Recovering From a Node Failure in Wireless Sensor-Actor Networks With Minimal Topology Changes

Ameer A. Abbasi, Mohamed F. Younis, *Senior Member, IEEE*, and Uthman A. Baroudi

***Abstract*—In wireless sensor-actor networks, sensors probe their surroundings and forward their data to actor nodes. Actors collaboratively respond to achieve predefined application mission. Since actors have to coordinate their operation, it is necessary to maintain a strongly connected network topology at all times. Moreover, the length of the inter-actor communication paths may be constrained to meet latency requirements. However, a failure of an actor may cause the network to partition into disjoint blocks and would, thus, violate such a connectivity goal. One of the effective recovery methodologies is to autonomously reposition a subset of the actor nodes to restore connectivity. Contemporary recovery schemes either impose high node relocation overhead or extend some of the inter-actor data paths. This paper overcomes these shortcomings and presents a Least-Disruptive topology Repair (LeDiR) algorithm. LeDiR relies on the local view of a node about the network to devise a recovery plan that relocates the least number of nodes and ensures that no path between any pair of nodes is extended. LeDiR is a localized and distributed algorithm that leverages existing route discovery activities in the network and imposes no additional prefailure communication overhead. The performance of LeDiR is analyzed mathematically and validated via extensive simulation experiments.**

***Index Terms*—Fault tolerance, network recovery, topology management, wireless sensor-actor network (WSAN).**

I. Introduction

RECENT years have witnessed a growing interest in the applications of wireless sensor-actor networks (WSANs). Of particular interest are applications in remote and harsh areas in which human intervention is risky or impractical. Examples include space exploration, battle field surveillance, search-and-research, and coastal and border protection. A WSAN consists of a set of miniaturized low-cost sensors that are spread in an area of interest to measure ambient conditions in the vicinity. The sensors serve as wireless data acquisition devices for the more powerful actor nodes that process the sensor readings and put forward an appropriate response. For example, sensors may detect a fire and trigger a response from an actor that has an extinguisher. Robots and unmanned vehicles are example actors in practice [1]. Actors work autonomously and collaboratively to achieve the application mission.

Given the collaborative actors' operation, a strongly connected inter-actor network topology would be required at all times. Actors usually coordinate their motion so that they stay reachable to each other. However, a failure of an actor may cause the network to partition into disjoint blocks and would thus violate such a connectivity requirement. The remote setup in which WSANs often serve makes the deployment of additional resources to replace failed actors impractical, and repositioning of nodes becomes the best recovery option [2]. In addition, tolerance of node failure cannot be orchestrated through a centralized scheme given the autonomous operation of the network. On the other hand, distributed recovery will be very challenging since nodes in separate partitions will not be able to reach each other to coordinate the recovery process. Therefore, contemporary schemes found in the literature require every node to maintain partial knowledge of the network state. To avoid the excessive state-update overhead and to expedite the connectivity restoration process, prior work relies on maintaining one- or two-hop neighbor lists and predetermines some criteria for the node's involvement in the recovery [3]–[5]. However, one-hop-based schemes often impose high node repositioning overhead, and the repaired inter-actor topology using two-hop schemes may differ significantly from its prefailure status.

Unlike prior work, this paper considers the connectivity restoration problem subject to path length constraints. Basically, in some applications, such as combat robotic networks and search-and-rescue operation, timely coordination among the actors is required, and extending the shortest path between two actors as a side effect of the recovery process would not be acceptable. For example, interaction among actors during a combat operation would require timeliness to accurately track and attack a fast moving target. A novel Least-Disruptive topology Repair (LeDiR) algorithm is proposed. LeDiR relies on the local view of a node about the network to relocate the least number of nodes and ensure that no path between any pair of affected nodes is extended relative to its prefailure status. LeDiR is a localized and distributed algorithm that leverages

Manuscript received September 23, 2011; revised February 20, 2012 and May 25, 2012; accepted July 10, 2012. Date of publication August 9, 2012; date of current version January 14, 2013. This work was supported in part by the King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, under Project IN090006 and in part by the National Science Foundation under Award CNS 1018171. The review of this paper was coordinated by Dr. L. Li.

A. A. Abbasi and U. A. Baroudi are with the Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia (e-mail: ameer_abbasi@kfupm.edu.sa; ameer@sofdigital.com; ubaroudi@kfupm.edu.sa).

M. F. Younis is with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD 21250 USA (e-mail: younis@cs.umbc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2012.2212734

existing route discovery activities in the network and imposes no additional prefailure communication overhead.

When a node fails, its neighbors will individually consult their possibly incomplete routing table to decide on the appropriate course of actions and define their role in the recovery if any. If the failed node is critical to the network connectivity, i.e., a node whose failure causes the network to partition into disjoint blocks, the neighbor that belongs to the smallest block reacts. The performance of LeDiR is validated both analytically and through simulation. The simulation results demonstrate that LeDiR outperforms existing schemes in terms of communication and relocation overhead.

The next section describes the assumed system model and defines the considered problem. Section III gives an overview of related work. Section IV explains LeDiR in detail. Section V describes the validation experiments and analyzes the simulation results. The paper is concluded in Section VI.

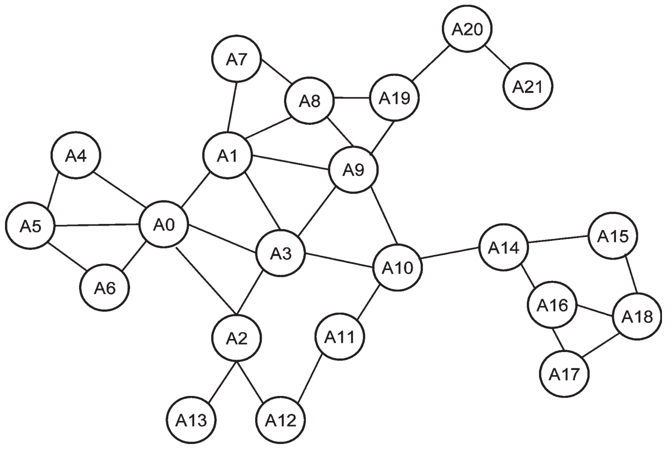


Fig. 1. Example one-connected inter-actor network. Nodes A_0, A_{10}, A_{14}, and A_{19} are cut vertices whose failure leaves the network partitioned into two or multiple disjoint blocks.

II. System Model and Problem Statement

As mentioned earlier, a WSAN involves two types of nodes: 1) sensors and 2) actors. Sensors are inexpensive and highly constrained in energy and processing capacity. On the other hand, actors are more capable nodes with relatively more onboard energy supply and richer computation and communication resources. However, the transmission range of actors is finite and significantly less than the dimensions of the deployment area. Although actors can theoretically reach each other via a satellite channel, the frequent inter-actor interaction required by WSAN applications would make the often intermittent satellite links unsuitable. It is thus necessary for actors to rely mostly on contemporary terrestrial radio links for coordination among themselves. Upon deployment, actors are assumed to discover each other and form a one-connected network using some of the existing techniques, such as in [6]. An actor employs ranging technologies and localization techniques to determine its position relative to its neighbor [7]. We assume that the actors can move on demand to perform tasks on larger areas or to enhance the inter-actor connectivity. Given the application-based interaction, an actor is assumed to know how many actors are there in the network. The focus of this paper is on restoring strong connectivity at the level of inter-actor topology. It is assumed that a sensor node can reach at least one actor over multihop paths and will not be affected if the actors have to change their positions. Thus, sensor nodes are not part of the recovery process. In the balance of this paper, actor and node are used interchangeably.

The impact of the actor's failure on the network topology can be very limited, e.g., a leaf node, or significant if the failed actor is a cut vertex. A node (vertex) in a graph is a cut vertex if its removal, along with all its edges, produces a graph with more connected components (blocks) than the original graph. For example, in Fig. 1, the network stays strongly connected after the loss of a leaf actor such as A_{21} or a nonleaf node like A_5. Meanwhile, the failure of the cut vertex A_0 leaves nodes A_4, A_5, and A_6 isolated from the rest of the network. In the rest of this paper, the terms cut vertex and critical node will be used interchangeably. To tolerate the failure of a cut vertex node, two methodologies can be identified: 1) precautionary and 2) real-time restoration. The precautionary methodology strives to provision fault tolerance by establishing a biconnected topology, where every pair of nodes A_i and A_j has two distinct paths with no common nodes other than A_i and A_j; therefore, the network stays connected after a single node failure. However, provisioning such a level of connectivity may require the deployment of a large number of actors and can thus be impractical due to the high cost. In addition, it may constrain the mobility of actors and negatively affect application-level functionality. On the other hand, real-time restoration implies a response only when a failure is detected. We argue that real-time restoration better suits WSANs since they are asynchronous and reactive in nature, where it is difficult to predict the location and scope of the failure. We further direct our attention to setups in which the interactions among actors are delay sensitive and the shortest data path between a pair of nodes should not get extended compared to its prefailure length.

This paper assumes that only nonsimultaneous node failures will take place in the network. To the best of our knowledge, most recovery schemes found in the literature assume no simultaneous faults. The rationale is that the probability for having multiple simultaneous failures is very small. If p is the probability for a node failure, the probability for two simultaneous faults is p^2, p^3 for three, etc. With p being a small fraction, the probability of multiple faults diminishes. In addition, the focus of LeDiR is on nodes that are critical to network connectivity, e.g., cut vertices in a graph. Uncritical nodes can be handled at the network layer of the communication protocol stack by performing topology maintenance, which may also involve node relocation [2], [6], [8]. Tolerance of uncritical nodes is usually straightforward since the network stays connected and appropriate topology adjustment can be orchestrated among the healthy nodes. The failure of critical nodes, on the other hand, is very challenging since the network gets partitioned into disjoint blocks.

To simplify the analysis, all nodes are assumed to have the same communication range. However, our proposed algorithms do not require such assumption. In addition, the presentation of our work focuses on the algorithmic part of the recovery without focusing on the link layer issue. In general, any distributed medium access arbitration scheme would suffice. It is also assumed that a node would transmit at its maximum power to repair broken data routes before declaring a major connectivity problem and invoking LeDiR.

III. Related Work

A number of schemes have recently been proposed for restoring network connectivity in partitioned WSANs [2]. All of these schemes have focused on reestablishing severed links without considering the effect on the length of prefailure data paths. Some schemes recover the network by repositioning the existing nodes, whereas others carefully place additional relay nodes. On the other hand, some work on sensor relocation focuses on metrics other than connectivity, e.g., coverage [9]–[15], network longevity [16], and asset safety [17], or to self-spread the nodes after nonuniform deployment [6], [18], [19], which is not our focus in this paper.

