Movement-Assisted Connectivity Restoration in Wireless Sensor and Actor Networks

Ameer Ahmed Abbasi, Mohamed Younis, *Senior Member*, *IEEE*, and Kemal Akkaya, *Member*, *IEEE*

Abbasi, A.A., Younis, M.F., & Akkaya, K. (2009). Movement-Assisted Connectivity Restoration in Wireless Sensor and Actor Networks. IEEE Transactions on Parallel and Distributed Systems, 20, 1366-1379.

Abstract—Recent years have witnessed a growing interest in applications of wireless sensor and actor networks (WSANs). In these applications, a set of mobile actor nodes are deployed in addition to sensors in order to collect sensors' data and perform specific tasks in response to detected events/objects. In most scenarios, actors have to respond collectively, which requires interactor coordination. Therefore, maintaining a connected interactor network is critical to the effectiveness of WSANs. However, WSANs often operate unattended in harsh environments where actors can easily fail or get damaged. An actor failure may lead to partitioning the interactor network and thus hinder the fulfillment of the application requirements. In this paper, we present DARA, a Distributed Actor Recovery Algorithm, which opts to efficiently restore the connectivity of the interactor network that has been affected by the failure of an actor. Two variants of the algorithm are developed to address 1- and 2-connectivity requirements. The idea is to identify the least set of actors that should be repositioned in order to reestablish a particular level of connectivity. DARA strives to localize the scope of the recovery process and minimize the movement overhead imposed on the involved actors. The effectiveness of DARA is validated through simulation experiments.

Index Terms—Connectivity restoration, controlled node mobility, fault tolerance, wireless sensor and actor networks.

1 Introduction

Wireless sensor and actor networks (WSANs) have attracted lots of interest in recent years due to their potential use in numerous applications such as boarder protection, battlefield reconnaissance, space exploration, search and rescue, etc. A typical WSAN consists of a larger set of miniaturized sensor nodes reporting their data to significantly fewer actor (actuator) nodes [1]. Sensors probe their surroundings and report their findings to one or multiple of actors, which process the collected sensor readings and respond to emerging events of interest. An actor's response would depend on its capabilities, which vary based on the application and the expected role the actor plays. For example, an actor can deactivate a landmine, extinguish a fire, and rescue a trapped survivor. It is worth noting that a heterogeneous set of actors may be employed and assigned complementary roles.

In most application setups, actors need to coordinate with each other in order to share and process the sensors' data, plan an optimal response and pick the most appropriate subset of actors for executing such a plan. For example, in forest monitoring applications, fire-extinguishing actors need to collaborate with each other in order to effectively control a fire and prevent it from spreading. The selection of actors that need to be engaged can be based on many factors such as the actor's capabilities, actor's proximity to the detected event, and actor's current load. All of these factors would require a frequent update of the actor's state. To enable such interactions, actors need to stay reachable from each other. In other words, a connected interactor network has to be maintained at all times.

An actor failure can cause the loss of multiple interactor communication links and may partition the network if alternate paths among the affected actors are not available. Such a scenario will hinder the actors' collaboration and thus have very negative consequences on the WSAN application. Therefore, the actors should be able to detect and recover from the failure of one of them. Given that WSANs usually operate autonomously and unattended, the recovery should be a self-healing process for the network and should be performed in a distributed manner. In addition, the network recovery should be both quick and lightweight. Rapid recovery is desirable in order to maintain the WSAN responsiveness to detected events. Moreover, the overhead should be minimized in order to ensure the availability of actors' resources for application-level tasks.

While restoring connectivity is crucial as justified above, it may still take a certain amount of time, which WSANs may not tolerate in delay-critical applications. In such a case, a possible solution is to be proactive and maintain two paths among every pair of actors in the network so that if one of the paths fails due to node failure(s), the second one can immediately be used. To achieve this, 2-connectivity should be maintained among the actors. That is, there should be at least two node-independent paths between any selected actors.

In this paper, we study the impact of a node failure on the interactor connectivity in WSANs. We present DARA, a Distributed Actor Recovery Algorithm, which opts to efficiently restore the connectivity of an interactor network to its pre-node-failure level. Based on the type of connectivity considered, two algorithms, namely, DARA-1C and DARA-2C, are developed to address 1 and 2-connectivity requirements, respectively. DARA is a localized scheme that avoids the involvement of every single actor in the network. DARA pursues a coordinated multiactor relocation in order to

- *A.A. Abbasi is with the Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran-31261, Saudi Arabia. E-mail: ameer_abbasi@kfupm.edu.sa.*
- *M. Younis is with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD 21250. E-mail: younis@cs.umbc.edu.*
- *K. Akkaya is with the Department of Computer Science, Southern Illinois University, Carbondale, IL 62901. E-mail: kemal@cs.siu.edu.*

Manuscript received 2 Feb. 2008; revised 7 Oct. 2008; accepted 7 Nov. 2008; published online 20 Nov. 2008.
Recommended for acceptance by A. Boukerche.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-02-0045.
Digital Object Identifier no. 10.1109/TPDS.2008.246.

reestablish communication links among impacted actors. The main idea of DARA-1C is to replace the dead actor by a suitable neighbor. The selection of the best candidate (BC) neighbor is based on the node degree and the physical proximity to the dead actor. The relocation procedure is recursively applied to handle any actors that get disconnected due to the movement of one of their neighbors (e.g., the BC that replaced the faulty actor). Similarly, DARA-2C identifies the nodes that are affected, i.e., lost their 2-connectivity property, due to the failed actor. Some of these nodes are then relocated in order to restore 2-connectivity. Although both DARA-1C and DARA-2C pursue node relocation to restore the desired level of connectivity, they fundamentally differ in the scope of the failure analysis and the recovery.

The main optimization objective of DARA is to minimize the total distance traveled by the involved actors in order to limit the overhead incurred by the movement. In addition, DARA strives to minimize the messaging costs in order to maintain scalability. The entire recovery process is distributed, enabling the network to self-heal without any external supervision. DARA is validated analytically and through simulation. The simulation results shows that DARA is both efficient in achieving minimal total traveled distances and lightweight in terms of required communication resources.

This paper is organized as follows: The next section describes the system model that we consider throughout the paper. Related work is covered in Section 3. Section 4 discusses the multiactor coordinated recovery process and describes the DARA-1C algorithm in detail. DARA-2C is discussed in Section 5. The validation experiments and performance results are discussed in Section 6. Section 7 concludes the paper with a summary and a highlight of our planned future extensions.

2 System Model

As mentioned earlier, a WSAN involves two types of nodes: sensors and actors. Sensors are inexpensive and highly energy constrained and have limited data processing capabilities. On the other hand, actors are more capable nodes with relatively more onboard energy supply and richer computation and communication resources. They are assumed to know their positions through GPS or other range-based localization techniques, e.g., [2]. The transmission range of actors is finite and significantly less than the dimensions of the deployment area. Although actors theoretically can reach each other via a satellite channel, frequent interactor interaction as required by WSAN applications would make the often-intermittent satellite links unsuitable. It is thus necessary for actors to rely mostly on contemporary interactor wireless links for coordination among themselves. The communication range of an actor refers to the maximum euclidean distance that its radio can reach. Meanwhile, the action range of an actor is defined as how far it can be effective from its current position. It is also assumed that the radio range of all actors is equal and limited and that the communication links are symmetric.

Actors are randomly deployed in an area of interest. Upon deployment, actors are assumed to discover each other and form a connected network using some of the existing techniques such as [3]. We also assume that the actors can move on demand in order to act on larger areas or to enhance the interactor connectivity. Fig. 1 articulates the considered WSAN model. An actor collects sensor data in its neighborhood and collaborates with other actors. Some of the actors

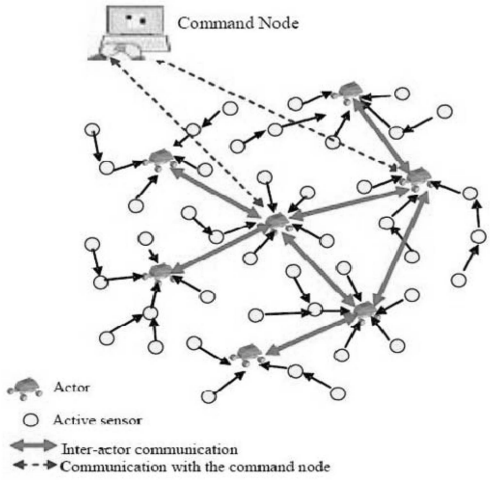

Fig. 1. An example WASN with a connected interactor network.

can interact with a remote command center through a long-haul communication link, e.g., through a satellite, to report on their activities and detected event/targets.

The following notations are used in the paper:

- 1-hop-Neighbors(A_i) or simply $\mathrm{Neighbors}(A_i)$ is the set of actors that are directly reachable or adjacent to A_i (i.e., lies within the communication range of A_i).
- 2-hop-Neighbors(A_i) is the set of actors that are reachable from A_i through A_j where $A_j \in \mathrm{Neighbors}(A_i)$.
- $\mathrm{Adjacent\_Siblings}(A_i, A_f)$ is the set of actors that are neighbors to two adjacent actors A_i and A_f, where $A_i \in \mathrm{Neighbors}(A_f)$, i.e., $\mathrm{Adjacent\_Siblings}(A_i, A_f) = \{A_k | A_k \in (\mathrm{Neighbors}(A_f) \cap \mathrm{Neighbors}(A_i)) \wedge A_i \in \mathrm{Neighbors}(A_f)\}$.
- $\mathrm{Dependents}(A_i, A_f)$ is the set of $\mathrm{Neighbors}(A_i)$ that are not neighbors of A_f, where $A_i \in \mathrm{Neighbors}(A_f)$, i.e., $\mathrm{Dependents}(A_i, A_f) = \{A_k | A_k \in \mathrm{Neighbors}(A_i) \wedge A_k \notin \mathrm{Neighbors}(A_f) \wedge A_k \neq A_f\}$. Alternatively, $\mathrm{Dependents}(A_i, A_f)$ can be defined as $\mathrm{Neighbors}(A_i) - \mathrm{Adjacent\_Siblings}(A_i, A_f)$.

