This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the WCNC 2008 proceedings.

Distributed Recovery of Actor Failures in Wireless Sensor and Actor Networks

Kemal Akkaya, Aravind Thimmapuram, and Fatih Senel
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL 62901
kemal@cs.siu.edu | {taravind, fsenel}@siu.edu

Suleyman Uludag
Department of Computer Science
University of Michigan - Flint
Flint, MI 48502
uludag@umich.edu

Abstract—**Wireless sensor and actor networks (WSANs) additionally employ actor nodes within the wireless sensor network (WSN) which can process the sensed data and perform certain actions based on this collected data. In most applications, inter-actor coordination is required to provide the best response. This suggests that the employed actors should form and maintain a connected inter-actor network at all times. However, WSANs often operate unattended in harsh environments where actors can easily fail or get damaged. Such failures can partition the inter-actor network and thus eventually make the network useless. In order to handle such failures, we present a connected dominating set (CDS) based partition detection and recovery algorithm. The idea is to identify whether the failure of a node causes partitioning or not in advance. If a partitioning is to occur, the algorithm designates one of the neighboring nodes to initiate the connectivity restoration process. This process involves repositioning of a set of actors in order to restore the connectivity. The overall goal in this restoration process is to localize the scope of the recovery and minimize the movement overhead imposed on the involved actors. The effectiveness of the approach is validated through simulation experiments.**

I. INTRODUCTION

In recent years, wireless sensor and actor networks (WSANs) have started to receive growing attention due to their potential in many real-life applications [1]. Such networks include miniaturized low-cost sensing nodes that are responsible for probing their surroundings and reporting their measurements to some actor nodes over wireless communication links. Actors process the sensed data, make decisions, and then perform the appropriate actions. The actor's response mainly depends on its capabilities and the application. For instance, actors can be used in lifting debris to search for survivors, extinguishing fires, chasing an intruder, etc. Examples of WSAN applications include facilitating/conducting urban search and rescue (USAR), detecting and countering pollution in coastal areas, performing in-situ oceanic studies of bird/fish migration and weather phenomena, detection and deterring of terrorist threats to ships in ports, destruction of mines in land and under water, and monitoring the environment for unusually high-level of radiation.

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 instance, in Urban Search and Rescue (USAR) applications, in case of events such as fires, earthquakes, disasters, etc., the survivors may be in desperate need of oxygen gas, water or even some sort of medicine within a short period. Therefore, the actors should collaboratively decide the best possible solution in terms of the number of actors to employ, their traveling time, and distance to the survivor. This process requires that all the actors should be able to communicate in order to become aware of the current states of others. To enable such communications, actors should form and maintain a connected inter-actor network at all times.

In such a USAR application, failure of an actor may cause the loss of multiple inter-actor communication links, partition the network if alternate paths among the affected actors are not available, and stop the actuation capabilities of the actor. Such a scenario will not only hinder the actors' collaboration but also may potentially risk the life of some survivors and thus has very negative consequences on the USAR application. Therefore, WSANs should be able to tolerate the failure of an actor and recover from it in a distributed, timely and energy efficient manner: First, the recovery should be distributed since WSANs usually operate autonomously and unattended. Secondly, rapid recovery is desirable in order to maintain the WSAN responsiveness to detected events. And finally, the energy overhead of the recovery process should be minimized to extend the lifetime of the WSAN.

In this paper, we present a distributed PArtition Detection and Recovery Algorithm (PADRA) which can determine possible partitioning in advance and restore the connectivity in case of such failures with minimized node movement and message overhead. Since partitioning is caused by the failure of a node which is serving as a cut-vertex (i.e., a gateway for multiple nodes), each actor node determines whether it is a cut-vertex or not in advance in a distributed manner. This is achieved by utilizing the connected dominating set (CDS) of the whole network. Once such cut-vertex nodes are determined, each node designates the appropriate neighbor to handle its failure when such a contingency arises in the future. The designated neighbor picks a node whose absence does not lead to any partitioning of the network (i.e., a dominatee) to replace the failed node when the node fails. The replacement is done through a cascaded movement where all the nodes from the dominatee to the cut-vertex are involved. The goal is to share the movement load so that the energy of the selected dominatees will not drain quickly as a result of long mechanical motion.