A. Recovery Through Node Repositioning

The main idea of this category of recovery schemes is to reposition some of the healthy nodes in the network to reinstate strong connectivity. LeDiR fits in this category. Published approaches differ in the level of involvement expected from the healthy nodes, in the required network state that needs to be maintained, and in the goal of the recovery process. For example, both Distributed Actor Recovery Algorithm (DARA) [3] and PArtition Detection and Recovery Algorithm (PADRA) [20] require every node to maintain a list of their two-hop neighbors and determine the scope of the recovery by checking whether the failed node is a cut vertex. DARA pursues a probabilistic scheme to identify cut vertices. A best candidate (BC) is selected from the one-hop neighbors of the dead actor as a recovery initiator and to replace the faulty node. The BC selection criterion is based on the least node degree and physical proximity to the faulty node. The relocation procedure is recursively applied to handle any disconnected children. In other words, cascaded movement is used to sustain network connectivity. On the other hand, PADRA identifies a connected dominating set to determine a dominatee node. The dominatee does not directly move to the location of the failed node; instead, a cascaded motion is pursued to share the burden. In [5], the focus is also on recovering from the failure of a cut vertex. Only a special case is considered where the failure causes the network to split into two disjoint blocks. To relink these blocks, the closest nodes are moved toward each other. The other nodes in the blocks follow in a cascaded manner. None of these approaches cares for the path length between nodes. While LeDiR also employs cascaded relocation, the criteria for selecting the lead node and other participants are different.

To ensure that the recovery process converges in an efficient way, the approaches in [3], [5], and [20] require each node in the network to be aware of its two-hop neighbors. The availability of two-hop list allows the nodes to detect cut vertices with high probability and limits the scope of the recovery to cases in which the network becomes partitioned. Recovery through Inward Motion (RIM) [4] and Least Distance Movement Recovery (LDMR) [21], on the other hand, defy that assumption and base the recovery process on the knowledge of direct, i.e., one-hop, neighbors. Simply, the neighbors of a node F detect that F has failed, and then move toward F until they can reach each other directly. In RIM, any lost link during the recovery will be reestablished through cascaded relocation. The collective effect seems like the network topology is shrinking inward. LDMR avoids the cascaded relocation by sending messages to find the replacement for the neighbors of F after they move. The advantage of RIM and LDMR is obviously the reduced prefailure communication overhead that is nonetheless provided at the expense of overreacting to failure of uncritical nodes. LeDiR utilizes the partial knowledge of a node about the network topology, gained during route discovery, to decide on which node participates and which node does not. No recovery-related explicit state update is required.

Unlike LeDiR, Connectivity Restoration through node Rearrangement (CRR) [22] avoids replacing the faulty node with a healthy node since the failure might be caused by hazards that may damage the substitute node as well. Instead, CRR rearranges the network topology in the vicinity of the faulty node. The network restoration is modeled as a Steiner tree approximation problem. A set of Steiner points are identified, and the one-hop neighbors of the faulty node are relocated at these points. In case the number of one-hop neighbors is not enough, the approach progresses as the DARA approach, as previously discussed. Prefailure planning was also pursued in [23] by designating a backup for each of the critical nodes. To get a bound on the performance of recovery schemes, Fadhly *et al.* [24] formulated the problem of finding the relocation schedule with the least travel distance and maximum coverage as an integer linear program. A similar idea was proposed in [25]. These approaches would fit more of a planned rather than a reactive recovery scenario, as targeted by LeDiR.

Upon the detection of network partitioning, LeDiR opts to identify the smallest block and limits the scope of the recovery to that block. The rationale is that fewer nodes will be involved and the overhead is reduced. Basu and Redi [26] have also pursued block movement. However, they considered a biconnected network where nodes still can exchange messages with each other to coordinate the recovery process even after failure. Obviously, the goal of the block movement is to restore network biconnectivity rather than repairing a disjointed network. Other similar studies have been reported by Das *et al.* [27], [28]. Some prior work cared about the coverage hole in the network when a node fails rather than connectivity [29], [30].

In addition to network connectivity, coverage is also an important performance metric for WSANs. While restoring the network connectivity, coverage loss is possible either because of the failure itself or due to the connectivity-limited focus of the recovery. Unlike the approaches previously discussed, Coverage Conscious Connectivity Restoration ($\mathrm{C}^3\mathrm{R}$) [31] tackles the loss of both coverage and connectivity. $\mathrm{C}^3\mathrm{R}$ involves one-hop neighbors of the faulty node in the recovery process. All the one-hop neighbors take turn in relocating to the position of the faulty node and return back to their original position. This leads to intermittent connectivity and monitoring of all originally covered spots. Finally, node relocation has been pursued to optimize network performance, including boosting connectivity, not necessarily to deal with node failure. A survey of such work can be found in [2].

B. *Recovery by Placement of Relay Nodes*

The foregoing algorithms aim to restore the network connectivity by efficiently relocating some of the existing nodes. However, in some setups, it is not feasible to move the neighbors of the failed node due to physical, logistical, and coverage constraints. Therefore, some schemes establish connectivity among the disjoint network segments by placing new nodes. The published schemes generally differ in the requirements of the newly formed topology. For example, SpiderWeb [32] and Distributed algorithm for Optimized Relay node placement using Minimum Steiner tree (DORMS) [33] opt to not only reestablish the network connectivity but also achieve a certain quality in the formed topology. Basically, both schemes try to avoid the introduction of cut vertices so that some level of robustness, i.e., load balancing and high node degree, is introduced in the repaired network topology. SpiderWeb and DORMS also strive to minimize the required number of relays. Both SpiderWeb and DORMS deploy relays inwards toward the center of the deployment area. The former considers the segments situated at the perimeter and establishes a topology that resembles a spider web. Meanwhile, DORMS initially forms a star topology with all segments connected through a relay placed at the center of the area. Then, adjacent branches are further optimized by forming a Steiner tree for connecting two segments and the center node to reduce the required relay count.

Meanwhile, in [34], intersegment connectivity ought to maintain some level of quality of service (QoS) while placing the least number of relay nodes. The proposed approach initially models the deployed area as a grid with equal-sized cells. Each cell is assessed based on the uncommitted capacity of the relay node residing in the cell. Finally, to meet the QoS requirement, optimization is done by finding the cell-based least cost paths and populating nodes along these paths. On the other hand, Zhang *et al.* [35] form a biconnected intersegment topology by placing redundant nodes so that the failure of a node can be tolerated and the network operation continues without interruption. Al-Turjman *et al.* [36] model the connectivity restoration as a node placement problem on a grid and reposition the deployed nodes to meet varying requirements on the intersegment traffic. As mentioned earlier, LeDiR is a reactive scheme that opts to restore connectivity while imposing the least travel overhead and in a distributed manner.

IV. Least-Disruptive Topology Repair

As mentioned earlier, the goal for LeDiR is to restore connectivity without extending the length of the shortest path among nodes compared to the prefailure topology. In this section, we first give an overview of LeDiR as a centralized solution and then explain the distributed implementation.

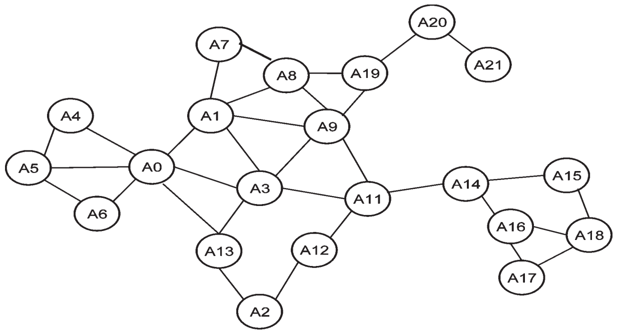


Fig. 2. How DARA [3] restores connectivity after the failure of node A_{10} in the connected inter-actor topology of Fig. 1.

A. *Problem and Solution Analysis*

Before explaining how LeDiR works, it is important to point out the effect of contemporary recovery schemes on the path length between nodes. Let us consider Fig. 1 and assume that node A_{10} fails. Connectivity restoration schemes that exploit node repositioning will replace A_{10} with one of its neighbors. For example, DARA [3] picks the neighbor with the least degree to limit the scope of relocation. Thus, A_{11} relocates to the position of A_{10}. The connectivity restoration process will be repeated with repositioning A_{12} to replace A_{11}, followed by relocating A_2 to where A_{12} was. Finally, A_{13} replaces A_2. The resulting topology is shown in Fig. 2. While A_0 and A_3 were directly reachable to A_2 before the failure, the repaired topology in Fig. 2 makes the shortest path one hop longer by involving A_{13}. As mentioned in Section I, this will not be acceptable for delay sensitive applications. LeDiR opts to avoid such a scenario by sustaining or even shortening the prefailure path lengths.

The main idea for LeDiR is to pursue block movement instead of individual nodes in cascade. To limit the recovery overhead, in terms of the distance that the nodes collectivity travel, LeDiR identifies the smallest among the disjoint blocks. For the previous example when A_{10} fails, LeDiR will only involve the block of node A_{14}. In addition, LeDiR opts to avoid the effect of the relocation on coverage and also limits the travel distance by stretching the links and moving a node only when it becomes unreachable to their neighbor. As mentioned in Section II, it is assumed that no simultaneous node failures would take place. It is important to stress the fact that the focus of LeDiR is on nodes that are critical to network connectivity, e.g., cut vertices.

To simplify the presentation, a centralized implementation of LeDiR is assumed, where every node is aware of the entire network topology prior to the failure and thus can build the shortest-path routing table (SRT) for every pair of nodes. This assumption is eliminated later in this section. LeDiR is a distributed scheme that does not need a network-wide state. The SRT can be populated through the route discovery activities in the network, e.g., when an on-demand routing protocol such as AODV is employed. The simulation results presented

TABLE I
PATH PREDECESSOR MATRIX

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	–	0	0	0	0	0	0	1	1	1	3	10	2	2	10	14	14	16	15	8	19	20
1	1	–	0	1	0	0	0	1	1	1	3	10	2	2	10	14	14	16	15	8	19	20
2	2	0	–	2	0	0	0	1	1	3	3	12	2	2	10	14	14	16	15	9	19	20
3	3	3	3	–	0	0	0	1	1	3	3	10	2	2	10	14	14	16	15	9	19	20
4	4	0	0	0	–	4	0	1	1	1	3	10	2	2	10	14	14	16	15	8	19	20
5	5	0	0	0	5	–	5	1	1	1	3	10	2	2	10	14	14	16	15	8	19	20
6	6	0	0	0	0	6	–	1	1	1	3	10	2	2	10	14	14	16	15	8	19	20
7	1	7	0	1	0	0	0	–	7	1	3	10	2	2	10	14	14	16	15	8	19	20
8	1	8	0	1	0	0	0	8	–	8	9	10	2	2	10	14	14	16	15	8	19	20
9	1	9	3	9	0	0	0	1	9	–	9	10	2	2	10	14	14	16	15	9	19	20
10	3	3	3	10	0	0	0	1	9	10	–	10	11	2	10	14	14	16	15	9	19	20
11	2	3	12	10	0	0	0	1	9	10	11	–	11	2	10	14	14	16	15	9	19	20
12	2	0	12	2	0	0	0	1	1	10	11	12	–	2	10	14	14	16	15	9	19	20
13	2	0	13	2	0	0	0	1	1	3	3	12	2	–	10	14	14	16	15	9	19	20
14	3	3	3	10	0	0	0	1	9	10	14	10	11	2	–	14	14	16	15	9	19	20
15	3	3	2	10	0	0	0	1	9	10	14	10	11	2	15	–	14	18	15	9	19	20
16	3	3	2	10	0	0	0	1	9	10	14	10	11	2	16	14	–	16	16	9	19	20
17	3	3	2	10	0	0	0	1	9	10	14	10	11	2	16	18	17	–	17	9	19	20
18	3	3	2	10	0	0	0	1	9	10	14	10	11	2	16	18	18	18	–	9	19	20
19	1	8	3	9	0	0	0	8	19	19	9	10	2	2	10	14	14	16	15	–	19	20
20	1	8	3	9	0	0	0	8	19	19	9	10	2	2	10	14	14	16	15	20	–	20
21	1	8	3	9	0	0	0	8	19	19	9	10	2	2	10	14	14	16	15	20	21	–

in Section VI confirm that LeDiR works well with partial topological information.