It is worth noting that although we consider such a system model, our algorithm is also applicable to mobile robot networks where no sensors are employed.

3 Related Work

The bulk of the research work on WSANs has focused on the appropriate positioning and/or engagement of actors in order to maximize coverage and responsiveness [3], [4], [5]. Interactor connectivity has only been studied in the context of actor placement without considering potential breaks in the interactor connectivity. For example, the goal of [6] is to maximize the coverage of actors without violating connectivity. Similar to DARA, actors' mobility is exploited. The idea is to model repelling forces among actors and from the boundaries of the deployment area so that the actors will spread for better area coverage. However, the approach does not handle network connectivity problems caused by the failure of an actor. On the other hand, some work has focused on failure detection and recovery. For example, Elhadef et al. [7] have developed a distributed mechanism

for detecting faulty actors and diagnosing the type of failure. Meanwhile, Ozaki et al. [8] have studied the establishment of a fault-tolerant operation of WSANs. The idea is to designate multiple actors to which a particular sensor reports its finding, and similarly, each actor is assigned multiple sensors that can report on the same event. Thus, an event is guaranteed to be detected and monitored by at least one actor even if a sensor or an actor fails. While tolerance of actor failure is the focus of DARA, it is in the context of maintaining the interactor connectivity rather than the reliable dissemination of a sensor's data reports.

Relocation of nodes has also been pursued in networks of mobile sensors to counter holes in coverage caused by sensor failures [9]. The idea is simply to identify some spare sensors from different parts of the network that can be repositioned in the vicinity of the faulty nodes. Since moving a node for a relatively long distance can drain a significant amount of energy, a cascaded movement is proposed if there are a sufficient number of sensors on the way. The idea is to determine intermediate sensor nodes on the path and replace those nodes gradually. While our work is similar in the sense that we use cascaded movements, connectivity is not considered in [9]. In addition, the movement of actors has been studied in the context of its impact on network operation. For instance, in [10], a mechanism to efficiently handle the mobility of actors is presented. This mechanism limits the broadcast of the updates of actors' location to within the Voronoi diagram of surrounding sensors. Sensors predict the movement of actors based on Kalman filtering of previously received updates. In this way, the sensors' energy consumed in keeping track of the actors' location is reduced significantly. Another approach is proposed in [5] to counter unreliable data collection caused by the relative short lifetime of sensors around an actor node. The idea is to continually reposition the actor to change the data paths. Our objective is rather to minimize the overhead incurred by actors while recovering from a failure. For discussion on other placement strategies for mobile sensors, the reader is referred to [11].

Establishing k-connectivity has been pursued in the general realm of wireless networks as a means to counter node and link failures. The rationale is that ensuring the availability of k node-independent paths would allow the network to tolerate $k-1$ failures. For example, Basu and Redi [12] have studied the problem of maintaining 2-connectivity in mobile robot networks even under link or node failure. Like DARA, their approach is based on moving a subset of the robots to restore the 2-connectivity lost due to the failure of a robot. However, their algorithm is centralized in its nature and requires complete and accurate knowledge of the entire network topology. Another similar study has been recently reported in [13]. The goal is to boost the connectivity level of a network from one to two. The basic idea is to eliminate all critical nodes, i.e., cut vertices, in the network by moving selected noncritical nodes. The approach is for each critical node "u" to direct two noncritical nodes among its neighbors to move in order to be reachable from one another, and thus, "u" will no longer be a cut vertex. The algorithm collects and stores p-hop neighbor information at every node. The higher the value of p is, the better the performance of the approach. In contrast, DARA-2C requires only two-hop neighbor information and outperforms the approach in [13], as will be shown in Section 6.

K-connectivity has also been studied in wireless sensor networks. Most of the published work focuses on how to set the network topology through careful node placement or configuration [11]. For instance, a heuristic for establishing a fault-tolerant WSN has been proposed in [14]. The goal is to form a k-connected topology that ensures coverage while engaging the least number of sensors. However, neither the handling of node failure nor node mobility has been considered. Another work that considers k-connectivity to achieve a fault-tolerant network topology has been presented in [15]. Initially, each node discovers all reachable nodes by broadcasting "hello" messages using maximum transmission power. Each node collects and stores neighbor information to construct a local k-connected baseline graph. The graph is then pruned to identify the minimal set of links that sustain k-connectivity while using the least energy to reach neighbors. The baseline graph is then consulted to reestablish a new k-connected topology if a neighbor becomes unreachable due to mobility. DARA-2C does not assume the availability of such a graph to use in the recovery. In addition, the approach appears to assume a very highly connected network, which may be impractical.

4 Restoring Connectivity through Actor Movement

This section investigates effective means for restoring 1-connectivity of WSANs that gets partitioned due to failure of only one particular actor. We opt to pursue a localized, distributed, and lightweight approach in order to suit the nature of WSANs. In this section, we describe the recovery problem and the proposed DARA-1C approach.

4.1 The Connectivity Restoration Problem

The impact of actor's failure on the network connectivity can be very limited and can also be very dramatic. When the lost actor is a leaf node, no other actors will be affected. Meanwhile, when the failed actor serves as a cut-vertex node in the network, playing the role of a gateway between two subnetworks, a serious damage to the network connectivity will be inflicted. Basically, with the loss of a cut vertex, the network gets partitioned into disjoint subnetworks. Fig. 2a shows an example interactor network. In that example, the loss of a leaf actor such as A_3 will not impact the connectivity of the network. The same applies to nonleaf nodes like A_{12} and A_{14} when alternate paths are available between the neighbors of the failed actor. Meanwhile, both A_1 and A_9 are cut vertices, and the failure of either of them will result in two or more disjoint blocks of actors.

In order to tolerate the actor failure that causes network partitioning, two methodologies can be identified: 1) precautionary and 2) real-time restoration. The precautionary methodology strives to provision fault tolerance in the network topology both at setup and during normal operation. The idea is to establish a k-connected topology such that every node can reach other nodes over at least k independent paths. Such arrangement will allow the network to seamlessly tolerate the failure of up to $k-1$ actors [12]. As we mentioned earlier, provisioning a high level of connectivity may require the deployment of a large number of actors and may thus be impractical due to their high costs. In addition, it may constrain the mobility of actors and negatively affect application-level functionality. We consider the case of 2-connectivity in Section 5. On the other hand, real-time restoration implements a recovery procedure when an actor failure is detected. We argue that real-time restoration better suits WSANs since they are asynchronous and reactive in nature

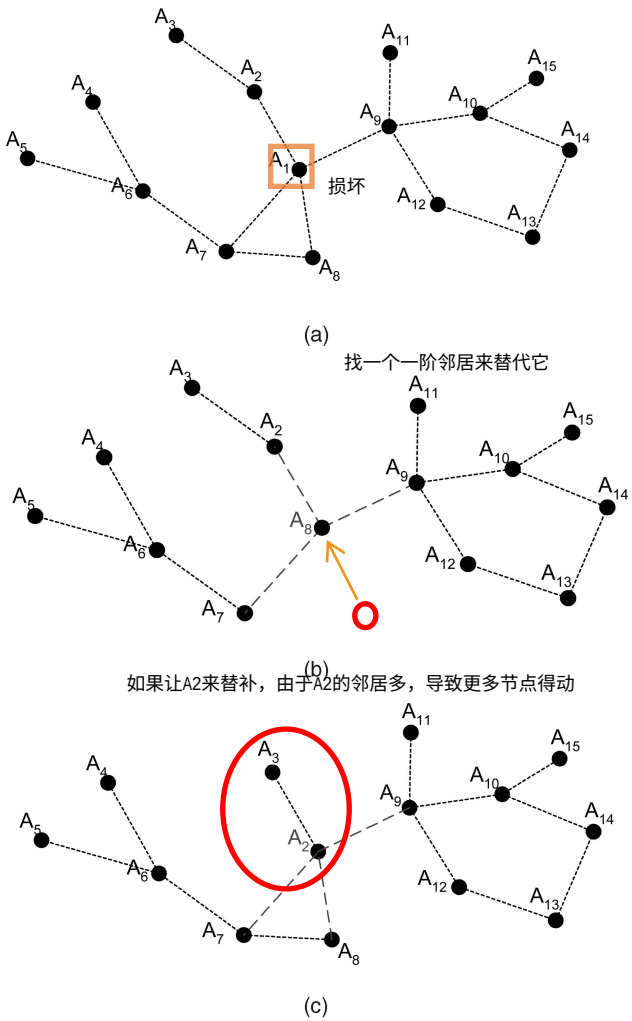

Fig. 2. (a) An example connected interactor network. (b) Node A_8 replaced the faulty actor and reestablish connectivity between actors. (c). The topology if node A_2 is picked to replace A_1. A_3 has also to move to sustain its connectivity to A_2.

and it is difficult to predict the location and the scope of the failure. Therefore, adaptive schemes can best scope the recovery process depending on the effect on the failure on the network connectivity.