1525-3511/08/$25.00 ©2008 IEEE

This paper is organized as follows. Related work is discussed in Section II. Section III covers the assumptions, problem definition and details of our approach. The experimental validation of the approach is discussed in Section IV. Section V concludes the paper with a summary of planned future work.

II. Related Work

In WSANs, fault tolerance has only been studied in a few works in different contexts. For instance, a fault-tolerant model for WSANs is discussed in [2]. The idea is to designate multiple actors to which a particular sensor reports events and similarly for an actor there will be multiple sensors reporting on the same event. Thus, an event is guaranteed to be reported to an actor even if a sensor fails and a sensor will find an actor to report to, even if an actor becomes inaccessible. Our fault-tolerant model in this paper is in the context of maintaining network connectivity and availability for performing actions not sensor-actor communication as studied in [2]. The closest work to ours is reported in [3]. This work presents DARA which also strives to restore the connectivity when a cut-vertex node fails. The idea is similar to ours in the sense that it explores cascading movement when replacing the failed node. However, there are many differences from our work. First, DARA does not provide a mechanism to detect cut-vertices. It is assumed that this information is available at the node which may require the knowledge of the whole topology. In that case, it may not make sense to try a restoration approach which is local. Our approach on the other hand determines the cut-vertices in-advance through the underlying CDS. Second, the selection of the node to replace the failed node is done based on the neighbors' node degree and distance to the failed node which may require excessive replacements until a leaf node is found. Our approach looks for a dominatee rather than a leaf node to replace the failed cut-vertex.

As the actors employed in WSANs can also be robots, some of the research in mobile robot networks can be applied to WSANs. One example of providing fault-tolerance in such networks has been studied in [4]. In this work, the initial network is assumed to be 2-connected, meaning that there are at least 2 node independent paths between every pair of nodes. The goal is to sustain such 2-connectivity even under link or node failure. The approach is based on moving a subset of the robots to restore the 2-connectivity that has been lost due to the failure of a robot. While the idea of movement of robots is similar to ours, it is performed based on block movements and requires a centralized approach. The same problem is also studied in [5] which rather presented a distributed approach. The approach tries to restore the 2-connectivity without requiring both the knowledge of the network and block movements. Unlike [4] and [5], our paper focuses on providing 1-connectivity.

The idea of repositioning has also been applied to mobile sensors in order to counter holes in sensing coverage [6]. The idea is simply to identify some spare sensors from different parts of the network that can be repositioned in the vicinity of the holes. Since moving a node for a relatively long distance can drain significant amount of energy, a cascaded movement is proposed if there are 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 [6].

III. CDS-based Partition Detection and Recovery

A. System Model and Assumptions

WSANs involve two types of nodes: sensors and actors. Sensors are inexpensive and highly energy-constrained with 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. The communication range of an actor refers to the maximum Euclidean distance that its radio can reach and is assumed to be larger than that of sensors. Sensors and actors are randomly deployed in an area of interest. Upon deployment, actors are assumed to discover each other and form a connected network. We would like to also note our approach can also be applied to mobile robot networks where no sensors exist. In that case robots will be actors.

B. Problem Definition

The impact of actor's failure on the network connectivity may be limited or dramatic. When the lost actor is a leaf node, no other actors will be affected. Meanwhile, when the failed actor serves as a gateway node in the network, a serious damage to the network connectivity will be inflicted. Such nodes are referred to as cut-vertices. Basically, with the loss of a cut-vertex, the network gets partitioned into disjoint sub-networks. Fig. 1a shows an example inter-actor network. In that example, the loss of a leaf actor such as A_3 will not impact the connectivity of the network. However, A_4 is a cut-vertex whose failure will result in two disjoint blocks of actors as seen in Fig. 1b.

