). In networks with 250 nodes, LINAR with $\beta = 0$ sends approximately 286 kb on average for $k = 1$, whereas this value is approximately 1252 kb for $k = 7$. In networks with 50 nodes, LINAR sends less bytes and the effects of k values on the sent bytes are not significant, yet, increasing the number of nodes increases the amount of sent bytes (Fig. 9d).

Fig. 10a shows the average movement costs of all algorithms against the node count. In networks with 50 nodes, MCCR, TAPU, and LINAR (with $\beta = 0$) algorithms generate approximately 17 m movement on average whereas for the same topologies the average movements of the *Localized*, *Greedy*, and *Basic* algorithms are higher than 26 m (52.9% higher than LINAR). By increasing the node count, the probability of a *Joint* node failure and also the distance between the nodes decrease which leads to shorter movements. In networks with 250 nodes, the average movement costs of MCCR, TAPU, and LINAR (with $\beta = 0$) are less than 12 m, whereas the average movement costs of the other three algorithm are higher than 20 m. Fig. 10b compares the average movement costs of the algorithms against the k values. The movement costs of MCCR, TAPU, and LINAR (with $\beta = 0$) are up to 34% lower than those of the other algorithms. LINAR with $\beta = 0.3$ generates more movements than LINAR with $\beta = 0$ because only this algorithm considers the coverage constraint. The average movement cost of LINAR for different k values

is presented in Fig. 10c. In general, for lower k values the networks have fewer *Joint* nodes than networks with higher k values. Hence, the probability of a *Joint* node failure for smaller k values is lower. Fig. 10c shows that for $k = 1$, the average movement cost is approximately 7 m in networks with 50 nodes. This cost is reduced to approximately 5 m in networks with 250 nodes. For $k = 7$, the average movement cost of LINAR varies between 33 m and 25 m. Fig. 10d reveals that the k value is more dominant for movement cost than for the node count. Increasing the node count slowly reduces the movement cost whereas increasing the k value rapidly increases the movement cost.

We measured the coverage loss of the network before and after the restoration for two cases (i.e., the primary coverage loss and the general coverage loss). Fig. 11a shows a sample MSN and Fig. 11b shows the difference between the primary and general coverage. We assumed that the sensing and communications ranges are equal [44]. We run LINAR with $\beta = 0$, $\beta = 0.3$, $\beta = 0.6$, and $\beta = 0.9$, however, because of the negligible difference between the results of $\beta = 0.6$ and $\beta = 0.9$, we present the results up to $\beta = 0.6$.

Fig. 12 shows the primary coverage loss percentage after node failures. Fig. 12a shows that we lose 3% (for $n = 50$ and $k = 7$) up to 16% (for $n = 250$ and $k = 1$) of the initial covered area without performing any restoration after the failures of 20% of the nodes. After k-connectivity restoration, we lose approximately 8% (for $n = 50$) up to 19% (for $n = 250$ and $k = 1$) of the covered area with $\beta = 0.6$ (Fig. 12b) and approximately 9% (for $n = 50$ and $k = 1$) up to 28% (for $n = 250$ and $k > 5$) with $\beta = 0.3$ (Fig. 12c). For $\beta = 0$ (Fig. 12d) the coverage loss varies between 10% (for $n = 50$ and $k = 1$) and 33% (for $n = 250$ and $k > 5$).

Fig. 13 shows the general coverage loss after node failures. Fig. 13a shows that we lose 1% (for $n = 50$ and $k > 2$) to 4% (for $n = 250$ and $k = 1$) of the covered area without restoration after failures of 20% of nodes. After k-connectivity restoration, we lose approximately 3% (for $n = 50$) to 5% (for $n = 250$ and $k = 1$) of covered area with $\beta = 0.6$ (Fig. 13b) and about 3% (for $n = 50$ and $k < 3$) up to 6% (for $n = 250$ and $k > 4$) with $\beta = 0.3$ (Fig. 13c). For $\beta = 0$ (Fig. 13d), the coverage loss percentage varies between 3% (for $n = 50$ and $k = 1$) and 8% (for $n = 250$ and $k > 5$).