4.2 Approach Overview

To repair partitioned actor networks, we propose DARA-1C, a distributed actor recovery algorithm. DARA-1C promotes real-time restoration and does not make any assumption about the network topology prior to an actor's failure. Before presenting the detailed steps, we first explain DARA-1C using the network topology in Fig. 2a. As we mentioned earlier, A_1 is considered a cut vertex. The failure of A_1 will cause the partitioning of the network into three disjoint subnetworks, namely, $\{A_2, A_3\}$, $\{A_4, A_5, A_6, A_7, A_8\}$, and $\{A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}\}$. Instead of running a block movement [12], DARA-1C pursues a cascaded relocation of few actors. Block movement, as defined in [12], is a solution based on the relocation of all nodes within a partition (block). Specifically, the neighbor of the failed node will take the lead and move toward the location of the failed node. The other nodes in the partition follow through in the same direction headed by the leader node and maintain their current links. We argue that cascaded relocation requires a less number of movements when compared to block movement, which rather causes all the actors in the subnetwork to move. In addition, block movement requires all the actors in the subnetwork to be aware of where and how far to move, which introduces extra messaging. DARA-1C only requires that each actor A_i form a two-hop neighbor list, i.e., the set 2-hop-Neighbors(A_i), which can be easily maintained.

Coming back to our example, DARA-1C expects nodes A_2, A_7, A_8, and A_9 to initiate the recovery procedure since they are adjacent to A_1. The basic idea of DARA-1C is that one of the one-hop neighbors of the faulty actor will relocate to where A_1 is to reconnect the disjoint partitions. Clearly, three issues exist: 1) how do we prevent further partitioning conditions that result from moving nodes, 2) which of the neighbors will move, and 3) how do we ensure that only one actor will replace the failed actor. To deal with the first issue, DARA-1C pursues cascaded node relocation in order to sustain connectivity. DARA-1C establishes selection criteria that are based on the node degree and the proximity of neighbors. DARA-1C prefers the node that has fewer neighbors since the scope of the cascaded relocation will be limited. Legible choices are filtered based on their proximity to the failed node since the travel distance will be shorter and thus the overhead will be limited. In case of ties, the node ID is used for final arbitration. Since every actor is aware of its two-hop neighbors, nodes A_2, A_7, A_8, and A_9 would independently come to the same conclusion, and only one of them will assume responsibility for conducting the recovery.

Among A_2, A_7, A_8, and A_9, A_9 has the highest node degree and will be thus excluded. Also, A_7 has node degree of two, which is larger than that of A_2 and A_8. Since A_2 and A_8 have the same node degree and are equidistant to A_1, node A_8 is picked based on the node ID. Fig. 2b shows the network topology after the recovery. It is worth noting that if A_2 is to be picked, A_3 may also need to move, as shown in Fig. 2c, and thus, the total travel distance of all involved actors will be longer than the case when node A_8 is selected. Therefore, DARA-1C may not always yield the optimal results, which is typical for a greedy approach. Nonetheless, as we later discuss, DARA-1C employs a few optimization techniques that proves to be effective in limiting excessive cascaded motion. For example, in Fig. 2c, if A_4 is reachable from A_3, we can allow A_3 to lose connectivity to A_2, and thus, we can further reduce the overhead.

4.3 Detailed DARA-1C Steps

DARA-1C is a distributed algorithm that requires little coordination. As described in the previous section, DARA-1C relies on the neighbors of a failed node for initiating the recovery process. Since these nodes will not be able to communicate after the failure of the cut-vertex node that used to connect them, they cannot coordinate with each other. Instead, they perform the same steps. DARA-1C guarantees that the recovery process converges. The following are the detailed steps.

4.3.1 Heartbeats and Neighbor List Maintenance

The only prefailure knowledge that DARA-1C requires for each actor to have is the list of neighboring actors. Each actor in the network should be aware of its one-hop and two-hop neighbors and should strive to keep this information current. The list of neighbors can be updated each time one of these neighbors changes its position. Under normal conditions, an actor can move to enhance performance,

conduct application-specific action, etc. [16]. The two-hop neighbor list is used in identifying the actor that may need to replace a failed node and in optimizing the recovery process. Each actor will also tabulate the node degree, position, and ID of all its neighbors.

The two-hop neighbor information table, which we refer to thereafter as TwoHopTable, will be used to identify cut vertices in the network using distributed algorithms like the one proposed in [17]. This type of algorithms generally trade off the need for a network-wide state with the accuracy of identifying cut vertices. It has been shown that the probability of missing a cut vertex is zero while a very high percentage of the picked nodes are really cut vertices [18]. Using two-hop information, it was shown that the accuracy can reach 90 percent [18]. The TwoHopTable will be updated by the individual nodes. Actors will periodically send heartbeat messages to their neighbors to ensure that they are functional and also report changes to the one-hop neighbors. Obviously, for a node B that is a neighbor to A, the one-hop neighbors of A other than B itself are among the two-hop neighbors of B. Formally, we have

$$1\text{-}hop\text{-}Neighbors(A) - \{B\} \subseteq 2\text{-}hop\text{-}Neighbors(B).$$

4.3.2 Detecting Actor Failure and Initiating the Recovery Process

Missing heartbeat messages can be used to detect the failure of actors. Depending on the actor's position in the network topology, major or no recovery may be needed. As discussed in the prior section, the failure of a node that does not hinder the connectivity of other nodes would not necessitate any adjustments to the network topology. Despite the fact that failure of such a node will not cause any problems to the interactor connectivity, it can create other problems such as forming coverage holes, disrupting the data collection from that particular region, etc. In such cases, depending on the application-level requirements, these problems need to be handled. We would like to note, however, that handling such problems is out of scope of this paper. We only focus on restoration of interactor connectivity when a cut-vertex node fails. Such a failed cut vertex will trigger the execution of DARA-1C. The entire detection and recovery process is localized. Actors that detect the failure of one of their neighbors will decide on the scope of the recovery. Again, if the failed actor is a cut vertex that causes the network to partition, the restoration process will be initiated on these neighboring actors. The failed actor is referred to thereafter as A_f.

4.3.3 Best Candidate Selection

DARA-1C restores the connectivity of a partitioned network by substituting A_f with one of its one-hop neighbors. This way, all broken links can be reestablished, and potential ambiguity about the signal propagation conditions is avoided. The latter is handled by moving the substitute actor to the exact position of A_f. The obvious question is which neighbor should be picked. DARA-1C strives to identify the BC for replacing A_f using the following criteria in order:

1. *Least node degree*. The rationale is that the impact of moving a node that has many neighbors will be significant. Thus, DARA-1C favors replacing the failed actor with the neighbor that has the least node degree.
2. *Closest proximity to failed actor*. In order to minimize the movement overhead, the nearest actor to A_f will be favored.
3. *Highest actor ID*. It is possible that among the neighbors of A_f, two or more actors have identical node degrees and are equidistant to it. The actor with the greatest ID will be picked to break the tie.

It is important to note that DARA-1C is executed simultaneously on all neighbors of A_f without any coordination among them (since they may not be connected anymore). These three criteria guarantee that the same BC is identified at all neighbors of A_f. We expect the use of node ID to be rare in practice, especially with the available high-resolution ranging technology for which the probability that two actors are equidistant to a third one is almost zero. Nonetheless, node IDs serve as safeguard for preventing conflicts in the selection of the BC.

4.3.4 Cascaded Node Relocation

The BC actor will prepare itself to move to the location of A_f and calculate the expected time it will take to reach the new location. In addition, before moving to the new location, the BC will inform all actors in the set $\texttt{Dependents}(\texttt{BC}, A_f)$ about its movement and the time it will take to reach to the new location by sending a "MOVING" message. The BC will then broadcast a "RECOVERED" message upon arriving at the destination.

The dependent neighbors (children) of the BC, i.e., those neighbors of the BC that were not one-hop neighbors of Af, keep waiting until they receive the "RECOVERED" message indicating the restoration process has been completed and that they are still connected. Such scenario happens when the relocated actor stays in the radio range of these dependent children. If some of the dependents do not hear the "RECOVERED" message, they will assume that they got disconnected and apply DARA-1C again as if their parent has stopped functioning. In other words, the recovery process will be applied recursively to trigger the cascaded relocation of affected actors. Thus, these detached dependents identify a BC at the children level to relocate to the position of their parent. Please note that the child BC will do exactly what its parent has done, i.e., broadcast "RECOVERED" message to its neighbors when it ceases motion. This process continues until every dependent child is connected. The pseudocode of the DARA-1C algorithm is detailed in Appendix A.

4.4 Optimization for Special Topologies

One of the scenarios, for which the performance DARA-1C in terms of the total movement distance may not be good, would be when the cut vertex is part of a cycle. In this case, the BC will be part of a network, which is 2-connected. That is, each actor can reach any of the actors through two independent paths. An example is depicted in Fig. 3. Since DARA-1C only considers two-hop information, it triggers movements until all nodes on the ringlike subnetwork are reconnected. For instance, in Fig. 3, when A_1 fails, the BC A_2 will replace it. Moving A_2 will result in successive motion of the other nodes on the ring until A_5. The reason is that none of these nodes will receive any "RECOVERED" messages. Therefore, in the worst case, a travel distance of $(m-2)r$ can result, where m is the number of nodes in the cycle. Clearly, in Fig. 3, relocating A_2 suffices to restore

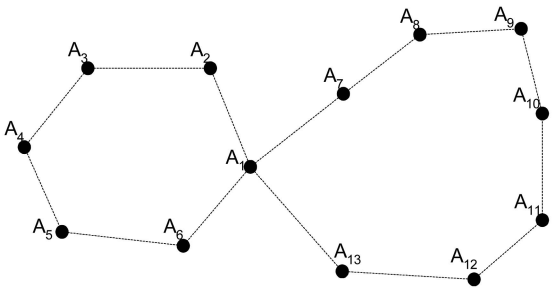

Fig. 3. A special topology for which DARA-1C does not perform well.

connectivity (i.e., optimal solution), which would only need a maximum total travel distance of r.

Since such ringlike topologies are not uncommon in certain WSAN applications, we introduce an optimization to DARA-1C in order to reduce the travel distance at the expense of higher message complexity. The idea is to start a local flooding at BC and check whether the picked child (e.g., A_3 in Fig. 3) is already connected through other siblings. In this case, the message is forwarded in the subnetwork and a response from one of the siblings is awaited. If such a message is received, no further relocation is performed. We call this version of the approach DARA-1CLF, where the suffix LF denotes local flooding.

