AOM: AN EFFICIENT APPROACH TO RESTORE ACTOR-ACTOR CONNECTIVITY IN WIRELESS SENSOR AND ACTOR NETWORKS

Azadeh Zamanifar[1], Omid Kashefi[2] and Mohsen Sharifi[3]

[123]Computer Engineering Department, Iran University of Science and Technology, Tehran, Iran
az_zamanifar@comp.iust.ac.ir, kashefi@iust.ac.ir, msharifi@iust.ac.ir

ABSTRACT

Wireless sensor and actor networks (WSANs) consist of powerful actors and resource constraint sensors that are linked together in wireless networks. They mostly rely on actors to make proper decisions and perform desired coordination to achieve the goals of the entire network. They are usually deployed in critical applications and actor-actor network connectivity is thus vital to their effective utilization. Since WSAN applications are mostly deployed in harsh environments, actor nodes may fail and so partition their network. We propose a comparatively more efficient distributed approach, nicknamed AOM, to restore actor-actor connectivity upon the failure of any actor. We identify critical actors by combining the result of determining critical actors using the Stojmenovich's method with the connectivity dominating set (CDS) of the network. This hybrid method of detecting critical actors helps in detecting critical nodes and candidate replacement actors more precisely while minimizing the total number of required messages for network restoration. The failure handling of actors is done in a proactive manner. Our proposed method minimizes both the restoration time of network and the total number of actor movements. 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 our approach compared to other works is shown by simulative experiments measuring two important parameters to WSANS, namely, the total number of transmitted messages and the total number of actor movements during actor-actor network reconnection process.

KEYWORDS

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

1. 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 causing 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 and decreasing the total number of transmitted messages and movements of actors are two most important issues in WSAN. Thus, we focus on the number of transmitted messages and the number of movements of actors as metrics for evaluating network reconnection solutions. 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. So 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 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 in the absence of cut vertex 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, a failed actor is not always replaced by other actors in a cascaded movement. Instead, 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 paper is organized as follows. Section 2 presents some related work. Section 3 describes our proposed approach. The empirical results are presented in Section 4, and Section 5 concludes the paper.

2. RELATED WORK

The fault tolerance issue in WSANs 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. 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. In order to detect the cut vertex precisely, a depth-first search is performed for each dominator. The main drawback of their approach is that a delegated actor exactly replaces a failed neighbor, which may be inefficient and unnecessary; as we will show that in most cases the network could be connected with less total number of movements. 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.

Only one work [7] is done to maintain the connectivity between sensors and actors and 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.

3. 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.

3.1. 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.

3.2. 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. If the actor is not a cut vertex but it creates a large hole in the network, it is desired to restore the network to cover the hole. A part of an actor network is shown in Figure 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.

3.3. Our Proposed Method

Our method, nicknamed AOM, 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.

3.3.1. Network Setup

After the network is deployed, the following steps are taken: 1) Determining the critical node and the CDS, and 2) Determining restoration policy for each critical actor.

3.3.1.1. Determining Critical Nodes 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. Figure 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 none cut-vertex node by the algorithm in [3] like *A* in Figure 1. The movement or failure of these nodes does not partition the network.
2) Dominator nodes that are detected as none cut vertex node by algorithm in [3], like *C* in Figure 1. The movement of these nodes may partition the network only if the cut vertex node in their neighbors fails and they relocate.
3) Dominator nodes that are detected as the cut vertex node like *K* in Figure 1. The failure of these nodes causes network partitioning.

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%.

3.3.1.2. Restoration Policy Determination

We use a proactive policy in order to restore the network in case of critical actor failure. After critical nodes in the network are determined, 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.

