Coverage-based Clustering of Wireless Sensor and Actor Networks

Brian McLaughlan and Kemal Akkaya
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL 62901
{brianm,kemal}@cs.siu.edu

Abstract—In addition to the miniaturized sensor nodes, Wireless Sensor and Actor Networks (WSANs) employ significantly more capable actor nodes that can perform application specific actions to deal with events detected and reported by the sensors. Since these actions can be taken at any spot within the monitored region, the actors should strive to provide maximal coverage of the area. In addition, minimizing the time to decide which actor should take what action is of utmost importance for responsiveness. In this paper, we propose a distributed actor positioning and clustering algorithm which employs actors as cluster-heads and places them in such a way that the coverage of actors is maximized and the data gathering and acting process times are minimized. Such placement of actors is done by determining the $k$-hop Independent Dominating Set ($k$-IDS) of the underlying sensor network. The performance of the approach is validated through simulations.

I. INTRODUCTION

Wireless sensor and actor networks (WSANs) have started to receive a growing attention from the research and engineering communities in recent years. Potential applications of WSANs include detecting and countering pollution in coastal areas, performing oceanic studies of bird/fish migration and weather phenomena, and monitoring the environment for unusually high-level of radiation, facilitating/conducting urban search and rescue (USAR), guarding borders against illegal infiltration and drug/human trafficking, detecting suspiciously active chemical/biological agents, etc. [1]. Such networks employ a large number of miniaturized low-cost sensing nodes that are responsible for measuring ambient conditions and reporting such measurements to some actor nodes over wireless communication links. Actors have the capability for processing the sensed data, making decisions and then performing the appropriate actions. Robotic Mule [1], which is an autonomous robot designed for the Army to detect mines in the battlefield, and the NASA JPL miniaturized rover [3], are some examples of possible actor nodes.

Coverage is one of the most important design goals in most applications of WSANs [3]. It is often required for the network to provide services at every part of the deployment area. In addition to coverage, actors’ responsiveness is usually desired in order for the network to be effective. For example in forest monitoring applications, actors such as fire trucks and flying aircrafts need to be engaged as rapidly as possible to control a fire and prevent it from spreading. Similarly for scientific studies or space applications actors should respond instantaneously to record rare phenomena, e.g., capture an image or record a weird behavior of a habitat.

In order to provide such services over large areas, clustering of the WSAN is often pursued given that each actor can also act as a sink [1][11]. In a clustered WSAN architecture, each actor acts as a cluster-head and takes certain actions based on the received data from the sensors in its cluster. However, before forming the clusters it is important to determine the location of the cluster-heads in order to maximize the coverage of all actors. A good coverage should minimize the overlap among the action ranges of the actors and include all the sensors deployed within the monitored region. Therefore, we need to position the actors as evenly as possible within the sensor network for maximizing coverage. For example in a forest monitoring application where unmanned vehicles capable of sprinkling water are used as actors, the positions of such actors should be decided based on the distribution of sensors in the forest. Given that sheer number of sensors is deployed, deterministic placement of sensors may not be possible and thus the sensors need to self organize and coordinate in order to determine the number and/or the best locations of actors needed in extinguishing the fire.

In order to achieve such good distribution of actors within the sensor network, we propose to determine the $k$-hop independent dominating set ($k$-IDS) of the sensor network and position the actors next to the location of the nodes in this set. Utilizing this positioning technique, this paper provides a clustering approach for WSANs where each actor leads a cluster consisting of the sensor nodes dominated by a dominator in $k$-IDS. Here, the parameter $k$ shows the number of hops for a sensor to reach its dominator and thus its cluster-head. The value of $k$ can be determined based on the available number of actors, action range of the actors and/or application level needs such as delay and throughput. Placing the actors at the locations of the nodes in $k$-IDS will provide two things: First, it will place actors in all parts of the sensor network which boosts the coverage of the actor network. Second, it will guarantee for each sensor to relay its data within $k$ hops to the cluster-head and thus help to reduce the decision time for the actors.

Since determining IDS and thus $k$-IDS of a network is an NP-Hard problem [15][22][23], one of the goals of this paper is to provide a heuristic approach to solve this problem. We provide a probabilistic and distributed heuristic which considers a neighborhood of $k$-hop radius for each node when
picking the dominators. The dominators are picked such that any two of the dominators will not have a direct link to ensure independence. We even strive to increase the number of links between any two dominators in order to minimize the overlaps among the action range of actors. Once the dominators are determined, each sensor joins to the closest dominator’s cluster by unicasting a message to the dominator node. Each dominator keeps a list of the nodes it dominates and then shares this cluster list with the actor which will be placed next to it. Our approach is validated through simulations and is shown to be both effective in achieving high coverage and reduced delay and number of actors employed.

