A Hybrid Approach to Actor-Actor Connectivity Restoration in Wireless Sensor and Actor Networks

Azadeh Zamanifar, Mohsen Sharifi and Omid Kashefi
Computer Engineering Department
Iran University of Science and Technology
Tehran, Iran
az_zamanifar@comp.iust.ac.ir, {msharifi, kashefi}@iust.ac.ir

Abstract—Wireless sensor and actor networks (WSANs) consist of powerful actors and resource constraint sensors, which are linked together in wireless networks. In some applications, actors must communicate with each other to make appropriate decisions and perform coordinated actions. Actor-actor network connectivity is vital to networks in such applications. Since WSAN applications are mostly deployed in harsh environments, actor nodes may fail and partition their network. This paper proposes a comparatively more efficient distributed approach to restore actor-actor connectivity upon the failure of any actor by identifying critical actors as well as the connectivity dominating set of the network. This hybrid method helps in detecting critical nodes and candidate replacement actors more precisely while minimizing the total number of required messages for network restoration. Handling the failure of the actors is done in a proactive manner. The proposed approach minimizes both the total actors’ movements and restoration time of network. When a failed actor is a critical node, actors in its neighborhood are relocated in a coordinated way to reconnect the actor network. The superiority of the proposed approach is validated through simulative experiments.

Keywords—Wireless Sensor and Actor Network; Network Restoration; Actor Connectivity; Cut Vertex; Connectivity Dominating Set (CDS)

I. INTRODUCTION

In recent years, wireless sensor and actor networks (WSANs) have been emerged as a new type of ad hoc networks that can be deployed as fully automated systems [1]. With the idea of eliminating human intervention from wireless sensor networks (WSNs), WSANs have received growing attention from the research community in the past few years. There have been numerous application areas for WSAN including monitoring environment for unusually high-level of radiation, conducting urban research and rescue (USAR), detecting and controlling pollution in coastal areas, and performing in-situ oceanic studies of bird/fish migration and weather phenomena [2].

In most of these applications, actors should coordinate and send messages to each other in order to take appropriate decisions and fulfill their pre-specified goals. Therefore, actor-actor connectivity is a key challenging issue in WSAN applications.

Actor's failure may partition the network, which may cause disruptive application result. Therefore, detecting and recovering from actor failure are two significant issues in this context. Recovery process must be preferably done autonomously and in a distributed manner in order to be efficient to cope with specified functional requirements.

In this paper, we present a distributed algorithm to detect and recover from actor failures. The aim is to reconnect the network with minimum restoration time, total movements of actors and number of messages. Most of the actor’s power would be dissipated in communication and movement. So decreasing the total number of messages and movements is one of the issues in WSAN. It is obvious that the longer the restoration time, the longer the network remains disconnected; therefore, minimizing the restoration time is the other concern of actor recovery in WSAN.

Critical actors (cut vertices) are those that the network will become partitioned without them. Critical actors in the network could be determined using the localized algorithm proposed by Stojmenovic [3]. Connectivity dominating set (CDS) of the network is also determined by using the algorithm proposed in [4]. This algorithm detects the nodes whose absence does not lead to any partitioning of the network (dominatee) and by applying the algorithm proposed in [3] this process is done precisely. By combining these two methods, firstly the critical nodes can be determined, and secondly, the redundant actors of the network are detected. This hybrid method restores the network with minimum total movements, restoration time and number of messages. If the actor is critical and there is any dominatee node (v) in its neighbor, it sends a message to v to notify that it must replace the failed actor in case of failure. If there is not any such actor, the cut vertex sends a request message to all its neighbors and asks them to send the maximum allowable movements that they can make in order not to be disconnected from their neighbors. In other words, each neighbor calculates the maximum distance that it can move towards the failed actor without being disconnected.
disconnected from its neighbors and sends a message to notify the cut vertex.