The following highlights the major steps.

1) *Failure detection*: Actors will periodically send heartbeat messages to their neighbors to ensure that they are functional, and also report changes to the one-hop neighbors. Missing heartbeat messages can be used to detect the failure of actors. Once a failure is detected in the neighborhood, the one-hop neighbors of the failed actor would determine the impact, i.e., whether the failed node is critical to network connectivity. This can be done using the SRT by executing the well-known depth-first search algorithm. Basically, a cut vertex F has to be on the shortest path between at least two neighbors of F. Consider Table I, which lists the entries of the SRT for the network topology in Fig. 1. After the failure of actor A_{19}, which is a cut vertex, node A_{20} will check what nodes are reachable through A_{19}, which are A_8 and A_9 in this example. Checking the entries for nodes A_8 and A_9 reveals that A_1, A_3, A_7, and A_{10} will become consequently unreachable. The same is repeated and finally leads node A_{20} to conclude that only A_{21} is reachable and A_{19} is indeed a critical node. The SRT can make the same conclusion for a node that is not a cut vertex but serves on the shortest path of all nodes. For example, in a wheel-shaped topology, the node at the center is not a cut vertex, yet it serves on the shortest paths among many nodes on the outer ring. The SRT points out the criticality of such a node and motives the invocation of the recovery process.
2) *Smallest block identification*: LeDiR limits the relocation to nodes in the smallest disjoint block to reduce the recovery overhead. The smallest block is the one with the least number of nodes and would be identified by finding the reachable set of nodes for every direct neighbor of the failed node and then picking the set with the fewest nodes. Since a critical node will be on the shortest path of two nodes in separate blocks, the set of reachable nodes can be identified through the use of the SRT after excluding the failed node. In other words, two nodes will be connected only if they are in the same block. For example, let us again consider the network topology provided in Fig. 1 and assume that node A_{19} failed. When nodes A_8, A_9, and A_{20}, the one-hop neighbors of A_{19}, confirm that A_{19} is indeed a cut vertex (critical node), they will be able to identify the disjoint blocks. For A_{20}, the analysis of the cut vertex detection step discussed previously will conclude that A_{20} can reach only A_{21}, and thus, A_{20} and A_{21} constitute a block. Now, A_{20} would check the column of A_{19} and find out that A_8 and A_9 are the other direct neighbors of A_{19}. Node A_{20} will then repeat the analysis and identify the other disjoint block(s) and determine the smallest block after A_{19} fails. Now, A_{20} will lead the recovery effort if it happens to belong to the smallest block, which is the case in this example. Nodes A_8 and A_9 will perform the same analysis and conclude that they are not part of the smallest block.
3) *Replacing faulty node*: If node J is the neighbor of the failed node that belongs to the smallest block, J is considered the BC to replace the faulty node. Since node J is considered the gateway node of the block to the failed critical node (and the rest of the network), we refer to it as "parent." A node is a "child" if it is two hops away from the failed node, "grandchild" if three hops away from the failed node, and so on. The reason for selecting J to replace the faulty node is that the smallest block has the fewest nodes in case all nodes in the block have to move during the recovery. As will be shown later,

the overhead and convergence time of LeDiR are linear in the number of nodes, and thus, engaging only the members of the smallest block will expedite the recovery and reduce the overhead. In case more than one actor fits the characteristics of a BC, the closest actor to the faulty node would be picked as a BC. Any further ties will be resolved by selecting the actor with the least node degree. Finally, the node ID would be used to resolve the tie.

4) *Children movement*: When node J moves to replace the faulty node, possibly some of its children will lose direct links to it. In general, we do not want this to happen since some data paths may be extended. For example, in Fig. 2, the path between A_2 and A_3 get extended because A_2 lost its link to A_{12} after A_{12} had moved. LeDiR opts to avoid that by sustaining the existing links. Thus, if a child receives a message that the parent P is moving, the child then notifies its neighbors (grandchildren of node P) and travels directly toward the new location of P until it reconnects with its parent again. If a child receives notifications from multiple parents, it would find a location from where it can maintain connectivity to all its parent nodes by applying the procedure used in RIM [4]. Briefly, suppose a child C has two parents A and B that move toward the previous location of node J. As previously mentioned, node J already moved to replace the faulty node F, and as a result, nodes A and B get disconnected from node J. Now, nodes A and B would move toward the previous location of J until they are $r/2$ units away. Before moving, these parents inform the child C about their new locations. Node C uses the new locations of A and B to determine the slot to which it should relocate. Basically, node C will move to the closest point that lies within the communication ranges of A and B, which is the closest intersection point of the two circles of radius r and centered at A and B, respectively. It is worth to mention that since parents A and B move toward a single point, that is, the position of node J, they get closer to one another. Thus, if both can reach C before they move, i.e., C lies within their range, their communication range must overlap after the move since they get closer to one another. This observation also applies for more than two parent nodes since there must be an intersection point of two circles which lies within the communication ranges of all the moved nodes. It has been proven in [38] that this relocation scheme sustains existing links in the connected component (block).

A simple example scenario is a 1-D smallest block (B_s), where each node is r units away from each other, as presented in Fig. 3(a). Simply, each child would move to the location of its parent, and thus, the entire block B_s would move r units toward node F. This would keep intrablock connectivity as is and would not extend any path within B_s. However, in reality, nodes within B_s can be closer than r units to each other. In this scenario, the movement of B_s would be performed in a way that the intrablock paths remain unchanged or get shorter, and the total travel distance is minimized, as depicted in Fig. 3(b). Node A moves to the location of F, and the children B and C get disconnected. To regain the connectivity

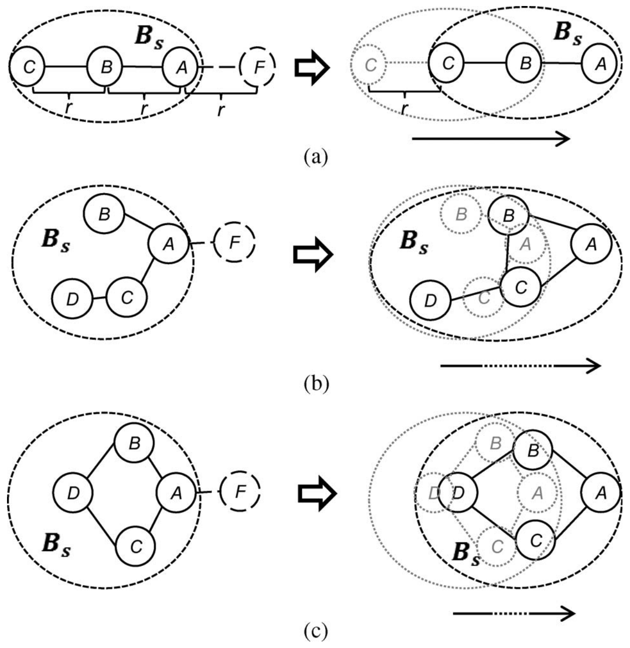


Fig. 3. Movement of block B_s in LeDiR to restore the network connectivity and to keep intrablock paths unchanged. (a) That entire B_s moved r units. (b) The collective effect of B_s participation in the recovery is stretching B_s toward F. (c) B_s is both stretched and moved with links within the B_s stretched to minimize the total travel distance. r is the actor's communication range.

to A, nodes B and C move toward the new location of A until becoming r units away. As a side effect, connectivity within B_s gets stronger and creates a new link between B and C. This makes the intrablock shortest path between B and C even shorter; however, premovement intrablock paths remain unchanged. In addition, to avoid unnecessary movement and minimize the total travel distance, node D does not move as it is still connected to its parent C. Moreover, it is worth to note that the shortest path from D to B has become one-hop shorter after recovery. Fig. 3(c) shows the situation where the entire B_s moves to preserve the intrablock paths with links between nodes stretched to minimize the total travel distance. As explained earlier, nodes B and C move toward A until they are r units away. Since node D has two parents, i.e., B and C, which move and break their links to D, node D relocates to the closest point that lays within the communication ranges of B and C.

Fig. 4 shows an example for how LeDiR restores connectivity after the failure of A_{10}. Obviously, node A_{10} is a cut vertex, and A_{14} becomes the one-hop neighbor that belongs to the smallest block [see Fig. 4(a)–(c)]. In Fig. 4(d), node A_{14} notifies its neighbors and moves to the position of A_{10} to restore connectivity. Disconnected children, i.e., nodes A_{15} and A_{16}, follow through to maintain communication link with A_{14} [see Fig. 4(e)]. Note that the objective of the children movement is to avoid any changes to the current routing table. Nodes A_{15} and A_{16} would notify their children A_{17} and A_{18} before they move. Since A_{18} had communication links with nodes A_{15}, A_{16}, and A_{17}, it moves to a new location where it can stay directly connected to these nodes [see Fig. 4(f)]. The links between A_{17} and nodes A_{16} and A_{18} are not affected by the relocation process, and thus, A_{17} would not need to reposition.

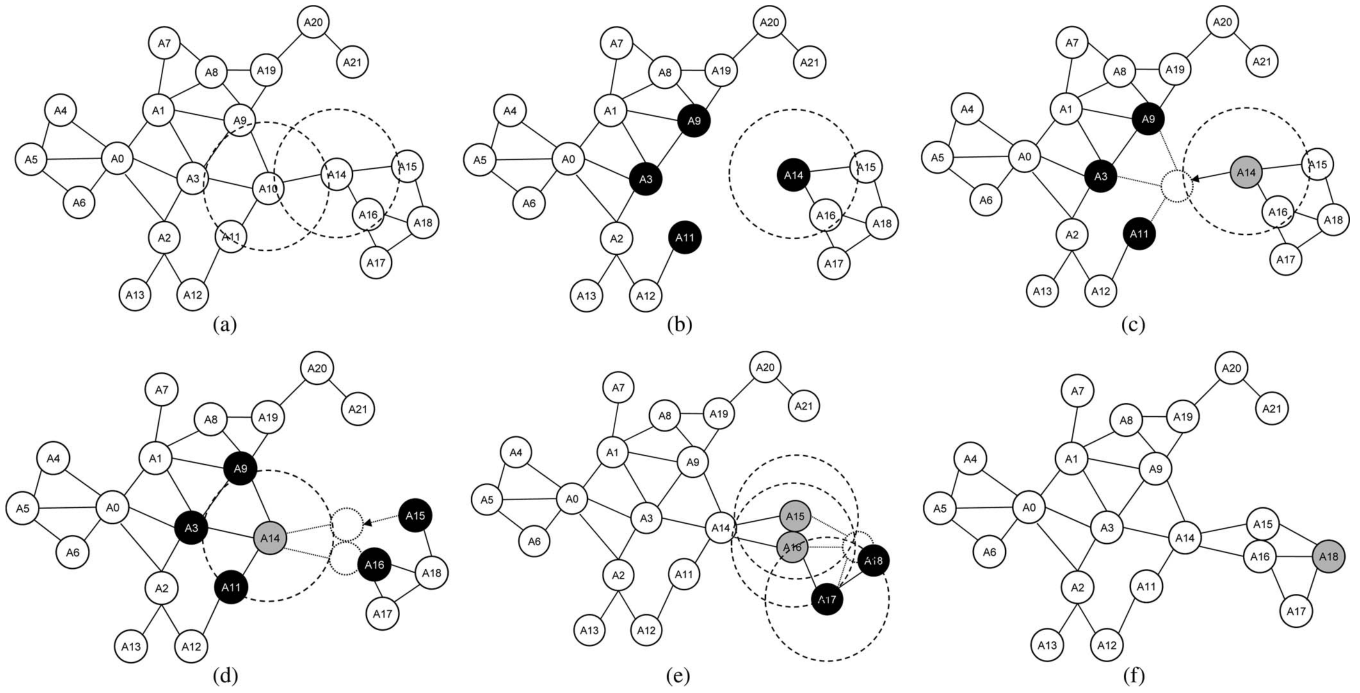


Fig. 4. How LeDiR restores connectivity after the failure of node A_{10} in the connected inter-actor topology of Fig. 1. Nodes in black are participating in the recovery process, and those in gray are selected to move.

