ACE: An Emergent Algorithm for Highly Uniform Cluster Formation

Haowen Chan and Adrian Perrig

Carnegie Mellon University
Pittsburgh PA 15213, U.S.A
{haowenchan, perrig}@cmu.edu

Abstract. The efficient subdivision of a sensor network into uniform, mostly non-overlapping *clusters* of physically close nodes is an important building block in the design of efficient upper layer network functions such as routing, broadcast, data aggregation, and query processing.
We present ACE, an algorithm that results in highly uniform cluster formation that can achieve a packing efficiency close to hexagonal close-packing. By using the self-organizing properties of three rounds of feedback between nodes, the algorithm induces the emergent formation of clusters that are an efficient cover of the network, with significantly less overlap than the clusters formed by existing algorithms. The algorithm is scale-independent — it completes in time proportional to the deployment density of the nodes regardless of the overall number of nodes in the network. ACE requires no knowledge of geographic location and requires only a small constant amount of communications overhead.

1 Introduction

Large-scale *distributed sensor networks* are becoming increasingly useful in a variety of applications such as emergency response, real-time traffic monitoring, critical infrastructure surveillance, pollution monitoring, building safety monitoring, and battlefield operations. Such networks typically consist of hundreds to tens of thousands of low cost *sensor nodes*, deployed via individual installation or random scattering. The nodes are usually highly power-constrained and have limited computation and memory resources. They typically utilize intermittent wireless communication. The sensor network is usually organized around one or more *base stations* which connect the sensor network to control and processing workstations or to an external communications network.

Clustering is a fundamental mechanism to design scalable sensor network protocols. A clustering algorithm splits the network into disjoint sets of nodes each centering around a chosen cluster-head. A good clustering imposes a regular, high-level structure on the network. It is easier to design efficient protocols on this high-level structure than

This research was supported in part by the Center for Computer and Communications Security at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and by gifts from Bosch and Intel Corporation. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, Carnegie Mellon University, Intel, or the U.S. Government or any of its agencies.

at the level of the individual nodes. Many efficient protocols rely on having a network partitioned into clusters of uniform size. Some examples of these protocols include routing protocols [14, 23], protocols for reliable broadcast [19, 20], data aggregation [10, 26], and query processing [6]. We further discuss clustering in Section 3.

Conventional algorithms that use centralized control and global properties of the sensor network have inherent difficulties in the properties of scalability and robustness, which are two important design goals for protocols in large-scale sensor networks. Centralized, top-down algorithms often need to operate with knowledge of the conditions and variables at every point of the network. In a very large network, the network traffic and time delay induced by the collection of this large amount of data may be undesirable. Finally, since some specific nodes, commands or data are usually of higher importance in a centralized protocol, an error in transmission or a failure of a critical node could potentially cause a serious protocol failure.

As an alternative to centralized algorithms, *localized algorithms* reduce the amount of central coordination necessary and only require each node to interact with its local neighbors [6]. While sometimes harder to design, these algorithms do not have the limitations of centralized algorithms and are often highly scalable, fast and efficient.

A class of localized algorithms that are particularly promising are *emergent algorithms*. Emergent algorithms have the additional characteristic that the individual agents (i.e., the sensor nodes in the case of distributed sensor networks) only encode simple local behaviors and do not explicitly coordinate on a global scale. Through repeated interaction and feedback at the individual level, global properties *emerge* in the system as a whole. Emergent behaviors are being studied extensively in biological, physical and social systems — such systems are often collectively termed *complex adaptive systems*. Examples include ant colonies, ecosystems, and stock markets. It is possible that emergent algorithms have the potential to be more flexible than non-emergent localized algorithms, which are constrained by the fact that a complex global property may be difficult to directly encode in a program that can act only upon local information.

In this paper, we provide an introduction to the definitions and motivations of localized and emergent algorithms. To demonstrate the potential of emergent algorithms in sensor networks, we present a new emergent protocol for node clustering called ACE (for Algorithm for Cluster Establishment). ACE has high cluster packing efficiency approaching that of hexagonal close-packing, and only incurs a small constant amount of communications overhead. ACE is scale-independent (it completes in constant time regardless of the size of the network) and operates without needing geographic knowledge of node positions or any kind of distance or direction estimation between nodes.

2 Localized protocols and emergent protocols

In this section we define localized and emergent protocols, and discuss the particular benefits and trade offs of using these protocols in sensor networks.

2.1 Localized protocols

Estrin et al. [6] offer a broad definition of a localized protocol:

Definition 1. *A **localized protocol** for a sensor network is a protocol in which each sensor node only communicates with a small set of other sensor nodes within close proximity in order to achieve a desired global objective.*

In this paper, we use a narrower definition of localized algorithms that better conveys the intuition of localized algorithms being free from centralized control:

Definition 2. *A **strictly localized protocol** for a sensor network is a localized protocol in which all information processed by a node is either: (a) local in nature (i.e. they are properties of the node's neighbors or itself); or (b) global in nature (i.e. they are properties of the sensor network as a whole), but obtainable immediately (in short constant time) by querying only the node's neighbors or itself.*

This narrower definition captures the notion that in a good localized protocol, each node should be capable of independent simultaneous operation in the protocol at any period. For example, consider a protocol that involves building a spanning tree in time proportional to the diameter of the network by doing a distributed breadth-first search involving only local communication (e.g. the Bannerjee and Khuller clustering algorithm [3]). Such a protocol would be a localized protocol by the first definition but not a *strictly* localized protocol by the second definition since a spanning tree is a global data structure and the entire network must be traversed before it can be computed.