After receiving all messages from its neighbors, the cut vertex calculates whether the neighbors can reconnect the network—by movement if the cut vertex fails or not. If the network cannot be reconnected, a message is sent to the nearest neighbor of the cut vertex to replace the cut vertex in case of failure. Every actor sends a heartbeat message to its 1-hop neighbor; thus, the failure of actor could be detected by its neighbors and the network would be restored in an appropriate time. The novelty of our algorithm is that unlike previous works—wherein a failed actor is always replaced by other actors in a cascaded movement, neighbors collaborate with each other and restore the network with minimum latency. The other advantageous point to note is that by detecting both the CDS and critical nodes, the restoration process is done more efficiently. It is shown that the proposed approach is more efficient in both total movements of actors and restoration time compared to other works [2], [5].

The rest of the paper is organized as follows: Section 2 describes some related work. Section 3 presents our proposed technique. The empirical results are presented in Section 4, and Section 5 concludes the paper.

II. RELATED WORK

The fault tolerance issue in WSAN has been studied in few works in different contexts [2], [5], [6]. A fault tolerant model was first introduced in [6] wherein fault-tolerance is achieved by means of redundancy. In other words, sensors send their sensed data to more than one actor and each actor receives the sensed information from multiple sensors in the event area. In this paper we focus on connectivity when an actor fails and we do not consider fault-tolerant communication between sensors and actors.

Another related work is the one first proposed by Akkaya et al. in 2007 [5]. The main idea was to detect the failure of an actor and replace the failed actor in a cascaded manner. Actors get the status of each other by sending heartbeat messages periodically. The alternative candidate is selected based on the distance to the dead actor and the number of neighbors namely node degree. The drawback of this method is that in choosing a candidate to replace a failed actor, it does not take into account whether the selected actor is critical or not. This blind movement may cause repartitioning of the network again, which is inefficient and boosts the restoration time and may cause lots of movement. In other words, the criticality of nodes had not been considered in their work.

Akkaya et al. enhanced their previous method in April 2008 [2]. They use the CDS of the whole network in order to detect the cut-vertex node while the CDS method is not accurate for cut vertex detection. To detect a cut vertex node precisely, they perform a depth-first search on every CDS resulted cut vertices to confirm that the node is really a cut vertex or not. After detecting these nodes, each node picks the appropriate neighbor to handle its failure in the case of failure in future. The objective is to choose a neighbor that may not partition the network again. The main drawback of their approach is that a delegated actor exactly replaces the failed neighbor that such movement may be inefficient and unnecessary. As we will show, the network could be connected without exact replacement and with less movement in some cases. We use a hybrid method to detect the cut vertex node, so our proposed method detects the cut vertex actors more exactly and restores the network more intelligently and more efficiently with the same number of messages compared to [2]. We propose a method to relocate the neighbors of the dead actor in a way that the network is reconnected by fewer total number of actors' movements and less restoration time compared to the pervious works.

Just one work [7] maintains the connectivity between sensors and actors. So it is irrelevant to actor-actor recovery issue in this paper.

Few works consider connectivity [8] at initial deployment of the network, while our objective is to provide connectivity after the failure of actors. In [8] the aim is to relocate the actor so that the maximum coverage is achieved while maintaining connectivity among actors. This is done by applying the concept of repelling forces among neighboring actors. Our method differs from them as we assume that the network may partition and our goal is to restore the network considering connectivity.

Connectivity management has been also studied in mobile sensor networks. In [9] inter-actor connectivity is maintained by providing 2-connectivity among each actor and the goal is to keep the 2-connectivity in the case that actors fail. The main idea is to move non-critical nodes while keeping critical nodes static unless they become non-critical. It assumes that the network is already 1-connected. However, providing 2-connectivity in the network may not be always possible. In [10] a 1-connected ad-hoc network is transformed to 2-connected network by moving certain nodes.

The idea of cascading movement has been deployed in many researches before [11], [12]. We combine this method with block movement.

III. OUR APPROACH

This section describes our proposed algorithm for restoring actor-actor connectivity in WSANs with minimum latency. Section 3.1, defines the network assumptions, Section 3.2, states the problem, and Section 3.3 presents the details of the proposed algorithm.

