Suplimentary Material for: Decentralized Robot Swarm Clustering: Adding Resilience to Malicious Masquerade Attacks

Mitali Gandhe[1] and Michael Otte[2]

[1] Georgia Institute of Technology, Atlanta, GA 30332
[2] University of Maryland, College Park, MD 20742

Abstract. This document contains supplimentary material for the paper titled "Decentralized Robot Swarm Clustering: Adding Resilience to Malicious Masquerade Attacks" that has been submitted to the Workshop on the Algorithmic Foundations of Robotics (WAFR'22). In particular, this document contains:

- A more comprehensive discussion of related work.
- Additional descriptions of algorithms using both images and psueodode.
- Additional pictures of the algorithms running on the Kilobot hardware testbeds.

The abstract for the paper now follows. We compare the resilience of four distributed robot swarm clustering algorithms to masquerade attacks launched from malicious robots within the swarm. The clustering algorithms are distributed variants of DBSCAN and k-Means that have been modified for use on a distributed robot swarm that only has access to *local* communication and *local* distance measurements. We subject two distributed swarm variants (one based on k-Means and one based on DBSCAN) to malicious masquerade attacks and observe how clustering performance is affected. We then modify each variant to include a distributed Intrusion Detection and Response System (IDRS) to detect malicious robots and maintain the swarm's integrity despite an attack. We evaluate all four variants in both simulation and in a hardware testbed containing a swarm of 25 Kilobot robots. We find that centralizing data within the swarm makes the swarm more vulnerable to malicious attacks, and that distributed IDRS relying on local message passing can effectively identify malicious robots and reduce their negative effects on swarm clustering performance.

1 More In Depth Discussion of Related Work

We now survey related work on distributed clustering algorithms, security in robot swarms, security in distributed networks, network attack classification, and other combinations of clustering and robot swarms. We conclude this section with a discussion of the most closely related work how it differs from our work.

1.1 Distributed Clustering Algorithms

Clustering algorithms, such as k-Means [32] and DBSCAN [26], were originally developed in the 1960s and 1970s to classify multivariate data for the purposes of data analysis. Originally invented when single-processor computers were the norm, the original versions of these algorithms were designed for use on centralized computational architectures. Clustering remains an important tool for data classification and an area of continued research [14, 8, 5, 36]. More recent variants of k-Means and DBSCAN have been developed to take advantage of distributed computing [4, 37, 47, 25, 33, 17, 9] including over peer-to-peer, wireless, and ad hoc networks [12, 51, 43, 41, 29, 27, 35, 50]. Data is partitioned among multiple computers and then each computer clusters its own partition of the data set. Communication is used to identify similar clusters found on multiple computers. Such work has largely focused on the potential time savings that can be achieved by using multiple CPUs or computers working together and/or the problem of clustering more data than can be stored on a single computer. However, work has also been done on distributed clustering algorithms that ensure the privacy of the data being clustered. [21, 38, 20, 39, 28, 3, 10].

The distributed clustering approaches listed above seek efficiency by performing as much computation as is practical on a single computer and minimizing communication between computers. In contrast, we explore the problem of distributed swarm clustering, in which each robot is in charge of a single datapoint—its own location—and clustering is the result of an emergeant process involving many local communication between robots but relatively simple computations on each robot.

1.2 Security in Robot Swarms

Existing work in swarm robotics security has focused on surveying the broad range of threats to the swarm and developing preventative security measures [19, 23, 24]. Thompson and Thulasiraman view the problem through a network security lens and use authenticated encryption to transmit classified information in a swarm of three simulated UAVs [45]. Others, such as Dolev *et al.*, successfully implemented public and private encryption key procedures within a swarm, ensuring confidential data transfer [13]. Wolf *et al.* recently published a study on how malicious robots could influence a decentralized swarm programmed to encircle a target [49]. The swarm relied on distances to their neighbors to identify the best positions to form a perimeter. Malicious robots within the swarm attempted to gain access to the target by creating a gap in the circle's arc (simulations show that malicious robots are able to control 6 - 23% of the perimeter).

Gil *et al.* detect *spoofing*, one robot claiming to be multiple robots to gain undue influence, by monitoring the directional wireless radio signatures associated with incoming messages [16]. Thus, Gil *et al.* show that if robots are equipped with appropriate sensors then hardware signatures and observed data can help identify malicious robots. This is one reason why we have chosen to focus on malicious attacks that do not involve spoofing the robot ID of the sending robot.

1.3 Security in Distributed Networks

A survey on Intrusion Response Systems indicates that many network monitoring frameworks, such as EMERALD and CITRA, use distributed surveillance and communication to check for suspicious activity, and respond to threats by taking action locally [44]. These algorithms provide evidence that security in a swarm can be handled locally, if swarm-specific vulnerabilities are addressed.

However, other work has shown that an IDS often struggles to determine what constitutes abnormal behavior, and falsely identifies outlier data as tampered data [23]. This issue can exacerbated in swarms if local interactions happen to differ from the swarm's overall emergent behavior [24]. The resilient swarm k-Means algorithm that we present includes the detection false positive malicious labelings. This provides a mechanism for reintegrating robots wrongly labeled as malicious.

1.4 Network Attack Classification

To disrupt the clustering algorithms in our work, we develop attacks that target specific weakness of the algorithms. According to one classification system, this type of attack is malicious, intentional, human-generated, and seeks to corrupt information [22]. A second taxonomy offers a more specific name: a masquerade attack [6]. Masquerading is an insider attack, as the malicious robot has gained access to the system, but may not have much knowledge of the systems functions or procedures.

Past research notes that responding to masquerade attacks is challenging, as any solution must be specifically tailored to the way the malicious robot disrupts the swarm [19].

1.5 Other Combinations of Clustering and Robot Swarms

Other clustering algorithm have been developed that take inspiration from swarms in the biological world. For example, in ant-inspired clustering and sorting algorithms, virtual 'ants' 'carry' data around a virtual space, where the probability of 'picking up' or 'dropping' data is determined by its similarity to other nearby datapoint [31, 46, 30]. Particle swarm optimization is a general technique inspired by communication in swarms of birds that has been adopted for clustering (as well and many other applications). Surveys of ant inspired clustering algorithms and particle swarm based clustering algorithms can be found in [18] and [1], respectively.

1.6 Closely Related Work

McCune and Madey introduce an ant-inspired pick-up, carry, and drop algorithm for resource aggregation they have called "Decentralized k-Means Clustering" drawing a parallel between k base stations (cluster centers/ant nest), mobile agents (transport nodes/ants), and stationary sensors (resources, food).

They test this idea using virtual simulations [34]. In contrast, the appropriate metaphor for our work is that robot locations are datapoints—and the objective is to find k cluster centers (also robots). Other differences between our work and [34] include the fact that we perform experiments in a hardware testbed and investigate both: (1) how swarm clustering algorithms can be hacked and (2) made more resilient to hacking.

Another set of closely related work includes k-median algorithms designed for use on graphs and trees [48, 11, 2]. See [7] for both an overview of this subfield as well as an example pertaining to facility placement on road networks. Because we use robot locations for both datapoints and the k cluster "centers," once could make the argument that we are using median values and not mean values. However, previous work on the k-Means and k-medians algorithms has defined the distinction between them as follows: k-Means seeks to minimize the sum of the L^2 norm from all nodes to their corresponding cluster's center, while k-medians seeks to minimize the sum of L^1 norms from all nodes to their corresponding cluster's center. In contrast, the swarm algorithms that we investigate consider distance as defined through/along the communication graph and not through the Euclidean space in which the graph is embedded.

Graph distance is an appropriate metric for robot swarms that communicate locally, lack GPS data, and can determine line-of-sight distance to neighboring robots using on board sensors. For this reason graph distance is commonly used in robot swarm algorithms. (Graph distance is also used for other problems at the intersection of robotics and graph theory; for example, the shortest path planning problem and traveling salesperson problem). Major differences between our work and preexisting work on k-Means and k-median algorithms is our focus on distributed algorithms, swarms of robots, and hardware experiments—and, in particular, our exploration of hacking and resilience to hacking.

1.7 Summary of Contributions of this Work vs. Previous Work

In contrast to past work on distributed clustering algorithms, in general, our research focuses on how malicious robots can influence the emergent behavior from within the algorithms procedures on real-robot swarms. We also extend previous work to study how distributed algorithms might identify and ignore data from a malicious robot. Finally, we examine the strengths and vulnerabilities that arise from the distributed nature of swarm algorithms, with the intent to build more secure and practicable robot swarms.

A preliminary and non-archival version of this work was presented as a poster at the IEEE International Symposium on Multi-Robot and Multi-Agent Systems in 2021 [15].

2 Preliminaries

2.1 Nomenclature

The robot swarm $S = \{r_1, \ldots, r_n\}$ contains n robots with unique IDs $1 \ldots n$. Let d_{ij} denote the distance between robots r_i and r_j. We assume that if robot r_i can

communicate with r_j then r_j can determine d_{ij}. We do **not** require robots to know their global positions. The distance between two neighboring robots can be determined without global positions, for example, by observing message time of flight or received signal strength. The communication graph over the swarm is $G = (V, E)$, where we abuse our notation by letting robots represent their (own) respective nodes in the node set $V \equiv S = \{r_1, \ldots, r_n\}$. The existence of a directed edge $(r_i, r_j) \in E$ indicates that robot r_i can communicate with r_j.