In this paper, when we mention "localized protocols" or "localized algorithms", we will be referring to *strictly* localized protocols and algorithms.

Localized protocols have the following benefits:

- **Scalability.** Localized protocols can enable nodes to act independently and simultaneously in various parts of the network. Hence, localized protocols often exhibit better scalability in large networks than centrally controlled protocols, which may have to wait for information to propagate across the network.
- **Robustness.** When information use is purely local and no centralized control infrastructure is needed, the chances for protocol failure due to transmission errors and node failure are reduced. It is also more likely for performance to degrade gracefully under communication error rather than simply fail or end up in an erroneous state. This is because if all information is local, then the impact of any datum of information is most likely also locally limited. For example, if no critical control messages need to be routed across the entire network in a localized algorithm, then if a node fails then it will most likely induce a failure of the protocol at most only within its own vicinity.

2.2 Emergent protocols

In this paper, we make use of the definition of an emergent algorithm as outlined by Fisher and Lipson [7]:

Definition 3. *An **emergent algorithm** is any computation that achieves formally or stochastically predictable global effects, by communicating directly with only a bounded number of immediate neighbors and without the use of central control or global visibility.*

Hence, an *emergent protocol* for a sensor network is a localized protocol in which the desired global property is neither explicitly encoded in the protocol nor organized by a central authority, but emerges as a result of repeated local interaction and feedback between the nodes.

One of the main distinguishing characteristics of emergent protocols over other localized protocols is the existence of *feedback* during protocol operation. Feedback occurs when some node A affects some node B, which then directly or indirectly affects node A again. Due to the reliance on repeated feedback, emergent protocols are commonly *iterative* in nature, requiring several rounds of communication between a node and its neighbors before the network as a whole converges on the desired global property.

The main advantages of emergent protocols are:

- **Sophisticated applications.** Emergent algorithms have the potential for more easily expressing complex global properties than localized algorithms. Iterated feedback allows the algorithm to sidestep the explicit coordination and calculation required for such tasks as efficient cluster formation and pattern formation.
- **Increased robustness against transient faults.** The iterated nature of emergent protocols further improves robustness against transient node failure, since a small number of missing or incorrect interactions are unlikely to have a large effect due to the fact that all interactions are repeated several times. This may allow the protocol to tolerate some error in consistency and synchronization between nodes.

Emergent protocols are often harder to design effectively than localized algorithms, since the repeated feedback can create complex interactions that are difficult to analyze. However, their increased expressive power and robustness make them an important class of algorithms, particularly in large-scale distributed sensor networks.

3 Overview of sensor node clustering and applications

Efficiently organizing sensor nodes into clusters is an important application in sensor networks. Many proposed protocols for both sensor networks and ad-hoc networks rely on the creation of clusters of nodes to establish a regular logical structure on top of which efficient functions can be performed. For example, clustering can be used to perform data aggregation to reduce communications energy overhead [10, 26]; or to facilitate queries on the sensor network [6]; clusters can be used to form an infrastructure for scalable routing [14, 23]; clustering also can be used for efficient network-wide broadcast [19, 20]. Single-level clustering is sufficient for many applications; for others, multi-level hierarchical clustering can be performed (by creating clusters of clusters, and so on).

The clustering problem is defined as follows. At the end of the clustering algorithm, the nodes should be organized into disjoint sets (*clusters*). Each cluster consists of a *cluster-head* (cluster leader) and several cluster *followers*, all of which should be within one communication radius of the cluster-head, thus causing the overall shape of the cluster to be roughly a circle of one communication radius, centered on the cluster-head. Each node belongs to exactly one cluster (i.e., every node chooses only one leader, even if there may be several leaders within range). Given these constraints, our goal is to select the smallest set of cluster heads such that all nodes in the network belong to a cluster. The problem is similar to the *minimum dominating set* problem in graph theory. We note that if every node is in *exactly* one cluster, then maximizing the average cluster sizes while maintaining full coverage is exactly equivalent to minimizing the number of clusterheads while maintaining full coverage. The purpose of minimising the number

of cluster heads is to provide an efficient cover of the network in order to minimize cluster overlap. This reduces the amount of channel contention between clusters, and also improves the efficiency of algorithms (such as routing and data aggregation) that execute at the level of the cluster-heads.

For brevity, we have defined the clustering problem as obtaining a single-level clustering. We note that, assuming that clusterheads can establish multiple-hop communications to neighboring clusterheads of the same hierarchy level, it is possible to generalize any single-level clustering protocol to multi-level hierarchical clustering by repeatedly executing the clustering protocol on the cluster-heads of each level to generate the cluster-heads of the next level, and so on.

We summarize in brief a few simple examples of efficient protocols that rely on the effective solution of the single-level clustering problem. A straightforward example is in data aggregation. In an unclustered network, if an aggregate query of sensors over a given sub-area is desired, the query needs to be forwarded to every sensor in the sub-area, each of which then needs to individually send its reply to the base station. In contrast, in a clustered network, a query of sensors over a given sub-area needs only be forwarded to the relevant cluster-head which will then query its followers and send a single aggregated reply.

As an example of the importance of highly uniform clustering with low overlap, consider the clustered broadcast protocol described by by Ni et al.[19]. In this protocol, the broadcast message is relayed from cluster-head to cluster-head, which then broadcast the message to their followers. In a clustering with few clusterheads and large cluster sizes, the clusters have minimal overlap and provide the best coverage of the network with the fewest clusters. Hence, the number of repeated broadcast transmissions over any area will be small, thus reducing the amount of transmission collisions and channel contention, allowing communications to become faster, more efficient and more reliable. On the other hand, a poor clustering with much cluster overlap and many cluster-heads loses much of the benefits of clustering as transmissions will be repeated in areas of overlap with significant channel contention.