This paper is organized as follows. Next section summarizes the related work and distinguishes our work from the previous research. In section III we provide the system model and then define the problem and present our approach for actor placement. Section IV includes the approach for cluster formation. The performance of our approach is evaluated in section V. Section VI concludes the paper with a summary and a highlight of our planned future extensions.

II. RELATED WORK

The coverage problem considered in this paper has been studied in the literature in the context of multi-robot systems. It is defined as the maximization of the total area covered by actors. This can be achieved either statically or dynamically [3]. Static coverage is the problem of deploying actors in a static configuration, such that every point in the area is under the actors’ shadow (i.e. covered) at every instant of time. Dynamic coverage, on the other hand, is addressed by algorithms which explore and hence cover the area with constant motion [3]. The network in multi-robot systems consists of robots which have sensing, vision and motion capabilities. The issue is how to locate the robots so that every point in the region will be under the shadow of a particular robot [5]. If the region is very large and the number of robots is limited, dynamic coverage and exploration can be considered [6]. In this case, the robots will continuously move and patrol the region for complete coverage. We consider a different interpretation of coverage in this paper. Our coverage definition is based on the location of sensors. It is basically the total number of sensors under the action ranges of all the actors. In addition, we consider a different system model, in which actors serve as cluster-heads for a set of sensors and need to stay available for extended time and thus have to conserve their energy.

Coverage has also been studied extensively in Wireless Sensor Networks (WSNs) as an important quality of service metric. Particularly, the recent works such as [8][9][10] studied the improvement of sensing coverage after the deployment of the sensors by assuming that the sensors can relocate. Our goal in this paper is not to improve the sensing coverage; we rather focus on actor coverage.

Regarding the clustering problem, there has been a lot of research in WSNs. Particularly, in the last few years, a lot of clustering algorithms have been proposed. Most of these algorithms aimed at generating the minimum number of clusters that maximize the network lifetime and data throughput [18][19][20], and provide load balancing [14] and fault tolerance [21]. Since coverage of cluster-heads is not an issue in WSNs, none of these clustering algorithms are geared for picking the cluster-heads such that a good coverage is achieved. We note that coverage and delay are the primary goals in our approach. Energy efficiency for sensors is ensured through reduced messaging overhead and multi-hop data forwarding as we will discuss later. In addition, we do not consider load balancing or fault-tolerance in clustering which is left for future research.

The area of WSANs is fairly new and thus there is not much work on coverage and clustering problems in WSANs. Quite recently, the characteristics and research challenges of WSANs have started to draw attention [1]. One of the few projects that address specific problems for WSANs is reported in [7]. The authors mainly tackled the problem of picking appropriate actors for responding to an event in a particular region. The paper focused on the problem of actor assignment to overlapping areas with the least amount of energy and packet delay. No clustering was considered in the paper.

The work in [24] considers load balanced clustering of WSANs with minimal message complexity. The proposed protocol focuses on improving the lifetime of the network and providing adequate coverage of sensors even when most of the sensors turn off their radio. In addition, actors are not employed as cluster-heads and their coverage are not considered in this work. Actors act as cluster-heads and this role rotates among all the sensors in order to extend the lifetime of the network. Our work on the other hand employs actors as cluster-heads in order to reduce the data gathering burden on sensors and expedite the decision making process of actors. In addition, we strive to maximize the coverage of the actors rather than sensors. Another recent work which considered improvement of coverage of actors is proposed in [11]. This work does not propose a new clustering technique however it employs one of the existing clustering strategies by using each actor as a cluster-head, as done in our approach. Each sensor picks the closest actor as its cluster-head. The idea of coverage in this work is based on area. It does not consider the distribution of sensors within the monitored region. Therefore, the approach may place few actors in an area that is densely populated with sensors while positioning numerous actors in areas with no sensors at all. Since there will be no events reported from the areas where no sensors are deployed, there is no need to keep actors there for possible actions. Our approach considering these situations defines a different metric for coverage as we elaborate next.