While communication between robots in the real world is generally non-symmetric, it is algorithmically convenient to impose symmetry in the communication graph such that $(r_i, r_j) \in E \iff (r_j, r_i) \in E$. This can be done by having robot r_j drop messages received from robot r_i whenever d_{ij} is greater than a user defined threshold distance d, i.e., such that the human user knows a swarm's communication hardware will reliably send/receive messages further than d. We now formalize this requirement. Define the "maximum disc-neighborhood distance" $d_{\max}$ as the maximum distance for which a bidirectional $d_{\max}$-disc communication sub-graph exists within the swarm's natural/raw communication network. $d_{\max} = \arg\max_d (\{(i,j) \,|\, d_{ij} \leq d\} \subset E)$. By construction, bidirectional communication exists between each pair of robots r_i and r_j such that $d_{ij} = d_{ji} \leq d_{\max}$. Thus, a d-disc communication graph with symmetry of the form "$(r_i, r_j) \in E \iff (r_j, r_i) \in E$" can be achieved by having robot r_j accept messages from robot r_i only if $d_{ij} \leq d$ for $d \leq d_{\max}$.

Let $E_d = \{(i,j) \,|\, d_{ij} \leq d\}$ be the set of all edges of length d or less. Define $G_d = (V, E_d)$. Having robots drop messages as described above, guarantees

$$((r_i, r_j) \in E_d) \wedge (d \leq d_{\max}) \implies ((r_j, r_i) \in E_d)$$

In other words, robot r_j can infer that if it (r_j) accepts a message from r_i then the sender (r_i) will accept messages from r_j in return. Cluster membership is determined based on the topology of the d-disc communication graph G_d.

The DBSCAN algorithm has two parameters: a distance threshold d that determines whether or not two nodes (robots) are neighbors (we assume $d \leq d_{\max}$), and the minimum number of neighbors m a node must have to be an 'internal' node. Each cluster found by DSCAN has a core of internal nodes. Thus, decreasing m tends to increase the resulting number of clusters. The k-means algorithm requires the user to define the number of clusters k.

2.2 Formal Problem Statements

We now define the swarm clustering and hacking problems.

Problem 1. Distributed Swarm Clustering: *Given a swarm of n robots that communicate locally, the swarm must collectively divide itself into mutually exclusive subgroups (clusters) based on robot's relative proximity such that robots within a particular cluster are closer to other robots in their own cluster then they are to robots in the other clusters.*

Problem 2. Swarm Clustering Masquerade Attack: *Given a swarm of n robots that communicate locally and that is attempting to solve Problem 1, one or more*

malicious robot(s) must cause the swarm to find a lower quality solution (or prevent the swarm from finding any solution) by injecting incorrect data into the distributed algorithm–but not by simply disrupting communication.

A notable element of Problem 2 is that the malicious robots seek to alter the outcome of the swarm algorithm instead of simply blocking communication.

Problem 3. Resilient Swarm Clustering: *Given a swarm of n robots that communicate locally and that are solving Problem 1, as well as one or more malicious robots that are attempting to hack the solution (by solving Problem 2), the swarm must identify (or otherwise neutralize) the malicious robots.*

2.3 Silhouette Coefficient Performance Metric

We use the silhouette coefficient [42] to evaluate clustering performance. This value requires, for all n nodes, the average distance to all nodes within the cluster, a_i, and the average distance to all nodes not in the cluster, b_i [40]. For each node, a silhouette value is calculated $s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$. A silhouette value is a number $-1 \leq s_i \leq 1$. Negative values indicate bad clustering and positive values indicate good clustering. The final silhouette coefficient summarizing the clustering accuracy for all nodes, is the maximum of the n silhouette values. If the whole swarm is classified as a single cluster, the silhouette coefficient is zero.

3 Swarm *k*-Means Algorithm

In Section 3.1 we introduce Distributed Swarm k-Means, a modification of k-Means designed to be run on by a robot swarm using local communication and locally determined robot-to-robot distances. In Section 3.2 we show how a malicious can attack Distributed Swarm k-Means by broadcasting incorrect data. In Section 3.3 we present Resilient Swarm k-Means, which uses a distributed IDRS to identify and block suspected malicious robots.

Discussion of swarm DBSCAN (including distributed algorithms, hacking, and hacking resiliency from IDRS) appears in Section 4.

3.1 Distributed Swarm *k*-Means

The distributed swarm k-Means algorithm is designed to be similar to the original k-Means algorithm, while working with (only) local communication and using a notion of distance defined along the communication graph.

Each of the k clusters are associated with a "root" robot that is analogous to a cluster "center" used in the original k-Means algorithm. At the beginning of the algorithm the k root robots can be chosen randomly (for unsupervised clustering) or selected by a user (for semi-supervised clustering).

The algorithm is iterative. At a high level, each iteration involves the following two different distributed processes:

1. Determine the cluster membership of all robots by calculating the graph distance of each robot to its nearest root robot (cluster center). Robots adopt the cluster label of the nearest root robot.
2. The new root robot of each cluster is selected to be closer to the center of all robots belonging to that cluster.

An example of the Distributed Swarm k-Means algorithm running appears in the main paper, while an example showing how the new root is chosen appears in Figure 1.

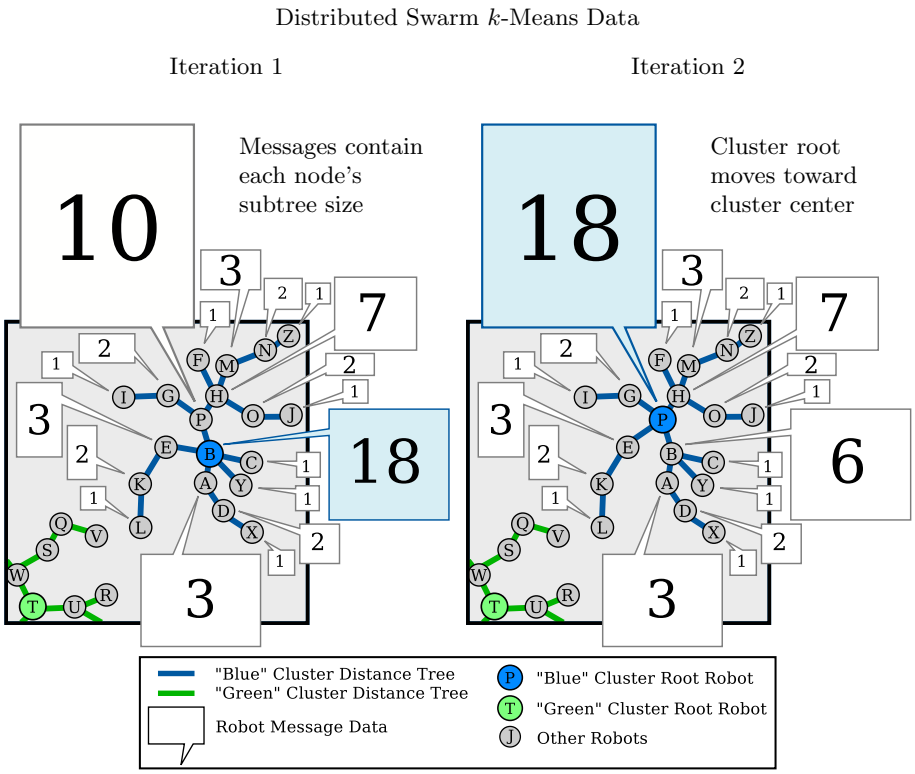


Fig. 1. Data passed in the Distribute Swarm k-Means Algorithm. We focus on the blue cluster in this figure. The shortest path forest is found between the set of root nodes and all other nodes (green and blue edges). Each robot calculates the number of nodes in the subtree of the shortest-path sub-tree of which it is the root by summing over the subtree sizes of its children and adding 1 to account for itself. Right: for the next iteration, the root status passes to the old root's child that has the largest subtree. In iteration robot P had a subtree of size 10 (compared to 3, 3, and 1 for robots E, A, and C, respectively; so, P becomes the new root. The algorithm terminates once any robot has been root more than a predetermined number of times.

The (step 1) distance calculation is accomplished by having the swarm create a shortest-spanning forest over all robots, where each tree in the forest is rooted at a particular cluster root. This is accomplished using a modification of distributed Dijkstra's algorithm—each robot chooses its parent to be the neighboring node (in the communication graph) through which it can reach a cluster root in the shortest amount of distance (through the communication graph). Each node's distance to a cluster root is calculated as the distance to its parent plus its parent's distance (through the communication graph) to its root. For brevity we use the language "the subtree of node i" as shorthand for "the subtree of the shortest spanning forest that descends from robot i," where node i may or may not be a cluster root.

In step 2, all nodes participate in a distributed calculation that provides each node with the size of its subtree as well as those of its children. One consequence of using (only) local communication is that the responsibility of being cluster root must pass from robot to robot (instead of jumping directly to the cluster's centroid as would be typical in standard k-Means). During each iteration, each of the k cluster roots compares the subtree sizes of its neighbors (all neighbors of cluster roots are children), and then selects the neighbor with the largest subtree to become that cluster's new root during the next iteration. The process of continually passing the cluster root to the best neighbor resembles a form of gradient assent, and can be seen in Figure1.

The gradient assent form of root relocation that we use is necessary in the decentralized communication case that we consider. However, a negative consequence of this distributed method is that root responsibility tends to settle into small cycles, where root status is passed back and forth between a small subset of nodes near local optima. This prevents convergence to a stationary clustering, and so we require a stopping criterion beyond the stabilization of roots at particular robots. To prevent cycling behavior we introduce the user parameter $z_{\max}$ which defines the maximum number of times any node may pass root responsibility to another. After a particular robot is root $z_{\max}+1$ times it retains root responsibility and the algorithm converges.

Pseudocode for the Distributed Swarm k-Means algorithm appears in Algorithm 1, with major subroutines appearing in Algorithms 2-5.

Pseudocode for the Distributed Swarm *k*-Means algorithm The algorithm is distributed such that Algorithm 1 is designed to run simultaneously on each robot in the swarm. The set of initial cluster centers $\mathbf{C}_{\text{init}}$ is provided (by a user or some other means such as random selection). Each robot initially knows its own ID number i and the value of $z_{\max}$. Values are initialized using the subroutine `initializeKmeans`() (line 1). Note that the subroutine `initializeKmeans`() itself is outlined in Algorithm 2. Each robot tracks the size of its subtree s_i and the subtree sizes of each neighbor j using an array entry s_j indexed by j. The array entry s_i is initialized to 1 (line 2) to reflect the fact that this robot is in its own subtree.