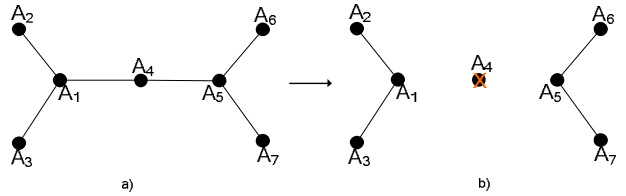

Figure 1. a) An example connected inter-actor network b) A4 fails and the network is partitioned into two disjoint sub-networks.

Our problem can be defined as follows: "N actors that know their locations are randomly deployed in an area of interest and they form a connected inter-actor network G. In the case of a failure of a particular node A_i, our goal is twofold: 1) Determine locally whether such failure causes any partitioning within the network G and; 2) If there is a partitioning, determine (again locally) the set of movements to restore the connectivity with minimum travel distance. With travel distance, we mean two different metrics: 1) Maximum Movement distance of all Individual actors (MMI) 2) Total Movement distance of all the Actors (TMA).

In order to reach our first goal, we present a CDS based approach which informs a particular node *u* in advance whether a partitioning will occur or not in the case of its failure. As a result, each actor node will know upfront how its failure will be handled. Obviously, if *u*'s failure does not cause any partitioning, no handling will be needed.

However, if failure of *u* leads to partitioning, it designates a set of nodes to be repositioned so that the connectivity is restored in the network. Determining such nodes and how they will be repositioned are our second goal. The algorithm should minimize both MMI and TMA as mentioned above. The first metric is important to provide fairness among the actors. Specifically, if a particular node moves very long distances, this node may deplete all of its energy and can die quicker than the other actors in the network. On the other hand, minimizing only the movement distance of a particular actor will not be efficient if this causes the majority of the actors in the sub-network to move. We would like to make sure that fewer actors are moving and each individual movement is minimized in order to extend the lifetime of the WSAN. Next, we describe our approach for cut-vertex determination.

C. Cut-vertex Determination

Determining whether a node is a cut-vertex or not can be easily done by using depth first search trees (DFS) [7]. However, this approach requires flooding the whole network and can be costly in terms of messaging. Thus, we investigate a distributed approach for such a purpose. Our approach will be based on connected dominating set (CDS) of the inter-actor network.

As every node can reach the nodes in a CDS, the connectivity of the network can be maintained as long as CDS is connected. We used Dai's distributed algorithm [8] in order to determine the CDS of a given network *G*. Note that this algorithm requires only local information. A node needs to know only its two-hop neighbors which can be done by transmitting two messages. As a result of running Dai's algorithm, each node will know whether it is a *dominator* (i.e., element of CDS) or a *dominatee* (i.e., can reach to a dominator within one-hop). When this information is available, determination of a cut-vertex will be handled as follows:

1) If an actor node *u* is a dominatee, it can not be a cut-vertex since its absence will not violate the connectivity of the network.

2) If an actor node *u* is a dominator, then there is a high probability that *u* will be a cut-vertex. In that case, there can be two cases: a) In the first case, *u* may have at least one dominatee, *v*, as its neighbor. If *v* does not have any neighbors, *u* will declare itself as a cut-vertex. b) In the second case, all the neighbors of *u* will be either dominators or dominatees which have some neighbors. For such circumstances, determination of the cut-vertex is challenging since the dominators/dominatees can be connected via some other nodes within the network. Thus, failure of *u* may not cause any partitioning. In order to decide whether *u* is a cut-vertex or not, one of the neighbors of *u* should start a local depth-first search (DFS) to look for all the remaining neighbors of *u*. If all of them can be reached, this indicates that there are alternative paths which can be used during the failure of *u* to maintain the connectivity of the network. Therefore, u will not be a cut-vertex. Otherwise, it will declare itself as a cut-vertex. These cases are depicted in Fig. 2.