A. Assumptions

There are a set of resource-constraint sensors and resource rich actors that are placed randomly throughout an application area. Since Actors are expensive, their number is limited in a network. The sensor network is dense and each
actor and sensor knows its position. After deployment, each actor finds its neighbors and a connected network is formed.

B. Problem Statement

Let us assume that there are $n$ actors and $m$ sensors where $n<<m$. Actors form a connected graph. Each actor knows its location. Actors may fail due to the intrinsic nature of the environment. If we consider each actor as a node, and draw an edge between the actors that are in their transmission range, the failed actor may be a cut vertex and partition the graph. The goal is to restore the network so that the graph is connected again. A part of an actor network is shown in Fig. 1 where $L$ is the cut vertex as its failure partitions the network. Therefore, we have to detect critical nodes and actor failures, and determine a set of movements to restore the network based on the type of the failed actor (i.e. critical or non-critical). The aim is to minimize the restoration time, number of messages and total amount of movements.

C. The Proposed Algorithm

The algorithm consists of three main steps or parts: 1) network setup, 2) monitoring the neighbors’ status and detecting failure of actors, and 3) restoring the network after actor failure.

1) Network Setup.

After the network is deployed, the following steps are taken:
   a) Determining the critical node and the CDS
   b) Determining restoration policy for each critical actor

a) Determining the Critical Node and CDS

A number of methods for determining critical node(s) have been proposed which are categorized as centralized, distributed and localized [13], [14]. Global algorithms are most accurate but more expensive and degrade the performance because they entail a huge amount of message transmissions. Localized algorithms are more favorable in this regard.

We use the localized method proposed by Stojmenovic [3]. It is not as accurate as global algorithms, but it detects the critical nodes quickly. The main concept is that each node sends a hello message to $k$-hop neighbors. If the k-hop neighbors construct a connected graph, then the node is not critical; otherwise, the node is critical. We use 2-hop neighbor’s information to detect critical nodes and use positional information of the neighbors to predict critical nodes more accurately.

Fig. 1 shows an example. By using 2-hop neighbors information, two sub-graphs $\{O, P\}$ and $\{M, L\}$ are determined for node $N$ which are two disjoint sub-graphs; thus node $N$ is critical. With the aid of 2-hop neighbors’ information, node $G$ is identified as non-critical. Using 2-hop information we also can calculate CDS of the network (dominator nodes) as proposed in [4]. So, the nodes are categorized as follows:

1. Dominatee nodes that are detected as the non-critical node, by the algorithm in [3]. The movement or failure of these nodes dose not partitions the network. –like A in Fig. 1
2. Dominator nodes that are detected as non-critical node, by algorithm in [3]. The movement of these nodes may partition the network only if the cut vertex node in their neighbors fails and they relocate. –like C in Fig. 1
3. Dominator nodes that are detected as the critical node. The failure of these nodes causes network partitioning. –like K in Fig. 1

Stojmenovic [3] state that the number of actors that are falsely declared as critical depends on the number of neighbors. Since the number of actors in the network is limited, with an average number of 4 to 11 neighbors, the average percentage of actors that are falsely detected as critical would be below 15%.

b) Determining Restoration Policy

We use a proactive policy in order to restore the network in case of critical actor failure. After determining critical nodes in the network, each critical node requests its neighbors to send the maximum distance that they can move towards the critical node without being disconnected from their other neighbors.

If the cut vertex node fails, the network is partitioned into two or more sub-networks. By applying the proposed algorithm, we could determine which partition each neighbor of the cut vertex node belongs to. If there is a dominatee node (v) in the neighbor of the cut vertex, a message is sent to v to notify that it must replace the cut vertex in case of failure. If there is not any such actor, the...
maximum allowable movement of each neighbor of the cut vertex is calculated. It must be noted that some neighbors of the cut vertex may belong to one partition. Therefore, from each partition, the nearest neighbors to cut vertex are selected and a request message is sent to each of them to send their maximum movement. Each neighbor of the cut vertex that receives the message calculates the location of the furthest actor in its own neighbor. Therefore, it would determine how much each neighbor of the cut vertex could move towards the cut vertex without violating the connectivity of the network.