Algorithm 1 Distributed Swarm k-Means

Require: i /∗ ID of this robot (node) ∗/
Require: $\mathbf{C}_{\text{init}}$ /∗ Set containing IDs of initial cluster centers ∗/
Require: $z_{\max}$ /∗ Number of times a node can transfer root ∗/
1: `initializeKmeans`() /∗ initializes variables ∗/
2: $s_i \leftarrow 1$ /∗ this nodes subtree size ∗/
3: **loop**
4: **if** received new message M **then**
5: $j \leftarrow M.senderID$
6: **if** $M.t < t$ **or** $d_{ji} > d$) **then**
/∗ message is out of date or beyond comms radius ∗/
7: **continue**
8: **if** $M.t > t$ **then**
/∗ a new k-Means iteration has started ∗/
9: `resetDistanceTree`()
10: $t \leftarrow M.t$
11: `updateDistanceTree`(M)
12: $s_i \leftarrow 1 + \sum_{j \in \mathbf{N}_c} s_j$ /∗ calculate this node's subtree size ∗/
13: `populateMessageMetaData`($\tilde{M}$)
14: $\tilde{M}.s \leftarrow s_i$
15: **if** $d_{\text{root}} = 0$ **and** $z \leq z_{\max}$ **and** `epochTimer`() **then**
/∗ pass cluster root to child of root with largest subtree ∗/
16: $\tilde{M}.newRootID = \arg\max_{j \in \mathbf{N}_c} s_j$
17: `Broadcast`($\tilde{M}$)

The algorithm works in a distributed fashion that relies on local information exchange via messages. Each distributed iteration of the algorithm is associated with a unique iteration number t. This is used to ensure that graph distances reflect the current cluster roots. Any message that is received is checked for validity (lines 4-7), where messages are ignored if they are relevant to a previous iteration or if they have been sent from robots beyond the allowed communication radius d. Whenever it is discovered that the iteration number has been increased, then the distance calculation is reset and this robot switches to using the new iteration number (Lines 8-10).

The shortest spanning forest distance data is updated to reflect most current information from all neighbors using the subroutine `updateDistanceTree`() (line 11). The subroutine `updateDistanceTree`() is discussed later and appears in Algorithm 4. For now it is sufficient to understand that `updateDistanceTree`() performs two important tasks. First, runs the distributed Dijkstra's variation that updates the cost to root values (used to calculate graph distances, parent relationships, and the topology of the shortest spanning forest). Second, it detects if/when this robot has been promoted to be a cluster root—in which case this robot increases the value of t and restarts the distributed distance calculation for the new iteration.

Algorithm 2 `initializeKmeans`()

```
1: parent ← ∅                                   /* the parent of this node */
2: t ← 0                                        /* k-Means iteration */
3: d_root ← ∞                                   /* distance to root */
4: N_c ← ∅                                      /* set of this node's children */
5: z ← 0                                        /* times this robot has been cluster root */
6: if i ∈ C_init then
7:     d_root ← 0
8:     t ← 1
```

Algorithm 3 `resetDistanceTree`()

```
1: d_root ← ∞
2: parent ← ∅
3: N_c ← ∅
```

Algorithm 4 `updateDistanceTree`(M)

```
1: if d_ji + M.d_root < d_root then
       /* a shorter path to a cluster root has been found */
2:     parent ← j
3:     d_root ← d_ji + M.d_root
4: if i = M.parent then
       /* this node i is the parent of the sending node j */
5:     N_c ← N_c ∪ {j}
6:     s_j ← M.s
7: else if i ≠ M.parent then
       /* this node i is not the parent of the sending node j */
8:     N_c ← N_c \ {j}
9:     if parent = j then
           /* sending node j is the parent of this node i */
10:        if M.newRootID = i then
               /* this node j is the new cluster root */
11:            parent ← ∅
12:            t ← t + 1
13:            z ← z + 1
```

This robot updates the number of nodes contained in its subtree (continuing in Algorithm 1, line 11). Basic message data related to iteration, tree distances, etc. is populated using the subroutine `populateMessageMetaData`() (line 13). Details of `populateMessageMetaData`() appear in Algorithm 5. If this robot is the root ($d_{\text{root}} = 0$) and this robot has not exhausted its root passing threshold $z \leq z_{\max}$, then this robot chooses the new cluster center the child with the most descendants, and adds the child's ID to the outgoing message (lines 15-16). The outgoing message is broadcast on line 17.

Algorithm 5 `populateMessageMetaData`($\tilde{M}$)

1: $\tilde{M}.senderID \leftarrow i$
2: $\tilde{M}.t \leftarrow t$
3: $\tilde{M}.parent \leftarrow parent$
4: $\tilde{M}.newRootID = \emptyset$

Pseudocode for the `updateDistanceTree()` subroutine We now describe the pseudeocode of subroutine `updateDistanceTree`() which appears in Algorithm 4. This subroutine is responsible for running the distributed Dijkstra's variation that updates the cost to root values, as well as restarting the distributed distance calculation if/when this node is promoted to be a cluster root. Whenever a neighbor is found that provides a shorter path to reach the goal, then that neighbor is set as this node's the parent, and the distance to the goal is updated accordingly (lines 1-3).

If a message is received from a node that has become one of this nodes children, then this node updates the set of children that it tracks internally, and also records the size of that child's subtree (lines 4-6). If this node receives a message from a non-child, then it ensures that the parent does not appear in its child list (lines 7-8). If the messages is from this node's parent and this node has been promoted to cluster root, then it records that it now has no parent, increases the algorithm iteration t, and also increases its count of the number of times it has been root z (lines 9-13).

3.2 Hacking Distributed Swarm *k*-Means

If a malicious actor gains control of one or more robots, the now-malicious robots may work within the algorithm's procedures to disrupt the intended emergent behavior. This masquerade attack, conducted by an insider with limited knowledge of the algorithm, can have disastrous effects.

A disruption to the Distribute Swarm k-Means clustering algorithm could potentially be anything that alters the robots final cluster labels, classifies multiple or all the clusters as a single cluster, or does not allow the algorithm to finish. However, in this paper we focus on the case where a malicious actor(s) attempt to gain root status of a cluster(s) by falsely advertising large subtree sizes.

A depiction of the malicious attack is shown in Figure 2. To disrupt clustering, a malicious robot falsely reports that its branch is very large. If the malicious is an immediate neighbor of a cluster root, then the root erroneously believes that the malicious robot should become root and passes it root responsibility. If the the malicious robot is not a direct neighbor of a cluster root, then well intentioned parents, grandparents, etc. of the malicious robot unknowingly report subtrees that are also erroneously large. As a result, the responsibility of being root is transformed one step closer to the malicious robot each iteration. Eventually

a direct neighbor of the malicious robot becomes root, and then the malicious robot itself.

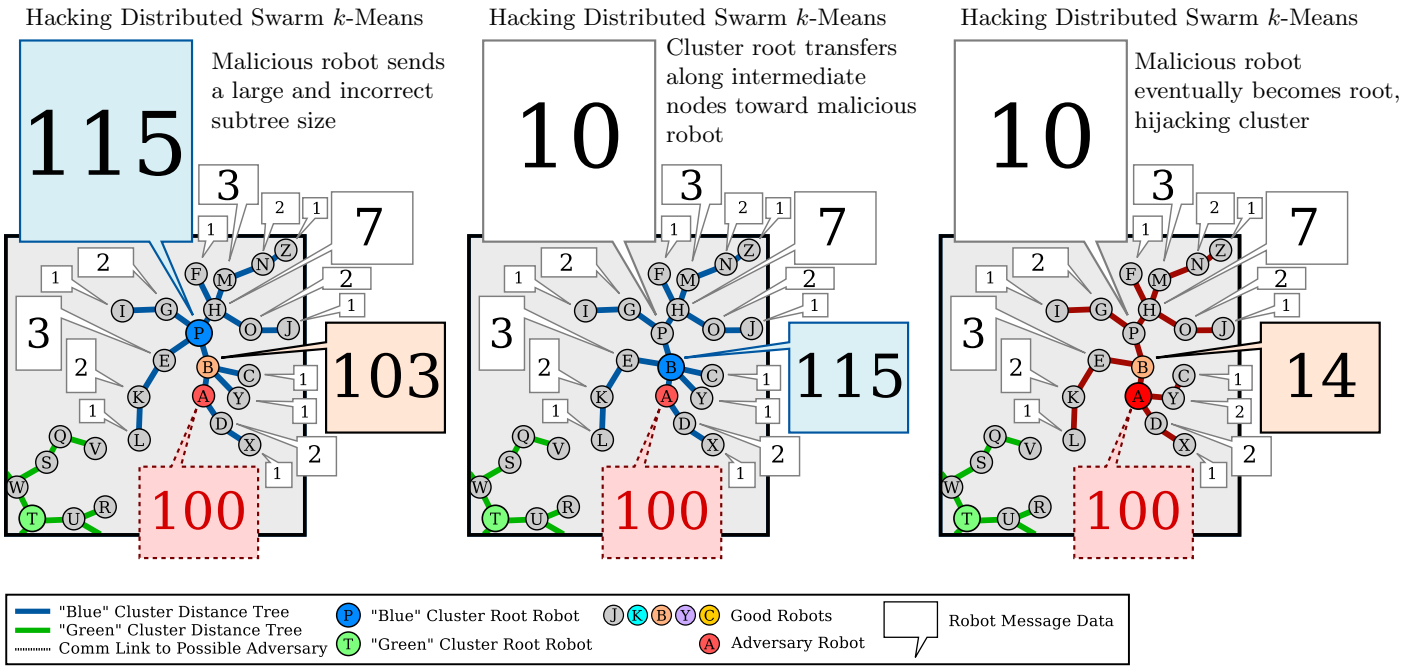


Fig. 2. A malicious attack in which a malicious robot becomes a cluster root by claiming inflated subtree sizes. Root status is transferred one step closer to the malicious robot each iteration. Eventually the malicious robot becomes a cluster root.

