LOCI: Local Clustering Service for Large Scale Wireless Sensor Networks

Vineet Mittal Murat Demirbas Anish Arora

Department of Computer & Information Science
The Ohio State University
2015 Neil Avenue, Dreese Lab 395
Columbus, Ohio 43210 USA

Abstract

We present a local, scalable, and fault-tolerant distributed clustering service, LOCI, that partitions a multi-hop wireless network into clusters of bounded physical radius $[R, mR]$ where m is a constant greater than or equal to 2. That is, each cluster has a leader node such that all nodes within distance R of the leader belong to the cluster but no node beyond distance mR from the leader belongs to the cluster.

LOCI is local in that each node only needs information about nodes that are at most distance $2R$ away. It is scalable in that each node maintains only a constant amount of state and completes its role in clustering in $O(R^4)$ time ($O(R^2)$ for a stronger system model), independent of the network size. It is fault-tolerant in that it is self-stabilizing in the presence of state corruption, node and link fail-stops, and node joins.

The network partitions generated by LOCI form a Voronoi tessellation as each non-clusterhead node joins the cluster of the nearest clusterhead and the number of clusters LOCI yields is within a constant factor approximation of the minimum number of clusters theoretically feasible. Our simulations demonstrate that, for $m = 2$, the number of clusters constructed by LOCI exceed the minimum number of clusters theoretically feasible only by a factor of 1.5 for a 1-D network and 2.3 for a 2-D network.

As hierarchical clustering is readily achieved by instantiating LOCI at multiple levels, LOCI provides a framework for scalable and fault-tolerant distributed tracking structure for *pursuer-evader scenarios that has arisen in our recent work in sensor networks. Furthermore, as part of our efforts towards developing sensor network services in the DARPA Network Embedded Software Technology (NEST) program, we have implemented LOCI in TinyOS on the Mica2 mote platform.*

Keywords: *Clustering, locality, self-configuration, self-stabilization, fault-tolerance, graph partition.*

1 Introduction

Large scale wireless sensor networks introduce challenges for self-configuration and self maintenance. Firstly, sensor nodes are deployed in an ad hoc manner; thus, centralized solutions that rely on pre-defined configurer or maintainer nodes are unsuitable. Secondly, sensor networks are fault-prone: message losses and corruptions (due to fading, collisions, and hidden node effect), node failures (due to crash and energy exhaustion), and network partitions (due to message losses and node failures) are the norm rather than the exception; thus, sensor nodes can lose synchrony and their programs can reach arbitrary states [16]. Finally, on-site maintenance is infeasible; thus, sensor networks should be self-healing.

The need for self-configuration, self-maintenance, and self-stabilization in wireless sensor networks has led us to consider alternative solutions for structuring networks via clustering. Clustering (and the derivative concept of hierarchical clustering) is a standard approach to achieving efficient, scalable control in networks, in part because it facilitates the distribution of control over the network as well as the insulation of changes and failures/joins in one part of the network from other parts. That said, achieving clustering itself in an efficient, scalable, and self-stabilizing manner has always been a challenge.

More specifically, we consider clustering based on geographic radius as it enables (i) structuring of the network independent of the number of nodes; (ii) efficient communication and energy expenditure in geographic regions; and (iii) efficient evaluation of spatial queries. Our criteria for an efficient clustering include: (i) constructing clusters of equal size for load balancing and uniformity of energy dissipation; (ii) minimizing the overlaps between clusters for minimizing energy requirements; (iii) minimizing the number of orphan nodes for maximizing available resources; (iv) minimizing communication overhead for saving energy (for increased network lifetime); (v) minimizing the number of clusters for minimizing memory requirements; and (vi) scalable cluster formation and scalable self-healing for minimizing space and time requirements and maximizing tolerance to faults.

Contributions of the paper. In this paper, we present LOCI, a distributed program that partitions a wireless sensor network into clusters, each with a distinguished leader node. Every cluster is uniform in that it includes all nodes within distance R from the cluster leader and no node outside of distance mR from the cluster leader, where m is a constant greater than or equal to 2. Furthermore, network partitions constructed by employing LOCI, form a Voronoi tessellation as each non-clusterhead node joins the cluster of the nearest clusterhead.

LOCI has several notable properties of locality, scalability, and fault-tolerance. It is local in that each node only needs information about nodes that are at most distance $2R$ away. It is scalable in that each node maintains a constant amount of state and completes its role in clustering in $O(R^4)$ time, independent of the network size. It is fault-tolerant in that it is locally self-stabilizing despite arbitrary state corruption and failure or repair of nodes and links. The number of clusters it yields is within a constant factor of the minimum number of clusters theoretically feasible. Lastly, LOCI is based on a relatively simple program, and additionally allows cluster radius to be a parameter.

We note that as part of our efforts towards developing sensor network services in the DARPA Network Embedded Software Technology (NEST) program, we have implemented LOCI [19] on the Mica2 mote platform [15]. Our simulation experiments demonstrate that, for $m = 2$, the number of clusters constructed by LOCI exceed the minimum number of clusters theoretically feasible only by a factor of 1.3 for a 1-D network and 2.2 for a 2-D network.

We show that a hierarchical partitioning of the network is readily achieved by instantiating LOCI at multiple levels. The resulting radius of a cluster at level *i* is in the range $[(2mR)^i * R,\ (\frac{(2mR)^{i+1}-1}{2mR-1}) * mR]$.

Finally, by way of application, LOCI provides a framework for scalable and fault-tolerant distributed indexing service for pursuer-evader tracking in wireless sensor networks. The hierarchical partitioning achieved via LOCI enables tracking information regarding an evader to be maintained over a small subset of nodes (only at selected clusterheads at increasingly higher levels) and with accuracy proportional to the distance from the evader.

1.1 Overview

Motivation for tolerance factor m. For local self-healing both in time and space, bounded radius range clustering is desirable as opposed to fixed radius clustering. We illustrate this observation with an example for a 1-D network (for the sake of simplicity).

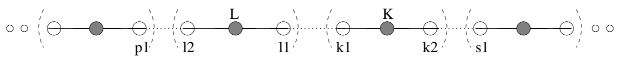


Figure 1. Each pair of brackets constitutes one cluster of unit radius, and colored nodes denote cluster leaders.

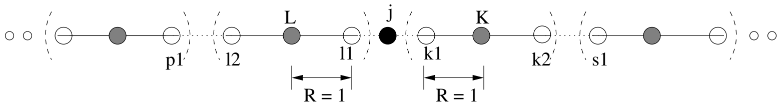


Figure 2. A new node j joins the network between clusters of cluster leaders L and K.

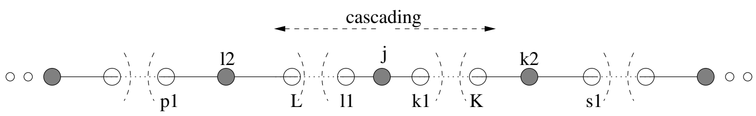


Figure 3. Node j forms a new cluster and leads to re-clustering of the entire network.

Consider a clustering scheme that constructs clusters of fixed radius $R = 1$. Figure 1 shows one such construction where all clusters have unit radius, i.e., all nodes that are within unit distance of cluster leaders belong to the same cluster as that of the leader. We show that for fixed radius clustering schemes, a node join can lead to re-clustering of the entire network. When node j joins the network (Figure 2), it cannot be subsumed in its neighboring clusters as j is not within unit distance of neighboring cluster leaders L and K. j thus forms a new cluster with itself as the cluster leader. Since all nodes within unit radius of a cluster leader should belong to the same cluster as that of the leader, j subsumes neighboring nodes l_1 and k_1 in its cluster. This leads to neighboring cluster leaders L and K to relinquish their clusters and election of l_2 and k_2 as the new cluster leaders (Figure 3). The cascading effect propagates further as the new cluster leaders l_2 and k_2 subsume their neighboring nodes leading to re-clustering of the entire network.

Thus a clustering scheme that constructs clusters of fixed radius is not locally self-healing. This motivates constructing clusters of bounded radius range [R, mR] as it results in scalable self-healing both in time and space. As an illustration, consider Figure 4 where clusters have tolerance factor $m = 2$ and radius R = 1. When node j joins the network it is subsumed by one of its neighboring clusters as it is within distance 2 of the neighboring clusterhead L, thus leading to local self-healing.