III. ACTOR PLACEMENT

The focus of the paper is to achieve maximal actor coverage when WSANs are clustered. In this section, we first describe the system model and assumptions and provide the problem definition. Then we describe our IDS-based actor placement approach in detail.
A. System Model and Assumptions

We assume that a set of sensors are spread throughout an area of interest to detect and track some events. The actors will be placed in the same area of interest in order to collect data from sensors and take necessary actions based on the received data. The actors will be able to communicate with each other as well as with a command node. The sensors are battery-operated with diverse capabilities and types and are empowered with limited data processing engines. We assume that the sensors are stationary, which is typical for WSNs. However, the actors are assumed to be mobile in order to relocate and act on different areas. All communication is over a single shared wireless channel. A wireless link can be established between a pair of sensor nodes if they are within the radio range of each other.

While sensors are deployed in abundance, the number of actor nodes is limited since robot-like nodes are usually used and they tend to be very expensive. The actors are both less-energy constrained and have larger transmission range than the sensors. The action range of an actor, which is defined as the maximum distance it can cover, is limited and assumed to be equal for all actors. Note that this is different than the radio range and is used to measure the coverage of an actor. Finally we assume that both the sensors and actors know their locations through mechanisms like GPS or other means. An example for the considered system model is depicted in Fig. 1.

B. Problem Definition

We define the problem as follows: “Given a set of sensors initially placed randomly in an area of interest and a number of available actors, we are interested in placing the actors and clustering the WSAN such that each cluster-head is an actor and total actor coverage is maximized.” Coverage in this context refers to the number of sensors that are under the control of the actors with respect to the total number of sensors. Note that this is a new interpretation of coverage as opposed to previous work. The idea is to consider the distribution of sensors within the monitored region as some parts may not have any deployed sensors at all. Even if some events may happen in those areas, the actors will not be able to know about those events since no sensors will be reporting about those events. Hence, it will be a waste of resource if an actor is deployed in such a place.

C. IDS based Actor Placement

An ideal solution to this problem should minimize the overlap among the action ranges of actors and cover all of the sensors deployed in the monitored region. However, this requires a central node which should know the number and location of all the deployed sensors. In most of the applications where sensors are dropped randomly from a flying vehicle, this may not always be possible. In addition, if the number of actors is small, the problem gets more challenging. In fact, the problem of determining the optimal locations for the actors has been proven to be NP-hard given the infinite number of possible locations to pick [13]. Therefore, we pursue distributed heuristics where sensors can determine an approximation to the problem of determining optimal location of cluster-heads.

Our goal is to spread the actors uniformly among the sensor nodes in the monitored region so that the overlap among the action ranges will be minimized. Before describing the details of our clustering approach, we note that in our approach the sensor network will be modeled as a unit disk graph $G = (V, E)$ where $V$ is the set of vertices and $E$ is the set of edges. In such a graph two nodes are connected by an edge only if they can communicate with each other based on the transmission range defined for sensor nodes. We provide the following definitions for $G$:

- **Independent Set (IS):** It is a subset of $V$ such that no two vertices within the set are adjacent in $V$.
- **Dominating Set (DS):** Let $S$ be the DS. Then $S$ is a subset of $V$ such that each vertex in $V$-DS is adjacent to at least one vertex in $S$.
- **$k$-Dominating Set ($k$-DS):** Let $S$ be the $k$-DS. Then $S$ is a subset of $V$ such that each vertex in $V$-$S$ can reach to at least one vertex in $S$ within at most $k$-hops.
- **Independent Dominating Set (IDS):** It is a DS such that no two dominators are adjacent within this set. That is, the dominators need to be within at least 1-hop distance (an edge) of each other. Finding IDS of a graph is NP-Hard [15].
- **$k$-Independent Dominating Set ($k$-IDS):** It is a $k$-DS such that any two dominators are within at least 1-hop distance of each other in this set.

Fig. 2 provides examples for DS, IDS, $k$-DS and $k$-IDS in a given network topology.

A good coverage can be achieved by determining an IS of the sensor network and place the actors next to the location of those nodes. Since no two nodes can have a direct link in an IS, it helps to reduce the amount of overlap among the action ranges. However, the size of IS should be large enough in order to cover the whole sensor network. This is exactly same as determining the DS of the sensor network so that each sensor will be under the control of a dominator (actor). Therefore, the ideal solution is to determine the IDS of the...
sensor network. This guarantees to cover all the sensors within the monitored region by minimizing the overlap among the action ranges. However, in order to minimize the number of actors employed and thus the number of clusters, we need to minimize the size of IDS, leading to the problem of determining k-IDS where each actor is at most k-hop away from the sensors it dominates.