A malicious robot that gains root responsibility could potentially exploit such a position to cause harm to the swarm and/or the environment (since clustering may be used as a prepossessing step for, e.g., sub-team formation, task allocation, etc.). However, in this paper we shall focus on the damage that a malicious robot can cause to the clustering itself. After becoming a cluster root we have the malicious robot replace the root's messages with an empty broadcast. Lacking messages from the root, other robots belonging to the now-dead cluster assume the root has shifted away, and join one of the other $k-1$ remaining clusters instead. This removes one of the k clusters from the swarm, resulting in poor clustering performance. This is demonstrated in the Attack row of Table 1.

3.3 Resilient Distributed Swarm *k*-Means

The resilient swarm kmeans algorithm is conceptually similar to the distributed swarm Kmeans algorithm, but is designed to detect malicious robots, and then

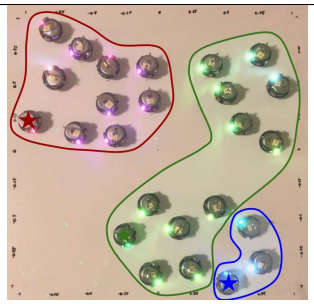
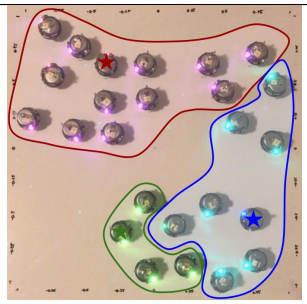
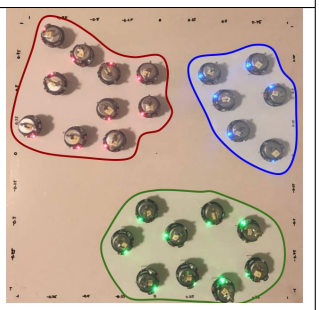
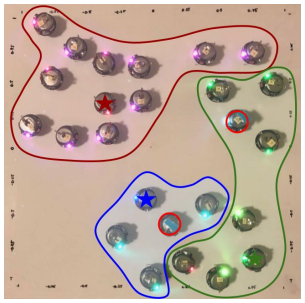
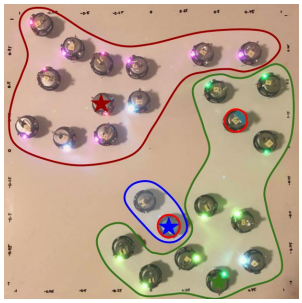
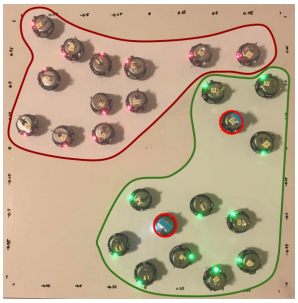
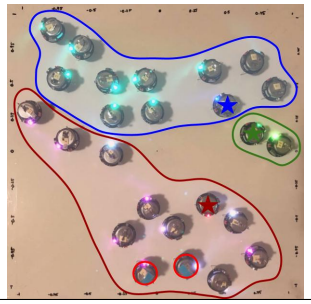
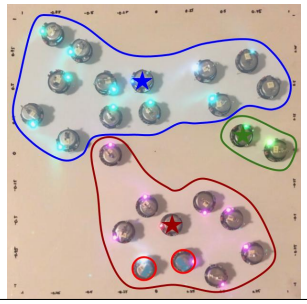
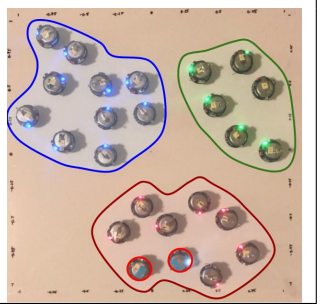
	After initialization	Midway	At end
Algorithm			
Attack			
IDRS			

Table 1. The k-Means algorithms in progress. Malicious robots in the Attack and IDRS algorithms are shown rimmed in red. Superimposed stars indicate the location of the root after initialization and midway through the algorithm. Each cluster has been encircled with a line of the cluster's color to better show the change in grouping over time.

remove their influence on the distributed algorithm. A High-level graphical description of the Distributed Swarm k-Means algorithm appears in Figure 3. The distributed IDRS works by having robots send the ID numbers of all nodes in their subtrees instead of only sending sub-tree size. If two robots claim to have the same robot as part of their sub-tree, then the other robots add both of them to a malicious actor list. While this does not completely eliminate the negative effects of bad actors, it does reduce the damage a single malicious robot can cause. All robots share their lists and ignore robots suspected of being malicious.

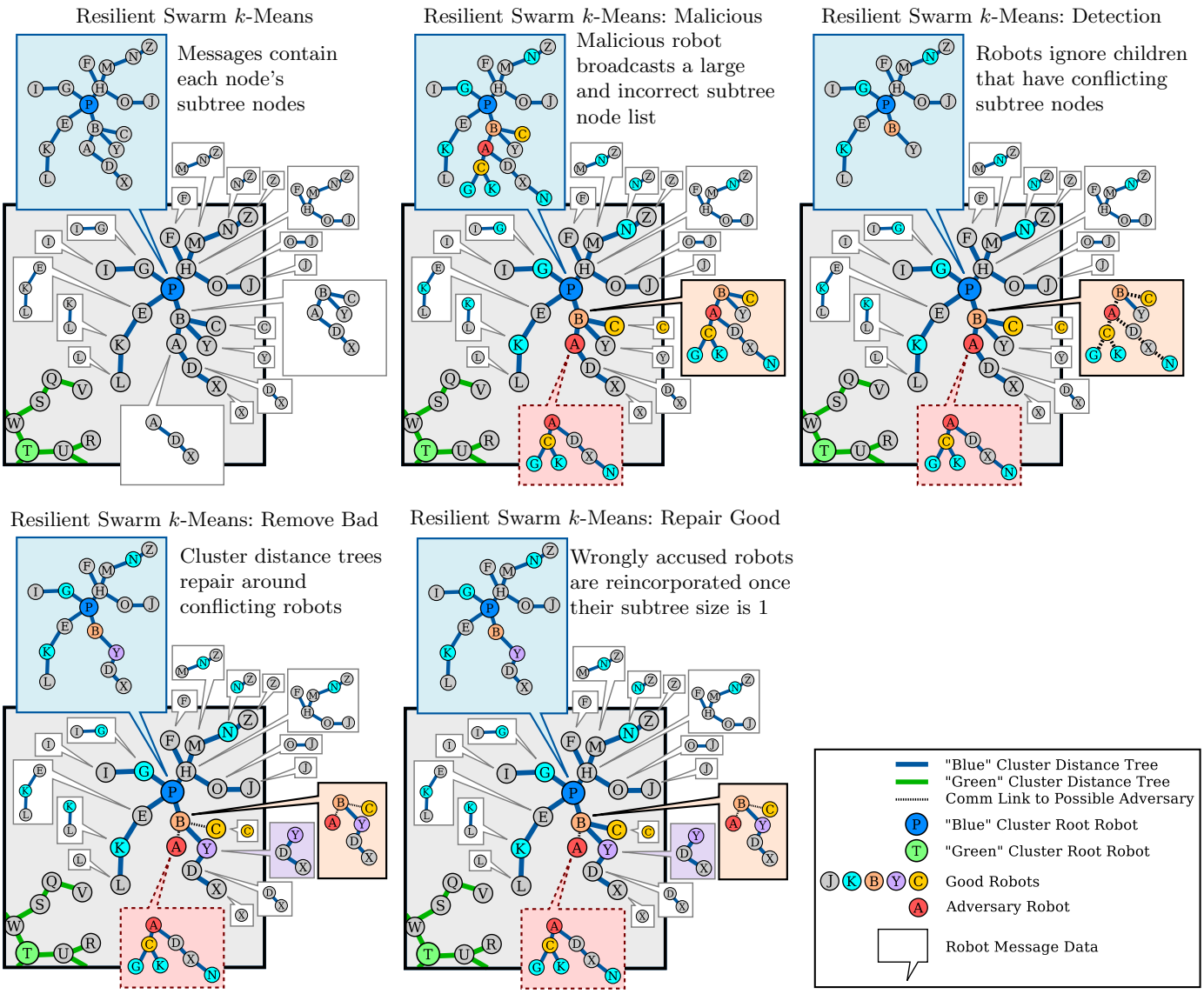


Fig. 3. Resiliant Swarm k-Means incoporates a distributed IDRS by having all nodes send the IDs of nodes in their subtrees (insead of only subtree size). Robots with confilciting descendent lists are suspected of being mallicios and ignored. The shortest path foreset then re=routes around malicious robots. Robots that were once considered adverasrial can be reinstaed once the subtree that they broadcast contains onely themself.

The shortest spanning forest eventually re-routes around all robots suspected of being malicious. This re-routing provides a means of detecting a subset of good robots that have been accidentally labeled as malicious—robots that broadcast subtrees containing only themselves are not malicious.