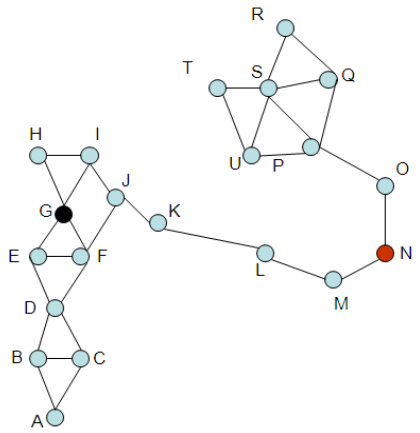

Figure 1. A network with node *N* as the cut vertex by 2-hop information

If the cut vertex node fails, the network is partitioned into two or more sub networks. By applying the proposed method, we can determine to which partition each neighbor of the cut vertex node belongs.

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 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, Figure 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 as: $t = (r - d') \times \cos(\theta)$ (1) 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, it checks whether there is a neighbor *v,* whose *maxMovement* covers the location of the cut vertex. This condition occurs if *v* does not have any other neighbor except the cut vertex, so it is a good candidate for replacement. If this criterion is held, *v* is responsible for handling the failure of the actor and it would replace the cut vertex in case of failure. If this criterion is not held, the cut vertex calculates whether the network could be reconnected if neighboring actors move by their *maxMovement*. Figure 2 shows the pseudo code for determining whether the network is still connected in the absence of cut vertex.

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 the transmission range of one or more actors.

As it is shown in Figure 2, the neighbor with minimum required movement is found at first and it is checked whether *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).

Connectivity Checking Algorithm

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

Figure 2. The pseudo code for determining the network connectivity

At lines 23 to 26, it is checked whether there is any actor in the neighbor 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.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 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.

3.3.2. 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 period of time from a particular actor, it is considered as having failed and the recovery process starts.

3.3.3. Restoring the Failed Actor

In the case of cut vertex failure, one of the following three states may hold based on the three conditions described at Section 3.3.1.2:

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. This movement causes network partitioning that based on its state of being critical node or not, we act differently. If it was a noncritical actor, it finds its nearest neighbor and sends it a message to be relocated too (*Relocation Message*). 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 method helps to restore the network earlier with fewer total movements of actors compared to other related work. For example, if *A3* fails in Figure 3, 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, meaning that *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.

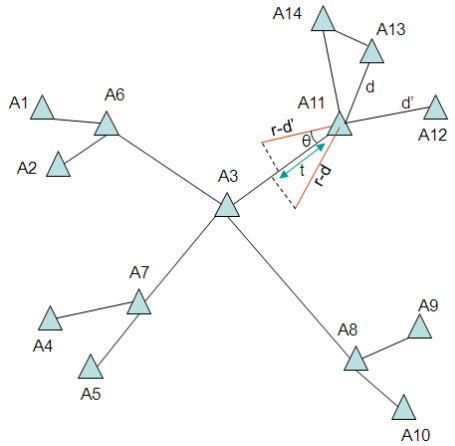

Figure 3. Maximum Allowable movement of each actor

The worst-case scenario occurs when all the neighbors of a failed actor are critical as in the example depicted in Figure 4. When all actors are critical, it is more probable that more actors need to be relocated in order to reconnect the network.

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'*.

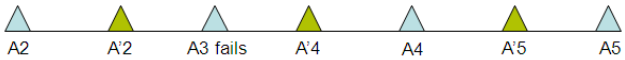

Figure 4. A worst-case topology with all critical nodes in the neighborhood

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.

4. EVALUATION

As we mentioned in Section 1, we consider the total number of transmitted messages and the total number of movements of actors as two major metrics for evaluating WSANs' connectivity approaches. Table 1 compares the detailed number of transmitted messages for each nodes type, separated by different phase of connectivity restoration process. As it is shown in Table 1, AOM is superior on transmitting fewer number of messages in most cases in contrast to other approaches like PADRA and PADRA+.