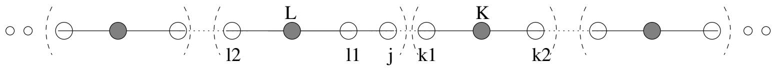


Figure 4. New node j joins one of its neighboring clusters.

We next illustrate how LOCI locally self-heals when all clusters are of radius mR and a new node joins the network (Figure 5). When j joins the network it elects itself as the clusterhead since it is not within 2 distance of clusterheads of its neighbors $l1$ and $k1$. Nodes $l1$ and $k1$ then join the cluster of clusterhead j because they are not within 1 distance of their respective clusterheads but are within 1 distance of clusterhead j, thus forming a legitimate cluster with j as the clusterhead (Figure 6).

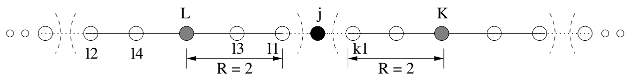


Figure 5. l_1 and k_1 are at distance 2 (radius range [1, 2]) from their cluster leaders L and K, respectively. $j's$ neighbors are l_1 and k_1.

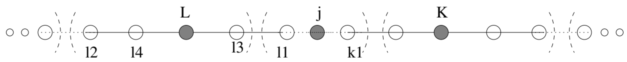


Figure 6. j becomes the clusterhead and neighboring nodes $l1$ and $k1$ join the cluster of j.

Overview of LOCI program. We assume a 2-D coordinate plane, where node distribution is dense and nodes have distance estimation capability relative to each other (we show how to relax this assumption in Section 8).

LOCI performs the desired clustering as follows. Each node waits for a random amount of time and then takes a local decision on whether or not to elect itself as a leader after checking the state of its neighboring nodes. Then, by subsuming all of its neighbors, each self-elected leader node forms a unit radius cluster in the shape of a circular disc, with itself at the center. We call a cluster formed by a self-elected leader an *autonomous* cluster. The autonomous cluster is then grown iteratively, with each iteration incrementing the radius of the circular disc by one, until a circular disc of physical radius R is formed. We call a cluster that properly contains a circular disc of physical radius R and contains no node beyond distance mR from the clusterhead a *legitimate* cluster.

We require clusters to be non-overlapping and cluster formation to be local and distributed; but two or more autonomous clusters might *overlap* with each other when they attempt to subsume all neighbors of their respective discs. We resolve overlaps among clusters by allowing only one of them to increment the radius of its disc. This is achieved by associating a dynamic, unique priority with each cluster, and by *reducing* the circular radius of the lower priority clusters by one when they overlap (by making the lower priority clusters to relinquish their peripheral nodes). More specifically, the prioritization favors clusters with larger radius and fewer overlaps, thus providing them with a greater chance of becoming legitimate. It also favors legitimate clusters over autonomous clusters that are not yet legitimate, thus a cluster remains legitimate once it achieves a circular disc of radius R. Notice that growing a cluster based on physical radius guarantees that radius of the cluster covers a circular region around the clusterhead, and subsequent iterative growth of the cluster guarantees that the number of overlaps that the cluster can have with other clusters of equal radius is bounded by a constant (i.e., $2\pi(1 + 1/r) \approx 12$, where r is the radius of the cluster).

To achieve local self-healing and network partitioning, we allow legitimate clusters to accept members that are up to mR distance from their respective leaders. Notice that legitimate clusters do not necessarily grow in a circular disc shape, i.e., they do not necessarily subsume all the nodes that are within a circular radius r, $r > R$, before subsuming nodes at distance $r + 1$. Moreover, we allow legitimate clusters another level of flexibility – they can lose members as long as they preserve their legitimacy (i.e., they can lose members that are within R to mR distance from the respective cluster leaders). This allows all nodes to join a legitimate cluster and additionally enables formation of a Voronoi tessellation as all non-leader nodes join the cluster of the nearest cluster leader. LOCI guarantees for $m \geq 2$ that all nodes are eventually included in some legitimate cluster.

LOCI achieves partitioning in $O(R^4)$ rounds of node execution, where a round is the least number of steps in which all the enabled nodes execute at least one action.

Organization of the paper. Section 2 describes the system model and LOCI problem statement. Section 3 describes LOCI program. Section 4 presents our simulation results in the Prowler [21] simulator. Section 5 describes how LOCI achieves multi-level clustering. Section 6 presents an application of LOCI that has arisen in our work on pursuer-evader tracking in wireless sensor networks. Section 7 discusses related work. Finally, Section 8 makes concluding remarks and includes further comments on the system model. For reasons of space, statistical density and fault model, analytical performance analysis of LOCI, improving the convergence of LOCI, and proofs are relegated to the Appendix; and implementation of LOCI is discussed in [19].

2 Preliminaries

In this section, we first present our system model and programming notation. We then present the fault model, some definitions and the problem statement.

System model. We consider a wireless sensor network where nodes lie in a 2-D coordinate plane. Nodes are connected iff they are within unit distance of each other. Nodes and edges are represented by the set V and E, respectively, and the resultant undirected graph by G, where $G = (V, E)$. All nodes within unit distance of any node j are its neighbors and the neighbor set of j is denoted by $nbr.j$. We make the following model assumptions:

- each node has a *unique ID* (to facilitate unique priority assignment to clusters);
- nodes have *distance estimation* capability relative to their neighbors [15] (to enable cluster construction based on geographic radius);
- *density*: nodes are distributed as per a homogeneous spatial Poisson process of intensity λ. (In Section 8 we show how to relax this assumption by presenting a weaker density model).

Programming notation. For ease of presentation we assume a *shared memory* model (in [19] we give implementation details for a *message passing* model). A *program* consists of a set of variables and actions at each node. Each variable has a pre-defined domain. The variables and actions of each node are $symmetric$ relative to node ID and neighbor set. A node can read variables of all of its neighbors and write its own variables at once. Each action has the form:

<*guard*> $\longrightarrow$ <*assignment statement*>

A $guard$ is a boolean expression over variables. An assignment statement updates one or more variables.

A $system$ is a wireless sensor network that consists of a set of nodes running the same program. A $state$ of the system is defined by a value for every variable in each node of the system, chosen from the predefined domain of that variable. An action whose guard is true at some state is said to be *enabled* at that state. A node is enabled if at least one action is enabled at that node. A $computation$ of the system is a maximal, fair sequence of system steps: in each step, some action that is enabled at the current state is executed, thereby yielding the next state in the computation. The fairness of a computation means that each continuously enabled action is eventually executed in the computation [13]. To measure progress of a computation it can be partitioned into logical units called *rounds* where a *round* is a minimal sequence of steps Q such that each node in the system that is enabled at some state along Q executes at least one action in Q [5]. A state is a *fixpoint* of a program *p* iff no action is enabled in that state. A state predicate *I* is closed in *p* iff each action of *p* preserves *I*. *I* is an invariant of *p* iff *I* is closed and holds for initial states of *p*.

In this paper, we use *j*, *k*, and *l* to denote the nodes and *var.j* to denote the variable residing at node *j*. The function of a *parameter* is to define a set of actions as one parameterized action. For example, let *k* be a parameter whose value is 0, 1, and 2; then an action *act* of node *j* parameterized over *k* abbreviates the following set of actions:

$$act(k = 0)\ []\ act(k = 1)\ []\ act(k = 2)$$

where act($k = i$) is *act* with every occurrence of *k* substituted with *i*. We use [] to separate the actions in the program.

Fault model. A node or an edge is *up* if it functions correctly, and it is *down* if it fail-stops. In a system, nodes and edges that are up can fail-stop, nodes and edges that are down can become up and join the system, and the state of a node can be arbitrarily corrupted. The program actions of a node cannot be corrupted.

Definitions. Let $dist.j.k$ denote the Cartesian distance between nodes j and k in G. A *cluster* is defined as a subset of vertices $S \subseteq V$ in G, such that the *subgraph* $G(S)$, $G(S) = (S, E')$, where E' consists of all the edges of G whose endpoints both belong to S, is connected.