4.5 Algorithm Analysis

In this section, we prove the correctness and analyze the performance of the DARA-1C algorithm. We introduce the following theorems.

Theorem 1. *DARA-1C guarantees connectivity and converges in a maximum of $N-3$ relocation steps, where N is the number of actors, regardless of the number of network partitions that a failure of the cut vertex creates.*

Proof. Since actors have the same radio range and an actor will be positioned at the same spot of the failed node, all the broken links will be reestablished. In addition, the successive node replacements will stop whenever a leaf node (i.e., a node with a node degree of 1) or a neighbor of the failed node is considered. No node will move more than once, even when the failed node is part of a cycle. For N actors, the worst case performance matches a one-dimensional network topology like the one shown in Fig. 4 and for which $N-4$ nonfaulty nodes have to move. Therefore, DARA-1C terminates in a maximum of $N-3$ steps. □

Theorem 2. *The message complexity of the DARA-1C is $O(N)$, where N is the number of actors.*

Proof. Since DARA-1C requires two-hop neighbor information to restore interactor connectivity, this requires two messages per node. In addition, every BC in DARA-1C broadcasts only two messages: 1) to inform its dependent children, i.e., members of $Dependents(BC, A_f)$, about its movement and 2) to announce its successful relocation. Thus, in the worst case, in which all nodes move, a total of $4 * N$ messages will be sent. For DARA-1CLF, a BC may need also to perform a local flooding that can span $N-3$ nodes in the worst case, as shown in Fig. 4. Therefore, a total of $4N+N-3$ messages would be needed. In conclusion, both versions of DARA-1C have a total message complexity of $O(N)$. □

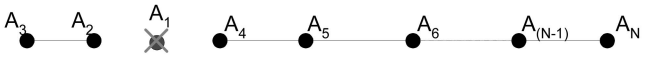

Fig. 4. The worst-case topology for DARA-1C for which $N-3$ actors would have to move since A_2 and A_4 has the same node degree, and they are equidistant to the failed A_1 node.

Theorem 3. *The maximum distance a node travels in DARA-1C is r, where r is the actor radio range.*

Proof. Since a failed node is replaced by one of its neighbors, the maximum travel distance in the worst case is the actor radio range, which is r. Similarly, a relocated node may be replaced by one of its neighbors that are at most at a distance of r. Since each node travels only once, the maximum travel distance for a node will be r. □

Theorem 4. *The convergence time of the DARA-1C algorithm to restore interactor connectivity is $O(p+r)$, where p is the time that the process takes for a node to become a BC, and r is the actor radio range.*

Proof. Let us assume that the maximum time for an actor to detect a failure and become a BC is p. In the movement process, the total distance traveled by such a BC can be at most r, as proved in Theorem 3. Assuming that the relocation of the BC triggers further relocation of k actors, each traveling a distance of r, the total time to restore 1-connectivity will be proportional to $(k+1) * r$ since the relocations are performed in a sequential manner. Thus, the convergence time of DARA-1C to restore 1-connectivity in the worst case is $p+(k+1) * r$, which is $O(p+r)$. □

5 Maintaining 2-Connectivity

This section focuses on restoring 2-connectivity after the failure of an actor. The interactor network is assumed to be initially biconnected. The following sections describe the 2-connectivity restoration problem and present the proposed DARA-2C algorithm.

5.1 Problem Analysis and Approach Overview

As shown in Section 4, when a critical actor fails, some of the nodes may be temporarily isolated until the network connectivity is restored, and thus, the network operation may be disrupted during the recovery. A way to avoid such short-lived disruption is to establish a 2-connected interactor network after deployment. In a 2-connected network, there are at least two node-independent paths among each pair of nodes. This type of connectivity would ensure continual interactor coordination even if an actor fails. Such robust operation is particularly important in WSANs where real-time decisions are made and when the network operates on remote and inaccessible areas such as other planets. Since the network should sustain such robustness throughout its lifetime, 2-connectivity should be restored after an actor fails.

The failure of a particular actor may or may not cause the network to lose its 2-connectivity property. For instance, in Fig. 5, the failure of A_9 will not affect the 2-connectivity of the network since every actor still maintains two node-independent paths to every other actor in the network. The same

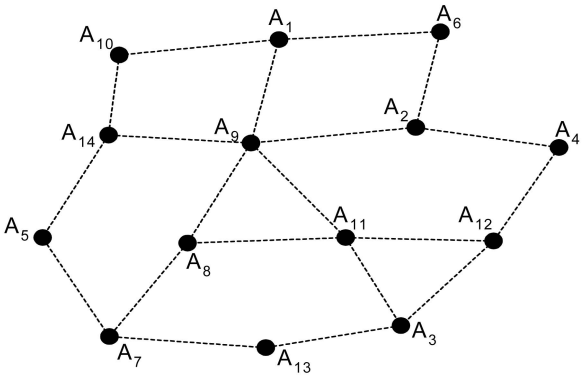

Fig. 5. An example of a 2-connected WSAN with 14 actors. No cut vertex exists.

applies to the failures of A_5, A_6, A_8, A_{10}, A_{11}, and A_{13}. As these types of actor failures do not affect the 2-connectivity property, no recovery is required. Meanwhile, the failures of A_{12} and A_7 will affect the 2-connectivity of A_4, A_5, and A_{13}. Thus, the 2-connectivity restoration problem can be defined more formally as follows: "given the failure of a node A_i in a 2-connected interactor network that violates the 2-connectivity property, the goal is to restore the 2-connectivity with the minimal travel distance of actors." The proposed DARA-2C algorithm opts to address this problem. Since we are utilizing the same idea of node relocation to restore the connectivity, we kept the name DARA but added the suffix "2C" in order to show that the algorithm is meant for restoring 2-connectivity as opposed to 1-connectivity. The following theorems analyze the problem and categorize the solution.

Theorem 5. *A network cannot be 2-connected if it has a cut vertex.*

Proof. The proof is well known and can be found in any Graph Theory books such as [20] □

Definition 1. *The geometric convex hull [20] is the minimal convex set of points containing a nonempty finite set of points.*

Definition 2. *A network periphery is the convex hull of the nodes of the network. Since the nodes are placed in a plane, the network periphery is a simple closed polygonal chain, unless the nodes are collinear.*

Definition 3. *A connectivity hole [21] is an area in which the edges between nodes form a closed polygonal without links between nodes that are not adjacent on the polygonal chain.*

Definition 4. *A boundary node is one that is located at the network periphery or on the closed polygonal chain surrounding a connectivity hole.*

Fig. 6 shows examples of boundary nodes. Nodes A_3, A_4, A_6, and A_9 are on the network periphery. Meanwhile, A_1, A_2, A_5, A_7, and A_8 are located on the closed polygonal chain of a hole in connectivity, the shaded area in which there is no node that connects the periphery of the network and the periphery of partition #2.

Theorem 6. *In a 2-connected network, a failure of a node A_f can cause another node A_c to be a cut vertex only if A_f is a boundary node.*

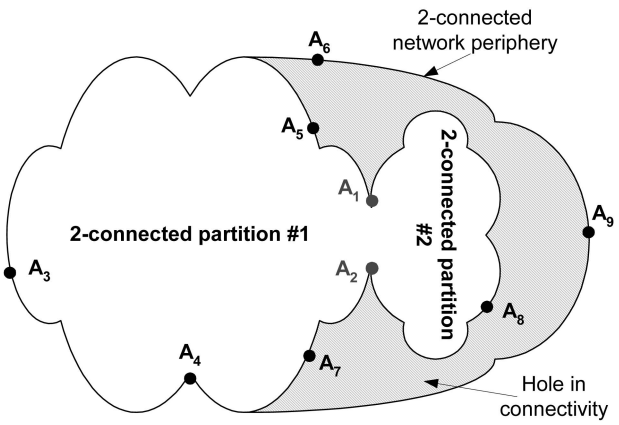

Fig. 6. Illustration of example boundary nodes in a network.

Proof. Without loss of generality, let us consider the 2-connected network topology articulated in Fig. 7. The network is composed of two 2-connected partitions that are linked through two vertices A_1 and A_2. Obviously, the failure of either A_1 or A_2 will cause the other to be a cut vertex and thus violate the network's 2-connectivity property (Theorem 5). Since A_1 and A_2 are the only nodes that link the two partitions, they must be boundary nodes that are located on the network periphery. The same conclusion can be easily reached for boundary nodes located on the edges of a connectivity hole in the network, e.g., nodes A_1 and A_2 in Fig. 6. □

Theorem 7. *In a 2-connected network, the biconnectivity lost due to a failure of a node A_f can be restored by repositioning the neighbors of A_f that are located at the network periphery or on the boundary of a connectivity hole.*

Proof. Based on Theorem 6, the biconnectivity property will be lost only if A_f is a boundary node due to introduction of a cut vertex A_c. The presence of A_f on the boundary implies that it links two other nodes A_a and A_b on the periphery (e.g., nodes A_3 and A_4 Fig. 6) or on the boundary of an internal hole, and each of these two nodes will belong to a distinct 2-connected partition after A_f fails. Since the two partitions are connected through A_c, moving A_a to link with the partition of A_b through a node other than A_c (or relinking A_b with the partition of A_a) will introduce a second path between the two partitions through A_a (or A_b) and will thus restore the network biconnectivity.

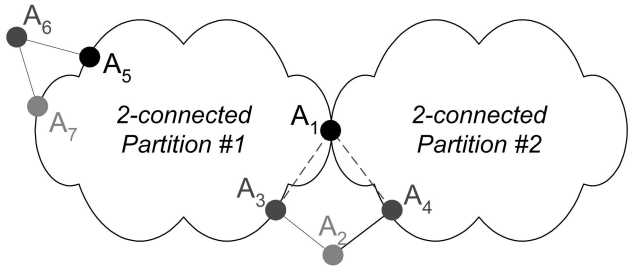

Fig. 7. Illustration of the possible cases for which a cut vertex is introduced when an actor fails.