4 ACE — Algorithm for Cluster Establishment

In this section, we present ACE (the Algorithm for Cluster Establishment), an emergent cluster formation algorithm. The algorithm consists of two logical parts — the first controls how clusters can spawn (by having a node elect itself to be leader) and the second controls how clusters *migrate* dynamically to reduce overlap. In general, clusters are only created when the overlap of the new cluster with existing clusters is small. After creation, clusters will move apart from each other to minimize the amount of mutual overlap, thus yielding a near-optimal packing in very few iterations.

4.1 Overview of the ACE protocol

We first present a high level overview of the protocol. ACE has two logical parts: the *spawning* of new clusters and the *migration* of existing clusters. New clusters are *spawned* in a self-elective process — when a node decides to become a cluster head, it will broadcast a RECRUIT message to its neighbors, who will become followers of the new cluster. A node can be a follower of more than one cluster while the protocol is running (it picks a single cluster for membership only at the end of the protocol).

Migration of an existing cluster is controlled by the cluster head. Each cluster head will periodically POLL all its followers (i.e., all its neighbors) to determine which is the best candidate to become the new leader of the cluster. The best candidate is the node which, if it were to become cluster head, would have the greatest number of nodes as followers while minimizing the amount of overlap with existing clusters. Once the best candidate is determined by the current cluster head, it will PROMOTE the best candidate as the new cluster head and ABDICATE its position as the old cluster head. Thus, the position of the cluster will appear to *migrate* in the direction of the new cluster head as some of the former followers of the old cluster-head are no longer part of the cluster, while some new nodes near the new cluster head become new followers of the cluster.

4.2 Detailed description of the ACE protocol

In ACE, time synchronization is not required — the nodes may in fact start the protocol at slightly different times due to network delay or clock discrepancies. During the protocol, nodes respond immediately to communications from other nodes, but will only *initiate* actions at random intervals to avoid collisions. Each time that an action can be initiated for a node is called a node's *iteration*. The iterations of different nodes do not need to be synchronized. The duration of the random time interval between iterations (the *iteration interval*) is uniformly random distributed.

We now describe the operation of ACE is described in detail. A node can have three possible states: it can be unclustered (not a follower of any cluster), clustered (a follower of one or more clusters) or it may be a cluster-head. In the beginning of the protocol, all nodes are unclustered. Each node waits for its next iteration (i.e., by waiting for a random iteration interval) before deciding on what action to take on that iteration, if any. When a node's iteration arrives, its available choice of actions depends on what state it is currently in.

If a node A is unclustered when its next iteration arrives, it assesses its surroundings and counts the number l of *loyal* followers it would receive if it declared itself a cluster-head of a new cluster. A *loyal* follower is a follower of only one cluster. Hence, in this case, this number is the same as the number of unclustered neighbors that A has. A knows how long it has been since it started the protocol; call this time t. It then computes the *spawning threshold* function $f_{min}(t)$ (the design of f_{min} will be described later). If $l \geq f_{min}(t)$ then A will spawn a new cluster. It does so by generating a random (unique with high probability) cluster ID and broadcasting a RECRUIT message. A's neighbors will receive this message and become followers of the new cluster.

If a node A is a cluster-head when its next iteration arrives, it prepares to migrate its cluster. It POLLs all of its neighbors to find the best candidate for the new cluster-head. The best candidate leader for a cluster is the node with the largest potential number of *loyal* followers in its neighbor set (recall that a loyal follower is a member of only one cluster). Hence, the best candidate for the new cluster-head is the node which has the largest number of nodes in its neighbor set which are either unclustered or have A's cluster as their only cluster. By counting only loyal followers and not counting nodes that lie on the overlap of two or more clusters, the best candidate node is generally in the direction of least overlap with other clusters. This generates a *repulsion* effect between clusters which leads to good packing efficiency. If the best candidate for cluster-head is A itself, then A does nothing. Otherwise, suppose the best candidate is some node B. A will now MIGRATE the cluster onto the new cluster-head B. It does so by issuing a

PROMOTE message to B. On receiving the PROMOTE message, B will issue a RECRUIT message with A's cluster ID. This is similar to spawning a new cluster except that an existing cluster ID is used instead of generating a new one. The effect of this is that the neighbors of B that were not in the cluster will now be added to the cluster (with B as the cluster-head), while the existing members of the cluster that are B's neighbors will realize that B is being promoted and thus update B as their new cluster-head. Once A observes B's RECRUIT message, it will then issue an ABDICATE message to its neighbors. The effect of this will be that common neighbors of A and B will have seen B's RECRUIT message beforehand and thus ignore the message; neighbors of A who are not neighbors of B will leave the cluster. The net effect of this sequence of actions is that leadership passes from A to B and the cluster as a whole *migrates* from being centered around A to being centered around B.

If a node is clustered (i.e., it is a follower in one or more clusters), then it does nothing during its iteration. It merely waits a random iteration interval for its next iteration to arrive.

Each node needs to be able to efficiently find out the number of loyal followers it may gain. This state can be efficiently maintained by having all nodes keep track of the list of clusters that each neighbor is in. Hence, whenever a node becomes a follower in a new cluster or leaves an existing cluster, it broadcasts an update locally to its neighbors. The overhead of this updating is low because clusters generally do not make drastic shifts in position during migration, hence the cluster sets of most nodes change only slowly with time. By keeping track of these periodic updates, each node can immediately compute how many loyal followers it can gain without needing to query its neighbors.