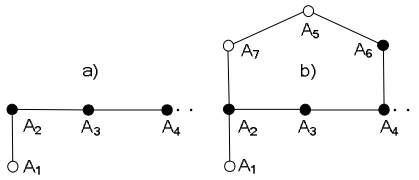

Figure 2. In a), actor A_1 is a dominatee and cannot be a cut vertex. A_2 is a dominator and has a dominatee A_1 which is not connected. Thus, A_2 is a cut-vertex. A3 is also a cut vertex in a) but will not be a cut-vertex in b).

It is important to note that for the cases of 2b, we may have an increased message overhead depending on the topology of the network. Especially, for the applications where message transmission is a concern (due to security reasons, network size etc.), the local DFS can be partially or completely eliminated.

Specifically, for complete elimination, any dominator which falls into category 2b will be assumed as a cut-vertex without performing any local DFS. For partial elimination, DFS will only be performed for the dominators which fall into category 2b but do not have any dominatees. In both cases, some nodes can be falsely identified as cut-vertices even though they are not. Later, in the experiments, we will assess the ratio of such false alarms with different network configurations and evaluate the impact on the travel distance. We will also assess the message overhead of DFS partial-usage/non-usage. Once the cut vertices are determined, the next step is to describe the failure handling approach when such cut-vertex nodes fail.

D. Handling a Cut-vertex Failure

Since the failure of a cut-vertex node partitions the network, the challenge here is to identify an actor within the region and replace it with the failed node to fix the connectivity of the inter-actor network with minimum movement and messaging cost. In this subsection, we describe when to identify such a node, which node to pick for replacement and how to move that node so that the mentioned goals are achieved.

In-advance message exchanging: Handling the failure of a cut-vertex node needs to be done in advance of the failure since the neighbors of the failed node will not be able communicate after the failure. As our goal is to fix the connectivity of the inter-actor network, for each cut-vertex, a dominatee node within the network should be found in order to replace it in case of a failure. Therefore, right after the network is deployed and the CDS is run to determine the cut-vertices, each cut-vertex node notifies its appropriate neighbor which is either a dominatee or a dominator that will determine the closest dominatee and thus will take care of its failure. In this way, the network recovery time is minimized as each node in the network will know how to react a failure in advance.

Determination of the closest dominatee: In order to minimize the movement distance, the closest dominatee in terms of

distance to the failed cut-vertex actor should be determined. We propose a greedy approach for determining the closest dominatee for a cut-vertex node.

The basic idea is as follows: If there is a dominatee among the neighbors of the cut-vertex, it will be designated as the node to replace the cut-vertex upon failure since it will be the closest dominatee to the failed node. Otherwise, the cut-vertex node designates its closest neighbor to handle its failure. In case of such a failure, the closest neighbor will apply the same idea in order to find the closest dominatee to itself. That is, it picks its closest neighbor and this continues until a dominatee is hit. For example in Fig. 3a, for node A_1, the closest neighbor will be A_2 if $|A_1A_2|$ is the closest distance among all the other links of A_1. Once A_1 determines its closest neighbor, it sends a message to A_2 and delegates it as the node to handle its failure when it fails. In case of failure, A_2 will pick A_5 and A_5 will find A_8 as the closest dominatee to A_2 assuming that $|A_2A_5|<|A_2A_3|$ and $|A_5A_8|<|A_5A_4|$. Note that with this approach only the nodes along the path to the dominatee transmit and receive messages which reduce the message traffic significantly. Once the closest dominatee is known, the next step is to relocate it for replacing the failed cut-vertex node.