A *cover* of the graph $G = (V, E)$ is a collection of clusters $\{S_1, S_2, \ldots, S_p\}$ such that $\bigcup_i S_i$ = V. A *partial partitioning* of G is a collection of disjoint clusters $\{S_1, S_2, \ldots, S_p\}$ with the property that $S_j \cap S_k = \phi$ for $j \neq k$. A *partitioning* of G is a collection of clusters that is both a *cover* and a *partial partitioning*.

The *r-neighborhood* of a node j is defined as $N_r(j) = \{k \mid dist.j.k \leq r\}$. The *radius* of a cluster S with respect to a node j, $j \in G(S)$, is defined as $max_{k \in G(S)} dist.j.k$. A cluster with radius r with respect to a node j is *circular*, if all nodes in the r-neighborhood of j belong to the cluster.

Problem statement. The clustering problem is to design a distributed, local, scalable and self-stabilizing program that, given a cluster radius interval [*R*, *mR*], where R is a natural number and m is a constant ≥ 2, constructs a partition $\{S_1, S_2, \ldots, S_p\}$ such that:

- a unique node is designated as a leader of each cluster,
- all nodes in the R-neighborhood of each leader belong to that cluster,
- maximum distance of a node from its leader is *mR*,
- each node belongs to the cluster of the closest clusterhead, and
- there exists a path from every node to its leader which consists of only the nodes that belong to the same cluster.

3 LOCI program

The LOCI program, as outlined in the Introduction, requires the following three tasks at each node j.

(1)	timeout(can_lead(j))	$\longrightarrow$ start_cluster(j)
[]		
(2)	can_join(j, S)	$\longrightarrow$ join(j, S)
[]		
(3)	j and k are clusterheads $\wedge$ overlapping_clusters(j, k)	$\longrightarrow$ resolve_overlap(j, k)

(1): A node j is *eligible* to become a clusterhead, if $j's$ random waiting time has expired and j does not belong to any cluster and j is not within mR-neighborhood of the clusterheads (if any) of its neighboring nodes (to guarantee that R-neighborhood of j does not overlap with R-neighborhood of a legitimate cluster). As shown in Figure 7, j is eligible to become a clusterhead but j' is not. j then *starts a cluster* by electing itself as the clusterhead and subsuming its neighboring nodes thus forming a unit radius circular disc. Recall that a cluster with circular disc radius greater than or equal to one is an *autonomous* cluster.

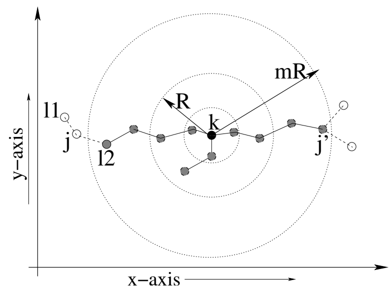


Figure 7. k is a clusterhead of a legitimate cluster. $l1$ does not belong to any cluster. j is not within mR-neighborhood of k. j can become a clusterhead but j' cannot.

(2): j *can join* the cluster S of one of its neighbors, say k, in three ways. First, if j does not belong to any cluster but is within mR-neighborhood of the new clusterhead. Second, if j already belongs to a cluster and j is not within R-neighborhood of its clusterhead but is within R-neighborhood of k's clusterhead. Thus, as outlined in the Introduction, legitimate clusters can lose nodes that are within R to mR distance from the respective clusterheads, e.g. $l2$ in Figure 7 can join $j's$ cluster. Finally, if both j and k belong to legitimate clusters and they are not within R-neighborhoods of their respective clusterheads and distance of j from $k's$ clusterhead is less than the distance of j from its current clusterhead (this allows nodes to join clusters of nearest clusterheads).

(3): When local clusterhead election leads to overlaps between *R*-neighborhoods of clusterheads (Figure 8), non-overlapping clusters are obtained by assigning to each cluster a unique, dynamic priority given by the lexicographical ordering <*Rad*, -*O*, *ID*>, where *Rad* is the radius of the cluster, *O* is the number of overlaps of the cluster with other equal radius clusters, and *ID* is the identity of the clusterhead. The rule for *cluster prioritization* is that the cluster with maximum radius has highest priority, followed by the cluster with minimum number of overlaps, and finally by the cluster with maximum ID.

For overlapping clusters j and k (shown in Figure 8), if the priority of cluster j is lower than cluster k, then cluster j reduces its radius by one, i.e., all the nodes at the periphery of cluster j relinquish the cluster. This allows the higher priority cluster k to subsume the overlapping nodes of cluster j (in this case node $l2$) thus incrementing its radius. The lower priority cluster j is either completely subsumed by a neighboring cluster, say k, or it grows again if all the nodes neighboring $j's$ cluster can join the cluster of j. This is the case when cluster k (to which cluster j lost) in turn loses to some other higher priority cluster thus decrementing its radius and allowing cluster j to grow again.

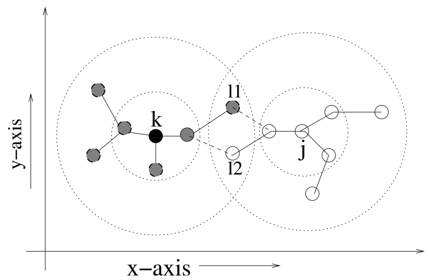


Figure 8. $l1$ and $l2$ join $k's$ and $j's$ cluster respectively, leading to overlap between the two clusters.

3.1 LOCI program actions

Concretely, the LOCI program for each node j is as follows:

$$
\begin{array}{l}
\textbf{(1)}\ timeout(d.j < 0 \ \wedge \ \ (\forall l : l \in nbr.j : d.l < 0 \ \vee \\
\qquad\qquad\qquad\qquad\qquad (d.l > 0 \wedge dist.j.(c.l) > mR))) \\
\\
\qquad\qquad \longrightarrow \ d.j, r.j, c.j, o.j := 0, 0, j, \phi \\
[] \\
\textbf{(2)}\ k \in nbr.j \ \wedge \\
\qquad ((d.j < 0 \ \ \wedge \ \ d.k \geq 0 \ \ \wedge \ \ dist.j.(c.k) \geq d.k \ \ \wedge \\
\qquad\qquad\qquad (r.k \leq dist.j.(c.k) \leq r.k + 1 \ \ \vee \\
\qquad\qquad\qquad\qquad (r.k = R \ \ \wedge \ \ dist.j.(c.k) \leq mR))) \\
\qquad \vee \ (d.j > R \wedge 0 \leq r.k < R \wedge dist.j.(c.k) \leq r.k + 1 \\
\qquad\qquad\qquad \wedge \ dist.(c.j).(c.k) > 2R) \\
\qquad \vee \ (d.j > R \ \ \wedge \ \ r.k = R \ \ \wedge \ \ dist.j.(c.k) < d.j)) \\
\\
\qquad\qquad \longrightarrow \ d.j, r.j, c.j, o.j := dist.j.(c.k), r.k, c.k, \phi \\
[] \\
\textbf{(3)}\ k \in o.j \ \wedge \\
\qquad\quad k \neq c.j \ \wedge \ d.j = 0 \wedge r.j < R \wedge |\{o.j\}| > 0 \\
\qquad\qquad \longrightarrow \text{if } priority.j < priority.k \text{ then } shrink.j
\end{array}
$$

In the above program, each node j maintains the following variables:

- $d.j$: distance of node j from the clusterhead of its cluster. A value of 0 denotes that j is the clusterhead, a value greater than 0 denotes that j belongs to some cluster, and a value less than 0 denotes that j does not belong to any cluster. The domain of $d.j$ is [-1, mR];
- $r.j$: circular radius of the cluster to which j belongs;
- $c.j$: ID of the cluster to which j belongs; and
- $o.j$: set of tuples that contains IDs of overlapping clusters and their circular radius, i.e., $o.j(t_k)$ = $<ID_k, r.ID_k>$; $o.j$ contains the IDs of only those clusters with equal or higher radius.

The first action formally defines the conditions under which a node j elects itself as the leader and is thus a trivial implementation of task (1) of LOCI abstract program.

The second action formally defines the conditions under which a node j joins cluster of neighboring node k, and is thus a straightforward implementation of task (2) of LOCI abstract program: each disjunct in the guard corresponds to the three respective cases explained previously.