Figs. 14a and 14b show the total sent bytes of LINAR with $\beta = 0$ and $\beta = 0.6$, which reveal that increasing β has a negligible effect on the sent bytes of LINAR. Figs. 14c and 14d

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

12   IEEE/ACM TRANSACTIONS ON NETWORKING
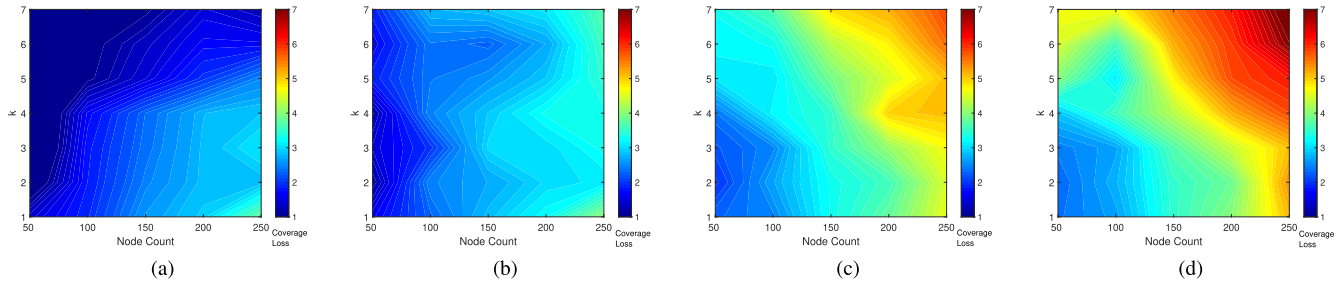
Fig. 13.   General coverage loss after node failures: a) without restoration, b) with $\beta = 0.6$, c) with $\beta = 0.3$, d) with $\beta = 0$.
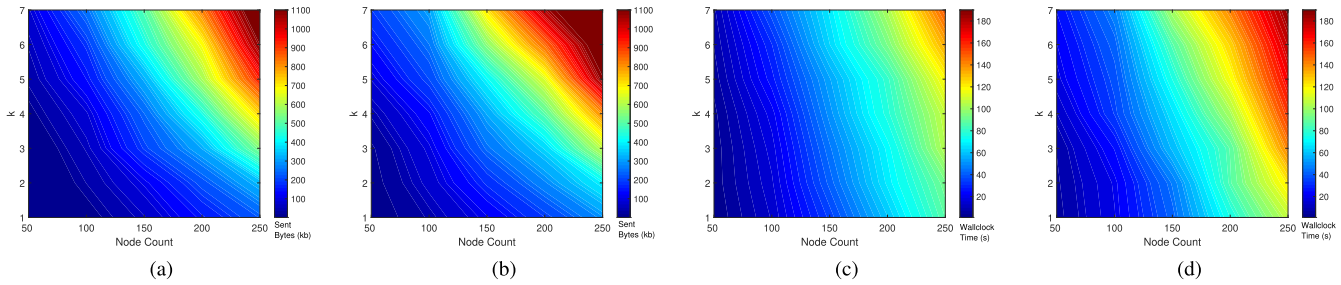

Fig. 14.   Sent bytes of LINAR: a) with $\beta = 0$, b) with $\beta = 0.6$; wallclock times of LINAR: c) with $\beta = 0$, d) with $\beta = 0.6$.
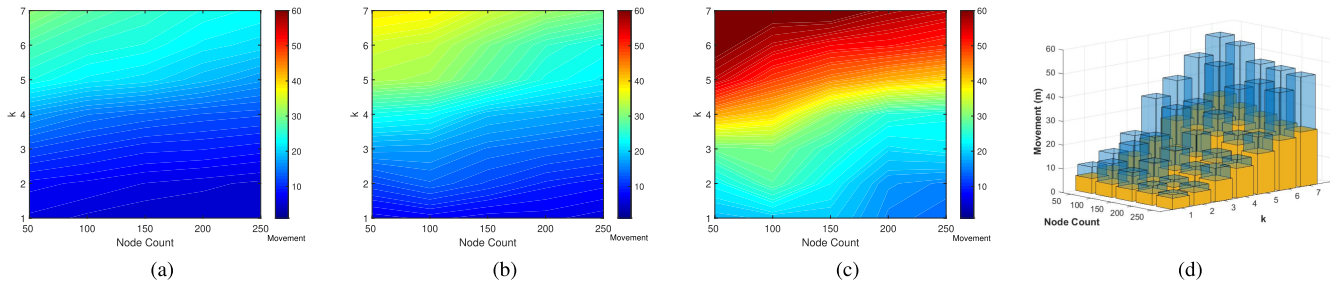

Fig. 15.   LINAR movements: a) with $\beta = 0$, b) with $\beta = 0.3$, c) with $\beta = 0.6$, d) with $\beta = 0$ (yellow) and $\beta = 0.6$ (blue).

show the wallclock times of LINAR with $\beta = 0$ and $\beta = 0.6$, which indicate that the difference between $\beta = 0$ and $\beta = 0.6$ is negligible.