Relocation of the closest dominatee: Moving the closest dominatee directly to the location of the failed cut-vertex will definitely restore the connectivity. However, since the movement distance can be very large, MMI will be unacceptably higher. In other words, the approach does not provide fairness and causes the moving actor die rather quickly as compared to other actors in the network. A possible solution to the unfairness problem here could be to move the partition as a block towards the failed actor. While all the actors will move the same distance with this approach, it will significantly increase TMA since all the nodes in that partition will be moving. Therefore, we propose a hybrid solution, which will combine the advantages of both approaches. The idea is to use cascaded movements from the closest dominatee to the failed node in order to maintain a maximum of r units of movement (i.e. MMI) for the individual actors where r is the actors' radio range and at the same time minimize the TMA by decreasing the number of moving actors. The approach can then be formulated as an optimization problem as follows:

$$Min \quad \sum_{\forall i \in A} M_i \text{ subject to } \quad \forall i \in A : M_i < r$$

where A denotes the set of actors to be moved and M_i denotes the total movement of a particular actor i. Specifically the approach will work as follows: In case of a failure, the closest dominatee will start a cascaded movement towards the location of the failed actor. That is, A_2 will replace A_1, A_5 will replace A_2 and finally A_8 will replace A_5 as shown in Fig. 3b. The idea is to share the load of movement overhead among the actors on the path to the failed cut-vertex node in order to extend the lifetime of each actor and thus the whole WSAN.

Note that this approach may not work when the network contains a cycle of dominators. We now describe how we address this problem.

Handling Cycles: Since the existence of a cycle may cause a series of replacements which will never be able to pick a dominatee node, a mechanism should be defined to stop the replacements of the nodes. Therefore, we introduce an extra confirmation message ACK to be received before a node u can start to move. The idea is as follows: In order to replace itself, node u should pick a node v which has not moved before. If node v has moved before, it will send a negative acknowledgement back to u and will not move anymore. Node u will understand that this indicates a cycle and thus a new neighbor other than v should be picked. If there are no other neighbors, then u will move once and no further movements will be performed so that the cycle can be broken.

An example is given in Fig. 3. In Fig. 3a, if $|A_3A_4| < |A_3A_7|$, $|A_3A_4| < |A_4A_6|$ and $|A_2A_5| < |A_5A_8|$ then A_3, A_4 and A_5 will end up replacing each other forever. Therefore, whenever a node moves, its state is changed to 'MOVED' (i.e., marking it as red) and a movement is performed only if the replacing node is not red. For instance, when its time for A_5 to replace A_4, it sends a message to A_3 and requests confirmation for moving (Fig. 3c). However, A_3 is already red and cannot move. Thus, it sends a negative ACK. Upon reception of this message A_5 understands that it sits on a cycle and thus looks for its second closest neighbor (A_8 in this case). Finally, A_8 replaces A_5 and A_5 can replace A_4 as seen in Fig. 3d.

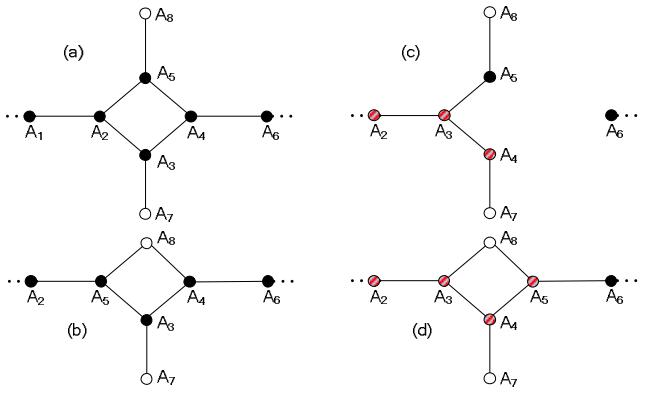

Figure 3. a) Initial network **b)** If A_2 fails, it is replaced with A_8 in a cascaded way. **c)** Cycle detection **d)** Cycle elimination.