The third action enables clusterhead j to resolve overlaps of its cluster with a neighboring clusterhead k. The $priority.j$ module enables clusterhead j to compute its priority with respect to other overlapping clusters. The $shrink.j$ module enables j to decrement the radius of its cluster by one, if $j's$ cluster overlaps with a higher priority cluster.

Module priority.j. Clusterhead j computes priority of its cluster with respect to other overlapping clusters from $o.j$ variable. Each peripheral node j' that belongs to the cluster of j updates $o.j'$ by the following action:

$$\boxed{\begin{array}{l} k \in nbr.j' \wedge k \notin o.j' \wedge d.j' \geq 0 \wedge d.k > 0 \wedge r.j' < R \wedge \\ r.j' \leq r.k \;\; \wedge \;\; dist.(c.j').k \leq r.j' + 1 \;\; \wedge \;\; c.j' \neq c.k \\ \qquad\qquad\qquad\qquad \longrightarrow o.j' := o.j' \cup <c.k, r.k> \end{array}}$$

The value stored in $o.j'$ variable at each peripheral node j' is propagated to the clusterhead j via an information feedback wave [10] (for efficiency the feedback wave may construct a spanning tree T of the cluster rooted at the clusterhead). If $j's$ cluster overlaps with another cluster of higher radius, j employs $shrink.j$ module to decrement its radius. Otherwise, j broadcasts the number of overlaps of its cluster via an information feedback wave to the peripheral nodes of the cluster, which enables neighboring clusters to compare their priorities with $j's$ cluster. Identically, it allows clusterhead j to compare its priority with other overlapping clusters of equal radius. The highest priority cluster then increments its radius and other lower priority clusters decrement radius of their clusters by employing $shrink.j$ module (which we explain next).

Module shrink.j. The $shrink.j$ module decrements the radius of the cluster by one (the peripheral nodes reset their variables), again via an information feedback wave. As above, for efficiency the information feedback wave might construct a spanning tree T. Specifically, the information is propagated from the clusterhead to all of its children, and each internal node of the tree after receiving the information from its parent, forwards it to all of its children until it reaches the peripheral nodes.

In a similar fashion, when a cluster increments its radius, the clusterhead broadcasts the new value of the circular radius $r.j$ to other nodes in the cluster. Notice that we allow the maximum value of the circular radius $r.j$ to be R to avoid updating the value of all the nodes of a legitimate cluster if it grows beyond R.

Since information feedback wave is a standard procedure [10], for the sake of brevity we do not present it in the paper.

3.2 Proof of correctness

Our proof proceeds by defining some state predicates for identifying an invariant I and a fixpoint F.

- $r.j$ denotes circular radius of the cluster.

  $r.j \equiv max_{1 \leq a \leq R}\{a : (\forall\ l \mid l \in N_a(c.j) : c.l = c.j)\}$

- $H.j$ denotes the predicate that the variables stored at a node j that belongs to some cluster $c.j$ have correct values.
  That is, $H.j \equiv$

  $(d.j = dist.j.(c.j)\ \wedge\ r.j = r.(c.j)\ \wedge$
  $(d.j \leq R \Rightarrow r.j \geq \lceil (d.j\text{-}1) \rceil \wedge (\forall k : k \in o.j : d.k = 0\ \wedge\ r.k \geq r.j \wedge dist.k.(c.j) \leq r.k + r.j + 2)) \wedge$
  $(d.j > R \Rightarrow d.(c.j) = 0\ \wedge\ r.j = \mathrm{R}\ \wedge\ |\{o.j\}| = 0\ \wedge\ (\forall k : k \in nbr.j\ \wedge\ c.k \neq c.j\ \wedge\ r.k = R :$
  $d.k > R\ \Rightarrow\ d.j \leq dist.j.(c.k)\ )))$

  The first and second conjunct mean that the distance of node j from its clusterhead is correct, and j and its clusterhead have the same value of circular radius $r.j$. The third conjunct means that if j is within R-neighborhood of its clusterhead then the value of circular radius $r.j$ is greater than distance of j from its clusterhead minus 1, and j contains only the identities of neighboring overlapping clusters. The fourth conjunct means that if $j's$ distance from its clusterhead is greater than R, then j must belong to a legitimate cluster and j must also belong to the cluster of the nearest clusterhead.

- $VC.j$ denotes that j is the clusterhead of a "valid" cluster.
  $VC.j \equiv (\forall l \in N_{min(r.j,R)}(j) : c.l = j\ \wedge\ H.l)$

- $LP.j$ denotes a "legitimate" path from a node j to its clusterhead such that all nodes in the path belong to the same cluster and the distance of a node in the path to the clusterhead is always less than the distance of the preceding node from the clusterhead. Formally,

  $LP.j\ \equiv\ (\exists k : k \in nbr.j : c.k = c.j\ \wedge\ H.j\ \wedge\ (d.k = 0\ \vee\ (d.k < d.j\ \wedge\ LP.k)))$

- $LC.j$ denotes that j is the clusterhead of a legitimate cluster.

  $LC.j \equiv (\forall k : k \in N_R(j) : c.k = j \ \wedge \ |\{o.k\}| = 0 \ \wedge \ H.k)$

Invariant. $I \equiv (\forall j : (d.j = 0 \Rightarrow VC.j) \ \wedge \ (d.j > 0 \Rightarrow H.j))$

I states that if a node belongs to a cluster then that cluster is a valid cluster and the values of all the nodes in that cluster are correct.

Theorem 1: I is an invariant of LOCI. □

Fixpoint. Let **F** $\equiv (\forall j : d.j \geq 0 \ \wedge \ (d.j = 0 \Rightarrow LC.j) \ \wedge \ (d.j > 0 \Rightarrow |o.j| = 0 \wedge LP.j))$

F implies that all nodes belong to a legitimate cluster and there is a legitimate path from every node to its clusterhead.

Theorem 2: *F* is a fixpoint of LOCI. □

Theorem 3: Starting from an invariant state *I*, the system eventually reaches a state in *F*. □

Theorem 4: The convergence time of LOCI from invariant state to fixpoint state is O(R^4) rounds. □

Proof sketch: We proceed by first showing that in the worst case within $O(R^2)$ distance of any node in the network, there exists at least one legitimate cluster in $O(R^2)$ rounds of node execution.

Every node either elects itself as the clusterhead (action 1 of LOCI program) thus forming a new cluster or joins a neighboring cluster (action 2) thus incrementing the radius of the cluster. Increase in cluster radius may lead to overlaps with other clusters (as depicted in Section 3). First, we consider the case when clusters of equal radius overlap (the number of overlaps is bounded by a constant). Since every cluster has a unique priority there exists at least one cluster amongst overlapping clusters with the highest priority. The highest priority cluster subsumes the overlapping nodes of neighboring clusters thus further increasing its priority (as the number of overlaps decreases). Since the number of overlaps cannot further increase and it eventually decreases, a cluster of radius r ($r < R$) eventually increments its radius to $r + 1$.

If a cluster of radius r overlaps with another cluster of radius r' where $r' > r$ ($r' < R$), then radius r' cluster will either eventually increment its radius to $r' + 1$ (by resolving overlaps) or will lose to a cluster with radius r'' where $r'' > r'$. Since the circular radius of the clusters is bounded by R eventually a cluster will grow to R and form a legitimate cluster. Furthermore, once a cluster becomes legitimate it remains legitimate.

Thus, the above chain of one cluster losing to another is guaranteed to be bounded by $\sum_{r=1}^{R} 2 * c_r * r = O(R^2)$ where c_r is the number of overlaps in the range (6, 12] ($\approx 2\pi(1+1/r)$) for radius r. Thus, in every circle of radius $O(R^2)$ there exists at least one legitimate cluster.

A legitimate cluster can further grow till distance mR and no node within distance $2R$ of the clusterhead of the legitimate cluster can become a clusterhead. Thus an autonomous cluster that is not yet legitimate cannot always lose to a legitimate cluster.