Fig. 7 again illustrates the proof of Theorem 7. If A_2 fails, moving A_4 toward partition #1 or A_5 toward partition #2 will make the network biconnected again. The same applies when A_7 fails by moving A_6 toward partition #1 so that it can establish a link to a node other than A_5. □

Theorem 8. *Theorem 7 represents a sufficient but unnecessary condition for restoring biconnectivity.*

Proof. Obviously, for a 2-connected network in which there are no two subnetworks that are connected by only two nodes, the failure of a boundary node will not risk biconnectivity, and no recovery action is required. For example, if multiple nodes connect partitions 1 and 2 in both Figs. 6 and 7, the failure of A_2 will not affect the network biconnectivity. Therefore, the recovery process stated in Theorem 7 is sufficient but is not necessary for all cases. □

DARA-2C strives to sustain 2-connectivity in the interactor network and thus avoids a possible network partitioning caused by a sequential failure of two actors. DARA-2C is a completely distributed and localized restoration mechanism that can work in real time. The main idea, which is based on the results of the analysis above, is to recover from the failure of boundary nodes in order to avoid the introduction of cut vertices in the network. Basically, since the network is 2-connected, each actor node will have at least two neighbors (i.e., a node degree of two). When an actor A_f fails, each neighbor of A_f runs DARA-2C. DARA-2C first checks whether A_f is a boundary node. If this is not the case, no recovery procedure is needed since the network is still 2-connected, as proved in Theorem 6 above. If A_f happens to be a boundary node, it is sufficient for its neighbors that are also boundary nodes to take action (Theorem 7). DARA-2C identifies the most appropriate boundary neighbor of A_f to relocate. Two versions of the recovery procedure are explored. The first (simplified version) simply applies Theorem 7 regardless of whether a cut vertex is introduced or not. The second version (base) involves more processing in determining the impact of the failure in order to avoid unnecessary relocation of nodes if a cut vertex does not exist in the network. Note that similar to DARA-1C, cascaded node relocations would be pursued if restoring the 2-connectivity of a node causes some of its neighbors to lose their 2-connectivity property.

In order to run DARA-2C, each actor needs to know its one-hop and two-hop neighbors, as assumed in DARA-1C. This includes the IDs, locations, and node degrees of these neighbors. In addition, an actor is assumed to know whether it is a boundary actor or not by running a localized boundary detection algorithm [19]. In the version of DARA-2C in which cut-vertex detection is pursued, we employ the distributed algorithm of [18], which probabilistically determines whether a node is a cut vertex or not. We next describe in detail which nodes are to be relocated and how to calculate the final location for these nodes.

5.2 Detailed Steps of DARA-2C

DARA-2C has mainly three steps. In the first step, an actor failure is detected, and the nodes that need to get engaged in the recovery process are determined. The second step determines where to move the nodes if deemed necessary. And finally, in the third step, the movement is performed.

5.2.1 Detecting Failure and Initiating the Recovery Process

Similar to DARA-1C, actors will periodically send heartbeat messages to their neighbors to ensure that they are functional. Missing heartbeat messages repetitively can be used to detect the failure of actors. The neighbors of a failed actor Af will individually decide on the scope of the recovery. Per Theorem 7, the restoration process will be initiated only on every actor $\mathrm{Aj} \in \mathrm{Neighbors(Af,)}$ based on the following criteria: 1) Af is a boundary node, 2) Aj is a boundary node, and 3) the failure of Af introduces a cut vertex in the network. The latter condition can be dropped to limit the required processing at the expense of occasionally performing recovery when not needed. Nodes that satisfy these conditions are referred to as candidates thereafter. DARA-2C restores the 2-connectivity of a network by relocating one of these candidates.

5.2.2 Selecting Which Node to Move and Where to Move It

DARA-2C employs the results of Theorem 7 to reconnect the neighbors of $\mathrm{A_f}$ that are boundary nodes (candidates) to each other. To do so, DARA-2C sets selection criteria that uniquely identify the best choice among these candidates to move. Each candidate will refer to its TwoHopTable and find out whether it is the most qualified BC. The main criterion for picking the BC is the following:

1. *Lowest node degree.* The rationale is that a node that has the least number of neighbors will yield few cascaded relocations. Obviously, this is not always the case, and it will depend on the number of two-hop neighbors, three-hop neighbors, etc., and how many of them will need to move. We argue that establishing a breadth-first-search tree routed on each candidate would expand the scope of the recovery and would require that each node maintains a network-wide state; something DARA-2C is geared to avoid.

The decision on where the BC will reposition depends on the cardinality of the set of candidates. If there are only two candidates, the selected BC will move toward the other candidate until it becomes within its radio range. However, if the cardinality of the set of candidates is three or more, the BC will move to the position of A_f. The rationale is that replacing A_f is sure to relink all its neighbors and restores the lost connectivity. It is worth to emphasize that we assume that all actors have the same radio range. In case multiple candidates have the least node degree, the following criterion is employed to qualify the best choice. Again, this applies only if there are more than two candidates.

2. *Least distance.* In order to minimize the travel overhead, the closest candidate to A_f , among those having the least node degree, is favored for BC.

Finally, if there is still a tie, the following criterion is applied:

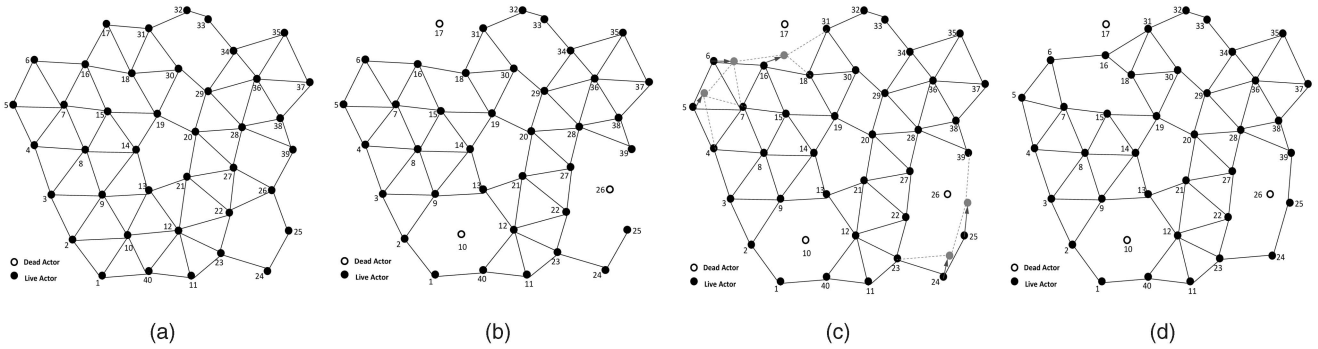

Fig. 8. (a) The original nonfaulty topology of the interactor network. (b) Articulation for the impact of the independent and nonsimultaneous failures of actors A_{10}, A_{17}, and A_{26}. (c) No recovery needed for A_{10}. Meanwhile, neighbors of A_{17} and A_{26} apply DARA-2C as shown. (d) The restored k-connected network after executing DARA-2C.

3. *Highest actor ID.* It is possible, although rare in practice, that there exist two or more candidates with identical node degrees and equidistant to A_f. In that case, the candidate with the greatest ID will be picked to break the tie. Note that this also applies for the case in which there are only two candidates, and both have the same node degree.

It is important to note that DARA-2C is executed simultaneously on all neighbors of a failed actor without any coordination among them.

5.2.3 Node Relocation

The BC actor, $\mathrm{A_{BC}}$, will notify all its neighbors that it is moving and tell them where it intends to reposition. The new position can be determined as explained above. Similar to DARA-1C, when $\mathrm{A_{BC}}$ stops, it broadcasts a "RECOVERED" message indicating the completion of the restoration process. The neighbors of $\mathrm{A_{BC}}$ will keep waiting for the "RECOVERED" message; if they receive it, they conclude that they are still connected. Otherwise, they will assume that they got disconnected and apply Theorem 7 again as if $\mathrm{A_{BC}}$ stopped functioning. This means that a node, $\mathrm{A_j} \in \mathrm{Neighbors(A_{BC})}$ will move only if it is a boundary node. In other words, the recovery process will be applied recursively to trigger the cascaded relocation of affected actors. The only difference here is that each of these boundary nodes will individually move toward $\mathrm{A_{BC}}$. Nonboundary neighbors of $\mathrm{A_{BC}}$ do no need to move (Theorem 7).

The example in Fig. 8 demonstrates the execution of DARA-2C for three different failure scenarios. The 40-node topology was initially 2-connected (Fig. 8a), before some actors fail (Fig. 8b). We stress that the failures of actors A_{10}, A_{17}, and A_{26} are independent and nonsimultaneous. Since A_{10} is not a boundary node, it does not risk the 2-connectivity of the network and, thus, no recovery is needed. On the other hand, A_{17} and A_{26} are boundary nodes, and their neighbors execute DARA-2C. The absence of A_{26} causes node A_{23} to be a cut vertex; meanwhile, the failure of A_{17} does not introduce any cut vertices in the network. For A_{26}, the 2-connectivity of the network would need to be restored and the recovery action, in terms of node movement, is similar for both the simplified and base versions of DARA-2C, meaning that investigating the presence of a cut vertex performed by the base version will be redundant in that case.