Detailed Pseudo Code: Briefly our algorithm shown below runs as follows: If the node i is a dominatee, no action for recovery will be needed (lines 1-2) and this is already known by itself and its neighbors. However, if the node i is a dominator (line 3), we check whether it has a dominatee which does not have any other neighbor (line 4). In that case, the dominator node i will be declared as a cut-vertex since the dominatee will be disconnected when i fails (lines 5-8). Otherwise, it will be checked through DFS to determine whether it is a cut-vertex or not (lines 10-15) which is optional. If a node is a cut-vertex, it determines either its closest dominatee, if any, or its closest neighbor which will handle its failure and sends it a message (lines 15-21). When a node i fails or moves, our failure recovery procedure kicks in as described before. We omit it due to space restrictions.

E. Algorithm Analysis

Travel Distance: When the TMA is considered, PADRA does not provide the optimal distance given that the distributed

```
Procedure NodeDesignation(i)
 1 Cut-vertex(i) ←false
 2 if isDominatee(i) = true then
 2   return //No action is needed for recovery
 3 else if isDominator(i) = true then
   //Check if i has a dominatee among its neighbors
 4   if ∃j|(j∈ N(i)&isDominatee(j) = true) then
 5     for each j do
 6       if |N(j)| = 0 then
 7         Cut-vertex(i) ←true
 8     end for
 /**** Optional for PADRA+ BEGIN *****/
 9     Cut-vertex(i) ←true
10   else
       //Check if it is really a cut-vertex
11     if DFSCheck(i)= false then
12       return //No need for recovery
13     Cut-vertex(i) ←true
14   end if
15 end if
 /**** Optional for PADRA+ END  *****/
16 if Cut-vertex(i) = true then
17   if ∃j|(j∈ N(i)&isDominatee(j) = true)
18     Unicast (i, "j, ACTION_FOR_ RECOVERY")
19   else
20     i←ClosestNeighbor(i) //failure handler
21     Unicast (i, "j, ACTION_FOR_ RECOVERY")
22   end if
23 end if
```

approach we follow may not always provide the shortest path to the closest dominatee. The optimal solution requires a dynamic programming approach which will require excessive messaging since all the possible paths should be explored before a decision is made. Therefore, we leave this as a future work. However, in terms of MMI we guarantee a movement distance of at most *r*.

Theorem 1: *PADRA guarantees a MMI of r for any topology.*

Proof: Since each node is replacing another node which is connected to it, the maximum movement distance will be *r*. In addition, each node moves at most once as dictated by the algorithm. Therefore, MMI cannot be larger than *r*. □

Total number of messages: Before calculating the message complexity for the worst case topology, we provide the type and number of messages sent in general in Table 1. For each node, two messages will be sent to determine the CDS of the whole network and one message will be needed to update the CDS when a node moves. Thus, totally three messages will be needed. Additionally, one message will be needed to notify the node which will be replacing a failed node unless the node is not a dominatee. Finally, when a failure happens 2k messages are sent to fix the connectivity where k is the number of hops to the closest dominatee.

TABLE 1. Types and number of messages used in PADRA(+)

Node Type	*# of Messages*			
	CDS	*DFS for PADRA+*	*Closest Node*	*Replace*
Dominatee	3	None	None	None
***Dominator**: Has at least one dominatee*	3	None	1	None
***Dominator**: Does not have any dominatees*	3	T + |Neighbors|	1	2k

Note that if a DFS is to be performed we name the approach as PADRA+ thereafter. DFS is performed as follows: The node picks the closest neighbor and start a DFS there. Each visited node is marked as gray. After a certain time, if all the other neighbors of the node are also gray, this indicates connectivity and thus the node will not be a cut-vertex. Thus, totally (T + |Neighbors|) messages will be needed where T represents the number of nodes within the sub-tree leaded by the closest neighbor.

The worst case behavior of PADRA and PADRA+ can be observed when the topology is a line as seen in Fig. 4.

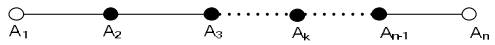

Figure 4. Worst case topology with dominatees at two ends.