In the slowest convergence, all the legitimate clusters are formed at the circumference of a circular region of radius $O(R^2)$. Since there can be O(R) such concentric circles within a circle of radius $O(R^2)$, and $O(R)$ legitimate clusters on the circumference of such circles, the convergence time is $O(R^2) * O(R) * O(R) = O(R^4)$. □

In Appendix A3, assuming that nodes have directional information sending capability, we show how to improve the convergence time of LOCI from $O(R^4)$ to $O(R^2)$. We achieve this by letting legitimate clusterheads assign cluster-leaders in their neighboring regions and by giving higher priority to these assigned clusters over autonomous clusters.

3.3 Regional healing using LOCI

Regional healing for node joins/fail-stops. When a new node joins the network it is subsumed by existing legitimate clusters if it is within mR distance of any neighboring clusterhead. Otherwise, the new node forms its own legitimate cluster by subsuming nodes that are not within R-neighborhood of clusterheads of neighboring legitimate clusters. Thus node joins are handled locally as only neighboring legitimate clusters are affected. Notice that legitimate clusters remain legitimate even in presence of node joins.

When nodes fail-stop within a cluster the effect of faults may either be contained within the same cluster or they may lead to collapse of the whole cluster, e.g. clusterhead failure. The remaining nodes of the cluster then either join neighboring legitimate clusters or form new legitimate clusters by subsuming nodes from neighboring legitimate clusters (as above). Thus in presence of node fail-stops only neighboring legitimate clusters are affected and hence faults are contained locally, i.e., spatially local failures disrupt an area proportional to the diameter of the failure. In presence of node fail-stops that globally partition the network, LOCI constructs legitimate clusters in each partition. Since it is impossible for the information to pass between the two partitions, nodes that are within R-neighborhood of a clusterhead but do not lie in the same partition of the network form clusters independently.

Furthermore due to arbitrary node fail-stops, a clusterhead may not be able to reach a node within its R-neighborhood without routing through nodes of another cluster. In these scenarios, such nodes either join a neighboring legitimate cluster or form their own clusters.

Regional healing for node state corruption. When local state at nodes is corrupted within a small region ($< O(R^2)$), the clusters affected either heal without affecting neighboring clusters or the clusters of the corrupted nodes collapse and as above only neighboring legitimate clusters are affected. If a large region ($> O(R^2)$) is corrupted, the effect of state corruption is contained within $O(R^2)$ distance, i.e., for large failures (node fail-stops or node state corruption) the effect of faults is contained within $O(R^2)$ distance of the failed node, independent of the extent of the failures.

3.4 Proof of self-stabilization

In the presence of faults, variables stored at any node may be arbitrarily corrupted. The stabilizing action for LOCI is to reset $d.j$ to -1 if any of the conditions hold: j is farther than $r.j + 1$ (as in the first disjunct of the guard); $r.j$ is corrupted (as in the second disjunct); there is no legitimate path from j to its clusterhead (as in the third disjunct); and variables $d.j$ or $r.j$ are corrupted (as in the fourth disjunct). That is, a node j disassociates itself from its current cluster when it detects a violation of the invariant.

$$
\begin{array}{l}
k \in nbr.j \;\wedge \\
(\quad d.j > r.j + 1 \\
\vee\; (d.j \geq 0 \;\wedge\; c.j \neq c.k \;\wedge\; r.j > d.j) \\
\vee\; (d.j > 0 \;\wedge \neg(\exists l \in nbr.j : 0 \leq d.l < d.j \wedge c.l = c.j)) \\
\vee\; (d.j > 0 \;\wedge d.k \geq 0 \wedge c.k = c.j \wedge (d.j \neq dist.j.(c.k) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee\; r.k \neq r.j))) \\
\qquad\qquad \longrightarrow \quad d.j := \text{-1}
\end{array}
$$

Theorem 5: Starting at an arbitrary state, every computation of the system reaches a state in *I* in $O(R)$ rounds. $\square$

For the proof of the theorem, we give a variant function on the number of invalid clusters in the network. We first show using the LOCI program actions that the number of invalid clusters never increases. We then prove that this number eventually decreases, since according to the stabilization action all the nodes in an invalid cluster relinquish the cluster within $O(R)$ rounds. We present the proof in the Appendix.

4 Simulation results

We have implemented LOCI on the Mica2 mote platform. To measure the effectiveness of LOCI, we simulate LOCI in an event-driven simulator Prowler [21], that simulates TinyOS environment and runs under MATLAB. We evaluate performance of LOCI in terms of the number of clusters constructed, convergence time and communication overhead; and compare LOCI with a theoretically optimum clustering scheme with global information.

To evaluate LOCI for different network sizes, we simulate 64-node, 256-node, and 625-node networks in both 1-D and 2-D network space with uniform distribution of nodes. For 1-D network space we consider 64 unit, 256 unit, and 625 unit area respectively; and for 2-D network space we consider 8 square, 16 square, and 25 square unit area respectively. Nodes in our simulations have unit transmission range. We simulate LOCI for constructing clusters of bounded radius interval $[R, 2R]$ with values of $R \in \{1, 2, 4, 8\}$. All our simulation results are averages of five runs.

Tables 2 and 3 below show the ratio of the number of clusters constructed by LOCI to the minimum number of clusters theoretically feasible for different values of radius R and number of nodes N in 1-D and 2-D networks respectively. Our simulation results (Table 2) demonstrate that the overhead in number of clusters is only 1.3 for LOCI for a 1-D network (the maximum overhead feasible is 2.0). For a 2-D network overhead for LOCI is 2.2 (the maximum overhead feasible is 4.0). Thus, in practice, the number of clusters constructed by LOCI are much less than the worst case. Notice that even in the worst case, LOCI bounds the number of clusters constructed by a constant. From our simulations, we observe that the overhead in number of clusters is independent of the radius R and size of the network N.

Number of nodes (N) Radius of the clusters (R)

N =	R=1	R=2	R=4	R=8
64	1.31	1.29	1.41	1.50
256	1.36	1.32	1.34	1.31
625	1.37	1.35	1.38	1.30

Table 2: Overhead in number of clusters (1-D network).

Number of nodes (N)	Radius of the clusters (R)			
N =	R=1	R=2	R=4	R=8
64	2.21	2.15	2.00	1.00
256	2.00	1.99	2.15	2.00
625	1.89	2.12	2.20	2.25

Table 3: Overhead in number of clusters (2-D Network).

During our simulations we observed that although small timeout values for local leader election allowed faster cluster formation they also led to more overlaps with neighboring clusters which slowed down the cluster formation (as clusterheads had to resolve overlaps). Thus timeouts for local leader election should be a function of radius R and network size, and should have a uniform distribution to minimize the overlaps and hence waiting time of a cluster.

In the Appendix, we give a simple mathematical worst case analysis for LOCI in terms of the percentage of uncovered nodes (for $m < 2$), number of clusters constructed, and communication complexity.

5 Hierarchical LOCI

Thus far we discussed partitioning of the network at only one level. LOCI allows multi-level partitioning in a bottom-up fashion. After the lowest level, i.e., level 0 partition is constructed, level 1 partitioning is achieved as follows. Clusters at level 0 are represented by their respective clusterheads. Since clusterheads of neighboring clusters, which are separated by $[2R, 2mR]$ distance, consider themselves as logical neighbors[1] at level 1, we redefine unit distance at level 1 to be $2mR$. Thus, the radius of the clusters at level 1 is

[1]This logical neighborhood relationship is implemented either via an underlying routing service or via distance constrained flooding that enables neighboring clusterheads to communicate.

$[(2mR)R, (2mR+1)mR]$. This process is repeated at higher levels forming clusters of bounded radius range $[(2mR)^i * R, (\frac{(2mR)^{i+1}-1}{2mR-1}) * mR]$ at each level i.

Stability. Changes at level i of the hierarchy should be contained within level i. We achieve this with high probability as follows. From our hierarchical construction, a level i clusterhead is also a clusterhead for every level up to i. When a level i clusterhead v fails, LOCI is executed to form a new cluster at the lowest level. The new cluster formed is represented by its clusterhead, say v', at the next upper level, and with high probability is elected as the clusterhead up to level i as it is closer to the failed clusterhead v (clustering hierarchy is constructed bottom-up). In a few pathological cases, it may be possible to percolate this reasoning up to a few levels, but with very high probability there will be a lower level i' whose clustering will be unaffected by the failure of v. Therefore a clusterhead failure at a level i is dealt locally by assigning this new clusterhead at level i' as the clusterhead of all the levels up to i.