To tolerate the failure of A_{26}, DARA-2C selects A_{25} as BC since it is a neighbor of A_{26} that is a boundary node and has the least node degree compared to other candidates (only A_{39} in this case). Node A_{25} moves toward A_{39} and is then followed by A_{24}, as shown in Fig. 8c. It should be noted that A_{23} does not move since A_{24} stays in its radio range after relocation. The handling of the failure of A_{17} demonstrates the fundamental difference between the base and simplified versions of DARA-2C. No cut vertex exists in the network, and obviously, nodes A_{16} and A_{31} stay 2-connected even after the loss of A_{17}. The base version of DARA-2C will not perform any node relocation since no cut vertices are introduced as a result of the failure of A_{17}. However, the simplified version will pick A_{16} as BC and move it toward A_{31} (Fig. 8c). Although nodes A_6 and A_7 get disconnected from A_{16}, only A_6 moves since it is a boundary node while A_7 is not. Fig. 8d shows the final topology. The pseudocode of the DARA-2C algorithm is provided in Appendix B.

5.3 Algorithm Analysis

In this section, the worst case performance of DARA-2C is analyzed.

Theorem 9. *The message complexity of DARA-2C is $O((k+1)N)$, where N is the number of actors in the network, and k is the number of candidates, i.e., boundary nodes that are neighbors of the failed actor.*

Proof. DARA-2C uses two-hop neighbor information, which requires two messages per node, i.e., a total of $2 * \mathrm{N}$ messages for the entire network. In addition, every candidate in DARA-2C broadcasts only two messages: 1) to inform its children before moving and 2) to announce its successful relocation. Furthermore, a candidate needs to check whether its neighbors are cut vertices or not. This involves a breadth-first search, which spans the whole network with N messages. Assuming that the number of candidates is k, the total number of messages is thus $2 * \mathrm{N} + (\mathrm{k} * (\mathrm{N}+2))$. Moreover, actors need to know whether they are boundary nodes or not. This is an $O(N)$ process [19], making the overall message complexity to be $O(kN)$. □

Theorem 10. *The maximum distance a node travels in DARA-2C is "r," where r is the actor radio range.*

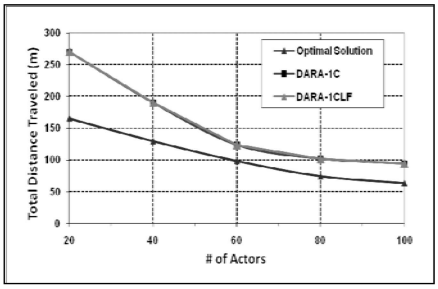

Fig. 9. Total distance traveled with varying numbers of actors (radio range = 100 m).

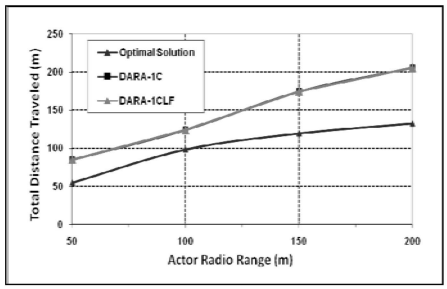

Fig. 10. Total distance traveled with varying actor radio ranges (number of actors = 60).

Proof. A BC would need to move to the position of failed actor A_f if there are more than two candidates or toward the other candidate otherwise. Since the BC had a link to A_f prior to its failure, the distance has to be less than or equal to "r." In addition, if there are only two neighbors of A_f that are boundary nodes, the distance between these two neighbors cannot be more than "$2r$" since they are two-hop neighbors. Therefore, one of them would need to move at most a distance "r" in order to be within the communication range of the other. □

Theorem 11. *The convergence time of the DARA-2C algorithm to restore 2-connectivity is $O(p + r)$, where p is the maximum time for an actor node to become a BC, and r is the actor radio range.*

Proof. Let p be the maximum time to detect a failure of an actor and become a BC. In the movement process, the total distance traveled by a BC is at most r, as proved in Theorem 10. Assuming that the relocation triggers further relocation of k descendants, each traveling a distance of r, the time to restore 2-connectivity will be proportional to $(k+1) * r$ since the relocations are performed in a serial manner. Thus, the convergence time of DARA-2C to restore 2-connectivity in the worst case is $p + (k+1) * 2r$, which is $O(p + r)$. □

6 Experimental Evaluation

The effectiveness of DARA-1C and DARA2-C are validated through simulation. This section describes the simulation environment, performance metrics, and experimental results.

6.1 Experiment Setup and Performance Metrics

In the experiments, we have created connected interactor networks consisting of varying numbers of actors (20 to 100). Actors are randomly placed in an area of 1,000 m × 600 m. The maximum transmission range for an actor node is assumed to be 100 m unless otherwise specified. Each simulation is run for 10 different network topologies, and the average measures are reported. We observed that with 90 percent confidence level, the simulation results stay within 6 percent-10 percent of the sample mean. We use the following two metrics to capture the performance:

1. *Total traveled distance.* This metric captures the total distance traveled by all actors when a failed node is replaced. It is used to determine the efficiency of DARA in terms of the energy overhead.
2. *Number of messages.* This is the total number of messages sent by all nodes in order to restore the connectivity of the network. It is another metric for assessing the energy overhead incurred by the nodes.

6.2 Performance Evaluation of DARA-1C

In the experiments, we compared DARA-1C and DARA-1CLF to the optimal cascaded motion solution that provides the least traveled distance. The optimal cascaded motion is a centralized approach that requires the knowledge of the state of the entire network. After identifying cut vertices in the generated actor topology, one of them is picked at random to be a faulty node in the network.

Movement performance. In order to assess the effectiveness of DARA-1C and DARA-1CLF in terms of the total traveled distance, we conducted experiments with varying numbers of actors. The results shown in Fig. 9 indicate that the total distance traveled by actors in DARA-1C and DARA-1CLF is exactly same. The travel distances achieved by DARA-1C get closer to the values obtained by the optimal cascaded motion when the network size grows, confirming the scalability of our approach. This can be attributed to the fact that with increasing the number of actors, the degree of connectivity in the network increases and, thus, fewer movements are required to restore connectivity. It is also interesting to note that local flooding in DARA-1CLF has not brought any advantages as the network is randomly deployed. We have repeated the same experiment with varying transmission ranges while fixing the number of actors at 60. The results shown in Fig. 10 indicate that the total travel distance grows when increasing the actor radio range. Such results may seem puzzling at the first look since the network connectivity usually improves with greater transmission ranges. However, cut vertices in this case are usually linking blocks that are far apart, and thus, the recovery will involve a travel for longer distances.

We have also conducted some experiments with special topologies (i.e., circular connected network) in order to assess the advantage of DARA-1CLF over DARA-1C in terms of total distance traveled. In such cases, DARA-1CLF performed significantly better than DARA-1C, almost matching the performance of the optimal cascading, as shown in Fig. 11. This is expected as in circular topologies, DARA-1C unnecessarily relocates all the nodes that are not indeed needed for maintaining the connectivity.

Message complexity. In order to compare the messaging overhead, we have recorded the total number of messages sent by all actors in DARA-1C and DARA-1CLF. Table 1

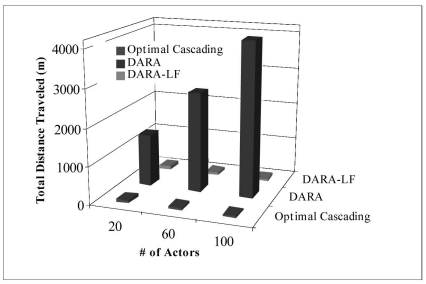

Fig. 11. Total distance traveled under ringlike topologies.

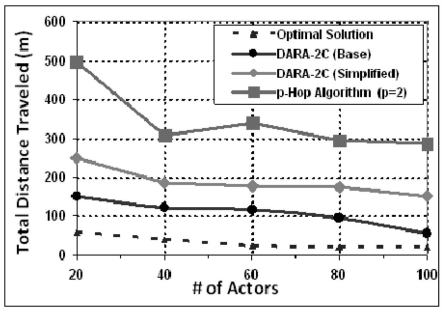

Fig. 12. Total distance traveled with varying numbers of actors (radio range = 100 m).

provides the statistics with varying numbers of actors and the actor radio range set to 100m. Such statistics confirm that DARA-1C and DARA-1CLF introduce significantly less interactor communication overhead than the optimal cascaded motion. This is more evident when the actor count is relatively large. The optimal cascading approach requires each actor to flood the network, leading to a message complexity of N^2. While it is conceivable to form a multicast tree in order to reduce the message count, forming this tree is still $O(N^2)$. On the other hand, the message complexity of DARA-1C and DARA-1CLF is linear in the number of actors, as seen in Table 1. Note that since DARA-1CLF requires local flooding, the number of messages sent in this approach is a little higher than DARA-1C. However, it brings reduction in the total distance movement for certain network topologies, as mentioned above.

6.3 Performance Evaluation of DARA-2C

For these experiments, we created 2-connected interactor networks consisting of varying numbers of actors (20 to 100) using the algorithm in [6]. We compared the base and simplified versions of DARA-2C to the optimal solution that provides the least movement distance and to the p-Hop algorithm [8], which moves some nodes in order to achieve biconnectivity of an initially 1-connected network. The symbol "p" signifies the number of hops up to which a node is aware of. Since DARA-2C utilizes two-hop information, we compare it to the two-hop version of the algorithm in [8].

Movement performance. We conducted experiments with both varying numbers of actors and varying transmission ranges. The results depicted in Figs. 12 and 13 show that both versions of DARA-2C significantly outperform the p-Hop algorithm in terms of the distance that actors collectively travel to restore the biconnectivity of the network. In many configurations, the base version delivers performance that is at least 300 percent better than the p-Hop algorithm. Even in the simplified version, actors often need a travel distance that is 60 percent-70 percent of that traveled under the p-Hop algorithm. The figures also indicate that the base version of DARA-2C delivers performance that is 25 percent-40 percent better than that of the simplified version. The network designer is sure to weigh such performance advantage against the corresponding increased cost in processing and communication.