For example, Fig. 2 shows how the maximum movement of each actor is calculated. In order to calculate the maximum allowable movement of A11, in each partition of its neighbors \{A13, A14, A12\} the nearest actor to A11 is selected \{A13, A12\}. Then, the farthest actor among them – A12 is selected. Each actor neighboring the cut vertex calculates \( t – \) the maximum allowable distance:

\[
  t = (r - d') \times \cos \theta
\]

where \( r - d' \) is the distance that A11 could move in order not to be disconnected from its farthest neighbor A12.

At the critical node, after receiving all the messages from neighbors, the cut vertex calculates whether the network could be reconnected if neighboring actors move by their maxMovement. The algorithm for checking whether the network is still connected or not in the absence of cut vertex (A) is depicted in Fig. 2.

To check the network connectivity —as it is shown in Fig. 2, at the first, the neighbor with minimum required movement is found and it is checked whether node A could see other neighbors of the cut vertex. Considering the maximum allowable movements of actors, the neighbors that could see node A, check whether they could see other neighbors that are not in the transmission range of A (lines 14 to 22). After that, it is checked whether there is any actor in the neighborhood of the cut vertex that is not connected to other neighbors by the means of its maximum movement. The procedure returns TRUE if there is not any such actor and returns FALSE otherwise (lines 23 to 26).

The newLocation of an actor is the location between the curlocation and curlocation + maxMovement that actor could be connected to other neighbors in that point. In other words, newLocation is the point that the actor would be in transmission range of one or more actors. Therefore, three conditions are checked in each cut vertex:

1. If there is a dominatee neighbor \( u \) in the neighborhood of the cut vertex, it is delegated to replace the cut vertex in case of failure and the cut vertex sends a message to notify \( u \).
2. If there is not any such actor and the network could be reconnected by moving the neighbors, the location of the neighbors (newLocation) is sent to them by the cut vertex.
3. If even maximum movements (maxMovement) of all actors in the neighborhood could not reconnect the network, the replacement is done through cascaded movement. In other words, the nearest actor is found and the cut vertex sends a message to notify that it must replace the cut vertex in case of failure and it must arrange a cascade movement of its neighbors too, to keep the network connectivity.

2) Detecting the Failure of an Actor.
Each actor periodically sends its ID and location to its neighbor. If the actor does not receive any messages for a specified amount of time from a particular actor, it is considered as having failed and the recovery process starts.

3) Restoring the Failed Actor.
In the case of cut vertex failure, based on the three conditions described at the bottom of the page, three states may occur:

1. A delegated dominatee actor \( u \), detects the failure of the cut vertex and replaces the cut vertex.
2. The neighbors would detect the cut vertex failure and move to their new location.
3. The neighbor of the cut vertex actor would replace the failed actor in a cascaded movement (cascade movement causes network partitioning). If the corresponding neighbor was a non-critical actor, the neighbor actor finds it nearest neighbor and sends it a message (Relocation Message) to be relocated too. If the neighbor is a critical actor, it finds a nearest neighbor in each partition and sends them the Relocation Message. Each actor that receives the Relocation Message from its neighbors runs the same algorithm If its movement to new location causes network partitioning, it sends the Relocation Message to its neighbor(s) too, and if not, it moves to its new location and sends an acknowledgment to the actor which has received a Relocation Message from it. Actors do not move before they receive an acknowledgment message from one of their neighbors. Upon receiving an acknowledgment, each actor moves to its known new location and sends an acknowledgment message to the actor that had sent it a Relocation Message.