IDS problem has been stated as NP-Hard in [15] and some approximation algorithms have been proposed for this problem [16]. In fact IDS is a special case for k-IDS when k=1. Therefore, k-IDS is also an NP-Hard problem [22][23]. Next, we describe our distributed and probabilistic heuristic for determining k-IDS of a network.

![Diagram](image)

**Fig. 2.** In this 14 nodes network, the following holds: 1) A₂, A₄, A₅, A₆, A₇, A₁₃ form a DS. 2) A₈, A₉, A₁₀, A₁₁ and A₁₂ form an IDS. 3) A₁ and A₁₁ form a 2-DS. 4) A₁ and A₁₁ form a 2-IDS. 5) A₁₄ forms a 3-DS.

### D. Detailed k-IDS Algorithm

While determining the k-IDS, we use some message exchanging within k-hop neighborhood of a particular sensor node in order to minimize the messaging and timing costs and thus extend the lifetime of the sensor nodes. We define three different messages as follows:

1) **ALIVE:** Sent by each sensor node to its k-hop neighbors to show its existence.

2) **DOMINATOR:** Sent by each sensor node to its k-hop neighbors which wants to become a dominator.

3) **BORDER:** Sent by a sensor node that is at k-hop hop from a dominator.

The idea behind our heuristic is to allow some nodes advertise themselves as dominators based on a computed probability. These nodes broadcast a DOMINATOR message to their k-hop neighbors. The probability of becoming a dominator depends on many factors such as the number of received ALIVE messages, the distance to the k-hop hop and some initial fixed parameters as we will explain in details below. Any node receiving a DOMINATOR message will be dominated by the originator of the message. If a node does not receive any such message and it is not a dominator, it also decides to advertise itself as a dominator within its neighborhood. In order to provide independence property, the nodes at the border (i.e., at the k-hop hop from a dominator) broadcast BORDER messages to their k-hop neighbors. With the increasing value of k, the probability of becoming a dominator reduces which gives chance to eliminate possible overlaps among the action ranges of dominators.

The complete algorithm pseudo-code is given below:

---

**Pseudo Code for k-IDS Algorithm**

```plaintext
// For each sensor i perform the following
k-IDS(i)
1 Broadcast(Msg(i,"ALIVE",Neighbors(i),k))
2 while true
3 Wait(i, T)
4 Tabulate received Msg(∀j,"ALIVE",i)
5 Tabulate received Msg(∀j,"DOMINATOR",i)
6 Tabulate received Msg(∀j,"BORDER",i)
7 Forward(∀Msg(j,"ALIVE",Neighbors(i),k-1))

//When a DOMINATOR message is received
8 if i ≠ Dominator and received Msg(j, "DOMINATOR", i, k) then
9 Dominated(i) ← TRUE
10 Forward(Msg(j),"DOMINATOR",Neighbors(i),k-1)
11 if Dist(j)<closest_known_dominator(i) then
12 closest_known_dominator(i) ← j
13 end if
14 if k-1 then //DOMINATOR message expires
15 Broadcast(Msg(i,"BORDER",Neighbors(i),k))
16 end if
17 end if

//When a BORDER message is received
18 if i ≠ Dominator and received Msg(j, "BORDER", i, k) then
19 if BTTL(i)<Msg(j,"BORDER", i, k) then
20 BTTL(i) ← Msg(j,"BORDER", i, k)
21 end if
22 Forward(Msg(i,"BORDER", Neighbors(i), k-1))
23 end if

//Suitability function S to become a dominator
24 S ← f(# of received Msg(∀j,"ALIVE",i), TTL(i))
25 Generate random chance R;
26 if R < Suitability then
27 Dominator(i) ← True;
28 Broadcast(Msg(i,"DOMINATOR",Neighbors(i),k))
29 end if
30 end while
```

As shown in line 1, the node broadcasts an ALIVE message to its neighbors. As with all new messages in this algorithm, the message has a Time-to-live (TTL) value of k. The node then begins a loop in line 2 that ends at line 30. In line 3, the node will wait for some random time period T. During this time, it will take note of any messages that it hears. This time period limits the chance of neighboring nodes acting at the same time. The exact time period for each node changes at each iteration.