Each node runs the protocol for at least a time cI where c is the desired average number of iterations per node and I is the expected length of the iteration interval. After a node has completed its iteration, if it still has not passed time cI counting from when it started running the protocol, then it will wait another random iteration interval until its next iteration.

After a node has passed time cI since it started running the protocol, it is ready to terminate the protocol. If the node is a cluster-head, it terminates immediately and informs its neighbors that it is done. If the node is a clustered node, it waits until all its cluster-heads have terminated before choosing one at random to become its final cluster-head (it does not need to notify its neighbors that it has terminated). After termination, the node will respond with "N/A" to leadership polls from clusters that have migrated into its range to indicate its unwillingness to return to the protocol.

Parameter selection. In the protocol, an unclustered node will spawn a new cluster by declaring itself a cluster head whenever it finds that it can gain at least f_{min} loyal followers if it were to become a cluster head. The function f_{min} is called the *spawning threshold function* and is dependent on the time that has passed since the protocol was initiated for that node. In general, f_{min} should decrease as the algorithm proceeds. This causes fewer clusters to form near the beginning of the algorithm. Fewer clusters in the beginning means that the clusters have more room to maneuver themselves apart from each other, in order to form the basis for an efficient clustering. As time advances, the algorithm then causes the gaps between the clusters to be filled in by spawning new clusters more and more aggressively. We observe that the unclustered gaps between clusters decrease roughly exponentially in size when cluster migration is taking place. Hence, in our implementation, we used an exponentially decreasing function for f_{min}:

$$f_{min} = (e^{-k_1 \frac{t}{cI}} - k_2)d$$

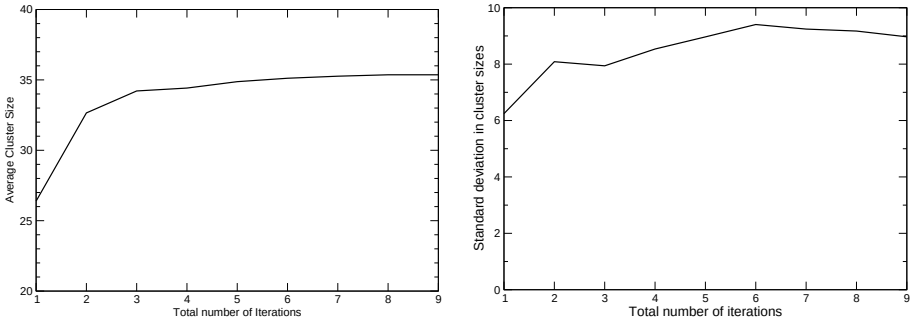
In this formula, t is the time passed since the protocol began and cI is the duration of the protocol as described earlier. d is the estimated average degree (number of neighbours) of a node in the network, and is pre-calculated prior to deployment. k_1 and k_2 are chosen constants that determine the shape of the exponential graph.

In practice, we have empirically found that $k_1 = 2.3$ and $k_2 = 0.1$ have produced good results. In this case, f_{min} starts at $0.9d$ at the beginning of the protocol and reduces to 0 by the final iteration. This ensures that any node left unclustered at the end of the protocol will declare itself a cluster head. A node A may (rarely) find itself unclustered at the end of the protocol if its cluster-head migrates away from A after A has completed its last iteration. To cover this case, an additional "clean-up" iteration should be run after the algorithm has completed for every node. During this final clean-up iteration, cluster migration is disabled, and any node that is still unclustered should declare itself as a cluster-head. This will ensure that evey node in the network is covered by a cluster.

An alternative parameter setting is $k_1 = 2.3$ as before, but setting $k_2 = 0$. In this case the function starts near d when the protocol commences and reduces to $0.1d$ at the end of the protocol. Since $0.1d > 1$ if $d > 10$, it is possible that there will be a small number of nodes that will not be within one hop radius of any cluster-head at the end of the algorithm. This means that this algorithm would not strictly satisfy the problem statement described in Section 3. However, this setting still has practical relevance because the number of unclustered nodes at the end of the algorithm is small. We observed in simulation that the number of nodes not within one-hop radius of a cluster-head is, on average, less than 4% of the total number of nodes in low node deployment densities, and around 2% for moderate to high node deployment densities (20 or more neighbors per node). These nodes that are not within one hop radius of any cluster-head can simply pick a clustered neighbor to act as their bridge to the cluster-head, thus becoming *two-hop followers* (because they take 2 hops to communicate with the cluster-head, instead of the usual 1 hop).

It remains to determine c, the number of iterations the algorithm should execute. Figure 1 reflects how the performance of ACE changes as it is given a longer number of iterations to operate. ACE was simulated in a 2D area with a uniform random distribution with an average deployment density d of 50 nodes per circle of one communication radius. Results for the simulation with $k_1 = 2.3$ and $k_2 = 0.1$ are shown (results for $k_2 = 0$ have similar characteristics). We note that increasing the number of iterations above 3 yielded only very slight improvements in average cluster size. In our simulations, the total number of iterations did not significantly affect the standard deviation in cluster sizes, which was between 6 and 10 for all iterations > 1. Based on these results, we choose $c = 3$ as a number of iterations for ACE that provides a good tradeoff between communication overhead and cluster size.