Theorem 2: *Both PADRA and PADRA+ uses less number of messages than the optimal cascading even in the worst case.*

Proof: Let us assume that A_3 failed and A_4 was the failure handling node. Then A_4 will start a replacement process until A_n is hit. This means a total of n-3 nodes will be replaced. At each replacement a request and an ACK messages were used. Thus, 2*(n-3) messages will be sent for fixing the connectivity. Adding n*3 messages for CDS and n*1 for closest node designation, the total number of messages in the worst case for PADRA will be 6n-6 which is in the order of *O(n)*. In the optimal cascading case, each needs to know the whole topology. This will require (n-1) broadcasts for A_1 and A_n and (n-2) broadcasts for the rest. Replacement of A_3 will require two more messages. Thus, total messages need will be in the order of $O(n^2)$ which is more than PADRA.

For PADRA+, a DFS is performed for each dominator, from A_3 to A_{n-2}. Thus, the number of nodes performing DFS will be n-4. For A_3, this will cost traversing each node until A_n. Thus, T=n-3 and |Neighbors| = 1, a total of n-2 messages will be sent. This is same for A_{n-1}. As a result, only two nodes will use (n-2) messages, the rest will use even smaller number of messages. However, in optimal cascading, each node uses at least (n-2) messages as shown above. Thus, number of messages in PADRA+ will be less than the optimal case. □

IV. EXPERIMENTAL EVALUATION

A. Experiment Setup and Performance Metrics

In the experiments, we created connected inter-actor networks consisting of varying number of actors randomly placed in an area of interest. Each simulation is run for 10 different network topologies and the average performance is reported. For each topology, one of the cut-vertices is picked in such a way that there will be no dominatees among the neighbors of the cut-vertex. We used the total travel distance (TMA) and the total number of messages as the two metrics to capture the performance. We compared PADRA and PADRA+ with DARA [3] and the optimal cascaded movement solution which provides the least travel distance.

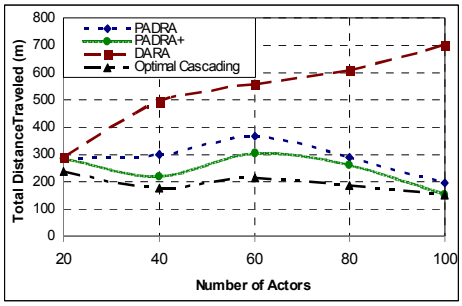

Figure 5. TMA with varying number of actors. Actor transmission range = 100m.

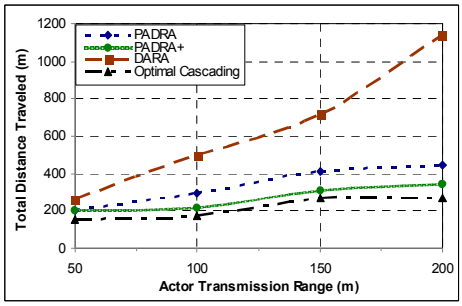

Figure 6. TMA with varying actor transmission range. # of actors is 40.

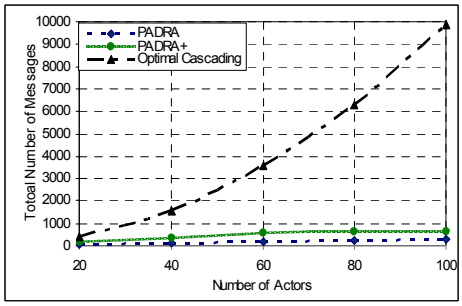

Figure 7. Total number of messages with varying number of actors.