6 Application: tracking in sensor networks

In this section, we discuss an application of LOCI in the context of pursuer-evader tracking in sensor networks where the objective is to construct a distributed indexing structure that enables a pursuer to query the location of an evader.

In [11], we have presented a self-stabilizing program for tracking in sensor networks. In our program, a dynamic "tracking tree" structure that is always rooted at the evader is maintained by the nodes collectively. To catch the evader, the pursuer follows the tree structure to the root: the pursuer queries the closest node about its parent, proceeds to the parent node, and continues in this fashion to eventually reach the root node (and hence the evader)[2]. This program achieves locality —hence, is scalable— with respect to the find operations as a find invoked within d distance of the evader requires $O(d)$ communication cost to return the location of the evader. However, it fails to achieve locality —hence, is not scalable— with respect to the move operations because regardless of the distance travelled by the evader the entire tracking tree is updated resulting in $O(N)$ communication cost, where N is the number of nodes in the network.

[2]We have implemented this program on the Berkeley's Mica mote platform [15] for a demonstration at the

We have recently [12] found that by employing a hierarchical partitioning using LOCI it is possible to maintain the tracking tree structure over a small number of nodes and with accuracy proportional to the distance from the evader; thus achieving locality and scalability with respect to both find and move operations.

More specifically, our tracking tree is now a *path* (each node has at most one child) rooted at the lowest level clusterhead where the evader resides. The idea of maintaining information at farther away nodes with lesser accuracy is achieved by maintaining the tracking path at increasingly higher level clusterheads as the distance from the evader increases, e.g., while the root is a level 0 clusterhead, the leaf is the clusterhead of the topmost level clustering. Each node in the tracking path points to a node (i.e., its parent in the path) that is closer to the evader, and hence that has more recent and more refined information about the location of the evader.

A find operation invoked at a node results in querying the clusterhead of the node and its neighbors at increasingly higher levels of the clustering hierarchy until a node in the tracking path is encountered. Once the tracking path is discovered, the find operation follows it to its root to return the location of the evader. Our find operations are local: a find operation invoked at a node d away from the object is guaranteed to hit the tracking path at level $\log_{2mR}(d)+1$ of the clustering hierarchy and hence requires $O(d)$ amount of communication cost. Our move operations are also local: a move of the evader to a distance d away requires updating only the lower $log_{2mR}(d) + 1$ levels of the tracking path, and hence $O(d)$ communication cost.

7 Related work

A number of clustering protocols have been proposed recently for constructing one-hop and multi-hop clusters in wireless networks [4, 6, 14]. Traditionally the protocols are evaluated in terms of the size of the clusters, number of clusters, convergence time, and communication overhead. In this thesis, we have evaluated LOCI by these metrics for comparison, and added

June 2002 DARPA–NEST retreat held in Bar Harbor, Maine. In our demonstration, a Lego Mindstorms^{TM} robot serving as a pursuer used our program to catch another Lego Mindstorms robot serving as an evader in a 4 by 4 grid of motes subject to a variety of faults. The source code and video shots for the demo are available at `www.cis.ohio-state.edu/~demirbas/peDemo`.

measures of scalability and local self-healing as well.

Cluster size and number. In [4], a distributed and scalable clustering scheme, Max-Min D-cluster, is presented that partitions the network into d-hop clusters, i.e., the farthest distance of a node from its clusterhead is at most d-hops. In this scheme, every node initiates $2d$ rounds of flooding, whereby at the end of the first d rounds the node with the maximum ID is elected as the leader. Then by using the values in the last d rounds, that exist at each node after the end of the first d rounds, a new leader is elected. The scheme has the advantage that it does not assume a connected network or a dense node deployment. The communication overhead and convergence time are also minimal. However, it does not guarantee the number of clusters formed, in fact in the worst case, the number of clusters generated may be equal to the number of nodes in the network (for a connected network).

In [14], a distributed and probabilistic clustering scheme, LEACH, is presented that forms 1-hop clusters. Nodes elect themselves as clusterheads based on a probabilistic function and broadcast their decisions. Each non-clusterhead node determines its cluster by choosing the clusterhead that requires the minimum communication energy. The energy load of being a clusterhead is evenly distributed among the nodes by incorporating randomized rotation of the high-energy clusterhead position among the nodes. Although the communication overhead and convergence time is minimal, the protocol offers no guarantee about the placement and/or number of clusterhead nodes.

Furthermore, since clustering is repeated periodically the scheme is implicitly fault-tolerant. The scheme is initially presented only for a 1-hop network where all nodes are within communication range of each other and a pre-determined node called the base station. This assumption is later relaxed by using collision-avoidance techniques and multihop routing approach.

In [20], a fault-tolerant and scalable clustering algorithm for amorphous computers is presented. The clustering algorithm, Clubs, forms 1-hop clusters. In this algorithm, a node waits for a random amount of time before electing itself as the leader and then sends a join request to all of its neighboring nodes to form a 1-hop cluster. If any of the neighboring nodes of the new leader is a leader, i.e. leaders are in communication range of each other, then both the clusters are collapsed and the process of electing leaders via random timeouts is repeated. A cluster can overlap with at most 24 other neighboring clusters, and an individual node can be a member of at most 9 clusters.

Scalability. In [23], a clustering scheme, GS^3, is presented that generates an approximate hexagonal close packing of clusters, with clusterheads at the center of the hexagons. In this scheme, a single pre-determined node starts a diffusing computation and selects nodes at/around pre-determined locations around it as clusterheads. The elected clusterheads, in turn, elect other nodes around them as clusterheads until the whole network is covered. To enable clusterhead assignment at pre-determined locations, directional information and distance estimation capability is assumed at each node and the network is initially assumed to be connected with additional requirement of dense node deployment. The number of clusters it yields and the communication overhead it entails is minimal, and convergence time is of the order of the diameter of the network. For self-healing it is assumed that regions in and around the clusterheads fail simultaneously such that the hexagonal structure can slide as a whole forming a new non-overlapping hexagonal structure.

In [6], a clustering algorithm is presented that forms bounded size (based on the number of nodes) clusters. The algorithm proceeds by first finding a rooted spanning tree of the network and then forming desired clusters from the subtrees. The algorithm gives a bound on the number of clusters constructed, and the communication cost is minimal. The convergence time of clustering the network is of the order of the diameter of the network. The scheme is locally fault-tolerant to node failures and joins but may lead to re-clustering of the entire network for some pathological scenarios.

In [8], a robust hierarchy is constructed via clustering in an amorphous network [1] by using persistent nodes [7]. The clusters formed have the property that all the nodes that are within r/2 hops of the leader belong to the same cluster as that of the leader and the farthest distance of any node from its leader is 3.5r hops. Any node that is farther than 2.5r hops from the nearest leader, elects itself as the leader with probability $\frac{2}{\rho \cdot (3r)^{(D+1)}}$ in every round, where ρ is the density of the nodes, and D is the dimensionality of the network. Once a leader is elected, it broadcasts this information up to 3.5r hops. If two leaders are within 2r hops of each other then the higher ID leader dominates the lower ID leader by destroying the lower ID leader and its cluster. The clustering scheme has the property that it avoids edges and congestion in the network. Every node that is within 2r hops of its leader, computes a heuristic value. The leader based on the heuristic values of its neighbors assigns the neighbor with the highest heuristic value as the new leader thus migrating the leader and hence the whole cluster, thus shifting the cluster away from edges and congestion. With high probability network cover is constructed in $O(r)$ rounds and communication overhead is $O(r^3)$.

Self-healing. Extant schemes that do not specifically deal with clustering but with fault locality are either local in time but not in space [17] or local in space but not in time [3] or none [5]. Convergence time and self-healing in LOCI are local both in time and space.

In [18], nodes in a wireless ad hoc network tune themselves to the minimum transmission power that guarantees connectivity of the network thus saving energy. In [9, 22], nodes that are redundant from a routing perspective are switched-off thus conserving energy. Since these schemes are power saving techniques, they are orthogonal to our work.