Fig. 4(f) shows the repaired network topology where the paths from nodes A_{14}, A_{15}, A_{16}, A_{17}, and A_{18} to the other nodes in the network are not extended.

B. Distributed LeDiR Implementation

The foregoing discussion has assumed that nodes are aware of the network topology and can assess the impact of the failure and uniquely identify which node should replace the failed actor. If every node in the network is communicating with all the other nodes, it would be possible to fully populate the routing table and for the individual nodes to reach consistent decisions without centralized coordination. However, in many setups, an actor may have only partial knowledge about the network with routes to some nodes missing in its SRT. This can happen due to changes in the topology caused by node mobility or due to the fact that a subset of actors do not need to interact and that a route has yet to be discovered. In general, a partially populated SRT can raise the following three issues for a distributed implementation of LeDiR: 1) A potential BC actor does not realize that its failed neighbor is a critical node; 2) every neighbor of the faulty node assumes that it is not part of the smallest block leaving the network topology unrepaired; and 3) more than one neighbor in different blocks step forward as BC. In the balance of this section, we discuss how LeDiR addresses these issues.

Let α be the percentage of entries, i.e., routes between actor pair (i, j), that each node has acquired over time. Hereafter, we will call this α as confidence level (CL). For example, if 50% entries of the node's A_i routing table are filled, we say node A_i has 50% CL. Since every node may potentially have different CL from others, upon the detection of a node failure, applying depth-first search at the neighboring nodes may yield an inconsistent assessment of the impact of the node loss on the network connectivity and on which actor is the BC for leading the recovery. For example, in Fig. 1, if node A_{11} was never on a route that has nodes A_{14}, A_{15}, A_{16}, A_{17}, and A_{18} as sources or destinations, node A_{11} will not know that A_{10} is a cut vertex. We argue nonetheless that this is rare in practice since the mobility pattern among actors is not typically high given their involvement in actuation activities. In addition, the operation in WSAN is collaborative in nature, and an actor usually communicates with many others; thus, the routing table would not be sparse or at least will include the important routes; in particular, the neighbors of a cut vertex would have more populated SRT compared to other nodes in the network as they would be passing packets among the actors in different blocks.

Furthermore, LeDiR may employ probabilistic cut vertex detection schemes that use two-hop information to boost the fidelity of the assessment [5], [39] and mitigate the effect of the missing entries in the SRT. It has been shown that these probabilistic schemes can achieve accurate detection of cut vertices up to 90%, i.e., no cut vertex will be classified otherwise, and only 10% of the time a node is claimed to be a cut vertex while it is not [5]. It is important to note that if LeDiR is applied while the failed node F turned out not to be a cut vertex, e.g., due to the inaccuracy of the probabilistic detection scheme, the shortest path lengths between nodes will not change since LeDiR sustains the links between nodes in the same block and the network will be in fact connected, i.e., one block. Determining the block size is always based on the entries of the SRT that neighbors of F have, regardless whether F is a cut vertex or not. Now, if the analysis to determine the block size is based on inaccurate assertion about whether F is a cut vertex, one of the neighbors F still becomes the BC and performs LeDiR successfully, i.e., proceeds to replace the faulty node. Children would follow BC to maintain connectivity, and so on.

The foregoing second and third issues are related to determining the BC, i.e., the neighbor of the failed node that belongs to the smallest block. If global topological information is available, i.e., the node has a fully populated SRT, determining the smallest block is straightforward, as we explained earlier. However, if a node has a low CL, it may not be able to accurately determine the smallest block. For example, if node A_{14} does not have sufficient entries in its SRT, it would not know that it belongs to the smallest block and would not thus initiate the recovery process by moving to replace A_{10}. Since the neighbors of A_{10} cannot reach each other, a partially populated SRT may lead to a deadlock, with none of the neighbors of A_{10} responding to the failure and leaving the network disconnected. To handle this issue, LeDiR imposes a timeout after which the neighbor(s) belonging to the second largest block will move. This time, multiple neighbors may be potentially moving toward A_{10}. To avoid having more than one actor replacing A_{10}, LeDiR requires these nodes to broadcast messages with their ID so that they pause as soon as reaching other neighbors of A_{10} that happen to be in a different block. The pause time would allow these neighbors to negotiate and pick the BC to continue on to the position of A_{10}. We study the effect of the CL on the performance through simulation in Section V. The pseudocode of LeDiR can be found in the Appendix.

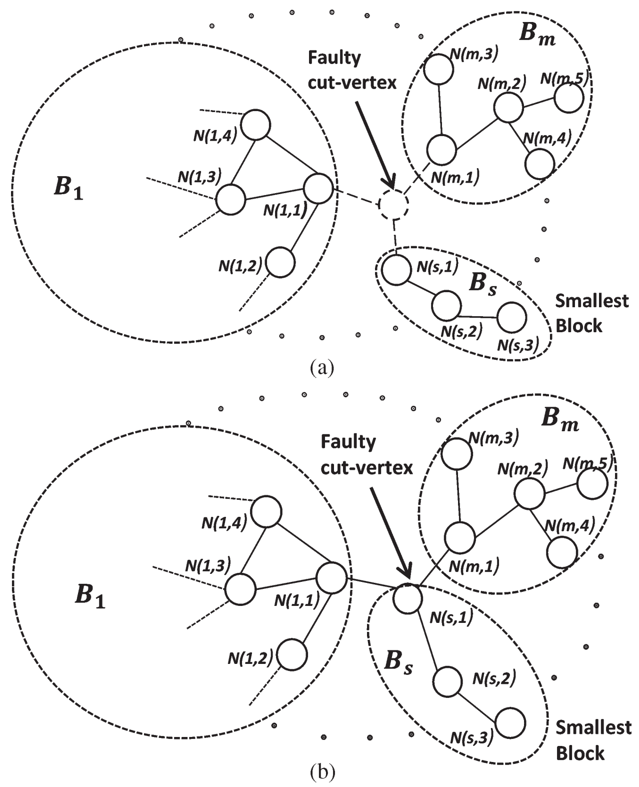


Fig. 5. LeDiR restores the network connectivity after the failure of a cut vertex (critical node). (a) WSAN before a cut vertex fails. (b) WSAN topology after applying LeDiR.

C. Algorithm Analysis

In this section, we prove the convergence and analyze the performance of LeDiR. We introduce the following theorems.

Theorem 1: LeDiR guarantees a localized network recovery without extending the shortest data path between any pair of nodes (i, j).

Proof: We assume that the network is partitioned into m blocks because of a faulty node F, which happens to be a critical node, i.e., a cut vertex. The scenario is illustrated in Fig. 5(a), where B_s is the smallest block. The node ID is represented by $N(b, i)$, where b is block number, and i is the node number within the block. LeDiR involves only one-hop neighbors of F, which is denoted hereafter as $Neighbors(F)$, in the process of block selection and moves only the node in $Neighbors(F)$ that belongs to B_s. Thus, the scope of the recovery is localized and affects only B_s.

To prove that the shortest path between any arbitrary nodes is not extended, it is sufficient to show that the interblock paths are not extended, and the intrablock paths after the recovery are not longer than before the failure takes place. Since the blocks are used to reach each other through F, the node F belongs to the shortest path between every pair of nodes $N(p, i)$ and $N(q, j)$, where $p \neq q$. Thus, replacing F with a healthy node will not extend any of these paths if the intrablock part of the path is not extended. In other words, if the paths between $N(p, i)$ and $N(p, 1)$ and between $N(q, j)N(q, 1)$ are not extended, LeDiR will sure achieve its goal for interblock paths. Since other than B_s none of the blocks will experience any changes in their intrablock topologies, the path between any pair of nodes $N(k, i)$ and $N(k, j)$ will stay intact after the recovery for all $k \neq s$. Hence, to prove the theorem, it will be sufficient to prove that the shortest paths between nodes in the moved block B_s are not extended.

When the node $N(s, 1)$ moves to replace F, the links to $Neighbors(N(s, 1))$ are maintained. If a neighbor $N(s, t)$ of $N(s, 1)$ was also a neighbor of F, the link between $N(s, t)$ and $N(s, 1)$ is not affected. Otherwise, $N(s, t)$ travels toward F to stay directly connected to $N(s, 1)$. Cascaded relocation also ensures that every node stays connected to all its neighbors. To maintain prefailure connectivity, a node that needs to move selects a new location that keeps it reachable to all its parents after the recovery. Since the motion of nodes in B_s is inward toward F, it has the effect of shrinking B_s toward node F, and the links of a node to its siblings are maintained. This is proven by [38, Lemmas 1 and 2].

The foregoing analysis shows that LeDiR not only keeps the intrablock shortest path between any pair of nodes in B_s but also may enhance the shortest path between blocks. Since F is no longer on paths between any pair of nodes $N(p, i)$ and $N(s, j)$, some of these paths are shorter. The most intuitive example is the path between any node $N(p, i)$ and $N(s, 1)$, which has replaced F. This proves that LeDiR achieves the objective to restore connectivity without extending the shortest data path between any pair of nodes in the network. ■

Theorem 2: The max number of nodes involved in the recovery process is $O(N)$, where N is the number of actors.

Proof: Consider the worst case where a 1-D network, as shown in Fig. 6, is split into two blocks (subnetworks) of

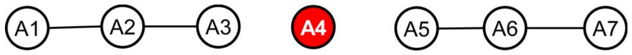


Fig. 6. Worst-case scenario topology where $N = 7$ and failure of A_4 has partitioned the network into two blocks of $\lceil 1/2(N-1) \rceil$ and $\lfloor 1/2(N-1) \rfloor$ nodes. LeDiR would pick the smallest block and, thus, involves a maximum of $\lfloor 1/2(N-1) \rfloor$ actors in the recovery process of either A_3 or A_5 selected to replace the faulty node followed by a series of interblock node relocation.

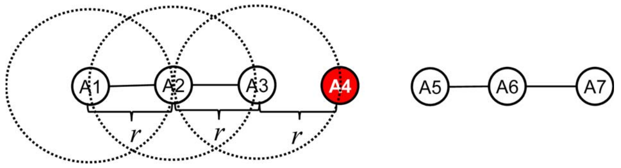


Fig. 7. Assuming the worst-case scenario presented in Fig. 6, LeDiR selected A_3 to replace the faulty node A_4 by traveling distance r. Once A_3 moved to the new position, A_2 will move behind it to maintain direct connectivity. Later, A_1 will do the same. Since the network is 1-D, and nodes are located r units away from each other, the maximum distance travel by a node is r.