Table 1. Comparative number of transmitted messages

<table>
<tr><th colspan="2" rowspan="3">Node Type</th><th colspan="8">Message Type</th></tr>
<tr><th colspan="2">Detecting Cut Vertexes</th><th colspan="2">DFS</th><th colspan="2">Neighbors</th><th colspan="2">Relocation</th></tr>
<tr><th>AOM</th><th>PADRA(+)</th><th>AOM</th><th>PADRA+</th><th>AOM</th><th>PADRA(+)</th><th>AOM</th><th>PADRA(+)</th></tr>
<tr><td colspan="2">Dominatee</td><td>3</td><td>3</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td colspan="2">Dominator:
has at least one dominatee in neighbor</td><td>3</td><td>3</td><td>-</td><td>-</td><td>1</td><td>1</td><td>-</td><td>-</td></tr>
<tr><td rowspan="3">Dominator:
does not have any dominatee in neighbor</td><td>non cut vertex</td><td>3</td><td rowspan="3">3</td><td>-</td><td rowspan="3">t + n</td><td>-</td><td rowspan="3">1</td><td>-</td><td rowspan="3">2*k</td></tr>
<tr><td>cut vertex (1*)</td><td>3</td><td>-</td><td>2 + n</td><td>-</td></tr>
<tr><td>cut vertex (2*)</td><td>3</td><td>-</td><td>2 + n</td><td>2*k</td></tr>
</table>

(1*) is the condition where the neighbor could reconnect the network as it is showed in Figure 2.
(2*) is the condition where cascaded movement is done
n is the number of neighbors, *k* is the number of hops to the closest dominates and *t* is the number of nodes within the sub tree headed by the closest neighbor.
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].

Figure 5 compares the total movements of actors in PADRA, PADRA+ [2], and Optimal Cascading with our approaches (AOM). In optimal cascading, each actor knows the topology of the whole network which needs that each node broadcast its status to all the other nods. Although in optimal cascading the shortest path to the nearest non critical actor could be determined. Optimal cascading approach has an excessively high overhead due to the large message transmission and forces substantial expends to WSAN that makes it incapable of being applicable in real experiments. It is observed that as the number of actors increase, the total movements of actors does not increase significantly. This is because, firstly, the number of cut vertices decreases, and secondly, as the number of cut vertices decreases, the number of actors that are falsely detected as a cut vertex by the Stojmenovich's method [3] decreases. Thus, as it is illustrated, our approach performs more efficient especially when the number of actors is high and consequently the number of cut vertices is low, compared to both PADRA and PADRA+ approaches because detecting the critical actor is done more precisely by our approach than other approaches. Figure 6 compares the number of transmitted messages in PADRA, PADRA+ and our approach (AOM). We eliminate the optimal cascading method from this comparison because of its far more actors' movements compared with other approaches that decreases the visual accuracy of the comparison chart depicted in Figure 6. Therefore, we separately compared the optimal cascading approach with our approach in Figure 7. As it is observed in Figure 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. Figure 6 shows the superiority of our approach on transmitting less number of messages through network reconnection process, compared to other works.

Figure 7 also shows that our approach is significantly more efficient in number of transmitted message for network reconnection compared to optimal cascading method.