Algorithm 6 Resilient Swarm k-Means

Require: i /∗ ID of this robot (node) ∗/
Require: $\mathbf{C}_{\text{init}}$ /∗ Set containing IDs of initial cluster centers ∗/
Require: $z_{\max}$ /∗ Number of times a node can transfer root ∗/
`initializeKmeans`() /∗ initializes variables ∗/
$\mathbf{S}_i \leftarrow \{i\}$ /∗ this nodes subtree set ∗/
$\mathbf{B} \leftarrow \emptyset$ /∗ list of bad actors ∗/
loop
  if received new message M **then**
    $j \leftarrow M.senderID$
    if $M.t < t$ **or** $d_{ji} > d$ **or** $j \in \mathbf{B}$ **then**
      /∗ message is out of date or beyond comms radius ∗/
      /∗ or message is from a suspected malicious ∗/
      continue
    /∗ synchronize bad actor list ∗/
    $\mathbf{B} \leftarrow \mathbf{B} \cup M.\mathbf{B}$
    $\mathbf{B} \leftarrow$ `calculateGoodActors`()
    `blockBadActors`($\mathbf{B}$)
    if $M.t > t$ **then**
      /∗ a new k-Means iteration has started ∗/
      `resetDistanceTree`()
      `resetConflictDurations`($\mathbf{N}_c \setminus \mathbf{B}, \mathbf{N}_c$)
      $t \leftarrow M.t$
  `updateDistanceTreeB`(M)
  if `settlingTimeOver`() **then**
    $\mathbf{B} \leftarrow$ `calculateBadActors`()
    $\mathbf{B} \leftarrow$ `calculateGoodActors`()
    `blockBadActors`($\mathbf{B}$)
  $\mathbf{S}_i \leftarrow \{i\} + \bigcup_{j \in \mathbf{N}_c} \mathbf{S}_j$
  `populateMessageMetaData`($\tilde{M}$)
  $\tilde{M}.\mathbf{S} \leftarrow \mathbf{S}_i$
  $\tilde{M}.\mathbf{B} \leftarrow \mathbf{B}$
  if $d_{\text{root}} = 0$ **and** $z \leq z_{\max}$ **and** `epochTimer`() **then**
    /∗ pass cluster root to child of root with largest subtree ∗/
    $\tilde{M}.newRootID = \arg\max_{j \in \mathbf{N}_c} |\mathbf{S}_j|$
  `Broadcast`($\tilde{M}$)

We note that this form of IDRS requires messages to increase in size from $O(1)$ to $O(n)$. While there are efficient means of passing ID lists (bit arrays, for example), it may not be appropriate in all cases. We also note that it is not always possible to reinstate good robots accidentally listed as malicious (a robots subtree will only decrease in size if there is a second communication path through which its descendants can re-route).

Pseudocode for Resilient Swarm *k*-Means Pseudocode for the resilient swarm kmeans algorithm appears in Algorithm 6. Difference vs. the swarm

kmeans algorithm are highlighted in blue. For brevity our discussion will focus on these differences. The set containing this node's subtree nodes is initialized to contain only this robot (line 2), while the bad actor list is initialized to the empty set (line 3). Messages are also ignored if they come from robots on the bad actor list (line 7).

There is a high consequence to two robots on different branches of a shortest-path tree erroneously reporting the same robot as belonging to their respective sub-trees (since this may cause them to be labeled as bad actors). However, such condition can temporarily occur—even if no bad actors are involved—in the normal course of distributed Dijkstra's algorithm[3]. It can also happen due to a dropped messages. Each time a conflict is detected two robots are placed on the bad actor list—yet only one is actually a bad actor. Therefore, some mechanism must be added to prevent pre-convergence data from being used to add robot to the bad actor list, and another to rehabilitate the status of robots that have been incorrectly placed on the bad actor list.

Avoiding problems caused by start-up effects is accomplished by defining a "conflict duration", and then only adding robots to the bad actor list if they report conflicting data for longer than the conflict duration. Conflicts are defined as ordered tuple, a conflict (j, ℓ) means that j is suspected of being a malicious robot due to a conflict with ℓ while (ℓ, j) means that ℓ is suspected of being a malicious robot due to a conflict with j. The conflict duration of one node conflicting with another is incremented (line 18) inside the subroutine `calculateBadActors`() (this subroutine is itself is described in Algorithm 8). The conflict duration for all non-bad-actors are also reset when there is a change in the root node (line 14) using the subroutine `resetConflictDurations`$(\mathbf{N}_c \setminus \mathbf{B}, \mathbf{N}_c)$, which resets all timers for conflicts (j, ℓ) such that $j \in \mathbf{N}_c \setminus \mathbf{B}$ and $\ell \in \mathbf{N}_c \setminus j$.

Robots are removed from the bad actor list once the subtree set they report contains only themselves (line 19) inside the subroutine `calculateGoodActors`() (this subroutine is itself is described in Algorithm 8). This will often happen after the shortest path forest rewires to avoid a wrongly accused node.

The bad actor list is updated (lines 9 and 18), and then used to remove suspected bad actor nodes a parents and children (lines 11 and 20) using the subroutine `blockBadActors`$(\mathbf{B})$ (this subroutine is itself is described in Algorithm 10).

Pseudocode for `updateDistanceTreeB`(M) subroutine This node updates its subtree information (line 16) using the subroutine `updateDistanceTreeB`(M), which is similar to `updateDistanceTree`(M) used in the original (non-IDRS) version, except that subtree set membership is tracked instead of set size. This node calculates its subtree set as the set union of its children's subtrees plus the set containing itself (line 21). Both set membership and bad actor lists are sent to other robots (lines 23-24).

[3]For example if the optimal path passes through many more robots than some other suboptimal path with fewer robots but larger graph distance.

Algorithm 7 `updateDistanceTreeB`(M)

```
1: if d_ji + M.d_root < d_root then
       /* a shorter path to a cluster root has been found */
2:     parent ← j
3:     d_root ← d_ji + M.d_root
4: else if i = M.parent then
       /* this node i is the parent of the sending node j */
5:     N_c ← N_c ∪ {j}
6:     S_j ← M.S
7: else if i ≠ M.parent then
       /* this node i is not the parent of the sending node j */
8:     N_c ← N_c \ {j}
9:     S_j ← M.S
10:    if parent = j then
           /* sending node j is the parent of this node i */
11:        if M.newRootID = i then
               /* this node j is the new cluster root */
12:            parent ← ∅
13:            t ← t + 1
14:            z ← z + 1
15:        else
16:            d_root ← d_ji + M.d_root
```

Algorithm 8 `calculateBadActors`()

```
   /* check for bad actors */
1: for j ∈ N_c \ B do
2:     for ℓ ∈ (N_c \ B) \ j do
3:         if S_j ∩ S_ℓ ≠ ∅ then
4:             if conflictDurationExceeded(j, ℓ) then
5:                 B ← B ∪ {j}
6:             else
7:                 conflictDurationIncreased(j, ℓ)
8: return B
```

Pseudocode for other subroutines The calculation of bad actors is accomplished using the subroutine `calculateBadActors`() (Algorithm 8). This subroutine checks each ordered pair of nodes that are not currently on the black list, to see if the subtree sets they report are in conflict. If a conflict is detected and the conflict time has been exceeded then the first node of the pair is added to the blacklist, if a conflict is detected but the conflict time has not yet been exceeded then the conflict time is updated.

Removal of wrongly suspected nodes from the bad actor list is accomplished using the subroutine `calculateGoodActors`() (Algorithm 9). This subroutine checks each node to see if the subtree that it reports contains only itself. If so, then the node is removed from the bad actor list.

Algorithm 9 `calculateGoodActors()`

```
   /* check for wrongly accused nodes */
1: for j ∈ B do
2:     if S_j = {j} then
3:         B ← B \ {j}
4:         for ℓ ∈ N_c \ j do
5:             conflictDurationReset(j, ℓ)
6: return B
```

Algorithm 10 `blockBadActors(B)`

```
1: N_c ← N_c \ B
2: if parent ∈ B then
3:     parent ← ∅
4:     d_root ← ∞
```

The subroutine `blockBadActors`($\mathbf{B}$) (Algorithm 10) is used to remove the negative effects of suspected bad actors. It prevents nodes suspected of being malicious from being the parents or children of other nodes.

4 Swarm DBSCAN

This section focuses on versions of DBSCAN we have modified for clustering robot swarms into subsets based on proximity. The organizational structure or this section is similar to the previous section. In Section 4.1 we introduce Distributed Swarm DBSCAN, a modification of DBSCAN that can be run on a distributed swarm of robots that have (only) local communication and local access robot-to-robot distances, but not global communication nor global position data. In Section 4.2 we show how a malicious robot can disrupt Distributed Swarm DBSCAN by broadcasting incorrect data. In Section 4.3 we present Resilient Swarm DBSCAN, which uses a distributed IDRS to identify and block suspected malicious robots.

4.1 Distributed Swarm DBSCAN

The original (centralized) DBSCAN algorithm is well suited to the case that clusters lie along mutually disjoint lower dimensional manifolds embedded in a higher dimensional space. The (original) algorithm works by first creating a d-disc graph, such that a particular node's neighbor set $\mathbf{N}$ contains all nodes within distance d of that node. Nodes are defined as being 'internal' nodes if $|\mathbf{N}| \geq m$ neighbors, 'boundary' nodes if $m > |\mathbf{N}| \geq 1$, and 'outliers' nodes if $|\mathbf{N}| = 0$, where m is a user defined parameter. Neighboring internal nodes are defined as being in the same cluster, while boundary nodes choose membership in one of their neighbor's clusters. Outliers are not considered to be in any cluster.