Fig. 12 indicates that the performance of all approaches seems to scale for large networks. This is mostly due to the localized nature of the recovery process; even for the central approach, the repositioning mostly involves neighbors of the failed actor. The increased actor population makes the network dense and shortens the distance to be traveled by relocating actors. Obviously, the performance of DARA-2C surpasses that of p-Hop, as we noted earlier.

Fig. 13 shows that total distance traveled increases with increasing actor radio range while the number of actors are static (40 actors). This is expected as the distance between nodes grows when the transmission range is increased. It is also notable that the base version of DARA-2C delivers close-to-optimal performance, especially for highly connected networks, for which the actor radio range is relatively large. This can also be related to Fig. 12 when the network is highly populated, making the base version of DARA-2C perform even better.

Message complexity. Fig. 14 provides the results for the total number of messages sent during the interactor network 2-connectivity restoration process with varying numbers of actors. The curve for the optimal solution is not shown in Fig. 14 to enable displaying the other curves at reasonable resolution. Since the optimal solution

TABLE 1
Total Number of Messages Sent with Varying Numbers of Actors

# of Actors	Optimal Cascading	DARA-1C	DARA-1CLF
20	400	87	98
40	1600	168	200
60	3600	249	299
80	6400	305	401
100	10000	406	499

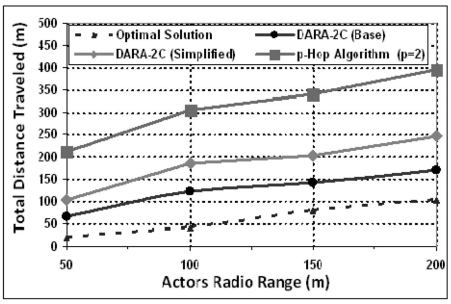

Fig. 13. Total distance traveled with varying actor radio ranges (with 40 actors deployed).

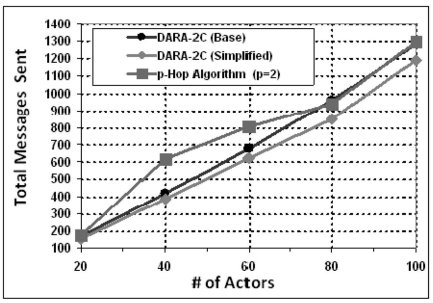

Fig. 14. Total number of messages sent with varying numbers of actors (radio range = 100 m).

requires the knowledge of the whole topology, as the network size grows, the number of messages sent increases significantly (message count = N^2). This is not the case in DARA-2C or the p-Hop algorithm as they are local approaches requiring only two-hop neighbor information for a node. As an example to highlight the difference in terms of the number of messages, the optimal solution will involve 1,600 messages for the 40-actor networks and 100,000 for the 100-actor one. DARA-2C still outperforms the p-Hop algorithm. The base version of DARA-2C uses about 8 percent more messages than the simplified version. Thanks to Theorem 7, the extra messaging is limited to checking for cut vertices only among boundary nodes.

7 Conclusion

WSANs are usually deployed in harsh environments where actor failure/damage is possible. Such failures can lead to a situation in which the interactor network connectivity is weakened and the network gets partitioned. In order to recover from these failures, we have presented DARA, a Distributed Actor Recovery Algorithm for WSANs. Two variants of the algorithm, namely, DARA-1C and DARA-2C, are developed to address 1 and 2-connectivity requirements. The main goal of DARA-1C is to restore the network connectivity by exploiting the mobility of actors. The idea is to pick one of the neighbors of a failed node in order to replace it. Since this may still disconnect the children of the moved actor, DARA-1C strives to pick the neighboring actor that will trigger the least number of cascaded movements. In this way, the total distance traveled by actors is minimized.

DARA-2C strives to restore the 2-connectivity of a 2-connected interactor network. The effect of an actor failure is analyzed and shown to depend on whether the actor is a boundary node or not. It has been proven that only the failure of a boundary node may risk the biconnectivity of the network. A sufficient condition is derived to categorize the recovery process. Two versions of DARA-2C are proposed for which a decrease in the algorithm complexity is traded off for a potential increase in the total distance that the actors collectively travel. Both DARA-1C and DARA-2C are completely distributed and require only the knowledge of two-hop neighbors, which reduces the communication overhead significantly.

We have validated DARA-1C and DARA-2C through simulation; comparing the performance to that of a centralized approach that considers the state of the entire network in providing a node relocation plan with the least total travel distance. The simulation results have confirmed the correctness and effectiveness of both algorithms; with close-to-optimal performance, DARA-2C is shown to outperform other applicable approaches in the literature. In the future, we plan to extend our approach to consider other metrics such as coverage and sensor-to-actor delay in determining the scope of the recovery and in selecting candidates for movement.

```
DARA-1C(ID, missing_node)
1   repeat
2    BestCandidateID ← FindBestCandidate(TwoHopTable)
3    if ID = BestCandidateID then
4      MoveToLocation(ID,loc(missing_node))
5      exit
6    else
7      Remove BestCandidateID from TwoHopTable
8      pause( 2 * time to travel for distance r)
9    end if
10  until Msg ('RECOVERED') is received

MoveToLocation(ID, newloc)
11  if Neighbor(ID) ≠ NULL then
12   for each j ∈ Neighbor(ID)
13    if IsDependentChild(j)== True then
14      Unicast(j, Msg('MOVING', Siblings(ID))
      // All dependent children will then invoke
      // ChildMovOptimizer(j, ID, Siblings(ID))
15    end if
16   end for
17  end if
18  Move(newloc)
19  Broadcast(Msg('RECOVERED'))
20  Update TwoHopTable

ChildMovOptimizer(ID, parent, parentSiblings)
21  for each k ∈ TwoHopTable // meaning ID.TwoHopTable
22   if k ∈ parentSiblings then
23    exit
24   end if
25  end for
26  if ID has not previously moved then
27    DARA(ID, parent)
28  end if
```

Fig. 15. Pseudocode for the distributed actor recovery algorithm.

Appendix A
Detailed Pseudocode for DARA-1C

Fig. 15 shows the complete pseudocode for DARA-1C. When an actor detects the failure of a neighbor A_f that is considered as a cut vertex, DARA-1C will be activated on this actor. This actor looks in its TwoHopTable for the BC to replace A_f, as shown in line 2. Specifically, the FindBestCandidate() function picks the actor node with the least node degree among the neighbors of A_f. In case of ties, the distance to A_f and the node ID are used to make the selection. The picked actor as a result of FindBestCandidate() moves to the location of A_f (line 4). If the actor is not the BC, it deletes the BC in its TwoHopTable since the BC will soon become a neighbor and waits until it hears a RECOVERED message from the BC actor (lines 7 and 8). This process is

repeated again if no recovery message is received, this time without considering the current BC, which has not responded as expected.

Moving the BC to the position of A_f may trigger further relocations among the dependent children of the BC, i.e., members of $\texttt{Dependents}(\texttt{BC}, A_f)$, in order to maintain the connectivity. Note that we make a distinction between the children (neighbors) of the BC that are one-hop and two-hop neighbors of A_f. A child A_c of the BC that used to be a neighbor of A_f (i.e., $A_c \in \texttt{connected\_siblings}(\texttt{BC})$) does not need to worry about its link to the BC since it eventually will be connected to the BC when the BC reaches the position of A_f. On the other hand, dependent children are only two-hop neighbors of A_f and need to move in order to remain reachable from the BC actor. Therefore, the BC checks whether it has any neighbors in line 11. If it has no neighbors, it moves to the location of A_f and broadcasts a "RECOVERED" message (lines 18 and 19). Otherwise, the BC sends a "MOVING" message to each of its dependent children. The "MOVING" message includes the list of other neighbors of A_f, i.e., list of siblings (lines 12-16). Each child receiving this message runs ChildMovOptimizer() in order to decide on whether to move or not.

The ChildMovOptimizer() function basically checks whether the actor node is already connected to one of the neighbors of A_f of the BC by referring its TwoHopTable (lines 21 and 22). If this is the case, the actor is indirectly connected to the BC, and no further action for restoring the connectivity is required. Therefore, the actor node terminates the algorithm (line 23). Otherwise and if it has not moved before, it starts a new coordinated recovery with its siblings, i.e., other neighbors of the BC actor, in order to replace the BC (line 27). The latter is simply performed by running DARA-1C again. Thus, the recovery continues recursively until all children are connected.

APPENDIX B
DETAILED PSEUDOCODE FOR DARA-2C

Fig. 16 shows the pseudocode for DARA-2C. When an actor detects a failure in its one-hop neighborhood, DARA-2C will be activated on this actor. This actor will check whether the lost neighbor was a boundary node or not (line 1). If a nonboundary node fails, no action is required, and the algorithm exits. When confirming the failure of a boundary neighbor, the actor will continue only if it is also a boundary node and a cut vertex exists in the network. The call to the `IsThereCutVertex` function in line 3 is underlined to indicate the difference between the base and simplified versions of DARA-2C. Basically, this call may be removed at the expense of occasional overreaction to a node failure. The performance of both versions of DARA-2C is studied in Section 6.