Lines 4-6 tabulate the received messages at the sensor node i. These are namely DOMINATOR, BORDER and ALIVE messages. The number of ALIVE messages heard represents the number of nodes within k hops of the node. In line 7, sensor i forwards each received ALIVE message to each of its neighbors by decrementing the TTL value of each message.
Lines 8-17 determine whether or not the sensor node should become dominated by another node based on the received messages. If the sensor node \( i \) receives a **DOMINATOR** message from node \( j \) and \( i \) is not already a dominator, then it will be dominated by \( j \) as long as it is the closest dominator that it has heard from. The sensor node \( i \) will also forward the **DOMINATOR** message to its neighbors if its TTL is not expired. Otherwise, the sensor node \( i \) will broadcast a **BORDER** message to its neighbors with the TTL set to \( k \).

Lines 18-23 execute if the node \( i \) hears a **BORDER** message and is not a dominator. If this condition is true, the remaining TTL of the **BORDER** message (i.e., TTL – 1) is stored and compared to the TTL of any other received **BORDER** messages. The maximum of these TTL values is saved as the BTTL value which will be used in the computation of the suitability formula for becoming a dominator. Node \( i \) then forwards the **BORDER** message to its neighbors if the message TTL has not yet expired.

Line 24 determines the suitability of a sensor node to become a cluster-head. The suitability score is a function \( S \) of the number of received **ALIVE** messages and the closest **BORDER** message. We utilize the following function in our algorithm:

\[
S_i = \left( \frac{B + M_i \times AC_i}{2^{BTTL}} \right)
\]

where:

- \( B \) is some base constant chance of becoming a dominator.
- \( M_i \) is some constant weight given to each other node within \( k \) hops of the node \( i \).
- \( AC_i \) is the total number of heard **ALIVE** messages by sensor node \( i \).
- \( BTTL \) is the TTL of the closest heard **BORDER** message.

This formula provides for several key behaviors. First, it allows any node to become a dominator regardless of the number of its neighbors. This allows the isolated nodes in partitioned networks to have a dominator and thus a cluster-head. Second, it favors the nodes which have more neighbors to be more suitable as dominators than those nodes that have fewer neighbors. In addition, the **BTTL** parameter in the formula exponentially reduces the chance of a node becoming a dominator based on its proximity to the border of another dominator. This in turn reduces the amount of overlap produced by the action ranges of actors which will be employed as cluster-heads at the position of dominators. For example, in a network where \( k = 3 \), a node three hops from a dominator will broadcast a **BORDER** message that will propagate for another three hops. Any node that receives the message with TTL = 3 will know that it is directly on the border of an already formed cluster. Therefore, if it were to become a dominator, it may create a significant overlap among the action ranges of two cluster-heads. Thus, it will most likely not become a dominator unless it goes for a significant amount of time without hearing a **DOMINATOR** message. However, nodes further from the border (e.g., nodes that receive a **BORDER** message with smaller TTL) will reduce their chance of becoming a dominator by a smaller amount.

Lines 24-30 generate a random probability that is then compared to the Suitability score to determine if the node will become a dominator. If this is the case, the node broadcasts a **DOMINATOR** message to all of its neighbors.

**E. Worst-case Message Complexity of k-IDS**

Our approach introduces very little message overhead. Each node sends **ALIVE** messages to discover its neighbors and **DOMINATOR** messages if it decides to be a dominator. Since, this message will travel \( k \) hop, each node within \( k \)-hop distance forwards it once. Finally, the nodes at the \( k^{th} \) hop broadcast **BORDER** messages.

In the worst case, a node can hear a **DOMINATOR** message from all the dominators. Let the \( s = |k-IDS| \), then the number of **DOMINATOR** messages a node will forward in the worst case would be \( s \) if \( k \) is considered large enough to reach all the sensors in the region. Adding the **ALIVE** and **BORDER** messages, totally at most \( (s+2) \) messages are transmitted by each node leading to a message complexity of \( O(s) \).

**IV. COVERAGE-BASED CLUSTERING**

In this section, we describe how clusters are formed and maintained after the IDS-based actor placement is performed.

**A. IDS-based Cluster Formation**

Once the dominators are determined, the next step is to form the clusters. In the determination phase of dominators, each sensor can hear the **DOMINATOR** message from multiple dominators and can pick its dominator based on the distance to and ID of the dominator as stated in the previous section. Since, the actors will be placed next to dominators, the dominators act as proxy cluster-heads until the actors are deployed. Therefore, a sensor sends a **JOIN** message to its dominator in order to join its cluster. The **JOIN** message is not broadcast, rather it is unicast. The path for such a unicast is obtained from the **DOMINATOR** message. This message will contain the path back to the dominator since each sensor node receiving **DOMINATOR** message adds its ID and broadcasts it to its neighbors, as used in DSR [17].