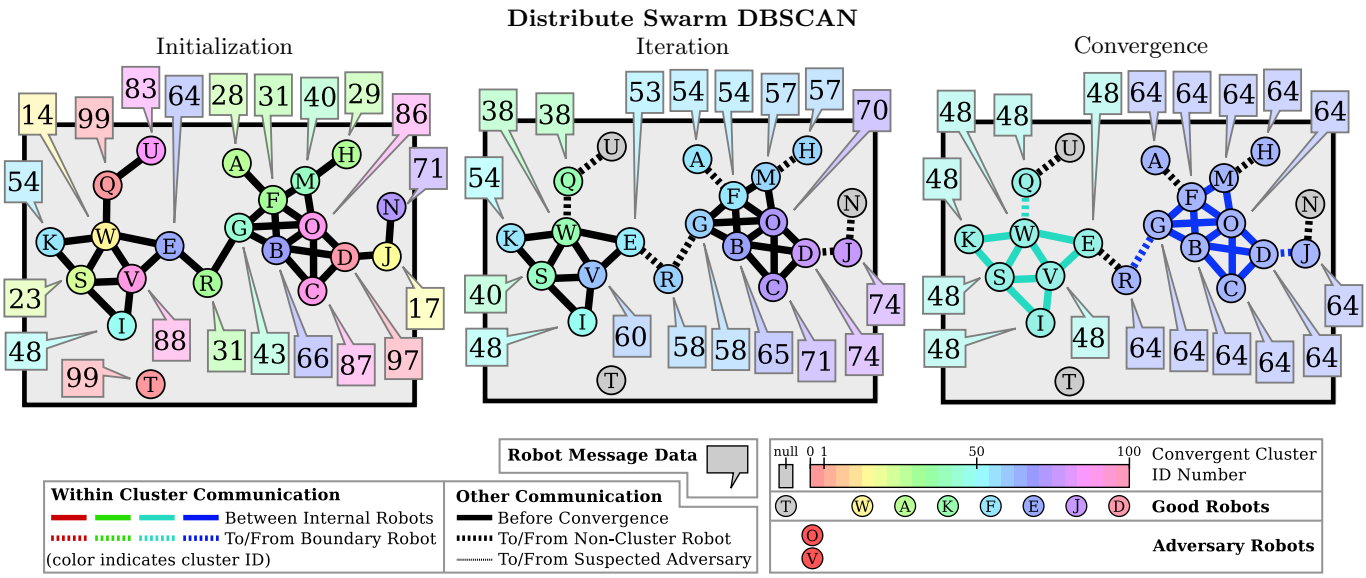


Fig. 4. Distributed Swarm DBSCAN. Nodes are defined as 'internal,' 'boundary,' or outlier depending on if they have, respectively, a number of neighbors $\geq m$, between 1 and m, or 0, respectively, where m is a user parameter. Internal nodes determine their cluster labels using a distributed consensus algorithm. Boundary nodes take the cluster labels of nearby internal nodes.

The fact that DBSCAN is naturally concerned with neighborhoods on a d-disc graph makes it well suited for operation on a robot swarm with local communication. The decentralized variant of DBSCAN that we explore—distributed swarm DBSCAN—simply requires that robots can actually communicate with all other robots closer than d. That is, we assume that for some physically defined $d_{\max}$, we are able to run distributed swarm DBSCAN for any $d \leq d_{\max}$. After nodes have determined if they are internal, boundary, or outlier, the internal nodes run a distributed consensus algorithm to agree on a cluster ID. This is accomplished by having internal nodes exchange and average randomly chosen integer values with their neighboring internal nodes until the internal nodes of each cluster have converged. An example of distributed swarm DBSCAN is depicted in Figure 4.

It is important to note that different clusters may accidentally converge to the same random value; however, the chances of this happening are small, especially if the sample space of random integers is many orders of magnitude larger than the size of the swarm (and the number of clusters). The chance of cluster ID collisions is relatively low for physical robot swarms (which typically contain 10s or 100s of robots) that use modern digital computers equipped with 16 or 32 bit unsigned integers (capable of representing 65536 and 4294967296 different values, respectively).

Algorithm 11 Distributed Swarm DBSCAN

Require: i /∗ ID of this robot (node) ∗/
Require: d /∗ distance param ∗/
Require: m /∗ min neighbors of internal nodes ∗/
1: `initDbscan`()
2: **loop**
3: **if** received new message M **then**
4: $j \leftarrow M.senderID$
5: **if** $d_{ji} > d$ **then**
6: **continue**
7: $\mathbf{N} \leftarrow \mathbf{N} \cup \{j\}$
8: **if** $M.status = internal$ **then**
9: $\mathbf{N}_{\text{int}} \leftarrow \mathbf{N}_{\text{int}} \cup \{j\}$
10: $C_j \leftarrow M.C$
11: `populateMessageMetaData`($\tilde{M}$)
12: **if** $|\mathbf{N}| \geq m$ **then**
/∗ this node i is an internal node ∗/
13: $C_i \leftarrow$ `integer` $\left(\left(C_i + \sum_{j \in \mathbf{N}_{\text{int}}} C_j\right) / (1 + |\mathbf{N}_{\text{int}}|)\right)$
14: $\tilde{M}.status \leftarrow internal$
15: **else if** $|\mathbf{N}| \geq 1$ **then**
/∗ this node i is a boundary node ∗/
16: $j \leftarrow$ closest member of $\mathbf{N}_{\text{int}}$
17: $C_i \leftarrow C_j$
18: $\tilde{M}.status \leftarrow bouandary$
19: **else**
/∗ this node i is an outlier node ∗/
20: $\tilde{M}.status \leftarrow outlier$
21: `Broadcast`($\tilde{M}$)

Pseudocode for Distributed Swarm DBSCAN Pseudocode for the distributed swarm DBSCAN algorithm appears in Algorithm 11. The algorithm requires as input the parameters d (neighborhood distance) and m (number of neighbors required to be considered an internal node). Each robot i assumed to have a unique ID i. Initialization of variables is performed (line 1) using the subroutine `initDbscan`(), itself appearing Algorithm 12. Within `initDbscan`() this node's parent is set to empty, its neighbor set $\mathbf{N}$ is set to empty, its internal neighbor set $\mathbf{N}_{\text{int}}$ (the set of neighbors that are also internal nodes) is set to empty, and its random integer C_i is drawn.

Back in Algorithm 11, distributed swarm DBSCAN relies heavily on message passing to transfer data throughout the swarm. Messages are received (lines 3-4) and then ignored if they are sent from robots beyond the neighborhood distance threshold (lines 5-6). Newly discovered neighbors are added to this node's neighbor set (line 7), internal neighbors are added to this node's internal neighbor set (lines 8-9), and the neighbors current (continually converging) cluster ID integer C_j is stored (line 10).

Algorithm 12 `initDbscan()`

1: $parent \leftarrow \emptyset$ /∗ the parent of this node ∗/
2: $\mathbf{N} \leftarrow \emptyset$ /∗ this node's neighbors ∗/
3: $\mathbf{N}_{\text{int}} \leftarrow \emptyset$ /∗ this node's internal neighbors ∗/
4: $C_i \leftarrow$ `randomInteger()` /∗ random initial cluster number ∗/

Algorithm 13 `populateMessageMetaDataB`($\tilde{M}$)

1: $\tilde{M}.senderID \leftarrow i$
2: $\tilde{M}.C \leftarrow C_i$

The second half of the algorithm focuses on sharing this robot's data with other robots (lines 12-21). Message data (this robot's ID i and current cluster ID integer C_i) is populated using the subroutine `populateMessageMetaDataB()` (line 10), this subroutine is detailed in Algorithm 13. The message is populated with this node's current internal, boundary, or outlier status (lines 14, 18, and 20). If this node is an internal node it calculates the new value of its converging integer (line 13), and if it is a boundary node then it randomly selects the cluster of its closest neighbor to join (line 16).

4.2 Hacking Distributed Swarm DBSCAN

Unlike the k-Means Algorithm, the proposed distributed swarm DBSCAN algorithm has no centralization that a malicious robot can use to its advantage.

In this paper we explore a DBSCAN attack in which multiple malicious robots, dispersed throughout the swarm, cause multiple clusters to converge to the same value. To disrupt clustering, a malicious robot classifies itself as an internal robot. Then, rather than averaging its cluster label with those of neighboring core robots, the malicious robot repeatedly broadcasts its own constant cluster label and forces the neighboring core robots to converge to that label. The final cluster label is the original cluster label of the malicious robots.

If two or more malicious robots are located in different clusters and all malicious robots share the same original constant cluster label, then each cluster containing a malicious robot converges to the same final cluster label. In this manner, multiple clusters reports the same cluster label and are incorrectly grouped into a single cluster. A graphical depiction of this attack appears in Figure 5.

4.3 Resilient Distributed Swarm DBSCAN

The resilient swarm DBSCAN algorithm is depicted in Figure 6. The main difference between this version and Distributed Swarm DBSCAN is that nodes track and share sets of suspected bad actors $\mathbf{B}$. The effects of bad actors are reduced by removing suspected bad actors from neighbor lists so that they cannot influence the convergent cluster ID integers or the internal, boundary, or outlier