$\lceil 1/2(N-1) \rceil$ and $\lfloor 1/2(N-1) \rfloor$ nodes. LeDiR involves only the smaller of the two blocks in the recovery process simply by moving it toward the other block.

Assuming that the network is sparse and nodes are r units away from each other, where r is the node's communication range, every node in the block would move and participate in the recovery process. Thus, the maximum number of nodes that get involved in the recovery process $\lfloor 1/2(N-1) \rfloor$ is $O(N)$. ■

Theorem 3: LeDiR guarantees to terminate in $O(N)$ iterations, where N is the number of actors.

Proof: Theorem 2 proves that in the worst case $\lfloor 1/2(N-1) \rfloor$ nodes are involved in the recovery. During the entire recovery process, a node can move only once, which means that LeDiR guarantees to terminate in $O(N)$ iterations. ■

Theorem 4: The message complexity of LeDiR is $O(N)$, where N is the number of actors.

Proof: LeDiR depends on the route discovery protocol to maintain the routing table on each node; thus, no special messaging is required to know the neighbors or network topology. If a node got involved in the recovery process and decided to move, it broadcasts one message to its children to notify them about its movement. Another message is broadcast to interact with the neighbors once a node has reached the new position. In other words, every node participating in the recovery process would broadcast only two messages. In the worst-case scenario, only $\lfloor 1/2(N-1) \rfloor$ nodes would participate in the recovery process, as proven in Theorem 2. Thus, the total message count is $2(\lfloor 1/2(N-1) \rfloor) + \lfloor 1/2(N-1) \rfloor$, which is to $O(N)$. ■

Theorem 5: The maximum distance a node travels in LeDiR is r, where r is the actor radio range.

Proof: In the worst case (see Fig. 7), LeDiR can select a one-hop neighbor to replace the faulty node that is at most r units away from it. When a node moves to replace the faulty node, possibly some of its children will lose direct links to it. If a child receives a message that the parent is moving, then the child would notify its neighbors and travel a maximum of r units to restore its link to the parent. If a child receives notifications from multiple parents, it would find a location from where it can maintain connectivity to all its parent nodes. In this case, the new location definitely is less than r, as proven in [38]. Thus, the maximum distance that a node travels is r. ■

Theorem 6: The maximum convergence time of the LeDiR algorithm to restore inter-actor connectivity is $O(N)$, where N is the number of actors.

Proof: Let us assume that no other failure occurs during the recovery process and s is the maximum time required for a node to find whether it belongs to the smallest block. The maximum time for a neighbor A of the failed node F to find out the block that it belongs to is $O(N.d)$. Basically, a node will have to check the column for F in the SRT to identify all the other $d-1$ neighbors of F. Node A then eliminates these $d-1$ actors, and all nodes that are reachable through them from its row in SRT. This step is applied at most $N-1$ times in a network of N actors, and node A is a leaf node in the network. To determine whether its block is the smallest, node A will repeat this process at most $d-1$ times for the other neighbors of F. Thus, the maximum time s for a node to identify the smallest block is $O(N.d^2)$.

Theorem 2 proves that the maximum number of nodes involved in the recovery process is $O(N)$. In addition, Theorem 5 proves that the maximum distance a node travels in the recovery process is r. Suppose t is the time to travel distance r. Assuming the worst case scenario where the node movement is sequential, the total time to restore network connectivity would be $(N) * t$. Thus, the maximum convergence time of LeDiR to restore inter-actor connectivity is $s + (N) * t$, which is $O(N[d^2 + t])$. For a uniform actor distribution, the value of d depends only on r [40]. Thus, both d and t are constant, and the inter-actor connectivity would be restored in $O(N)$. ■

Table II provides a comparison of the analytical performance bounds for LeDiR to those of DARA [3] and RIM [4]. As indicated by the table, LeDiR outperforms both baseline approaches when considering the recovery overhead at the network level in terms of the number of nodes participating in the recovery and the distance that these nodes collectively travel. RIM still does better in terms of balancing the travel overhead on all nodes in the network yet matches LeDiR at the network level. It is important to reiterate that unlike LeDiR, neither RIM nor DARA provides any guarantee on the internode path length.

V. Performance Analysis

LeDiR is validated through simulation. This section discusses the simulation environment and results.

A. Simulation Environment and Performance Metrics

The experiments are performed on a Wireless Sensor Networks (WSN) simulator developed in Visual C++. In the experiments, we have created connected topologies consisting of varying number of actors (20 to 100) with fixed transmission range ($r = 100$ m). All nodes are assumed to transmit at the maximum power set for the individual experiment, and thus, the detection of failure of a critical node would justify the invocation of LeDiR. In addition, we run simulation experiments with fixed node count (100 actors) while varying the communication range (25 to 200 m). Actors are randomly placed in an area of 1000 m × 600 m.

TABLE II
COMPARISON OF ANALYTICAL RESULTS OF LEDIR, RIM, AND DARA

Property	LeDiR	RIM	DARA
Maximum number of nodes to be involved	$\left\lfloor \frac{1}{2}(N-1) \right\rfloor$	*N-1*	*N-3*
Maximum messages to be send	$\left\lfloor \frac{3}{2}(N-1) \right\rfloor$	*2N-1*	*5N-3*
Maximum distance travelled by a node	r	*r/2*	r
Maximum distance travelled by all engaged nodes	$\left\lfloor \frac{r}{2}(N-1) \right\rfloor \approx \frac{1}{2}\, r\, N$	$\frac{r}{2}(N-1) \approx \frac{1}{2}\, r\, N$	$r(N\text{-}3) \approx r\, N$

After identifying the cut vertices in the generated topology, one of them is designated at random to be the faulty node. The Floyd–Warshall algorithm is used to form the SRT. This implies that every node is aware of the entire network topology. We then mimic the effect of CL (α) by randomly removing $(1-\alpha)\%$ entries from the copy of the global SRT stored at the individual nodes to capture the performance of a distributed implementation. Basically, the SRT of a node is considered, $(1-\alpha)\%$ rows (nodes) in the table are picked using a uniform random distribution, and then, these rows are removed as well as their corresponding columns. This is repeated for the SRT of each node in the network. In other words, the SRT of the different nodes is not the same. Obviously, quite a few of the picked nodes will be one- and two-hop neighbors of the failed node, particularly for small α values. In practice, each actor will collect these entries during route discovery in case a reactive scheme such as AODV is employed or while routing data traffic if other schemes are used, e.g., greedy forwarding. With the way these routing schemes work, it is likely that the missing entries will be random, which is consistent with our simulation.

The following parameters are used to vary the characteristics of the topology in the different experiments:

1) Number of deployed actors (N): This parameter affects the node density and the WSAN connectivity. Increasing N makes the WSAN topology highly connected.
2) Communication range (r): All actors are assumed to have the same communication range r. The value of r affects the initial WSAN topology. While a small r creates a sparse topology, a large r boosts the overall connectivity.

The following metrics are used to measure the performance of LeDiR in terms of recovery overhead.

1) Total travelled distance: reports the distance that the involved nodes collectively travel during the recovery. This can be envisioned as a network-wide assessment of the efficiency of the applied recovery scheme.
2) Number of relocated nodes: reports the number of nodes that moved during the recovery. This metric assesses the scope of the connectivity restoration within the network.
3) Number of exchanged messages: tracks the total number of messages that have been exchanged among nodes. This metric captures the communication overhead.

Furthermore, the following metrics are used to validate the path length performance of LeDiR:

1) Number of extended shortest paths: reports the total number of shortest paths between pairs of nodes (i, j) that get extended as a result of the movement-assisted network recovery. Shortest paths are calculated by using the Floyd–Warshall algorithm. This metric validates our claim that LeDiR avoids extending the shortest path between any pair (i, j) of node while restoring connectivity. Thus, for LeDiR, this metric must be zero in all experiments.
2) Shortest paths not extended: reports average number of shortest paths that are not extended per topology: This metric assesses how serious the potential path extension concern for contemporary approaches and further validates the correctness of LeDiR. This metric should be 100% for LeDiR.

In addition to the foregoing metrics, the coverage loss relative to the prefailure level has been tracked and, except for RIM, the difference among the variants of LeDiR and the baseline approaches is mostly insignificant. Finally, each simulation setup is run for 30 different network topologies, and the average measures are reported. We observed that with 90% CL, the simulation results stay within 6%–10% of the sample mean.

B. Simulation Results

As we mentioned earlier, LeDiR strives to restore network connectivity while minimizing the recovery overhead and maintaining the shortest path lengths at their prefailure value. We group the results into two sets: 1) overhead related metrics and 2) path length validation metrics. We compare the performance of LeDiR to RIM [4] and DARA [3], which are the most effective published solutions for the tolerance of a single node failure in WSAN.

The first set compares LeDiR, which runs in a distributed manner, to a centralized version that provides the least traveled distance. We also compare LeDiR to RIM in terms of recovery overhead. LeDiR selects the smallest partition and tries to maintain the existing communication links between nodes within the block that will perform the recovery. The movement technique and operation are closer to RIM; in other words, RIM can achieve the same objective while DARA cannot guarantee it. Therefore, in the first set, we compare LeDiR with RIM and not DARA.

In addition to the centralized version of LeDiR and RIM, the second set of simulation experiments compares LeDiR to DARA. The reason is that both RIM and DARA are designed particularly to restore network connectivity. However, RIM and DARA do not care whether a prefailure shortest path gets

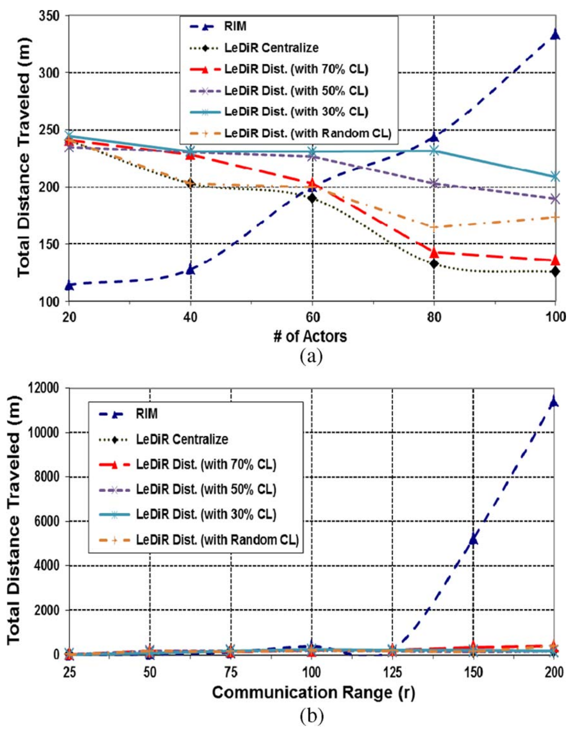


Fig. 8. (a) Effect of network size on the total distance traveled by actor nodes under RIM and LeDiR, where CL is varying (with $r = 100$). (b) Impact of an increased actor's communication range on the relocation overhead for a network of 100 actor nodes.