8 Concluding remarks

The properties of our clustering approach that make it suitable for large scale wireless sensor networks are its: *(1)* locality, in that each node is only affected by nodes within distance $2R$, *(2)* scalability, in that each node maintains only constant state, and its execution time is $O(R^4)$ rounds ($O(R^2)$ rounds is achievable with the cluster-leader assignment procedure) independent of network size, and finally *(3)* self-healing, in that it is self-stabilizing and tolerates node failures, joins and state corruptions locally.

More specifically, LOCI accomplishes our criteria for efficient clustering in the Introduction as follows: (i) constructing clusters of equal size for load balancing and uniformity of energy dissipation —LOCI constructs clusters such that all nodes within R-neighborhood of the clusterhead belong to the same cluster as that of the clusterhead, and additionally forms a Voronoi tessellation; (ii) minimizing the overlaps between clusters for minimizing energy requirements —LOCI constructs clusters with no overlaps; (iii) minimizing the number of orphan nodes for maximizing available resources —LOCI results in no orphan nodes; (iv) minimizing communication overhead for saving energy (for increased network lifetime) —LOCI has $O(R)$ communication overhead; (v) minimizing the number of clusters for minimizing memory requirements —LOCI gives a constant upper bound on the number of clusters constructed; and (vi) scalable cluster formation and scalable self-healing for minimizing space and time requirements and maximizing tolerance to faults —LOCI constructs clusters in time independent of the size of the network and heals locally.

The above properties were achievable in the geometric setting described in this paper, intuitively because independently growing clusters can have no more than a constant number of overlaps with neighboring clusters of equal radius. This fact suggests a prioritization of clusters such that only higher priority cluster survive/grow when there are overlaps, and eventually clusters of radius at least R are formed. We also showed that for guaranteeing both a partitioning of the network and local self-healing in presence of arbitrary node joins and fail-stops, clusters must have a range m of at least 2 in their radii.

Although we presented LOCI only in the context of a 2-D coordinate plane, the extension to the 3-D case is straightforward. In a 3-D coordinate space, a cluster would grow in the shape of a sphere (so as to have no more than a bounded number of overlaps with other clusters of same size) and increment its radius till the sphere grows till radius R. Clusters would resolve the overlaps just as they did in a 2-D plane, and the same properties of LOCI would hold.

For our Poisson distribution model wherein with high probability every node has a neighbor in each $60°$ cone, we can relax the model assumption that nodes have the capability of distance estimation with respect to neighbors. In this model variant, the clusters are grown on the basis of logical hops instead of geographic distance. That is, in the first round, all nodes within 1-hop (transmission radius) of the leader node would join its cluster, and in the next round, nodes that are within 2-hops of the cluster leader would join the cluster, and so on for R rounds.

It is also possible to relax our Poisson distribution density model. For one, LOCI works without any modification in the following weaker density model: for any two non-neighboring nodes j and k in a connected subgraph, there exists a neighbor l of j such that $dist.l.k < dist.j.k$. In other words, for any two connected nodes there exists a *local path*. LOCI can also be modified to work under $150°$ cone model [18], although a slightly more sophisticated program is needed since the local path assumption may not necessarily hold in this model.

For simplicity, we presented LOCI in a shared memory model and assumed that a node can read all of its neighbors values at once and also write its own variables at once. LOCI is easily refined to a lower atomicity model, for example, our LOCI implementation [19] is in a message passing model.

Future work. Our group [2] has developed a tracking service called "Line in the Sand" for the DARPA/NEST project. We are currently working on integrating LOCI and the hierarchical tracking program [12] to achieve scalability and fault-local stabilization in our "Line in the Sand" tracking service.

We are also investigating the role of geometric, local clustering in designing efficient data structures for evaluation of spatial queries in the context of sensor networks.

References

[1] Amorphous computing. http://www.swiss.ai.mit.edu/projects/amorphous.

[2] Stabilization in NEST. http://nest.netlab.ohio-state.edu.

[3] Y. Afek and S. Kutten. The local detection paradigm and its applications to self-stabilization. *PODC*, pages 199–229, 1997.

[4] A. Amis, R. Prakash, T. Vuong, and D.Huynh. Max-min d-cluster formation in wireless networks. *In Proceedings of IEEE INFOCOM*, 1999.

[5] A. Arora and M. G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[6] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. *IEEE INFOCOM*, 2001.

[7] J. Beal. Persistent nodes for reliable memory in geographically local networks. *AI Memo 2003-011, MIT*, 2003.

[8] J. Beal. A robust amorphous hierarchy from persistent nodes. *AI Memo 2003-011, MIT*, 2003.

[9] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Proceedings of the seventh annual international conference on Mobile computing and networking*, pages 85–96, 2001.

[10] A. Cournier, A.K. Datta, F. Petit, and V. Villain. Self-stabilizing pif algorithms in arbitrary networks. *International Conference on Distributed Computing Systems (ICDCS)*, pages 91–98, 2001.

[11] M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. *Proceedings of the Sixth Symposium on Self-Stabilizing Systems(SSS'03)*, pages 1–16, June 2003.

[12] M. Demirbas, A. Arora, T. Nolte, and N. Lynch. Self-stabilizing hierarchical tracking service for sensor networks. Technical Report OSU-CISRC-4/03-TR19, The Ohio State University, April 2003.

[13] N. Francez. Fairness. *Springer-Verlag*, 1986.

[14] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. Application specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Networking*, 2002.

[15] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS*, pages 93–104, 2000.

[16] M. Jayaram and G. Varghese. Crash failures can drive protocols to arbitrary states. *ACM Symposium on Principles of Distributed Computing*, 1996.

[17] S. Kutten and D. Peleg. Fault-local distributed mending. *PODC*, pages 20–27, 1995.

[18] L. Li, J. Y. Halpern, P. Bahl, Y. M. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. *PODC*, pages 264–273, 2001.

[19] V. Mittal, M. Demirbas, and A. Arora. Loci: Local clustering service for large scale wireless sensor networks. *Technical report OSU-CISRC-2/03-TR07, The Ohio State University*, February 2003.

[20] R. Nagpal and D. Coore. An algorithm for group formation in an amorphous computer. *Proceedings of the Tenth International Conference on Parallel and Distributed Systems (PDCS)*, October 1998.

[21] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. *IEEE Aerospace Conference*, March 2003.

[22] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. *ACM Mobicom*, pages 70–84, 2001.

[23] H. Zhang and A. Arora. GS^3: Scalable self-configuration and self-healing in wireless networks. *PODC*, pages 58–67, 2002.

Appendix

A1. Statistical density and fault model

We assume that nodes are distributed randomly as per a spatial Poisson distribution such that at an average within a unit area circular region there are λ nodes. Thus the probability of occurence of α nodes within a unit radius circle is $P(\alpha, \lambda A) = e^{-\lambda A}(\lambda A)^{\alpha}/\alpha!$, where A is the area of the region. For example, assuming that the expected value of λ is at least 1, the probability that no nodes occur in a unit radius circle is $e^{-\pi}$ (= 0.043). Thus with 0.957 probability there will be multiple nodes within a unit radius circle.

For iterative growth of any cluster with radius r, $r < R$, all nodes within $(r+1)$-neighborhood of the clusterhead should join the cluster before the next iteration. Thus for local cluster formation, for every node k that is within $(r + 1)$-neighborhood of the clusterhead there must exist a node j within r-neighborhood of the cluster leader such that the length of the path from j to k is some constant function of the distance $dist.j.k$ between the nodes. More specifically, this is guaranteed if for any two non-neighboring nodes j and k in a connected subgraph that are at most R distance apart, there exists a neighbor l of j such that $dist.l.k < dist.j.k$. We denote such a path as a *local* path.

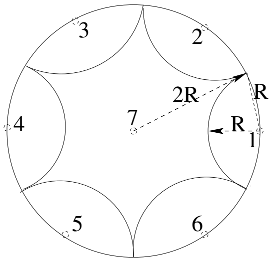
Of course, the local path assumption may not hold everywhere in the network, i.e., there may exist empty regions of area A in the network that violate it. In particular, if there are n legitimate clusters constructed by LOCI, the expected number of illegitimate clusters constructed are approximately,
$n \cdot (e^{\lambda A/2} + 2 \cdot \sum_{r=1}^{R} (1 - e^{(-\lambda A/8)(r-1)^2}) \cdot e^{(-\lambda A/8)r^2})$.