B. *Performance Evaluation*

TMA: We evaluated the movement performance of our approach by varying the number of actors (20 to 100). The results are depicted in Fig. 5 have shown that both PADRA and PADRA+ performed very close to the optimal cascading while significantly outperforming DARA. Our approach maintains similar TMAs even when the network size grows as seen in Fig. 5 indicating its scalability. This is due to termination of the replacements when a dominatee is hit. Since the dominatees can be anywhere in the network, there is a high probability of reaching it earlier than a leaf node and this is independent of the network size. Note that, this is not the case in DARA and it needs to look for a leaf node to stop; making the performance even worse when the network size grows. Optimal cascading performs slightly better than PADRA and PADRA+ as it determines the shortest path to the closest dominatee each time.

We also conducted a similar experiment by varying the transmission range (50 to 200m) as seen in Fig. 6. Similarly, PADRA outperformed DARA due to same reason mentioned above. Note however, that as the transmission range increases, surprisingly the travel distance also increases. PADRA keeps the same rate of increase as the optimal solution while DARA performs worse at higher transmission ranges. One explanation for this increase is the increased transmission range and thus the travel distance from a node to another. In addition, cut-vertices with higher transmission ranges are rare since the network connectivity is improved. Therefore, if there is a cut-vertex, it usually connects blocks that are far apart which eventually increases travel distance. Since the path for DARA replacements is longer, travel distance gets even worse with the increased transmission range.

When comparing PADRA and PADRA+, the latter performs better as it minimizes the error in identifying the cut-vertices. Note that we also conducted experiments for assessing this error but they are omitted due to space constraints. We found out that especially with smaller transmission ranges the error of PADRA is around 2%. With larger ranges such as 200m the error increases up to 15%. However, PADRA+ minimizes this ratio by doubling the message cost as we discuss next.

Total number of Messages: We also compared the number of messages sent when determining the cut-vertices, designating failure handlers and replacing the failed nodes. The simulation results confirmed that our approach requires significantly less number of messages for the whole failure handling process when compared to optimal cascading. Even for PADRA+, we observed that in average it is much better although it doubled the number of messages compared to PADRA (See Fig. 7.)

V. CONCLUSION

In this paper, we presented a local, distributed and movement efficient protocol PADRA which can handle the failure of any node in a connected WSAN. As it is unknown at the time of failure whether such failure caused a partitioning or not, we provided a technique based on CDS of the inter-actor network which can decide whether a node is a cut-vertex or not before the failure happens. Basically, if a node finds out that it is a cut-vertex, the closest dominatee/neighbor is delegated to perform failure recovery on behalf of that node. The failure recovery is done by determining the closest dominatee node and replacing it with the failed node in a cascaded manner so that the movement load is shared among the actors that sit on the path to the closest dominatee.

Simulation results confirmed that PADRA performed very close to the optimal solution in terms of travel distance while keeping the approach local and thus minimizing the message complexity. In addition, our approach outperformed DARA in terms of travel distance which requires the knowledge of two-hops for each node.

REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu, "Wireless Sensor and actor networks: Research Challenges," *Elsevier Ad hoc Network Journa*l, Vol. 2, pp. 351-367, 2004.

[2] Keiji Ozaki et al., "A Fault-Tolerant Model for Wireless Sensor-Actor System," in the *Proceedings of IEEE HWISE'06,* Vienna, Austria, April 2006.

[3] A. Abbasi et al., "A Distributed Connectivity Restoration Algorithm in Wireless Sensor and Actor Networks, in the *Proceedings of IEEE Local LCN'07,* Dublin, Ireland, Oct. 2007.

[4] 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, August 2004.

[5] S. Das et al., "Localized Movement Control for Fault Tolerance of Mobile Robot Networks," in the *Proceedings of the First IFIP International Conference on WSANs*, Albacete, Spain, Sept. 2007.

[6] G. Wang et al., "Sensor Relocation in Mobile Sensor Networks," in the *Proceedings of the IEEE INFOCOM'05,* Miami, FL, March 2005.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "*Introduction to Algorithms*," Second Edition. MIT Press and McGraw-Hill, 2001.

[8] F. Dai and J. Wu, "An extended localized algorithms for connected dominating set formation in ad hoc wireless networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, no. 10, Oct. 2004.