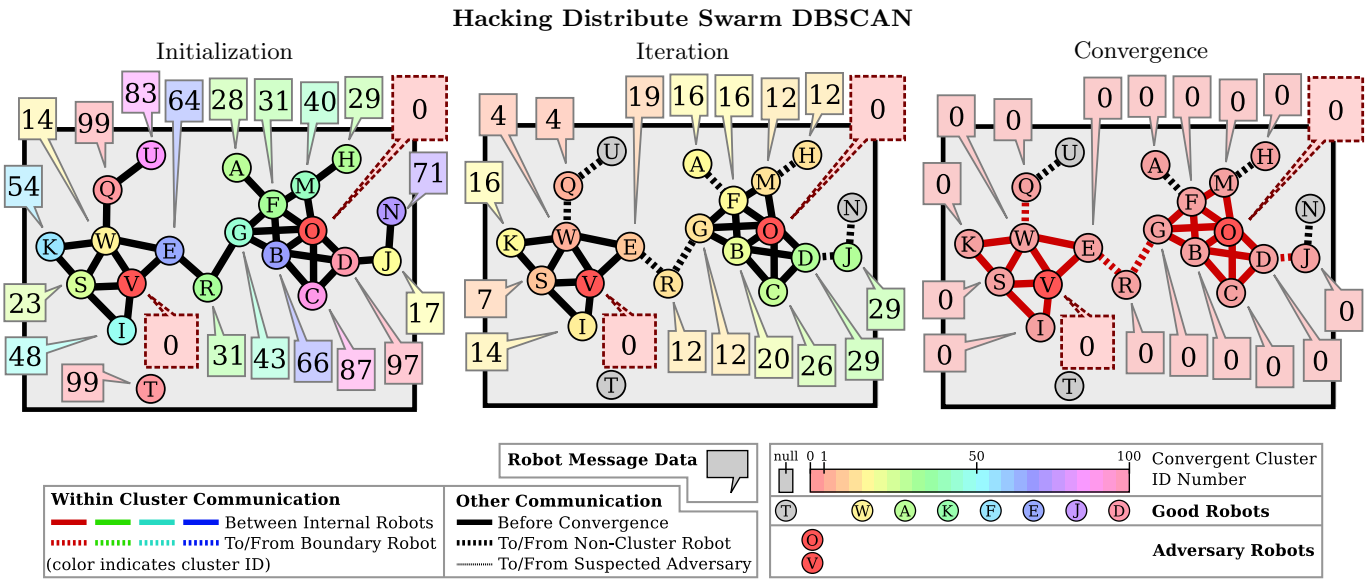


Fig. 5. An malicious attack on Distributed Swarm DBSCAN in which multiple malicious robots are located in different parts of the swarm. By sending the same values over and over, both clusters containing the malicious robots converge to the same cluster label.

status of other nodes. Bad actors are defined as nodes that have not updated their convergent values in more than $y_{\max}$ communications, where $y_{\max}$ is a user defined parameter.

This method of detection has been chosen primarily to enable us to explore the resilience of distributed swarm DBSCAN variants in the event that bad actors can be detected. The use of this particular detection method assumes that malicious robots are naive and simply sends the same bad value over and over. It is important to note that this method cannot detect malicious robots that send random numbers or change their value very slowly. However, in the latter cases it is easy to imagine similar but more involved forms of value analysis that robots could potentially be used to detect suspicious (non)converging integer values inside the subroutines `recordBadActorStatistics`() and `calculateBadActorsB`(). The simple method used here is useful because it enables us to compare how detecting malicious robots verses not detecting malicious robots affects algorithmic performance.

If the cluster label of a neighboring core robot remains constant over a period of time, the IDRS adds the suspected malicious ID to the malicious robot list—revoking its participation privileges in the swarm. Like the k-Means IDRS, all broadcasts from blacklisted robots are ignored. Therefore, the unchanging cluster label can no longer affect the final cluster label. To undo any effect the malicious robot may have already caused, the robot re-initializes its cluster label to a randomly generated value, then rejoins the distributed process of converging to a cluster label.

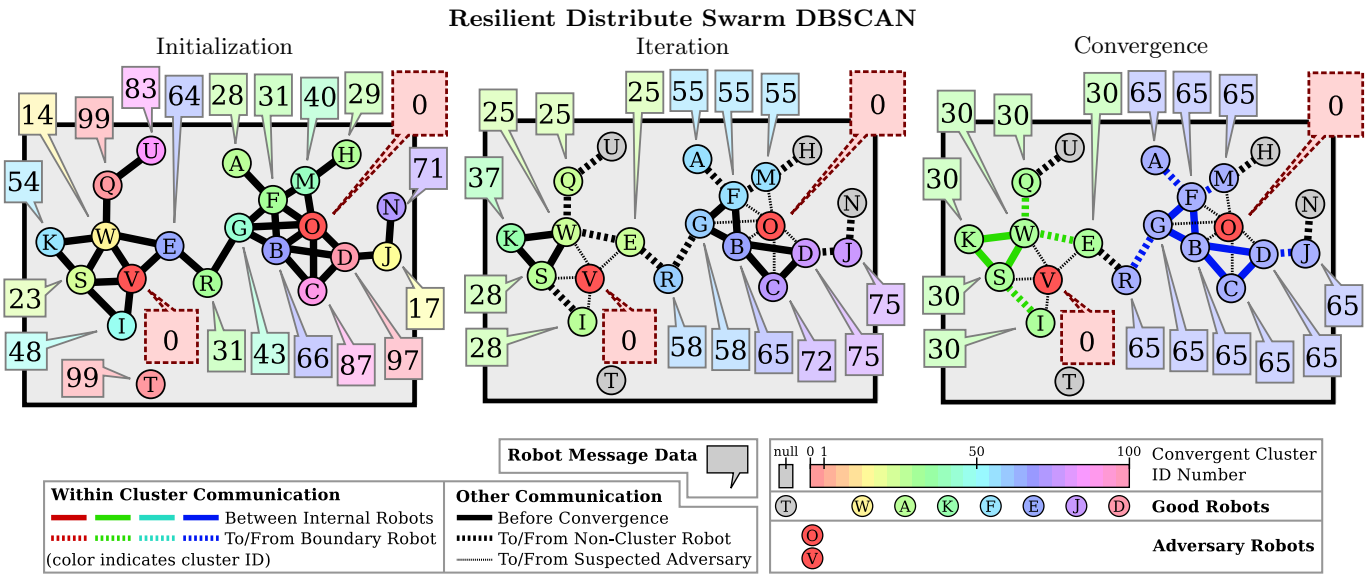


Fig. 6. Resilian Swarm DBSCAN incoporates an IDRS by ignoring robots that never change thier data (during the distributed convergence).

However, unlike the k-Means IDRS, the DBSCAN IDRS does not share the suspected malicious robot list throughout the swarm. Instead, each robot maintains its own separate list. To isolate a malicious robot from the swarm, all neighboring robots must detect it separately (by observing its unchanging cluster value). A robot that has been incorrectly identified as malicious by one robot will still contribute to the swarms behavior through its other neighbors.

Pseudocode for Resilient Swarm DBSCAN Pseudocode for the resilient swarm DBSCAN appears in Algorithm 14. Difference vs. the distributed swarm DBSCAN algorithm are highlighted in blue. The main differences are the calculation of bad actors based on message data (lines 11 and 13), the rejection of messages from suspected bad actors (lines 7-8), and blocking of bad actors (14) by removing them from neighbor lists via the subroutine `blockBadActors`($\mathbf{B}$).

Acknowledgments This work was partially supported by a joint Northrop Grumman and UMD Seedling Grant and by ONR grant N000142012712.

References

1. Alam, S., Dobbie, G., Koh, Y.S., Riddle, P., Rehman, S.U.: Research on particle swarm optimization based clustering: a systematic review of literature and techniques. Swarm and Evolutionary Computation 17, 1–13 (2014)
2. Anderson, B.J., Gross, D.S., Musicant, D.R., Ritz, A.M., Smith, T.G., Steinberg, L.E.: Adapting k-medians to generate normalized cluster centers. In: Proceedings of the 2006 SIAM International Conference on Data Mining. pp. 165–175. SIAM (2006)

Algorithm 14 Resilient Swarm DBSCAN

Require: i /* ID of this robot (node) */
Require: d /* distance param */
Require: m /* min neighbors of internal nodes */
Require: $y_{\max}$ /* max identical data sends allowed */
`initDbscan`()
$\mathbf{B} \leftarrow \emptyset$ /* list of bad actors */
$\mathbf{D}^{\text{same}} \leftarrow \{0, \ldots, 0\}$ /* counts neighbor's duplicate sends */
loop
  if received new message M **then**
    $j \leftarrow M.senderID$
    if $d_{ji} > d$ **or** $j \in \mathbf{B}$ **then**
      /* message is out of neighborhood radius */
      /* or message is from a suspected malicious */
      continue
    if $M.status = internal$ **then**
      $\mathbf{N}_{\text{int}} \leftarrow \mathbf{N}_{\text{int}} \cup \{j\}$
      `recordBadActorStatistics`(M)
    $C_j \leftarrow M.C$
  $\mathbf{B} \leftarrow$ `calculateBadActorsB`()
  `blockBadActors`$(\mathbf{B})$
  `populateMessageMetaData`$(\tilde{M})$
  if $|\mathbf{N}| \geq m$ **then**
    /* this node i is an internal node */
    $C_i \leftarrow$ `integer` $\left(\left(C_i + \sum_{j \in \mathbf{N}_{\text{int}}} C_j\right) / (1 + |\mathbf{N}_{\text{int}}|)\right)$
    $\tilde{M}.status \leftarrow internal$
  else if $|\mathbf{N}| \geq 1$ **then**
    /* this node i is a boundary node */
    if $\mathbf{N}_{\text{int}} \neq \emptyset$ **then**
      $j \leftarrow$ closest member of $\mathbf{N}_{\text{int}}$
    $C_i \leftarrow C_j$
    $\tilde{M}.status \leftarrow bouandary$
  else
    /* this node i is an outlier node */
    $\tilde{M}.status \leftarrow outlier$
  `Broadcast`$(\tilde{M})$

Algorithm 15 `recordBadActorStatistics`(M)

if $C_j \neq M.C$ **then**
  $\mathbf{D}_{\mathbf{j}}^{\text{same}} \leftarrow 0$
else if $C_j \neq C_i$ **then**
  $\mathbf{D}_{\mathbf{j}}^{\text{same}} \leftarrow \mathbf{D}_{\mathbf{j}}^{\text{same}} + 1$

3. Anikin, I.V., Gazimov, R.M.: Privacy preserving dbscan clustering algorithm for vertically partitioned data in distributed systems. In: 2017 International Siberian

Algorithm 16 `calculateBadActorsB()`

/∗ check for bad actors ∗/
1: **for** $j \in \mathbf{N}_{\text{int}}$ **do**
2: **if** $\mathbf{D}_{\mathbf{j}}^{\text{same}} > y_{\max}$ **then**
3: $\mathbf{B} \leftarrow \mathbf{B} \cup \{j\}$
4: **return B**