Figure 2 illustrates the ACE algorithm operating in simulation (with $k_1 = 2.3$ and $k_2 = 0$). The little circles represent nodes. Cluster-heads are highlighted in black, and their range is indicated with a large black circle (nodes within the circle are in that cluster). The clusters migrate away from each other in successive iterations to produce a highly efficient cover of the area. Clusters tend to center over areas where nodes are dense. The clusters overlap minimally, and when they do overlap, they tend to overlap in areas where nodes are sparse. Figure 2d provides a qualitative visual comparison of



(a) Average cluster size (b) Standard deviation of cluster size

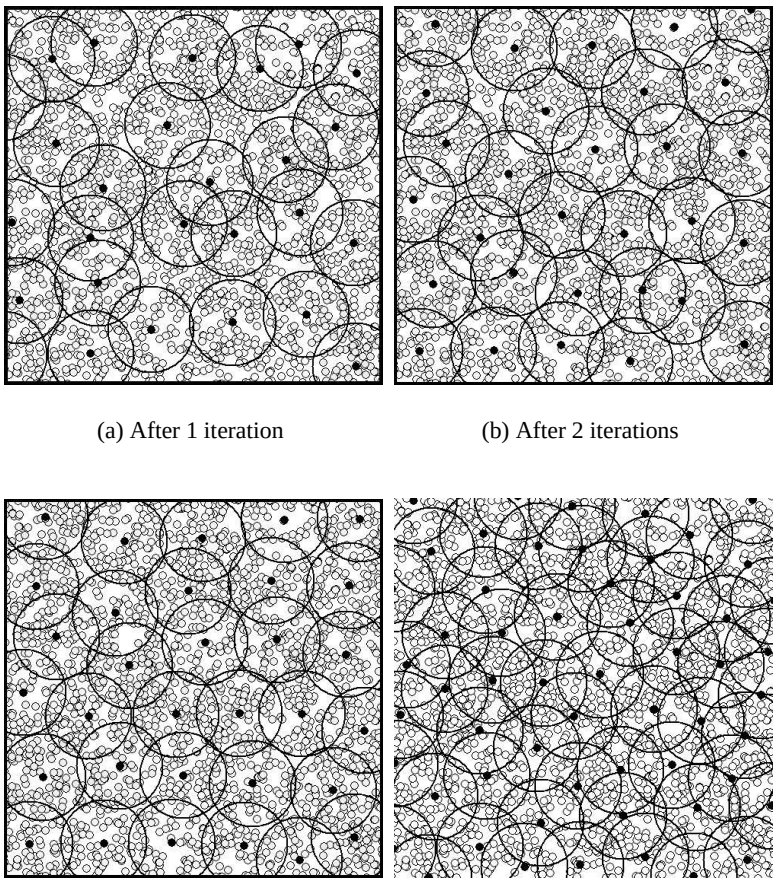
Fig. 1. Performance of ACE at various maximum iterations, $d = 50, k_1 = 2.3, k_2 = 0.1$

the Node ID algorithm with ACE. It can be observed that ACE provides a packing with signifi cantly less cluster overlap than Node ID.

5 Performance evaluation of ACE

To assess ACE's performance, ACE was simulated and its performance was compared with a well-known 2D packing (hexagonal close packing) as well as two other clustering algorithms, the Node ID algorithm and the Node Degree algorithm. In our simulations we simulated both ACE with full coverage ($k_1 = 2.3, k_2 = 0.1$), which we called ACE-1 and also ACE with parameters that leaves a small fraction of nodes uncovered ($k_1 = 2.3, k_2 = 0$), which we call ACE-2.

Hexagonal close-packing (HCP) is the well-known honeycomb packing that minimizes overlap between uniform circular clusters while ensuring full coverage. In general this packing is diffi cult to achieve unless nodes have very specifi c information about their geographic locations, e.g. as assumed Zhang and Arora [27], and even then a centralized algorithm needs to be used to coordinate the honeycomb structure, which leads to lower scalability. The Node ID algorithm is a generic name for the class of algorithms related to LCA (Linked Cluster Architecture) [2, 5]. In this algorithm, the node with the highest ID elects itself as a cluster-head, followed by the node with the next highest ID that is not already a follower, and so on until all the nodes have been covered. The basic concept has been revisited in various architectures such as those described by Lin et al. and Gerla et al. [9, 17]. Amis et al. [1] improved the algorithm for multi-hop clusters by adding a second pass in which clusters with low node IDs are expanded to better balance cluster size; however this improvement has no effect on 1-hop clusters hence we do not simulate it here. Nagpal and Coore propose a variation of the Node ID algorithm where the nodes generate a random number and start counting down from it; when the counter reaches zero and the node is not already a follower in some cluster then the node elects itself as cluster-head [18]. This algorithm is similar to the Node ID algorithm with the benefi t that it can be repeated for re-clustering on successive epochs. Using the degree of connectivity instead of node ID as a metric for which nodes to elect as cluster-heads



(a) After 1 iteration

(b) After 2 iterations

(c) After 3 iterations

(d) *The Node ID algorithm (for comparison)*

Fig. 2. The ACE algorithm (with $k_1 = 2.3$ and $k_2 = 0$)

has been proposed Basagni [4] and Gerla et al. [9]. This causes nodes in denser areas to become cluster-heads first. We model this algorithm as the Node Degree algorithm.

The various algorithms were simulated on various deployments of 2500 nodes in a square area where each node's coordinates were uniformly random. In our simulation, we assume that the communication links were bi-directional and that the communication range of all the nodes is uniform. 500 simulations per algorithm were run for each of the node densities (expected number of neighbors in a circle of one communication radius) of $d = 10, 20, 50, 100$.