Fig. 15 presents the total generated movements by LINAR for different β values. Fig. 15a shows that the generated movements of the algorithm with $\beta = 0$ vary between 4 m (for $n = 250$ and $k = 1$) and 32 m (for $n = 50$ and $k = 7$). The generated movements with $\beta = 0.3$ (Fig. 15b) vary between 6 m (for $n = 250$ and $k = 1$) and 39 m (for $n = 50$ and $k = 7$). With $\beta = 0.6$ (Fig. 15c) the algorithm generates movements from 9 m (for $n = 250$ and $k = 1$) to 66 m (for $n = 50$ and $k = 7$). Fig. 15d provides a comparison between the generated movements with $\beta = 0$ and $\beta = 0.6$, which shows that preserving the coverage and k-connectivity considerably increases the movement cost.

VI. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

In MSNs, connectivity and coverage should be maintained to achieve reliability and functionality. In this study, we present LINAR, which is a distributed algorithm for coverage-aware movement-based k-connectivity restoration in MSNs. Our algorithm is the first distributed movement-based

k-connectivity restoration algorithm designed for arbitrarily large k values, whereas all such algorithms in the literature are designed for $k \leq 2$. We introduce the theoretical foundations and show that most of the nodes can find their status using their local subgraphs. After a failure in a *Joint* node, a *Trusted* node or a chain of *Joint* nodes change their position(s) to preserve the k value with the minimum movement cost. LINAR is also capable of providing coverage preservation by sacrificing moderately from the movement minimization objective. In fact, the coverage conservation ratio (β), which is an input parameter, determines the extent of the tradeoff between coverage conservation and movement minimization. The comprehensive experimental and simulation results reveal that LINAR can restore k-connectivity with significantly better performance than the other algorithms in the literature. Furthermore, the obtained measurements also reveal that the difference between the primary coverage loss percentages before and after the restoration is, at most, 3% for $\beta = 0.6$.

Although we assume that paths and mobility costs between different locations are readily available to the nodes, in many real-word deployment scenarios, nodes have to discover paths

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

AKRAM *et al.*: COVERAGE-AWARE DISTRIBUTED k-CONNECTIVITY MAINTENANCE ALGORITHM

13

and movement costs to other locations, which itself would not be available without significant exploration. In our algorithm, the movement paths are from one location to another established location, however, moving to a new location can result in better coverage and retain k-connectivity. In many real-life MSN deployments, accommodating sensing models other than a disk-shaped sensing abstraction (as in this study) is necessary. However, in this case, it is challenging to estimate what each node senses. In this study, our priority is to restore k-connectivity in a coverage-aware manner. However, it can also be argued that, for most applications, connectivity/coverage should be restored only after a part of the network becomes disconnected or after coverage of some area is lost. The proposed algorithm guarantees k-connectivity restoration after a node failure. Yet, our algorithm cannot guarantee the restoration of k-connectivity for all failure scenarios (e.g., when all neighbors of the sink node fail simultaneously). Nevertheless, all of the aforementioned considerations warrant future research.

REFERENCES

[1] L. Song, C. Liu, H. Huang, H. Du, and X. Jia, "Minimum connected dominating set under routing cost constraint in wireless sensor networks with different transmission ranges," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 546–559, Apr. 2019.

[2] M. R. Henzinger, S. Rao, and H. N. Gabow, "Computing vertex connectivity: New bounds from old techniques," *J. Algorithms*, vol. 34, no. 2, pp. 222–250, Feb. 2000.

[3] P. Szczytowski, A. Khelil, and N. Suri, "DKM: Distributed k-connectivity maintenance in wireless sensor networks," in *Proc. 9th Annu. Conf. Wireless Demand Netw. Syst. Services (WONS)*, Jan. 2012, pp. 83–90.

[4] V. K. Akram and O. Dagdeviren, "DECK: A distributed, asynchronous and exact k-connectivity detection algorithm for wireless sensor networks," *Comput. Commun.*, vol. 116, pp. 9–20, Jan. 2018.

[5] K. Censor-Hillel, M. Ghaffari, and F. Kuhn, "Distributed connectivity decomposition," in *Proc. ACM Symp. Princ. Distrib. Comput. (PODC)*, 2014, pp. 156–165.

[6] O. Dagdeviren, V. K. Akram, and B. Tavli, "Design and evaluation of algorithms for energy efficient and complete determination of critical nodes for wireless sensor network reliability," *IEEE Trans. Rel.*, vol. 68, no. 1, pp. 280–290, Mar. 2019.

[7] J. L. Bredin, E. D. Demaine, M. T. Hajiaghayi, and D. Rus, "Deploying sensor networks with guaranteed fault tolerance," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 216–228, Feb. 2010.