A dominator receiving **JOIN** messages builds a list and stores the IDs of the sensors sending this message and thus forms the cluster list. This list will then be transmitted to the actor which will be placed just next to this dominator.

When such an actor is placed/relocated to this location, it sends an **ACTOR** message and the dominator at the same location receiving this message sends the IDs and routes of all sensor nodes in the cluster to the actor node. The actor can then unicast a **CLUSTER-HEAD** message to each sensor node in order to notify them about its existence and being ready to collect data. This message will also include the ID of the actor which uniquely identifies the cluster.

Note that with the addition of this **JOIN** message, each sensor now transmits three messages in order to complete the clustering process. As in the case of **DOMINATOR** messages,
JOIN messages are also forwarded through the dominator. Therefore, in the worst case a sensor node within 1-hop distance to the dominator will have to forward the JOIN messages of all the sensor nodes within k-hop distance. In this case, it will forward k JOIN messages including its message. Therefore, for the entire clustering process in the worst case, a sensor will have to send \((1+s+k)\) messages which will be \(O(s)\) or \(O(k)\) depending on the values of \(s\) and \(k\).

B. Handling Actor Mobility

Since we assumed that actors are following an on demand coverage approach (i.e., stay stationary most of the time) sensor associations and routes from sensors to actors do not change very frequently. The only situation where we need cluster re-organization is when the actor (i.e., cluster-head) has to move in order to act in larger areas or help other actors. In those cases, actors will be absent for certain amount of time and thus sensors’ data should be collected by other actors. For this purpose, we propose to re-associate sensors temporarily with the closest actors through the help of border nodes. Recall that the border nodes of a dominator were sending a BORDER message to their neighbors when TTL of DOMINATOR message was 1. Therefore, the nodes sending BORDER message will identify themselves as the border nodes for the cluster they belong to. When the routes are created, the border nodes will piggyback their IDs and locations to all the sensor nodes on their routes to the cluster-head. In this way, every sensor node can learn its corresponding border node and keep the path to its border node (i.e., the reverse route on the message). When the actor node moves, it sends a LEAVING message to all the sensors in its cluster through the reverse routes of sensors. A sensor receiving this LEAVING message will begin sending its data to its border node thereafter. The border node broadcasts a GATEWAY message to its neighbors in order to determine the closest border node in another cluster. It will thus forward the data from the sensors to this border node so that the data will be routed towards a new actor in that cluster. When the actor comes back, it sends a BACK message and sensors start forwarding their data to the actor again. Note that with this cluster update mechanism, there is almost no overhead for the sensors. Determining the border node for a sensor node does not cost anything as it will be learnt from the routed data. Only the border nodes need to identify the closest border node in other clusters which can be done by exchanging a message with the neighbors.

C. Communication with Command Node

In order to utilize our IDS-based clustering approach in real-life applications, there are two issues to consider for the command node: 1) Deciding the value of parameter \(k\) 2) Getting the size of \(k\)-IDS from the sensor network.

In this paper, we assume that the sensor network is initially aware of a predefined \(k\)-value and thus the sensors know how far they will broadcast DOMINATOR messages in the proposed \(k\)-IDS algorithm. The value of \(k\) can be decided based on the \(a/s\) ratio where \(a\) and \(s\) represent the actors’ action range and sensors’ radio range respectively. For example if \(a = 2s\), then it may make sense to pick \(k\) as at least 2 in order to prevent overlaps and thus maximize the coverage. In addition, \(k\) value can be selected based on the application level requirements such as number of available actors, end-to-end delay and action time as mentioned before. For instance, time critical applications such as target tracking and destroying may require hard real-time constraint and thus the value of \(k\) should be selected as small as possible.

Another issue is to receive the information from the sensor network about the number of actors needed and their locations. This is collected by the command node and then the actors can either be programmed to move to the desired locations or they can directly placed at those locations depending on the type of application. If the region is accessible, direct placement would be more efficient since it will eliminate the movement of actors. Otherwise, if it is for instance a battle environment, there is no other option than dropping/placing them to a certain place and let them to relocate from there to their new locations. The command node can be notified in two ways: One possible solution is to get the dominators directly send their IDs and locations to the command node. In the other case, the dominators can relay the information via a powerful sink node which is deployed within the monitored region.

V. EXPERIMENTAL EVALUATION

The effectiveness of the approach is validated through simulation. This section describes the simulation environment, performance metrics, and experimental results.