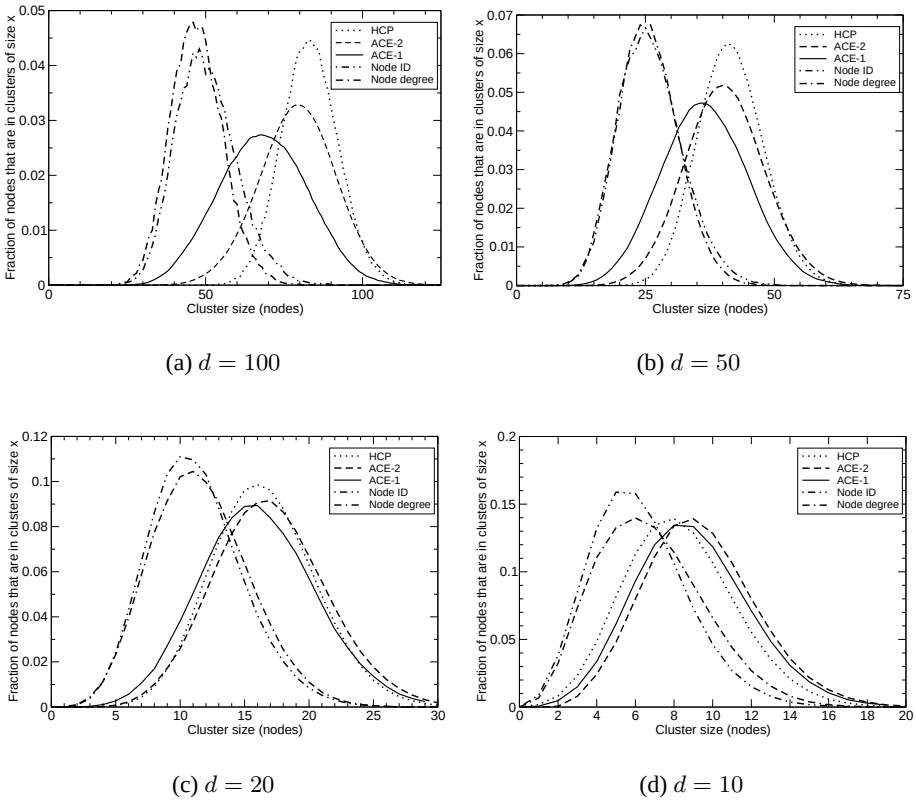
Figure 3 shows the relative distributions of cluster sizes for the various algorithms under the various node densities simulated. Figure 4 compares the average cluster sizes of the various algorithms as d varies.

Algorithm 5.1 ACE

```
procedure SCALE_ONE_ITERATION()
  if myTime > 3× EXPECTED ITERATION LENGTH then
    if myState = CLUSTER-HEAD then
      return DONE
    else if myState = CLUSTERED then
      wait for my cluster-heads to terminate, then pick one as my cluster-head
      return DONE
    else if myState = UNCLUSTERED then
      pick a random clustered node to act as my proxy after it terminates
      wait for it to terminate, then return DONE
    end if
  else if myState = UNCLUSTERED
  and numLoyalFollowers() ≥ f_min(myTime) then
    myClusterID ← generate_New_Random_ID()
    locally_broadcast(RECRUIT, myID, myClusterID)
  else if myState = CLUSTER-HEAD then
    bestLeader ← myID
    bestFollowerCount ← numLoyalFollowers
    for all n where n is a potential new cluster-head do
      followerCount = Poll_For_Num_Loyal_Followers(n, myClusterID)
      if followerCount > bestFollowerCount then
        bestLeader ← n
        bestFollowerCount ← followerCount
      end if
    end for
    if bestLeader is not myID then
      send(bestLeader, PROMOTE, myClusterID)
      wait for bestLeader to broadcast it's RECRUIT message
      locally_broadcast(ABDICATE, myID, myClusterID)
    end if
  end if
end procedure
```

It is clear that ACE exhibits superior packing efficiency to either the Node ID or Node Degree algorithms. ACE-1 exhibits consistent performance of around $0.7d$ average cluster size for all node densities. ACE-2 exhibits performance around $0.8d$ average cluster sizes. For reference, the average cluster size for the ideal 2D packing of HCP is $0.83d$. ACE sometimes exceeds the ratio for HCP because it intelligently forms clusters around areas where nodes are most densely distributed, while choosing areas of overlap where the nodes are least densely distributed. In comparison, both Node ID and Node Degree converge towards only $0.5d$ for large d and never perform better than $0.7d$ even at low node densities.

Figure 3 shows that the variance in cluster sizes for ACE is small and only slightly larger than the baseline variance of the number of nodes in a given area (this is reflected in the variance of cluster sizes for HCP). The low variance and high average cluster sizes reflect that the ACE algorithm produces good packing efficiency.