[8] F. Deniz, H. Bagci, I. Korpeoglu, and A. Yazıcı, "An adaptive, energy-aware and distributed fault-tolerant topology-control algorithm for heterogeneous wireless sensor networks," *Ad Hoc Netw.*, vol. 44, no. 7, pp. 104–117, 2016.

[9] S. Lee, M. Younis, and M. Lee, "Connectivity restoration in a partitioned wireless sensor network with assured fault tolerance," *Ad Hoc Netw.*, vol. 24, pp. 1–19, Jan. 2015.

[10] Y. Zeng, L. Xu, and Z. Chen, "Fault-tolerant algorithms for connectivity restoration in wireless sensor networks," *Sensors*, vol. 16, no. 1, pp. 3:1–3:15, Dec. 2015.

[11] H. M. Almasaeid and A. E. Kamal, "On the minimum k-connectivity repair in wireless sensor networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2009, pp. 1–5.

[12] X. Han, X. Cao, E. L. Lloyd, and C.-C. Shen, "Fault-tolerant relay node placement in heterogeneous wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 9, no. 5, pp. 643–656, May 2010.

[13] V. Ranga, M. Dave, and A. K. Verma, "Relay node placement for lost connectivity restoration in partitioned wireless sensor networks," in *Proc. Int. Conf. Electron. Commun. Syst. (ECS)*, vol. 2015, pp. 170–175.

[14] S. Lee, M. Younis, B. Anglin, and M. Lee, "LEEF: Latency and energy efficient federation of disjoint wireless sensor segments," *Ad Hoc Netw.*, vol. 71, pp. 88–103, Mar. 2018.

[15] S. Shriwastav and D. Ghose, "Round-table negotiation for fast restoration of connectivity in partitioned wireless sensor networks," *Ad Hoc Netw.*, vol. 77, pp. 11–27, Aug. 2018.

[16] X. Liu, A. Liu, T. Qiu, B. Dai, T. Wang, and L. Yang, "Restoring connectivity of damaged sensor networks for long-term survival in hostile environments," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 1205–1215, Feb. 2020.

[17] K. P. Sharma and T. P. Sharma, "Distributed connectivity restoration in networks of movable sensor nodes," *Comput. Electr. Eng.*, vol. 56, pp. 608–629, Nov. 2016.

[18] N. Atay and B. Bayazit, "Mobile wireless sensor network connectivity repair with k-redundancy," in *Algorithmic Foundation of Robotics VIII* (Springer Tracts in Advanced Robotics), G. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. Berlin, Germany: Springer, 2009, pp. 35–49.

[19] H. Wang, X. Ding, C. Huang, and X. Wu, "Adaptive connectivity restoration from node failure(s) in wireless sensor networks," *Sensors*, vol. 16, no. 10, p. 1487, Sep. 2016.

[20] Y. Zhang, J. Wang, and G. Hao, "An autonomous connectivity restoration algorithm based on finite state machine for wireless sensor-actor networks," *Sensors*, vol. 18, no. 2, p. 153, Jan. 2018.

[21] K. Akkaya, F. Senel, A. Thimmapuram, and S. Uludag, "Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 258–271, Feb. 2010.

[22] D. V. Dimarogonas and K. H. Johansson, "Decentralized connectivity maintenance in mobile networks with bounded inputs," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2008, pp. 1507–1512.

[23] X. Liu, "Survivability-aware connectivity restoration for partitioned wireless sensor networks," *IEEE Commun. Lett.*, vol. 21, no. 11, pp. 2444–2447, Nov. 2017.

[24] P. Yang, R. A. Freeman, G. J. Gordon, K. M. Lynch, S. S. Srinivasa, and R. Sukthankar, "Decentralized estimation and control of graph connectivity for mobile sensor networks," *Automatica*, vol. 46, no. 2, pp. 390–396, 2010.

[25] E. Uzun, F. Senel, K. Akkaya, and A. Yazici, "Distributed connectivity restoration in underwater acoustic sensor networks via depth adjustment," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 6357–6362.

[26] I. F. Senturk, K. Akkaya, and S. Janansefat, "Towards realistic connectivity restoration in partitioned mobile sensor networks," *Int. J. Commun. Syst.*, vol. 29, no. 2, pp. 230–250, Jan. 2016.

[27] D. Zhang, W. Shi, R. Elhabyan, and M. St-Hilaire, "A coverage and obstacle-aware clustering protocol for wireless sensor networks in 3D terrain," *Comput. Commun.*, vol. 146, pp. 48–54, Oct. 2019.