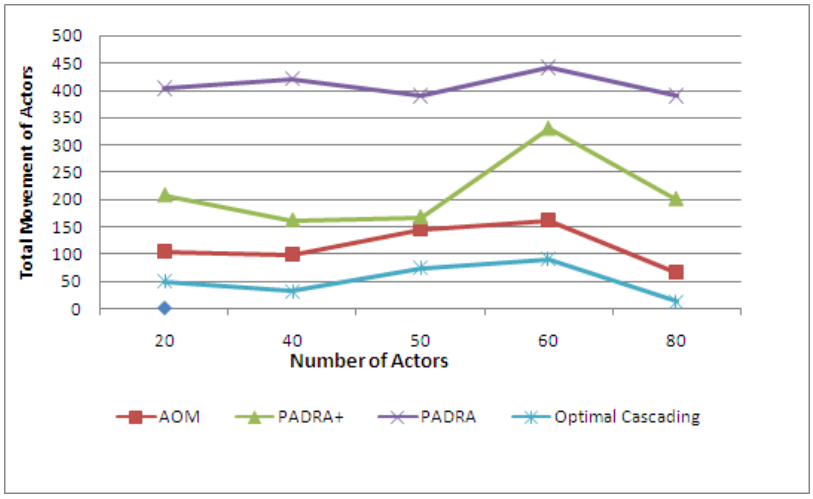

Figure 5. Comparative total movements of actors

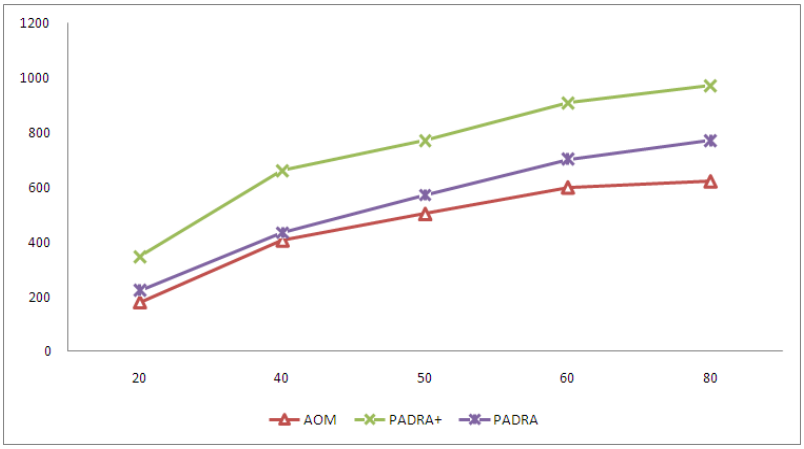

Figure 6. Comparative total number of messages

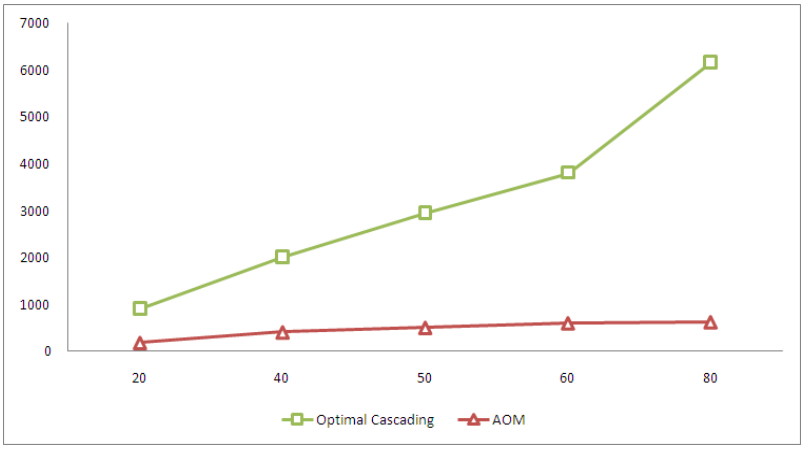

Figure 7. Comparative total number of messages

5. CONCLUSION AND FUTURE WORKS

This paper presented an efficient method to detect and repair the failure of actors in WSANs. The proposed restoration process aimed to reconnect 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 by the Stojmenovich method [3] and the CDS method [4], our method 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. Another advantage of our approach was in its significant reduction in restoration time of the network.

Future expected works include the consideration for coverage as a factor for restoring the actor network after actor failure, sensor-actor delay, obstacles that could affect the actor movements in real networks and therefore the restoration policy, and three-dimensional actor locations instead of two-dimensional locations assumed in our current work. Given the high rate of actor

failures in harsh environments, finding proper solutions for simultaneous failures of critical actors in a neighborhood merits further studies.

REFERENCES

[1] Akyildiz, F. and Kasimoglu, I. H. (2004) Wireless sensor and actor networks: research challenges. *Elsevier Ad Hoc Network Journal*. 2 351-367.

[2] Akkaya, K., Thimmapuram, A., Senel, F. and Uludag, S. (2008) Distributed recovery of actor failures in wireless sensor and actor networks. Proceedings of the IEEE Wireless Communications and Networking Conference.