Algorithm 17 `blockBadActors(B)`

1: $\mathbf{N} \leftarrow \mathbf{N} \setminus \mathbf{B}$
2: $\mathbf{N}_{\text{int}} \leftarrow \mathbf{N}_{\text{int}} \setminus \mathbf{B}$

Conference on Control and Communications (SIBCON). pp. 1–4. IEEE (2017)
4. Balcan, M.F., Ehrlich, S., Liang, Y.: Distributed k-means and k-median clustering on general topologies. arXiv preprint arXiv:1306.0604 (2013)
5. Bello-Orgaz, G., Menéndez, H.D., Camacho, D.: Adaptive k-means algorithm for overlapped graph clustering. International journal of neural systems 22(05), 1250018 (2012)
6. Ben Salem, M.: Towards effective masquerade attack detection (2012), https://academiccommons.columbia.edu/doi/10.7916/D8J96DBT
7. Benkoczi, R., Bhattacharya, B., Chrobak, M., Larmore, L.L., Rytter, W.: Faster algorithms for k-medians in trees. In: International Symposium on Mathematical Foundations of Computer Science. pp. 218–227. Springer (2003)
8. Bu, Z., Li, H.J., Zhang, C., Cao, J., Li, A., Shi, Y.: Graph k-means based on leader identification, dynamic game, and opinion dynamics. IEEE Transactions on Knowledge and Data Engineering 32(7), 1348–1361 (2019)
9. Cordova, I., Moh, T.S.: Dbscan on resilient distributed datasets. In: 2015 International Conference on High Performance Computing & Simulation (HPCS). pp. 531–540. IEEE (2015)
10. Dai, B.R., Lin, I.C.: Efficient map/reduce-based dbscan algorithm with optimized data partition. In: 2012 IEEE Fifth international conference on cloud computing. pp. 59–66. IEEE (2012)
11. Dasgupta, S., Frost, N., Moshkovitz, M., Rashtchian, C.: Explainable k-means and k-medians clustering. In: Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria. pp. 12–18 (2020)
12. Datta, S., Giannella, C., Kargupta, H.: Approximate distributed k-means clustering over a peer-to-peer network. IEEE Transactions on Knowledge and Data Engineering 21(10), 1372–1388 (2009)
13. Dolev, S., Lahiani, L., Yung, M.: Secret swarm unit: Reactive k-secret sharing. Ad Hoc Networks 10(7), 1291 – 1305 (2012), http://www.sciencedirect.com/science/article/pii/S1570870512000613
14. Galluccio, L., Michel, O., Comon, P., Hero III, A.O.: Graph based k-means clustering. Signal Processing 92(9), 1970–1984 (2012)
15. Gandhe, M., Otte, M.: Intrusion detection and response system in swarm robotics clustering algorithms. In: International Symposium on Multi-Robot and Multi-Agent Systems (MRS), poster track (nonarchival). Cambridge, UK (2021)
16. Gil, S., Kumar, S., Mazumder, M., Katabi, D., Rus, D.: Guaranteeing spoof-resilient multi-robot networks. Autonomous Robots 41 (08 2017)

17. Götz, M., Bodenstein, C., Riedel, M.: Hpdbscan: highly parallel dbscan. In: Proceedings of the workshop on machine learning in high-performance computing environments. pp. 1–10 (2015)
18. Handl, J., Meyer, B.: Ant-based and swarm-based clustering. Swarm Intelligence 1(2), 95–113 (2007)
19. Higgins, F., Tomlinson, A., Martin, K.: Threats to the swarm: Security considerations for swarm robotics. International Journal on Advances in Security 2 (2009)
20. Jagannathan, G., Pillaipakkamnatt, K., Wright, R.N.: A new privacy-preserving distributed k-clustering algorithm. In: Proceedings of the 2006 SIAM international conference on data mining. pp. 494–498. SIAM (2006)
21. Jagannathan, G., Wright, R.N.: Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 593–599 (2005)
22. Jouini, M., Rabai, L.B.A., Aissa, A.B.: Classification of security threats in information systems. Procedia Computer Science 32, 489 – 496 (2014), http://www.sciencedirect.com/science/article/pii/S1877050914006528
23. Kolias, C., Kambourakis, G., Maragoudakis, M.: Swarm intelligence in intrusion detection: A survey. Computers & Security 30(8), 625 – 642 (2011), http://www.sciencedirect.com/science/article/pii/S016740481100109X
24. Laing, T., Martin, K., Ng, S., Tomlinson, A.: Security in Swarm Robotics, pp. 42–66. IGI Global (Dec 2015)
25. Liang, Y., Balcan, M.F., Kanchanapally, V.: Distributed pca and k-means clustering. In: The Big Learning Workshop at NIPS. vol. 2013. Citeseer (2013)
26. Ling, R.F.: On the theory and construction of k-clusters. The Computer Journal 15(4), 326–332 (01 1972), https://doi.org/10.1093/comjnl/15.4.326
27. Ling, S., Yunfeng, Q.: Optimization of the distributed k-means clustering algorithm based on set pair analysis. In: 2015 8th International Congress on Image and Signal Processing (CISP). pp. 1593–1598. IEEE (2015)
28. Liu, J., Huang, J.Z., Luo, J., Xiong, L.: Privacy preserving distributed dbscan clustering. In: Proceedings of the 2012 Joint EDBT/ICDT Workshops. pp. 177–185 (2012)
29. Liu, Q., Fu, W., Qin, J., Zheng, W.X., Gao, H.: Distributed k-means algorithm for sensor networks based on multi-agent consensus theory. In: 2016 IEEE International Conference on Industrial Technology (ICIT). pp. 2114–2119. IEEE (2016)
30. Liu, S., Dou, Z.T., Li, F., Huang, Y.L.: A new ant colony clustering algorithm based on dbscan. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826). vol. 3, pp. 1491–1496 vol.3 (2004)
31. Lumer, E.D., Faieta, B.: Diversity and adaptation in populations of clustering ants. In: Proceedings of the third international conference on Simulation of adaptive behavior: from animals to animats 3: from animals to animats 3. pp. 501–508 (1994)
32. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. vol. 1, pp. 281–297. Oakland, CA, USA (1967)
33. Mao, Y., Xu, Z., Li, X., Ping, P.: An optimal distributed k-means clustering algorithm based on cloudstack. In: 2015 IEEE International Conference on Information and Automation. pp. 3149–3156. IEEE (2015)
34. McCune, R., Madey, G.: Decentralized k-means clustering with manet swarms. In: Proceedings of the 2014 Symposium on Agent Directed Simulation. pp. 1–8 (2014)

35. Merk, A., Cal, P., Woźniak, M.: Distributed dbscan algorithm–concept and experimental evaluation. In: International Conference on Computer Recognition Systems. pp. 472–480. Springer (2017)
36. Nie, F., Zhu, W., Li, X.: Unsupervised large graph embedding based on balanced and hierarchical k-means. IEEE Transactions on Knowledge and Data Engineering (2020)
37. Oliva, G., Setola, R., Hadjicostis, C.N.: Distributed k-means algorithm. arXiv preprint arXiv:1312.4176 (2013)
38. Patel, S., Garasia, S., Jinwala, D.: An efficient approach for privacy preserving distributed k-means clustering based on shamir's secret sharing scheme. In: Dimitrakos, T., Moona, R., Patel, D., McKnight, D.H. (eds.) Trust Management VI. pp. 129–141. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
39. Patel, S., Patel, V., Jinwala, D.: Privacy preserving distributed k-means clustering in malicious model using zero knowledge proof. In: International Conference on Distributed Computing and Internet Technology. pp. 420–431. Springer (2013)
40. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research 12, 2825–2830 (2011)
41. Qin, J., Fu, W., Gao, H., Zheng, W.X.: Distributed k-means algorithm and fuzzy c-means algorithm for sensor networks based on multiagent consensus theory. IEEE transactions on cybernetics 47(3), 772–783 (2016)
42. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics 20, 53–65 (1987)
43. Sasikumar, P., Khara, S.: K-means clustering in wireless sensor networks. In: 2012 Fourth international conference on computational intelligence and communication networks. pp. 140–144. IEEE (2012)
44. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response system. International Journal of Information and Computer Security 1 (01 2007)
45. Thompson, R.B., Thulasiraman, P.: Confidential and authenticated communications in a large fixed-wing uav swarm. In: 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA). pp. 375–382 (2016)
46. Thrun, M.C., Ultsch, A.: Swarm intelligence for self-organized clustering. Artificial Intelligence 290, 103237 (2021)
47. Weiwei, N., Jieping, L., Zhihui, S.: An effective distributed k-means clustering algorithm based on the pretreatment of vectors' inner-product. Journal of Computer Research and Development 42(9), 1493 (2005)
48. Whelan, C., Harrell, G., Wang, J.: Understanding the k-medians problem. In: Proceedings of the International Conference on Scientific Computing (CSC). p. 219. The Steering Committee of The World Congress in Computer Science, Computer (2015)
49. Wolf, S., Cooley, R., Borowczak, M.: Adversarial impacts on autonomous decentralized lightweight swarms. In: 2019 IEEE International Smart Cities Conference (ISC2). pp. 160–166 (2019)
50. Yang, K., Gao, Y., Ma, R., Chen, L., Wu, S., Chen, G.: Dbscan-ms: Distributed density-based clustering in metric spaces. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 1346–1357. IEEE (2019)
51. Zhou, J., Zhang, Y., Jiang, Y., Chen, C.P., Chen, L.: A distributed k-means clustering algorithm in wireless sensor networks. In: 2015 International Conference

on Informative and Cybernetics for Computational Social Systems (ICCSS). pp. 26–30. IEEE (2015)