The actor, which is now considered a candidate, then applies the selection criteria stated in the previous section in order to identify the BC among the boundary neighbors of the failed node (line 4). If the actor turns out to be the BC, it calls `"CalculateNewLoc"` in line 6 to determine its new location, which is the position of the failed node if there are more than two candidates or a point that is "r" away from the second candidate otherwise. The actor then moves to the new position and exits (lines 7 and 8). Otherwise, the actor removes the BC from its TwoHopTable (line 10) since it is supposed to move and becomes a neighbor rather than two-hop neighbor. The node then waits in line 11 to hear the confirmation from the BC that the recovery is complete. If for some unexpected reason, the BC does not move as expected, the actor repeats the procedure again (lines 2-14) to identify a new BC. When the recovery is complete, the actor updates the entries in its neighbor lists and exits DARA-2C.

```
DARA-2C(ID, MissingNode)
1  if IsBoundaryNode(MissingNode) == True then
2   repeat
3    if IsBoundaryNode(ID) == True &&
       IsThereCutVertex(ID, MissingNode) == True then
       // Among neighbors of failed actor that are
       // boundary nodes AND a cut-vertex resulted
4     BestCandidateID ← FindBestCandidate(TwoHopTable)
5     if ID == BestCandidateID then
6       NewLoc ← CalculateNewLoc(ID, MissingNode)
7       MoveToLocation(ID,NewLoc)
8       Exit
9     else
10      Remove BestCandidateID from TwoHopTable
11      Pause enough time for Best Candidate to move
12    end if
13   end if
14  until Msg ('RECOVERED') is received
15  Add BestCandidateID to Neighbors
16  Add Neighbors(BestCandidateID) to TwoHopTable
17 end if

MoveToLocation(ID, NewLoc)
18 if Neighbor(ID) ≠ NULL then
19   for each j ∈ Neighbor(ID)
20     if IsDependentChild(j) == True then
21       Unicast(j,Msg('MOVING'), NewLoc)
         // All dependent children invoke ChildMovOptimizer(j, ID)
22     end if
23   end for
24 end if
25 Move(NewLoc)
26 Broadcast(Msg('RECOVERED'))
27 Update TwoHopTable

IsThereCutVertex(ID, F)
28 if Visited(ID) then
29   return (FALSE)
30   // request come back indicating no cut vertices
31 else
32   if ID is a cut vertex in the absence of F then
33     return (TRUE)
34   else
35     Visited(ID)= TRUE
36     for each j ∈ Neighbor(ID)
37           if IsBoundaryNode(j) then
38       Unicast(j, F)
         // j checks with next neighbor on boundary
         // and invokes IsThereCutVertex(j, F)
39       Wait for reply(j) //node waits meanwhile
40       if reply(j) then
41         return(TRUE)
42       end if
43           end if
44    end for
45   end if
46 end if
47 return (FALSE)

ChildMovOptimizer(ID, ParentNewLoc)
48 if ID has not moved && IsBoundaryNode(ID) == True then
49   NewLoc ← CalculateNewLoc(ParentNewLoc)
50   MoveToLocation(ID,NewLoc)
51 end if
```

Fig. 16. Pseudocode for the DARA-2C algorithm.

Fig. 16 also shows the pseudocode for three additional functions called from DARA-2C. The `"MoveToLocation"` function is similar to its counterpart in DARA-1C and

initiates the cascaded relocation of neighbors of the BC. On the other hand, the function "ChildMovOptimizer" differs from the version in DARA-1C. This function basically limits the scope of a child involvement in the cascaded relocation to boundary nodes since the goal is to sustain 2-connectivity and given the results of Theorem 7. The function "IsThereCutVertex" checks the presence of a cut vertex in the network because of the failed actor. The function is simply a recursive implementation of a breadth-first search for which the request is spread from candidates, i.e., the neighbors of the failed actor that are boundary nodes, to other actors in the network (lines 36-44). Each of these nodes will apply a cut-vertex detection algorithm such as [18] and return a confirmation of the result (lines 33-35). Per Theorem 6, it is sufficient to check for cut vertices among boundary nodes.

ACKNOWLEDGMENTS

This work was supported by the US National Science Foundation under Contract 0000002270.

REFERENCES

[1] I.F. Akyildiz and I.H. Kasimoglu, "Wireless Sensor and Actor Networks: Research Challenges," *Elsevier Ad Hoc Network J.*, vol. 2, pp. 351-367, 2004.
[2] A. Youssef, A. Agrawala, and M. Younis, "Accurate Anchor-Free Localization in Wireless Sensor Networks," *Proc. First IEEE Workshop Information Assurance in Wireless Sensor Networks (WSNIA '05)*, Apr. 2005.
[3] K. Akkaya and M. Younis, "COLA: A Coverage and Latency Aware Actor Placement for Wireless Sensor and Actor Networks," *Proc. IEEE 64th Vehicular Technical Conf. (VTC-Fall '06)*, Sept. 2006.
[4] T. Melodia et al., "A Distributed Coordination Framework for Wireless Sensor and Actor Networks," *Proc. ACM MobiHoc '05*, May 2005.
[5] R.W.N. Pazzi and A. Boukerche, "Mobile Data Collector Strategy for Delay-Sensitive Applications over Wireless Sensor Networks," *Computer Comm.*, vol. 31, no. 5, pp. 1028-1039, 2008.
[6] K. Akkaya and M. Younis, "Coverage-Aware and Connectivity-Constrained Actor Positioning in Wireless Sensor and Actor Networks," *Proc. 26th IEEE Int'l Performance Computing and Comm. Conf. (IPCCC '07)*, Apr. 2007.
[7] M. Elhadef, A. Boukerche, and H. Elkadiki, "A Distributed Fault Identification Protocol for Wireless and Mobile Ad Hoc Networks," *J. Parallel and Distributed Computing*, vol. 68, no. 3, pp. 321-335, 2008.
[8] K. Ozaki et al., "A Fault-Tolerant Model for Wireless Sensor-Actor System," *Proc. Second IEEE Int'l Workshop Heterogeneous Wireless Sensor Networks (HWISE '06)*, Apr. 2006.
[9] G. Wang et al., "Sensor Relocation in Mobile Sensor Networks," *Proc. IEEE INFOCOM '05*, Mar. 2005.
[10] T. Melodia, D. Pompili, and I.F. Akyildiz, "A Communication Architecture for Mobile Wireless Sensor and Actor Networks," *Proc. Third IEEE Conf. Sensor, Mesh and Ad Hoc Comm. and Networks (SECON '06)*, Sept. 2006.
[11] M. Younis and K. Akkaya, "Strategies and Techniques for Node Placement in Wireless Sensor Networks: A Survey," *J. Ad-Hoc Networks*, vol. 6, no. 4, pp. 621-655, 2008.
[12] P. Basu and J. Redi, "Movement Control Algorithms for Realization of Fault-Tolerant Ad Hoc Robot Networks," *IEEE Networks*, vol. 18, no. 4, pp. 36-44, Aug. 2004.
[13] S. Das et al., "Localized Movement Control for Fault Tolerance of Mobile Robot Networks," *Proc. First IFIP Int'l Conf. Wireless Sensor and Actor Networks (WSAN '07)*, Sept. 2007.
[14] D. Li, J. Cao, M. Liu, and Y. Zheng, "K-Connected Target Coverage Problem in Wireless Sensor Networks," *Proc. First Int'l Conf. Combinatorial Optimization and Applications (COCOA '07)*, Aug. 2007.
[15] L. Zhang, X. Wang, and W. Dou, "A K-Connected Energy-Saving Topology Control Algorithm for Wireless Sensor Networks," *Proc. Sixth Int'l Workshop Distributed Computing (IWDC '04)*, Dec. 2004.
[16] K. Akkaya, M. Younis, and M. Bangad, "Sink Repositioning for Enhanced Performance in Wireless Sensor Networks," *Computer Networks*, vol. 49, pp. 434-512, 2005.
[17] X. Liu, L. Xiao, A. Kreling, and Y. Liu, "Optimizing Overlay Topology by Reducing Cut Vertices," *Proc. ACM Int'l Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV '06)*, May 2006.
[18] K. Akkaya et al., "Distributed Recovery of Actor Failures in Wireless Sensor and Actor Networks," *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC '08)*, Mar. 2008.
[19] C. Zhang, Y. Zhang, and Y. Fang, "Detecting Coverage Boundary Nodes in Wireless Sensor Networks," *Proc. IEEE Int'l Conf. Networking, Sensing and Control (ICNSC '06)*, Apr. 2006.
[20] R. Diestel, *Graph Theory*. Springer, 2005.
[21] G. Destino and G. Thadeu Freitas de Abreu, "Network Boundary Recognition via Graph-Theory," *Proc. Fifth IEEE Workshop Positioning, Navigation and Comm. (WPNC '08)*, pp. 271-275, Mar. 2008.
[22] S. Funke, "Topological Hole Detection in Wireless Sensor Networks and Its Applications," *Proc. Third Intl Workshop Foundations of Mobile Computing (DIAL-M-POMC '05)*, Sept. 2005.

Ameer Ahmed Abbasi received the MS degree in information systems from the University of Karachi, Pakistan, in 2000. He is currently an instructor in the Computer Science Department, Al-Hussan Institute of Management and Computer Science, Dammam, Saudi Arabia. In addition, he is a part-time graduate student in the Computer Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. His research interests include mobile computing and ad hoc and wireless sensor networks. He has published several papers in refereed international conference proceedings and journals.

Mohamed Younis received the BS degree in computer science and the MS degree in engineering mathematics from Alexandria University, Egypt, in 1987 and 1992, respectively, and the PhD degree in computer science from the New Jersey Institute of Technology in 1996. He is currently an associate professor in the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County (UMBC), Baltimore. His technical interests include network architectures and protocols, wireless sensor networks, embedded systems, fault-tolerant computing, and distributed real-time systems. He has five granted and two pending patents. He served on multiple technical committees and published more than 100 technical papers in refereed conference proceedings and journals. He is a senior member of the IEEE.

Kemal Akkay received the BS degree in computer science from Bilkent University, Ankara, Turkey, in 1997, the MS degree in computer science from the Middle East Technical University (METU), Ankara, Turkey, in 1999, and the PhD degree in computer science from University of Maryland Baltimore County, Baltimore, in 2005. Currently, he is an assistant professor in the Department of Computer Science, Southern Illinois University, Carbondale. His research interests include self-deployment and organization in wireless sensor and actor networks, security, and quality-of-service provisioning in wireless ad hoc, multimedia, and mesh networks. He is a member of the IEEE.