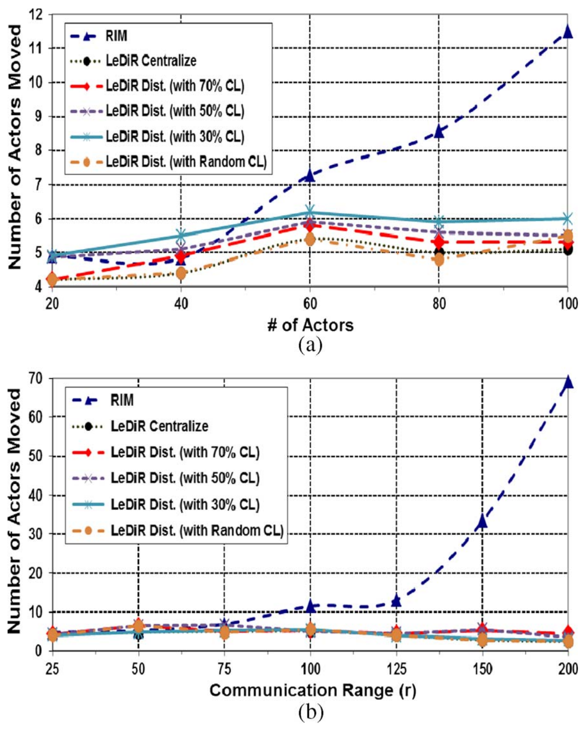


Fig. 9. Number of actors that moved during the recovery while varying (a) the network size (with $r = 100$) (b) the communication range (with $N = 100$).

extended or not. Therefore, the path length validation metrics would assess how frequent the shortest paths are affected by contemporary schemes and LeDiR's contribution to sustaining the prefailure path lengths.

Overhead Related Metrics: Fig. 8 shows the distance that actor nodes collectively travel during the recovery under varying r and N, respectively. Fig. 8(a) shows that LeDiR scales well with dense topologies and outperforms RIM significantly. Although in sparse topologies LeDiR does not appear to have advantage over RIM, RIM does not prevent the paths from being extended, as shown later in this section. More specifically, in networks with a low degree of connectivity, most nodes have few neighbors, and RIM often yields a topology that has some longer paths between pairs of nodes compared to the prefailure topology. When the node count increases, LeDiR demonstrates distinct performance and dominates RIM even without considering the path length between nodes. Fig. 8(b) captures the impact of changing r for a network of 100 nodes. Obviously, LeDiR performs very well in highly connected networks and matches the performance of RIM for low ranges while meeting the internode path length goal.

As we mentioned earlier, the decrease in the CL level means fewer entries in the actor's SRT and less information for the actor to make the right assessment of the scope of the failure and define the most appropriate recovery plan. As pointed out in Section IV, LeDiR may not make the right assessment by reacting to the failure of uncritical nodes and may not pick the smallest block if the SRT is sparsely populated. This leads to an increase in the likelihood of wrong decision making and results in more travel overhead. However, Fig. 8 shows LeDiR stays robust and yields close to optimal results when CL is 70%. Even under very low CL scenarios, e.g., 30%, the performance of LeDiR is not far from the centralized version.

While simulating LeDiR, initially, we assume that all the nodes are deployed together and have the same CL. In Fig. 8, saying 70% CL means that all nodes in the network have only 70% of the entries populated in SRT. The same is true when we say 50% or 30% CL. We have further tested the performance of LeDiR with unequal CL values. This mimics the case when nodes are deployed in batches and the case when the traffic density is different throughout the network. In Fig. 8, the curve for LeDiR with random CL reflects the performance with unequal CL values, and the results are very close to those of a centralized implementation of LeDiR. To set the CL in this case, we pick values in the range [30, 70] using uniform random distribution. The average CL observed during the simulation was approximately 57%.

Fig. 9 clearly indicates that LeDiR outperforms RIM by moving fewer nodes during the recovery, particularly for dense and highly connected topologies. Unlike RIM, LeDiR tries to relocate nodes that belong to the smallest block to avoid triggering large-scale movement of child actors. In addition, networks with high node density or large radio range are highly connected, and thus, cut vertices usually exist close to the network periphery. Determining the smallest block would then limit the scope of the recovery and make LeDiR more advantageous.

Table III reports the performance of LeDiR in terms of correct block selection under varying CL and presents the percentage of simulated topologies for which the smallest block was selected, a block other than the smallest block was picked, and when the available network state was insufficient to assess the criticality of the failed node and trigger recovery. Clearly,

TABLE III
PERFORMANCE OF LEDIR IN TERMS OF BLOCK SELECTION

CL (α)	Smallest Block Selected (% of Cases)	None-Smallest Block Selected (% of Cases)	None of the Block Selected (% of Cases)
100%	100	-	-
70%	92	08	-
50%	78	22	-
30%	61	39	-
20% or Less	07	15	78
Random CL	79	21	-

TABLE IV
NO. OF MESSAGES SENT WITH VARYING NUMBER OF ACTORS

# of Actors	RIM	LeDiR				
		Centralized	Distributed			
			70% CL	50% CL	30% CL	Rand. CL
20	30	406	6	6	9	7
40	56	1608	9	10	14	12
60	85	3613	13	15	17	14
80	115	6406	11	19	25	17
100	152	10017	17	20	30	23

TABLE V
NUMBER OF SENT MESSAGES WITH VARYING ACTOR RADIO RANGE

Radio Range	RIM	LeDiR				
		Centralized	Distributed			
			70% CL	50% CL	30% CL	Rand. CL
25	112	10010.9	11	11	13	12
50	113	10009.4	14	14	9	16
75	121	10010.7	21	15	24	17
100	163	10017.3	17	23	26	22
125	143	10013.4	25	25	23	26
150	351	10016.2	71	29	87	33
200	1072	10021.1	78	56	98	62

LeDiR performs very well with fully populated SRT and high CL; however, the frequency of optimal block selection diminishes as CL decreases, and at CL of 30%, the probability of picking the nonoptimal block reaches about 40%. It is worth mentioning that having very low CL ($\leq 20\%$) would not enable the network to know whether a major connectivity problem exists after the failure of a cut vertex due to the very limited network state. We argue that this is not practical unless the node fails right after deployment and before the network becomes fully operational.

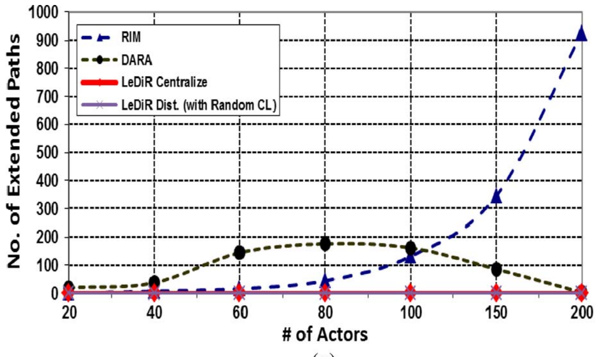
With respect to the number of messages, LeDiR introduces significantly less messaging overhead to enable and during the recovery in comparison to the centralized version and RIM, as shown in Tables IV and V. Actually, in the centralized version, each node must be aware of the complete network topology, which involves N^2 messages required for maintaining the network status, as pointed out earlier. Thus, the messaging overhead dramatically grows as the node count increases. On the other hand, RIM requires maintaining one-hop neighbor information for performing the recovery. Thus, an extra N message overhead is considered for RIM to exchange information initially at the network startup. Conversely, LeDiR leverages the available route discovery process and does not impose prefailure messaging overhead. The only communication cost incurred during the recovery is when a node informs its children about its movement or broadcasts the successful relocation. Nonetheless, as previously noted, the avoidance of explicit state update comes at the cost of increased travel overhead.

It is important to note that for the results in Tables IV and V, no heartbeat messages are counted during the network operation for all approaches. In practice, heartbeat messages may or may not be explicitly transmitted. Typically, a node that stays quiet for a long time has to send a message to confirm its healthy status. Otherwise, messages that are part of the normal network operation, such as route update, data packet generation, inter-actor coordination, etc., would suffice. We argue that the number of heartbeat messages would vary from node to node and over time. It is our view that they are not part of the recovery process in case a node failure is to be tolerated. Therefore, we did not factor in heartbeat messages in the results of LeDiR, RIM, and the centralized approach.

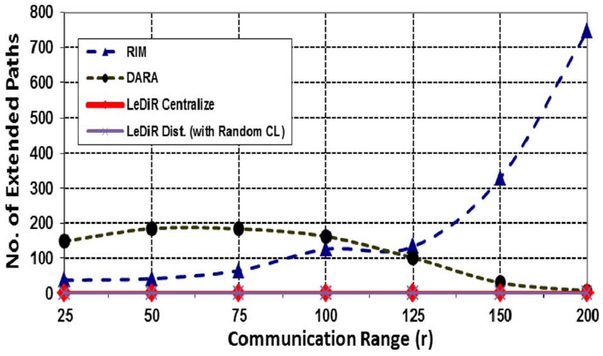
Path Length Validation Metrics: Fig. 10 validates that LeDiR does not extend the shortest path between any pair of nodes. We compare LeDiR to RIM and DARA. As expected, LeDiR achieves its design objective and does not extend any shortest path unlike RIM and DARA. RIM engages all neighbors of the failed node and triggers subsequent cascaded relocation. This can be tolerated in sparse topologies. However, in highly connected networks, i.e., large N or r values, many nodes are involved in the recovery process, as indicated by Fig. 9. As a result, the scope of node movement grows dramatically, and the number of extended paths increases, as shown in Fig. 10.

On the other hand, DARA performs very close to LeDiR in highly connected topologies. In sparse networks, DARA does not do well with significant number of extended paths. However, after a certain point, the number of extended paths in the network started to decline. In Fig. 10(a), this started to happen when the number of actors reaches 80 and, in Fig. 10(b), when the actor's radio range exceeds 75 m. The reason is that in highly connected topologies, cut vertices are found at or near the network periphery. This is particularly very advantageous for DARA since the network would be partitioned into a very large block and few small blocks. In such a case, DARA would select the BC node, i.e., that with the least node degree, from a small block since the large block would be highly connected.

DARA performs significantly worse than LeDiR in sparse topologies. This is attributed to the fact that DARA selects the

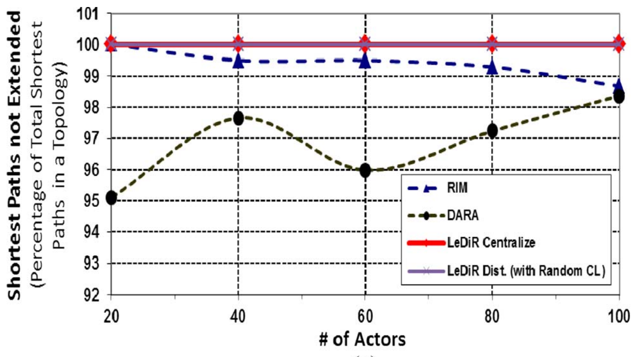


(a)

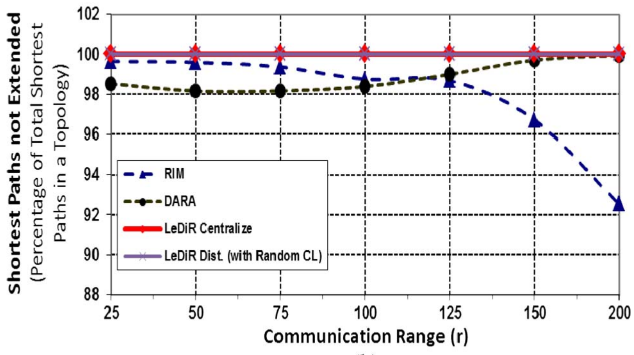


(b)

Fig. 10. Number of extended paths per topology (average over 30 runs) after performing the recovery while varying (a) the network size (with $r = 100$), and (b) the communication range (with $N = 100$).