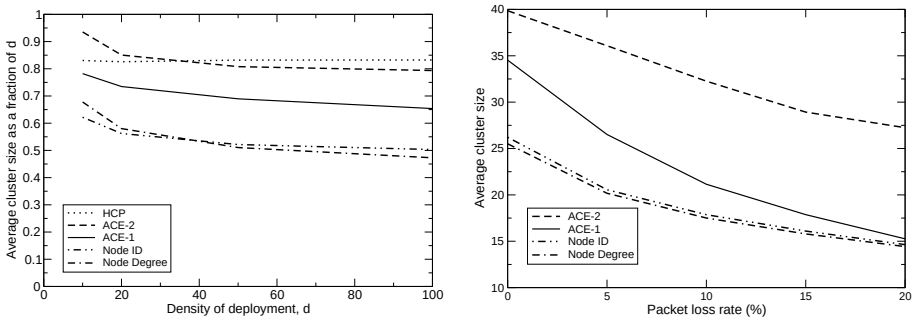


(a) $d = 100$ (b) $d = 50$

(c) $d = 20$ (d) $d = 10$

Fig. 3. Distribution of cluster sizes for various clusterings

We investigated the performance of the various algorithms under conditions of packet loss. Various deployments of $d = 50$ were simulated with packet loss rates ranging from 0 to 20%. Packet loss was simulated by having the simulated nodes ignore an incoming message with a probability corresponding to the packet loss rate. Figure 4b reflects the results. The performance of each protocol degrades gracefully under packet loss. ACE-2 maintains its large advantage over Node ID and Node Degree even under conditions of heavy packet loss. ACE-1 degrades at a higher rate and approaches the performance of Node ID and Node Degree under conditions of high packet loss, but never actually performs worse than either algorithm under the simulated conditions. We note further that since the ACE algorithm is localized and require no central direction, it is highly resistant to transmission errors and random node failure. For example, a centralized algorithm utilizing a BFS tree (e.g. Bannerjee and Khuller's algorithm [3]) could suffer the loss of an entire subtree if one of the nodes high in the tree suffers a failure, thus leading to failure of the protocol. In our protocols, in the worst case, the loss of a cluster-head node would leave at most one cluster of nodes unclustered. If a cluster-head node fails while executing the protocol, and the failure is detected by the



(a) Average Cluster Sizes (normalized to d)

(b) Cluster Sizes under packet loss, $d = 50$

Fig. 4. Average cluster sizes of various schemes

d	10	20	50	100
ACE-1	6.68	6.80	7.07	7.32
ACE-2	5.41	4.96	4.47	4.56
Node ID	1.17	1.09	1.04	1.02
Node Deg.	1.17	1.09	1.04	1.02

Fig. 5. Average communications overhead (per node per epoch)

followers, they can reset their own states to "unclustered", thus allowing neighboring clusters to migrate into the new vacant space or allowing a new clusterhead to spawn a cluster within the vacant space. Hence, the protocol has an innate self-stabilization property. These additional adaptations for unreliable communications were not simulated; if they were implemented they would likely further improve the protocol's resilience towards random node failures and communication errors.

We also measured the communications overhead of the various algorithms. Each transmission was considered one unit of communication, and the final cluster handshake where all nodes confirm membership with their cluster-heads was also considered an additional unit of communication. The results are tabulated in Figure 5. Because of the low number of iterations needed by ACE (only 3 iterations), the communications overhead is small, only averaging around 4 to 8 communications per node per epoch. Each communication is brief (at most a message identifier, a node and cluster identifier, and a number). Hence the overall communications overhead is small compared with the normal communications load for the sensor network.

ACE exhibits scale independence (perfect scalability). The protocol takes a fixed amount of time, $O(d)$, to complete regardless of the total number of the nodes in the network. This is because it is a *strictly localized* algorithm (see definition in Section 2), where each node is capable of operating immediately and independently on local information without needing any global information to be computed by the network. As a result, both running time and per-node communications overhead of ACE are independent of the total size of the network.

6 Related Work

In this section, we review related work in localized and emergent algorithms in sensor networks, as well as clustering algorithms in general.

Currently, few practical emergent algorithms have been developed for use in sensor networks. Henderson suggests using Turing's reaction-diffusion equations [24] for forming patterns in sensor networks [11]. These approaches are promising and indicative of the future potential of emergent algorithms.

We now discuss related work in clustering. Many clustering protocols currently known are *self-elective* protocols, where a node creates a new cluster by declaring itself as a cluster-head. They differ in the heuristic used to select the nodes which will declare themselves. The node ID and node degree heuristics have been discussed in Section 5. Examples of node ID based clustering protocols include [1, 2, 5, 9, 17].Basagni proposes a node degree based clustering protocols [4]. Some researchers propose using a random number as a heuristic for cluster-head selection [8, 12, 18, 25]. Estin et al. propose using the remaining energy level of a node as another heuristic for cluster-head selection [6].

Ramanathan and Steenstrup [21], and Krishnan, Ramanathan, and Steenstrup [15] propose a clustering algorithm that controls the size of each cluster and the number of hierarchical levels. Their clustering approach follows the node ID approach. In general, these self-elective protocols all suffer from the same problem of being unable to prevent two nodes which are just over one cluster radius apart from simultaneously electing themselves as cluster-heads, thus leading to a large overlap in their clusters. Such overlap occurs sufficiently frequently to make the resultant cluster packing inefficient, as can be seen in our evaluation in Section 5.

The *minimum dominating set* (MDS) problem in graph theory has been addressed by several algorithms. The clustering problem is a special case of the MDS problem applied to random geometric graphs. While these algorithms have provable theoretical *asymptotic* bounds on performance on arbitrary graphs, their *actual* average performance on a random geometric graph is undetermined. We implemented in simulation two algorithms described by Jia et al. [13] and Kuhn et al. [16], however neither of them had comparable performance to even the simple Node-degree or Node-ID algorithms under our particular simulation conditions. We speculate that the relatively poor performance of these algorithms in simulation may be due to the fact that they are designed for arbitrary graphs while dedicated clustering algorithms are optimized for random geometric graphs. Hence, we did not run full simulation comparisons against these algorithms.