A. Network Operation and Experiment Setup

We assessed the performance of our approach in a simulated target tracking application environment. Sensors detect targets and relay information about the location of the targets to the actor nodes which can take certain actions on the detected targets (e.g. shoot at them, video tape, etc.). In the experiments, the network consists of varying number of sensor nodes randomly placed in an area of size 400m by 400m. The number of actors depends on the size of \(k\)-IDS. The positions for the actors are determined either by using IDS-based approach or randomly. In case of random placement, the number of random locations matches with \(|k\)-IDS]. We assumed a transmission range of 50m for sensors. In addition, we assume an action range of 100m for each actor which is the circular area where it can perform certain actions. For the suitability function \(S\), \(B\) and \(M\) are assumed to be 0.5% and 2% respectively. These values are confirmed to provide the best results through experimentation.

B. Performance Metrics

In the experiments, we used the following metrics to capture the performance:

- **Coverage**: Defined as the total number of sensors under the action ranges of all actors divided by the total number of sensors in the region.
- **End-to-end delay**: This is defined as the number of hops for a sensor to reach its cluster-head.
We varied the number of sensors and $k$ value and observed the coverage and delay performance. In the experiments, we applied 100 distinct seeds in order to generate network topologies based random uniform distributions. Separate simulation runs were performed for each topology and the results are averaged over these 100 runs.

C. Performance Evaluation

In this section, we report on the performance results observed for the above metrics. We compared our IDS based clustering approach (IDSC) to three other baseline clustering techniques. For each experimental run, the $k$-IDS algorithm was executed and evaluated. Next, the same number of actors was randomly redeployed and the clusters are formed based on the proximity to the actors. Each sensor picks the closest actor as its cluster-head. We name this clustering as random proximity based clustering and refer it as (RPC) in the graphs.

We also used two other baselines for only coverage purposes where the actors are deployed based on the region size. First we placed enough number of actors uniformly in the region such that the action areas of the actors touch but do not overlap. The clusters are formed based on proximity again. This technique is referred as uniform proximity based clustering (UPC). Finally, enough actors are deployed to fully cover the whole area with enough overlap among the action ranges which is named full proximity based clustering (FPC).

Coverage Performance: Fig. 3 shows the difference in actor coverage when IDSC, RPC, UPC and FPC are used. In this experiment, the number of sensor nodes varied from 300 to 700 while $k$ was held at 3.

The IDSC algorithm maintains a steady coverage at near 100% which is very close to the coverage of FPC. The lack of perfection can be attributed to the difference in actor range versus the communication range of the nodes. In these experiments, the theoretical maximum distance that a node can be from its cluster-head is farther than the actor range. For instance, a node that was three 50m hops in a straight line directly away from the cluster-head would be outside the 100m range of the actor. However, if 100% coverage is mandated, the $k$ value can be set to 2 which can ensure matching the coverage of FPC in a distributed manner. Nonetheless, the number of actors employed will also be close to that of FPC. There is a trade-off here between the coverage and the number of actors employed and thus the $k$ value. We investigate such trade-off later in Fig. 4. For best results, $k$ can be tuned based on the available number of actors, the desired coverage and end-to-end delay.

The ability of IDSC to maintain a consistently high coverage can be attributed to the dynamic nature of the algorithm. Cluster-heads are formed where needed, providing every sensor node with an actor within $k$ hops. Thus, the algorithm is highly scalable as confirmed in Fig. 3. RPC on the other hand produced less impressive results. Actors may be evenly distributed or may bunch together, producing a less desirable average coverage. The lower values at lower population sizes can be attributed to the nature of the network.

At lower population sizes, the network tends to be scattered into several smaller, almost unconnected networks. The IDSC algorithm can handle these networks as well as any other, but the RPC algorithm tends to miss trailing tendrils. Coverage of the UPC and FPC algorithms stayed constant. The FPC algorithm maintains 100% coverage by definition. Since the UPC algorithm may have coverage gaps, it can only cover around the 80% of all the sensors deployed in the region.

In order to observe the effect of $k$ value on the coverage and the number of actors employed, we conducted an experiment by keeping the number of sensors at 500. $k$ was varied from one to four hops. The results of this experiment are shown in Fig. 4.