Our approach helps to restore the network earlier with fewer total movements of actors compared to other related works. For example, if A3 fails in Fig. 4, according to [2], A7 is responsible for finding a nearest dominatee actor A4 to replace the failed actor. The network is then restored by cascaded movement – A4 is replaced by A7 and A7 is replaced by A3. However, according to our approach, the network is restored earlier with fewer total movements of actors.
Network Connectivity Checking Algorithm

1: IsNetworkConnectedByMaxMovement(node X)
2: {
3:   neighborsList = GetSortedNeighborsOnMaxMovement(X)
4:   A = neighborsList[0]
5:   Add A to ConnectedNodesList
6:   A.newLocation = A.curLocation + A.MaxMovement
7:   foreach node B in neighborsList
8:     if B is in A.transmissionRange
9:       Add B to ConnectedNodesList
10:      B.newLocation = MaxCoverPoint(B, A)
11:   Else
12:     Add B to NotConnectedNodesList
13:   Endloop
14:   foreach node C in ConnectedNodesList
15:     foreach node D in NotConnectedNodesList
16:       if D is in C.transmissionRange
17:         Remove D from NotConnectedNodesList
18:         Add D to ConnectedNodesList
19:         D.newLocation = MaxCoverPoint(D, C)
20:     Endif
21:   Endloop
22:   if (NonConnectedNodesList is Empty)
23:     return true
24:   Else
25:     return false
26:   }
27:   MaxCoverPoint(B, A) = P, where Distance(P, B.curLocation) <= B.MaxMovement and
28:   P is in A.TransmissionRange

Figure 2. The pseudo code for checking the network connectivity

The worst case scenario occurs when all the neighbors of a failed actor are critical as in the example depicted in Fig. 3. Each neighbor (A2, A4) of the failed actor A3 moves so that the network is reconnected again. Before A4 moves to its new location A4' it finds that its movement causes network partitioning. Therefore, it sends a relocation message to A5. The movement of A5 does not violate the actor connectivity anymore; therefore, it moves to its new location and sends an acknowledgment to A4. Upon receiving the acknowledgment message, A4 moves to its new location A4'. Therefore, the network needs less total movements of actors in order to be reconnected compared to the case that actor is being replaced in a cascaded movement [2]. In the worst case that actors are exactly located in R (transmission range of actor) distance from each other, the total movements of actors is equal to the total movements of actors reported in [2] but the amount of movements of each actor decreases, so it helps actors to have longer lifetime.

Figure 3. A worst case topology with all critical nodes in the neighborhood

IV. EVALUATION

In order to evaluate our approach, we created a connected WSAN whose actors were initially connected to each other
and were spread randomly. Simulation was carried out in Ptolemy II [15]. In each simulation, one critical node whose neighbors were all critical was selected as a failed actor. Five different topologies were simulated at each run. The total number of messages and the total movements of actors were considered and the average was calculated.

We considered messages for determining the cut vertices, detecting and handling the failure of actors and restoring the network. Each run was compared to PADRA+ [2].

Fig. 5, shows that as the number of actors increases, the total movements of actors does not increase significantly. This is because, firstly, the number of cut vertices decreases when the number of actors increases, and secondly, as the number of cut vertices decreases, the number of actors that are falsely detected as a cut vertex by the algorithm in [3] decreases. Thus, as it is shown in Fig. 5, our proposed approach performs better specially when the number of actors is high and consequently the number of cut vertices is low. As it is shown in Fig. 6, the number of sent messages grows as the number of actors increase. This is because the determination of cut vertices requires more messages to be sent to a high number of actors. The number of messages decreased compared to PRADA+ because our approach uses a depth-first search to detect critical nodes precisely. The results are depicted in Fig. 6.

![Figure 5. Total movements of actors vs. different number of actors](image)

![Figure 6. Total number of messages vs. different number of actors](image)

### V. CONCLUSION

This paper presented an algorithm to detect and repair the failure of actors in WSANs. The proposed restoration process aimed to reconnect the partitioned networks. Unlike the previous works like [2] that replace a failed actor with the aid of cascaded movements, the neighbors of a failed actor communicate with each other in order to reach an agreement and restore the network. By combining the methods for detecting critical nodes [3] and connectivity dominating set (CDS) [4] our proposed algorithm decreased the total number of messages, individual movements of each actor as well as the total movements of all actors in the network compared to [5]. It also prevented extra movements especially when all the neighbors of a failed actor were critical.

### REFERENCES