[28] Z. Zhang, J. Willson, Z. Lu, W. Wu, X. Zhu, and D.-Z. Du, "Approximating maximum lifetime k-coverage through minimizing weighted k-cover in homogeneous wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3620–3633, Dec. 2016.

[29] R. Elhabyan, W. Shi, and M. St-Hilaire, "Coverage protocols for wireless sensor networks: Review and future directions," *J. Commun. Netw.*, vol. 21, no. 1, pp. 45–60, Feb. 2019.

[30] Z. Zhou, S. Das, and H. Gupta, "Connected K-coverage problem in sensor networks," in *Proc. 13th Int. Conf. Comput. Commun. Netw.*, Oct. 2004, pp. 373–378.

[31] S. Wang, X. Mao, S.-J. Tang, X. Li, J. Zhao, and G. Dai, "On 'movement-assisted connectivity restoration in wireless sensor and actor networks,'" *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 4, pp. 687–694, Apr. 2011.

[32] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logistics Quart.*, vol. 2, nos. 1–2, pp. 83–97, Mar. 1955.

[33] V. K. Akram and O. Dağdeviren, "TAPU: Test and Pick up-based k-connectivity restoration algorithm for wireless sensor networks," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 27, no. 2, pp. 985–997, 2019.

[34] V. K. Akram, O. Dagdeviren, and B. Tavli, "Distributed k-connectivity restoration for fault tolerant wireless sensor and actuator networks: Algorithm design and experimental evaluations," *IEEE Trans. Rel.*, early access, Mar. 16, 2020, doi: 10.1109/TR.2020.2970268.

[35] X. Bai, D. Xuan, Z. Yun, T. H. Lai, and W. Jia, "Complete optimal deployment patterns for full-coverage and k-connectivity (k≤6) wireless sensor networks," in *Proc. 9th ACM Int. Symp. Mobile ad hoc Netw. Comput. (MobiHoc)*, 2008, pp. 401–410.

[36] Z. Yun, X. Bai, D. Xuan, T. H. Lai, and W. Jia, "Optimal deployment patterns for full coverage and k-connectivity (k≤6) wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 18, no. 3, pp. 934–947, Jun. 2010.

[37] D. Tian and N. D. Georganas, "Connectivity maintenance and coverage preservation in wireless sensor networks," *Ad Hoc Netw.*, vol. 3, no. 6, pp. 744–761, 2005.

[38] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 35, no. 2, pp. 619–632, 2012.

[39] J. Li, L. Andrew, C. Foh, M. Zukerman, and H.-H. Chen, "Connectivity, coverage and placement in wireless sensor networks," *Sensors*, vol. 9, no. 10, pp. 7664–7693, Sep. 2009.

[40] S. Sengupta, S. Das, M. D. Nasir, and B. K. Panigrahi, "Multi-objective node deployment in WSNs: In search of an optimal trade-off among coverage, lifetime, energy consumption, and connectivity," *Eng. Appl. Artif. Intell.*, vol. 26, no. 1, pp. 405–416, 2013.

[41] J. N. Al-Karaki and A. Gawanmeh, "The optimal deployment, coverage, and connectivity problems in wireless sensor networks: Revisited," *IEEE Access*, vol. 5, pp. 18051–18065, 2017.

[42] *IRIS Datasheet*. Accessed: Nov. 17, 2020. [Online]. Available: http://www.memsic.com/

[43] *Yujinrobot Kobuki*. Accessed: Nov. 17, 2020. [Online]. Available: http://kobuki.yujinrobot.com/

[44] C.-F. Huang and Y.-C. Tseng, "The coverage problem in a wireless sensor network," *Mobile Netw. Appl.*, vol. 10, no. 4, pp. 519–528, 2005.

Orhan Dagdeviren (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from Izmir Institute of Technology and the Ph.D. degree from the International Computer Institute, Ege University. He is currently an Associate Professor and the Director of the Network Science Laboratory, International Computer Institute. His interests lie in distributed computing, applied graph theory, and computer networking areas.

Vahid Khalilpour Akram (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer engineering from Islamic Azad University and the Ph.D. degree from the International Computer Institute, Ege University. He is currently an Assistant Professor with the International Computer Institute, Ege University. His research interests include wireless sensor networks, parallel and distributed algorithms, and graph theory.

Bulent Tavli (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Rochester, Rochester, NY, USA, in 2005. He is currently a Professor with the Department of Electrical and Electronics Engineering, TOBB University of Economics and Technology, Ankara, Turkey. His research interests include network science, algorithm design and analysis, telecommunications, privacy and security, smart grids, embedded systems, optimization, machine learning, digital twins, and blockchain.