Many centralized (non-localized) clustering algorithms are known, which deal with the topology of the entire network as a whole. This class of algorithms often uses graph-theoretic properties for clustering. In general, such algorithms are not as robust or scalable as localized algorithms, eventually requiring significant communications or computation overhead for very large networks. For example, Krishna et al. [14] proposes a technique where each cluster forms a clique, however their approach has $O(d^3)$ overhead. Some researchers proposed tree-based constructions for network partitioning. Thaler and Ravishankar propose to construct a top-down hierarchy, based on an initial root node [22]. Banerjee and Khuller also propose a tree-based clustering algorithm [3]. A drawback for using their algorithm in sensor networks is that only one node needs to initiate the clustering, and that the protocol still requires $O(n)$ time in linear networks. Zhang and Arora present a centralized scheme to produce an approximate hexagonal

close packing [27]. However, they assume that each node knows its precise location, which may be difficult to achieve in sensor networks. In general, besides scalability issues, most these nonlocalized algorithms also suffer from increased vulnerability of the protocol to node failure in certain key parts of the network (usually near the root of the tree, or near the base station).

7 Conclusion

We present ACE, the Algorithm for Cluster Establishment. ACE is an emergent algorithm that uses just three rounds of feedback to induce the formation of a highly efficient cover of uniform clusters over the network. This efficiency of coverage approaches that of hexagonal close-packing. ACE is fast, robust against packet loss and node failure, and efficient in terms of communications. It completes in constant time regardless of the size of the network and uses only local communications between nodes. The algorithm does not require geographic location information or any kind of distance or directional estimation between nodes. Besides its practical usefulness, ACE is a good demonstration of the power and flexibility of *emergent* algorithms in large-scale distributed systems.

References

1. Alan D. Amis, Ravi Prakash, Thai H.P. Vuong, and Dung T. Huynh. Max-Min D-Cluster Formation in Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM 2000*, pages 32–41, 2000.
2. D.J. Baker, A. Ephremides, and J.A. Flynn. The Design and Simulation of a Mobile Radio Network with Distributed Control. *IEEE Journal on Selected Areas in Communication*, 2(1):226–237, January 1984.
3. Suman Banerjee and Samir Khuller. A Clustering Scheme for Hierarchical Control in Wireless Networks. In *Proceedings of IEEE INFOCOM 2001*, April 2001.
4. Stefano Basagni. Distributed Clustering for Ad Hoc Networks. In *Proceedings of the IEEE International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, pages 310–315, June 1999.
5. A. Ephremides, J.E. Wieselthier, and D.J. Baker. A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling. *Proceedings of IEEE*, 75(1):56–73, 1987.
6. Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 263–270, August 1999.
7. David A. Fisher and Howard F. Lipson. Emergent Algorithms: A New Method for Enhancing Survivability in Unbounded Systems. In *Proceedings of the Hawaii International Conference On System Sciences*, January 1999.
8. M. Gerla, T.J. Kwon, and G. Pei. On Demand Routing in Large Ad Hoc Wireless Networks with Passive Clustering. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2000)*, September 2000.
9. M. Gerla and J.T. Tsai. Multicluster, Mobile, Multimedia Radio Network. *ACM/Kluwer Journal of Wireless Networks*, 1(3):255–265, 1995.
10. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00)*, January 2000.

11. T. C. Henderson, M. Dekhil, S. Morris, Y. Chen, and W. B. Thompson. Smart Sensor Snow. In *Proceedings of IEEE Conference on Intelligent Robots and Intelligent Systems (IROS)*, October 1998.
12. X. Hong, M. Gerla, Y. Yi, K. Xu, and T. Kwon. Scalable Ad Hoc Routing in Large, Dense Wireless Networks Using Clustering and Landmarks. In *Proceedings of IEEE International Conference on Communications (ICC 2002)*, April 2002.
13. Lujun Jia, Rajmohan Rajaraman, and Torsten Suel. An Efficient Distributed Algorithm for Constructing Small Dominating Sets. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 33–42, 2001.
14. P. Krishna, N. H. Vaidya, M. Chatterjee, and D. Pradhan. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, 27(2):49–65, April 1997.
15. Rajesh Krishnan, Ram Ramanathan, and Martha Steenstrup. Optimization Algorithms for Large Self-Structuring Networks. In *IEEE INFOCOM '99*, 1999.
16. Fabian Kuhn and Roger Wattenhofer. Constant-Time Distributed Dominating Set Approximation. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing*, pages 25–32, 2003.
17. Chunhung Richard Lin and Mario Gerla. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.
18. Radhika Nagpal and Daniel Coore. An Algorithm for Group Formation in an Amorphous Computer. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems (PDCS'98)*, October 1998.
19. Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceedings of the Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 151–162, August 1999.
20. Elena Pagani and Gian Paolo Rossi. Reliable broadcast in mobile multihop packet networks. In *Proceedings of the Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '97)*, pages 34–42, 1997.
21. R. Ramanathan and M. Steenstrup. Hierarchically-Organized, Multihop Mobile Wireless Networks for Quality-of-Service Support. *ACM/Baltzer Mobile Networks and Applications*, 3(1):101–119, June 1998.
22. David G. Thaler and Chinya V. Ravishankar. Distributed Top-Down Hierarchy Construction. In *Proceedings of IEEE INFOCOM*, pages 693–701, 1998.
23. P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *Symposium Proceedings on Communications Architectures and Protocols (SIGCOMM '88)*, pages 35–42, 1988.
24. Alan Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal society of London B237*, pages 37–72, 1952.
25. Kaixin Xu and Mario Gerla. A Heterogeneous Routing Protocol Based on a New Stable Clustering Scheme. In *IEEE MILCOM 2002*, October 2002.
26. Ya Xu, Solomon Bien, Yutaka Mori, John Heidemann, and Deborah Estrin. Topology Control Protocols to Conserve Energy in Wireless Ad Hoc Networks. Technical Report 6, University of California, Los Angeles, Center for Embedded Networked Computing, January 2003.
27. H. Zhang and A. Arora. GS^3: Scalable Self-configuration and Self-healing in Wireless Networks. In *21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, 2002.