The IDSC algorithm showed a little degradation of performance as the number of hops increased, going from 100% coverage at $k=1$ to approximately 95% coverage at $k=3$ and then dropping to 90% when $k=4$. Like the discrepancies noted in the first experiment, the coverage problems were due to the distance from the node to its cluster-head being potentially larger than the action range of the actor. While the coverage is still acceptable for large values of $k$, the number of required actors is much smaller with increasing $k$ values which are shown in the upper part of the graph. Note that IDSC outperformed RPC for all different values of $k$ in terms of coverage. The performance of the random algorithm began to
degrade significantly as $k$ increased.

Given a 95% of coverage with $k=3$ from Fig. 4, we conducted experiments to check how many actors would be needed in the ideal case in order to provide 100% coverage. We varied the number of sensors and investigate the performance of IDSC in terms of the number of actors required. The number of actors required by FPC algorithm to guarantee the coverage of all the sensors is solely a function of the size of the operational area which does not change with the number of sensors. The experiment results depicted in Fig. 5 demonstrated that IDSC algorithm maintained values less than the baseline for an area of 400mX400m. As the sensor population grew, the number of required actors increased until a saturation point and then held constant. This was due to the sprawling nature of smaller networks. As the network grew in size, it covered more area of the graph. At larger populations, the new nodes simply filled in gaps in the graph. At this point, no new actors were needed because no new ground was to be covered.

**Degree of Overlap:** Although a goal is complete actor coverage, it is often important that unnecessary redundancy be reduced. To that end, the degree of overlap among the action ranges produced by the IDSC and RPC algorithms were compared visually in a sample network of 4 actors and 50 sensor nodes. Figs. 6 and 7 illustrate the differences between the two algorithms. The solid squares denote cluster-heads and their related actors and the large circles show the actors’ action areas.

Because of the nature of the network, neither algorithm can realistically completely eliminate redundancies. However, due to the intelligent placement of cluster-heads in places where they are most needed, the IDSC algorithm typically produces minimal overlap only at the borders of action areas while the RPC algorithm may generate significant overlap as well as producing areas without adequate coverage.

The IDSC algorithm also has significant advantage when compared to UPC algorithm. For instance, in Fig. 6, there are areas -particularly in the upper right portion of the deployment area- where sensor nodes are not present. UPC would place actors in these areas. Actors placed in these locations would not be beneficial, as the actors would not have any...
information on which to react. The IDSC algorithm places the actors in areas that are producing data, making each actor an efficient asset.

**End-to-end Delay Performance**: We also looked at the end-to-end delay performance of our approach. The delay was measured as being a function of the number of hops from a node to its cluster-head. As with the first experiment, the population size was varied from 300 to 700 nodes while k was held at 3. We compared the delay with RPC algorithm. Same number of actors is deployed in RPC. The results are depicted in Fig. 8.

Due to the scalable nature of the IDSC algorithm, the delay remained essentially constant despite the increased number of nodes. In addition IDSC provides up to 25% decrease in average end-to-end delay when compared to RPC algorithm given that utilizing k-IDS can guarantee k hop delay in the worst case.

The preceding experiments illustrate the nature of the IDSC algorithm. It provides coverage almost as complete as an overlapping uniform distribution with guaranteed delay and significantly less actors, placing actors only where necessary. By varying the value of k, a balance can be achieved that rivals the best qualities of all the baseline comparisons.

VI. CONCLUSION

Wireless sensor and actor networks (WSANs) are gaining popularity in a number of civil and military applications such as urban search-and-rescue, border protection, etc. Actors collected sensor’s data and collaboratively perform tasks in response to detected events/targets. In most applications actors are randomly deployed and are expected to cover an area of interest. In this paper, assuming a coverage metric based on the covered sensor count, we presented a distributed coverage-aware placement and clustering mechanism for WSANs. Our approach utilizes k-IDS of the underlying sensor network in order to determine the number of cluster-heads and their locations. The actors are then positioned to the location of these cluster-heads in order to maximize coverage. This way, we also guarantee a k-hop delay for each packet from a sensor to actor.

The simulation results confirmed that our approach IDSC can almost match the coverage of the FPC algorithm for appropriate values of k and thus with significantly less number of actors as the degree of overlap among the action ranges is strived to be minimized. IDSC also provides up to 40% increase in coverage and 25% decrease in end-to-end delay when compared to RPC. In addition, these improvements are achieved in a distributed manner with a message complexity in the order of the size of k-IDS or the parameter k which are very small when compared to the number of sensors deployed in the region.

Our future work will focus on extending the current clustering approach in order to provide load balancing and connectivity among the actors. In addition, we plan to investigate distributed approaches in order to locate the actors next to the location of sensors in the k-IDS.

REFERENCES