[3] Jorgic, M., Stojmenovic, I., Hauspie, M. and Simplot-ryl, D. (2004) Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks. Proceedings of the 3rd Annual IFIP Mediterranean Ad Hoc Networking Workshop pp. 360-371.

[4] Dai, F. and Wu, J. (2004) An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks. *IEEE Transaction on Parallel and Distributed Systems*. 15(10) 908-920.

[5] Abbasi, A. A., Akkaya, K. and Younis, M. (2007) A Distributed Connectivity Restoration Algorithm in Wireless Sensor and Actor Networks. Proceedings of the 32nd IEEE Conference on Local Computer Networks IEEE Computer Society.

[6] Ozaki, K., Watanabe, K., Itaya, S., Hayashibara, N., Enokido, T. and Takizawa, M. (2006) A Fault-Tolerant Model ofWireless Sensor-Actor Network. Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing IEEE Computer Society.

[7] Wu, J., Yang, S. and Cardei, M. (2008) On Maintaining Sensor-Actor Connectivity in Wireless Sensor and Actor Networks. Proceedings of the 27th Conference on Computer Communications, IEEE INFOCOM pp. 286-290.

[8] Akkaya, K. and Younis, M. (2007) C2AP: Coverage-aware and Connectivity-constrained Actor Positioning in Wireless Sensor and Actor Networks. Proceedings of the IEEE Internationa Performance, Computing, and Communications Conference pp. 281-288.

[9] Das, S., Liu, H., Kamath, A., Nayak, A. and Stojmenović, I. (2007) Ttile., Wireless Sensor and Actor Networks.

[10] Basu, P., Redi, J., Technol, B. and Cambridge, M. (2004) Movement control algorithms for realization of fault-tolerant ad hoc robot networks. *IEEE network*. 18(4) 36-44.

[11] Li, X., Santoro, N. and Stojmenovic, I. (2007) Ttile., Ubiquitous Intelligence and Computing.

[12] Wang, G., Cao, G., Porta, T. L. and Zhang, W. (2005) Sensor relocation in mobile sensor networks. Proceeding of the 24th Annual IEEE Conference on Computer Communications.

[13] Goyal, D. and J. Caffery, J. (2002) Partitioning Avoidance in Mobile Ad Hoc Networks Using Network Survivability Concepts. Proceedings of the Seventh International Symposium on Computers and Communications IEEE Computer Society.

[14] Seada, K. and Helmy, A. (2004) Efficient geocasting with perfect delivery in wireless networks. Proceedings of the IEEE Wireless Communications and Networking Conference.

[15] Baldwin, P., Kohli, S., Lee, E. A., Liu, X. and Zhao, Y. (2004) Modeling of sensor nets in Ptolemy II. Proceedings of the 3rd international symposium on Information processing in sensor networks ACM.

Azadeh Zamanifar received her M.Sc in Computer Engineering (Software) in 2008 from Iran University of Science and Technology (IUST), Tehran, Iran. She is currently a member of distributed systems research group in the Computer Engineering Department of IUST and also a researcher in the Computer Systems Department, Niroo Research Institue, Tehran, Iran.

Omid Kashefi is currently a postgraduate student of Information Technolgy (Software Design and Development) in the Computer Engineering Department of IUST. He received his B.Sc. in Computer Engineering (Software) from IUST in 2006. He is a member of distributed systems research group, as well as a senior project manager and designer in Noor Computer Research Center, Tehran, Iran.

Mohsen Sharifi is an Associate Professor of Software Engineering currently chairing the Computer Engineering Department of IUST. He directs a distributed systems research group and laboratory in the department. His main interest is in the development of kernel level and middleware level distributed system software for use in mission critical applications requiring dependable high performance computing capabilities. He received his B.Sc., M.Sc. and Ph.D. in Computer Science from the University of Manchester in the United Kingdom.