(a)



(b)

Fig. 11. Percentage of shortest paths that are not extended per topology during the network recovery (average over 30 runs) while varying (a) the network size (with $r = 100$), and (b) the communication range (with $r = 100$).

neighbor with the least degree to replace the faulty actor. DARA does not care whether the selected node belongs to the smallest block. As a result, a node from a significantly large block may move to replace the faulty node. This causes many cascaded movements that extend many paths between nodes.

The graphs in Fig. 11(a) and (b) show the percentage of total number of shortest paths in a topology that does not get extended. Clearly, for LeDiR, the curve stays at 100%. DARA improves when adding more nodes or increasing the radio range since the network connectivity grows. However, RIM performs very close to LeDiR in sparse networks and quite poorly in dense topologies, as also noted in Fig. 10.

Fig. 12 illustrates the difference between LeDiR and the baseline approaches when the node density around the failed node is low. To restore connectivity after the failure of actor A_{14}, LeDiR would move actor A_{17} from the smallest block. RIM would stretch the links from both disjoint blocks and move A_{11} and A_{17} toward A_{14} to reconnect the network. The cascaded relocation for either LeDiR or RIM would not increase any shortest path.

However, when applying DARA, actor A_{11} will move to replace A_{14}. Actors A_{12}, A_2, and A_{13} will move during the cascaded relocation. Thus, as a result, many shortest paths get increased, e.g., the path from A_3 to A_{17}, causing DARA to perform poorly compared with LeDiR or RIM.

Another very important question is whether increasing the size of the network has any effect on the level of path growth in the repaired topology. For example, if we could run DARA for a network of 200 actors, would that lengthen some paths by three hops or more? In our simulations experiments, we have observed that the extension in the path length is independent of the number of nodes in the network and is entirely dependent on which node was selected to replace the faulty actor. For example, if node F fails and A, B and C are the one-hop neighbors, one of them should replace F to reestablish the network connectivity. Now suppose C is also a leaf node; thus, moving C to replace F would not extend any shortest path, and DARA would act like LeDiR, regardless of the size of the network.

Basically, in DARA, the length of the shortest path may grow because a parent moves to replace a faulty node and a child node is inserted in the path to bridge the gap that was created due to the parent departure. In addition, the cascaded nature of movement in DARA and RIM also plays a role since it may get a node to depart one shortest path and join another during the recovery operation. Thus, it is obvious that more paths get extended when the network grows. However, the increase in terms of the number of hopes would remain low, i.e., one or two in many cases. The increase in path length boosts packet loss and data delivery delay and negatively impacts the application. LeDiR strives to avoid that.

General Comments: We would like to make a few additional notes about the performance and applicability of LeDiR. First, LeDiR is designed to recover from a single node failure. Simultaneous node failure may cause conflicting conditions for LeDiR to converge successfully. As mentioned earlier, the probability for multiple nodes to fail at the same time is very small and would not be a concern for LeDiR. Second, LeDiR tends to shrink the smallest block inward toward the failed node; it may negatively affect the node coverage. In general, the

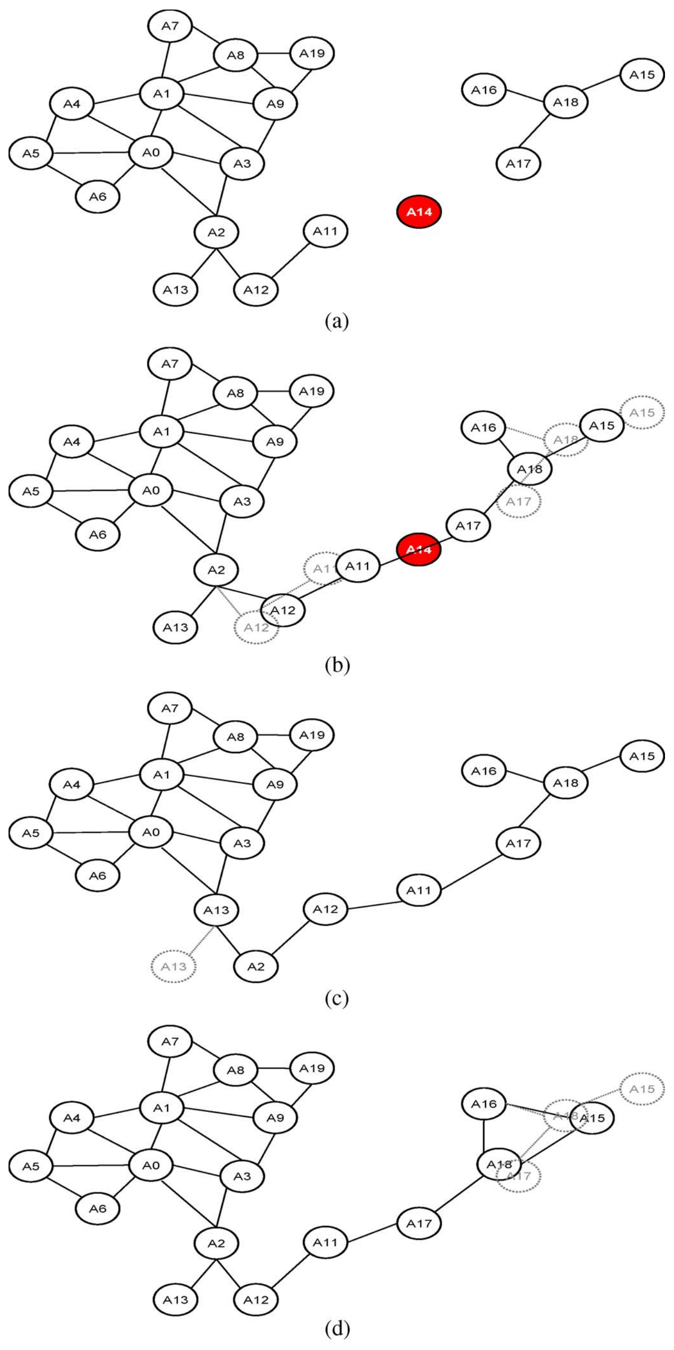


Fig. 12. (a) WSAN with a sparse one-connected topology and a faulty node. (b) Topology recovered by using RIM. (c) Topology recovered by using DARA. (d) Topology recovered by using LeDiR.

```
// EVERY NODE BUILDS ITS SHORTEST PATH ROUTING TABLE (SRT) BASED
// ON THE ROUTE DISCOVERY ACTIVITIES THAT IT INITIATES OR SERVE IN, E.G.
// WHILE EXECUTING A DISTRIBUTED ROUTING PROTOCOL.

LeDiR(J)
1 IF node J detects a failure of its neighbor F
2    IF neighbor F is a critical node
3      IF IsBestCandidate(J)
4          Notify_Children(J);
5          J moves to the Position of neighbor F;
6          Moved_Once ← TRUE;
7          Broadcast(Msg('RECOVERED'));
8          Exit;
9       END IF
10    END IF
11 ELSE IF J receives (a) notification message(s) from F
12    IF Moved_Once || Received Msg('RECOVERED')
13          Exit;
14    END IF
15    NewPosition ← Compute_newPosition(J);
16    IF NewPosition != CurrentPosition(J)
17       Notify_Children(J);
18       J moves to NewPosition;
19       Moved_Once ← TRUE;
20    END IF
21 END IF

IsBestCandidate (J)
// Check whether J is the best candidate for tolerating the failure
22 NeighborList[]← GetNeighbors (F) by accessing column F in SRT;
23 SmallestBlockSize ← Number of nodes in the network;
24 BestCandidate ← J;
25 FOR each node i in the NeighborList[]
      //Use the SRT after excluding the failed node to find the set of
      //reachable nodes;
26    Number of reachable nodes ← 0;
27    FOR each node k in SRT excluding i and F
28       Retrieve shortest path from i to k by using SRT;
29       IF the retrieved shortest path does not include node F
30          No. of reachable nodes ← No. of reachable nodes + 1;
31       END IF
32    END FOR
33    IF Number of reachable nodes < SmallestBlockSize
34       SmallestBlockSize ← Number of reachable nodes;
35       BestCandidate ← i;
36    END IF
37 END FOR
38 IF BestCandidate == J
39    Return TRUE;
40 ELSE
41    Return FALSE;
42 END IF
```

Fig. 13. Pseudocode for the LeDiR algorithm.

impact on coverage would depend on the relationship between the radio and sensing ranges. One would argue that unless the coverage range is significantly larger than the radio range, the loss of a node will have a more dominant impact on the coverage than the connectivity restoration process. The focus of LeDiR is on connectivity and does not factor in the impact on coverage. We plan to consider a joint connectivity and coverage recovery metric in the future.

VI. Conclusion

In recent years, wireless sensor and actor (actuator) networks (WSANs) have started to receive growing attention due to their potential in many real-life applications. This paper has tackled an important problem in mission critical WSANs, that is, reestablishing network connectivity after node failure without extending the length of data paths. We have proposed a new distributed LeDiR algorithm that restores connectivity by careful repositioning of nodes. LeDiR relies only on the local view of the network and does not impose prefailure overhead. The performance of LeDiR has been validated through rigorous analysis and extensive simulation experiments. The experiments have also compared LeDiR with a centralized version and to contemporary solutions in the literature. The results have demonstrated that LeDiR is almost insensitive to the variation in the communication range. LeDiR also works very well in dense networks and yields close to optimal performance even when nodes are partially aware of the network topology.

LeDiR can recover from a single node failure at a time. Generally, simultaneous node failures are very improbable unless a part of the deployment area becomes subject to a major hazardous event, e.g., hit by a bomb. Considering such a problem with collocated node failure is more complex and challenging in nature. In the future, we plan to investigate this issue. Our future plan also includes factoring in coverage and ongoing application tasks in the recovery process and developing a testbed for evaluating the various failure recovery schemes.

APPENDIX

Fig. 13 shows the pseudocode for LeDiR, which is executed independently by each neighbor J of the failed node F. When an actor J detects the failure of a neighbor F, it applies depth-first search to determine that F is indeed a critical node (lines 1-2); J checks its eligibility for replacing F in line 3 by consulting the SRT to find out whether it belongs to the smallest block.

If J qualifies, it will move to the location of F after notifying all its children (lines 4–10). Otherwise, node J checks whether it is to perform a movement to sustain current communication links (line 11), and if so, it identifies a new position and notifies its children before moving (lines 15–20). Nodes only move once (lines 12–14). The $Compute\_newPosition(J)$ procedure identifies where a node k (a child of J) would need to reposition based on the notifications that it has received from nodes other than J. The details of the $Compute\_newPosition(J)$ procedure and corresponding analysis are available in [38].

A new position for a node k would be computed only if k loses its direct communication link to one or multiple parent neighbors, as we already mentioned in Section IV under child movement. The $IsBestCandidate(J)$ procedure identifies the smallest block using the SRT, as described earlier in Section IV, and whether the node J belongs to that block.

REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: Research challenges," *Ad Hoc Netw. J.*, vol. 2, no. 4, pp. 351–367, Oct. 2004.

[2] M. Younis and K. Akkaya, "Strategies and techniques for node placement in wireless sensor networks: A survey," *J. Ad Hoc Netw.*, vol. 6, no. 4, pp. 621–655, Jun. 2008.

[3] A. Abbasi, M. Younis, and K. Akkaya, "Movement-assisted connectivity restoration in wireless sensor and actor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 9, pp. 1366–1379, Sep. 2009.

[4] M. Younis, S. Lee, S. Gupta, and K. Fisher, "A localized self-healing algorithm for networks of moveable sensor nodes," in *Proc. IEEE GLOBECOM*, New Orleans, LA, Nov. 2008, pp. 1–5.

[5] 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.

[6] K. Akkaya and M. Younis, "COLA: A coverage and latency aware actor placement for wireless sensor and actor networks," in *Proc. IEEE VTC*, Montreal, QC, Canada, Sep. 2006, pp. 1–5.

[7] A. Youssef, A. Agrawala, and M. Younis, "Accurate anchor-free localization in wireless sensor networks," in *Proc. 1st IEEE Workshop Inf. Assurance Wireless Sensor Netw.*, Phoenix, AZ, Apr. 2005.

[8] S. Vemulapalli and K. Akkaya, "Mobility-based self route recovery from multiple node failures in mobile sensor networks," in *Proc. 10th IEEE Int. Workshop WLN*, Denver, CO, Oct. 2010.

[9] S. Yang, M. Li, and J. Wu, "Scan-based movement-assisted sensor deployment methods in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 8, pp. 1108–1121, Aug. 2007.

[10] G. Wang, G. Cao, and T. La Porta, "Movement-assisted sensor deployment," *IEEE Trans. Mobile Comput.*, vol. 5, no. 6, pp. 640–652, Jun. 2006.

[11] X. Li, H. Frey, N. Santoro, and I. Stojmenovic, "Localized sensor self-deployment with coverage guarantee," *ACM Sigmobile Mobile Comput. Commun. Revi.*, vol. 12, no. 2, pp. 50–52, Apr. 2008.

[12] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," in *Proc. 22nd Annu.Joint Conf. INFOCOM*, San Francisco, CA, Apr. 2003, pp. 1293–1303.

[13] G. T. Sibley, M. H. Rahimi, and G. S. Sukhatme, "Robomote: A tiny mobile robot platform for large-scale sensor networks," in *Proc. IEEE ICRA*, Washington, DC, May 2002, pp. 1143–1148.

[14] Z. Shen, Y. Chang, H. Jiang, Y. Wang, and Z. Yan, "A generic framework for optimal mobile sensor redeployment," *IEEE Trans. Veh. Technol.*, vol. 59, no. 8, pp. 4043–4057, Oct. 2010.

[15] R.-S. Chang and S.-H. Wang, "Self-deployment by density control in sensor networks," *IEEE Trans. Veh. Technol.*, vol. 57, no. 3, pp. 1745–1755, May 2008.

[16] K. Dasgupta, M. Kukreja, and K. Kalpakis, "Topology-aware placement and role assignment for energy-efficient information gathering in sensor networks," in *Proc. 8th ISCC*, Kemer-Antalya, Turkey, Jun. 2003, pp. 341–348.

[17] W. Youssef, M. Younis, and K. Akkaya, "An intelligent safety-aware gateway relocation scheme for wireless sensor networks," in *Proc. ICC*, Istanbul, Turkey, Jun. 2006, pp. 3396–3401.

[18] H. Liu, X. Chu, Y.-W. Leung, and R. Du, "Simple movement control algorithm for bi-connectivity in robotic sensor networks," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 7, pp. 994–1005, Sep. 2010.

[19] G. Tan, S. A. Jarvis, and A.-M. Kermarrec, "Connectivity-guaranteed and obstacle-adaptive deployment schemes for mobile sensor networks," in *Proc. 28th ICDCS*, Beijing, China, Jun. 2008, pp. 429–437.

[20] K. Akkaya, A. Thimmapuram, F. Senel, and S. Uludag, "Distributed recovery of actor failures in wireless sensor and actor networks," in *Proc. IEEE WCNC*, Las Vegas, NV, Mar. 2008, pp. 2480–2485.

[21] A. Alfadhly, U. Baroudi, and M. Younis, "Least distance movement recovery approach for large scale wireless sensor-actor networks," in *Proc. Workshop FedSenS*, Istanbul, Turkey, Jul. 2011.

[22] M. Younis and R. Waknis, "Connectivity restoration in wireless sensor networks using Steiner tree approximations," in *Proc. IEEE GLOBECOM*, Miami, FL, Dec. 2010, pp. 1–5.

[23] M. Imran, M. Younis, A. M. Said, and H. Hasbullah, "Localized motion-based connectivity restoration algorithms for wireless sensor actor networks," *J. Netw. Comput. Appl.*, vol. 35, no. 2, pp. 844–856, Mar. 2012.

[24] A. Fadhly, U. Baroudi, and M. Younis, "Optimal node repositioning for tolerating node failure in wireless sensor actor networks," in *Proc. 25th IEEE QBSC*, Kingston, ON, Canada, Jun. 2010, pp. 67–71.

[25] M. Sir, I. Senturk, E. Sisikoglu, and K. Akkaya, "An optimization-based approach for connecting partitioned mobile sensor/actuator networks," in *Proc. 3rd Int. Workshop WiSARN*, Shanghai, China, Apr. 2011, pp. 525–530.

[26] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Netw.*, vol. 18, no. 4, pp. 36–44, Jul./Aug. 2004.

[27] S. Das, Liu. H, A. Kamath, A. Nayak, and I. Stojmenovic, "Localized movement control for fault tolerance of mobile robot networks," in *Proc. 1st IFIP Int. Conf. WSAN*, Albacete, Spain, Sep. 2007, pp. 1–12.

[28] S. Das, H. Liu, A. Nayak, and I. Stojmenovic, "A localized algorithm for bi-connectivity of connected mobile robots," *Telecommun. Syst.*, vol. 40, no. 3/4, pp. 129–140, Apr. 2009.

[29] G. Wang *et al.*, "Sensor relocation in mobile sensor networks," in *Proc. 24th Annu. Joint Conf. INFOCOM*, Miami, FL, Mar. 2005, pp. 2302–2312.

[30] K. R. Kasinathan and M. Younis, "Distributed approach for mitigating coverage loss in heterogeneous wireless sensor networks," in *Proc. 3rd IEEE Int. Workshop MENS*, Houston, TX, Dec. 2011.

[31] N. Tamboli and M. Younis, "Coverage-aware connectivity restoration in mobile sensor networks," in *Proc. IEEE ICC*, Dresden, Germany, Jun. 2009, pp. 1–5.

[32] F. Senel, M. Younis, and K. Akkaya, "Bio-inspired relay node placement heuristics for repairing damaged wireless sensor networks," *IEEE Trans. Veh. Technol.*, vol. 60, no. 4, pp. 1835–1848, May 2011.

[33] S. Lee and M. Younis, "Recovery from multiple simultaneous failures in wireless sensor networks using minimum Steiner tree," *J. Parallel Distrib. Comput.*, vol. 70, no. 5, pp. 525–536, May 2010.

[34] S. Lee and M. Younis, “QoS-aware relay node placement in a segmented wireless sensor network,” in *Proc. IEEE ICC*, Dresden, Germany, Jun. 2009, pp. 1–5.

[35] W. Zhang, G. Xue, and S. Misra, “Fault-tolerant relay node placement in wireless sensor networks: Problems and algorithms,” in *Proc. 26th Annu. Joint Conf. INFOCOM*, Anchorage, AK, May 2007, pp. 1649–1657.

[36] F. Al-Turjman, H. Hassanein, and M. Ibnkahla, “Optimized node repositioning to federate wireless sensor networks in environmental applications,” in *Proc. IEEE Int. GLOBECOM*, Houston, TX, Dec. 2011, pp. 1–5.

[37] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*., 1st ed. Cambridge, MA: MIT Press, 1990, pp. 558–565.

[38] M. Younis, S. Lee, and A. Abbasi, “A localized algorithm for restoring internode connectivity in networks of moveable sensors,” *IEEE Trans. Comput.*, vol. 59, no. 12, pp. 1669–1682, Dec. 2010.

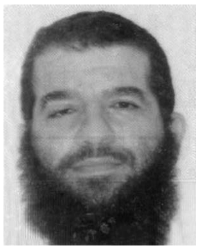
[39] X. Liu, L. Xiao, A. Kreling, and Y. Liu, “Optimizing overlay topology by reducing cut vertices,” in *Proc. ACM Workshop Netw. Operating Syst. Support Digital Audio Video*, Newport, RI, May 2006, pp. 1–6.

[40] A. Savvides, C. C. Han, and M. Srivastava, “Dynamic fine-grained localization in ad-hoc networks of sensors,” in *Proc. Annu. ACM Int. Conf. MOBICOM*, Rome, Italy, Jul. 2001, pp. 166–179.

Ameer A. Abbasi received the B.A and M.A. degrees in computer technology in 1999 and 2000, respectively, from the University of Karachi, Karachi, Pakistan, and the M.S. degree in computer engineering in 2011 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, where he is currently working toward the Ph.D. degree.

He cofounded SofDigital Systems: an information technology (IT) firm that provides consultation and services in the field of software engineering, web technologies, and IT infrastructure. He has a pending U.S. patent and has published several technical papers in refereed conferences and journals. His research interests include fault tolerance systems and topology management for mobile, ad hoc, and wireless sensor networks.

Mohamed F. Younis (SM’08) received B.S. degree in computer science and the M.S. degree in engineering mathematics from Alexandria University, Egypt, in 1987 and 1992, respectively, and the Ph.D. degree in computer science from New Jersey Institute of Technology, Newark, in 1997. He was with the Advanced Systems Technology Group, Honeywell International Inc., which is an aerospace electronic systems R&D organization. While at Honeywell, he led multiple projects for building integrated fault tolerant avionics and dependable computing infrastructure. He also participated in the development of the redundancy management system, which is a key component of the vehicle and mission computer for NASA’s X-33 space launch vehicle. He is currently an Associate Professor with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County. He has five granted and two pending patents. He serves/served on the editorial boards of multiple journals and the organizing and technical program committee of numerous conferences. He has published over 170 technical papers in refereed conferences and journals. His technical interest includes network architectures and protocols, wireless sensor networks, embedded systems, fault tolerant computing, secure communication, and distributed real-time systems.



Uthman A. Baroudi received the B.Sc. and M.S. degrees in electrical engineering from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Concordia University, Montreal, QC, Canada, in 2000.

In 2000, he joined Nortel Networks, Ottawa, ON, Canada, to work on Research and Development for next-generation wireless networks. Since January 2002, he has been an Assistant Professor with the Computer Engineering Department, KFUPM. He has extensive teaching and research experience. He taught and developed several graduate and undergraduate COE courses on wireless and computer networking and computer system performance evaluation. He has published several papers in refereed international journals and conferences. He has over 40 publications in international journals and conference proceedings and one U.S. patent. His research interests lies in the areas of radio resource management and quality-of-service provisioning for next-generation wireless networks, wireless ad hoc, and sensor and actuator networks.