A2. Analytical performance analysis of LOCI

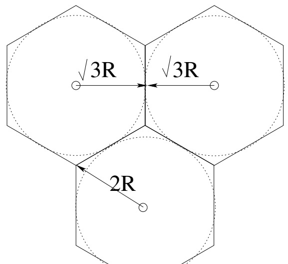
In this Section, we first present a simple *worst case* mathematical analysis of the percentage of uncovered nodes, for $m < 2$, for any clustering scheme in presence of arbitrary node joins and fail-stops. We then present, for LOCI, a worst case analysis of the overhead in the number of clusters constructed, and communication complexity.

Percentage of uncovered nodes. For $m \geq 2$, LOCI partitions the network into $[R, mR]$ radius clusters. In presence of arbitrary node joins and failures, we present a worst case analysis of the percentage of nodes that do not belong to any legitimate cluster, irrespective of the clustering scheme used, for different values of $m \leq 2$ (Table 1). More specifically, the following scenario illustrates a possible construction that results in minimum coverage, i.e., tiling an infinite region of 2-D space with equilateral triangles with sides of length $2\sqrt{3}R$. The cluster leaders are at the vertices of the triangles and their clusters cover a circular region of radius R around the vertices with maximum radius of any cluster being mR. Nodes lying in the uncovered regions can neither form nor join a legitimate cluster.

m	1.0	1.2	1.5	1.7	2.0
p	69.8%	56.5%	32.1%	12.7%	0.0%



Circular region of radius 2R. LOCI constructs seven clusters in the worst case.
(a)



(b)

Table 1: Percentage p of uncovered nodes for $m \leq 2$
Figure 9.

Number of clusters constructed. Since $m = 2$ is the minimum value of m that guarantees partitioning of the network, we consider clustering schemes that construct clusters of bounded radius range $[R, 2R]$. We compute the overhead in number of clusters constructed by LOCI compared to a globally optimum scheme that results in minimum number of clusters. The following scenario illustrates a possible construction. Consider a continuous circular region of radius 2R (Figure 9(a)). The globally optimum scheme constructs only one cluster with clusterhead at the center of the region. In the pathological case, LOCI constructs seven clusters as shown in Figure 9(a) where one clusterhead is at the center of the circular region forming a cluster of circular radius R, and six clusterheads are at the circumference of the circular region forming semi-circles of circular radius at least R. thus resulting in an overhead of seven[3]. In our simulations results we observe that even for large values of R compared to the size of the network ($R = 8, N = 64$), LOCI has overhead of only 1.5 for a 1-D network and 1.00 for a 2-D network.

For a continuous region of space and small values of R compared to the size of the network, the overhead in number of clusters constructed by LOCI is 3. More specifically, for the centralized clustering scheme the following best case scenario illustrates a possible construction that results in partitioning of the network with minimum number of clusters, i.e., tiling an infinite region of 2-D space with hexagons of radius $2R$ (and circular radius $\sqrt{3}$ R) with clusterheads at the centers of the hexagons (Figure 9(b)). (Notice that to construct network partitioning, all clusters in the network cannot be of circular radius $2R$ (Figure 10)). Next, consider the following worst case construction where LOCI partitions the network with maximum number of clusters by tiling an infinite region of 2-D space with hexagons of radius $2/\sqrt{3}$R and circular radius R with clusterheads at the centers of the hexagons. Our simulation results discussed in Section 4 show that practically LOCI has an overhead of only 1.3 for a 1-D network and 2.2 for a 2-D network.

Communication complexity. In Section 3, we presented LOCI in a shared memory model, here in order to compute the communication overhead incurred by LOCI, we analyse it in a message passing model.

We assume that the number of nodes neighboring a cluster increase as a linear function of the radius of the cluster. In LOCI, since cluster growth is iterative, to grow till radius r, $r < R$,

[3]Notice that, this is the case only for large values of R whereby a single cluster can cover the whole network.

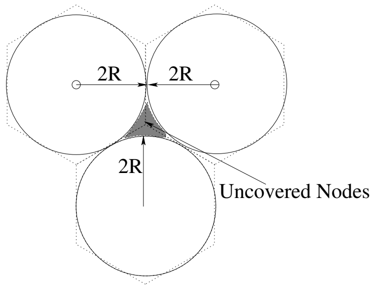


Figure 10.

information has to propagate upto distance r from the clusterhead and back. This is repeated every time the cluster radius is incremented. As an illustration, for a cluster to grow till radius $r = 1$, a single $join$ request by the clusterhead is followed by deg reply messages and to grow till radius $r = 2$, another $1 + deg$ *join* messages and $3 * deg$ *reply* messages are exchanged (where deg is the number of neighbors of a node). Notice that the radius update messages are piggybacked with the *join* messages. Thus to form a legitimate cluster of radius R, the number of messages exchanged within a cluster are $\sum_{r=1}^{R}(1 + deg * r^2) = O(deg * R^3)$. Furthermore, a cluster may have to resolve its overlaps with other neighboring clusters before becoming legitimate thus incurring additional $O(R)$ message exchanges.

For an optimal clustering scheme that constructs clusters in a one-way diffusing computation, communication cost is $O(deg * R^2)$. The communication cost for each node is $O(R)$ in LOCI, and for an optimal clustering scheme it is $O(1)$. Thus LOCI has a communication overhead of $O(R)$ when compared to the optimal scheme both for cluster formation and at an individual node (in the worst case).

A3. Improving the convergence

Here, in order to improve the convergence time of LOCI, we introduce a new action, namely *clusterhead assignment*. We also assume a stronger system model, that is nodes have directional information capability.

When a node becomes the clusterhead of a legitimate cluster, it assigns at most six leaders of other clusters in its neighboring region. The assigned clusterheads then grow their cluster until all nodes in R-neighborhood of the clusterhead join the cluster. Assignment of clusterheads by legitimate clusterheads may lead to overlaps between *R*-neighborhoods of assigned and legitimate clusters, among assigned clusters themselves, and between autonomous and assigned clusters. The priorities of clusters are in the following order: legitimate clusters have the highest priority followed by assigned clusters and then by autonomous clusters. When a legitimate and an assigned cluster overlap then the assigned cluster collapses. Similarly, when a valid but not yet legitimate autonomous cluster and an assigned cluster overlap, then the assigned cluster dominates the autonomous cluster.

When an assigned cluster overlaps with other assigned clusters it grows till circular radius *R* irrespective of the overlaps with neighboring assigned clusters. Non-overlapping clusters are obtained by assigning a unique priority to each assigned cluster which is given by the same *lexicographical* ordering $<R, -O, ID>$, however, now *O* is the number of overlaps of the cluster with neighboring *assigned* clusters, and the resolve overlap action is not considered until the radius R of the assigned clusterhead reaches R.

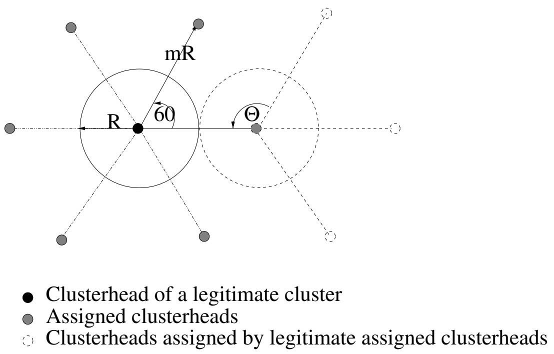


Figure 11. Clusterhead assignment

The cluster with minimum number of overlaps with other assigned clusters dominates. If the clusters have the same number of overlaps then the cluster with maximum ID dominates. When the assigned clusters become legitimate they assign at most three clusterheads in their neighboring region as shown in Figure 11. If j is a clusterhead of an autonomous cluster then variable $p.j$ is set to j. If j is a clusterhead of an assigned cluster, then $p.j$ is the clusterhead that assigned j. We add the following action to node j to assign clusterheads.

Module assign.j Clusterhead of a legitimate cluster assigns clusterheads at a distance of *mR* from it. An autonomous clusterhead node assigns at most six clusterheads in an arc of 360 degrees, with each clusterhead at an angle of 60 degrees with the neighboring assigned clusterheads as shown in Figure 4. The assigned clusterheads assign at most three clusterheads in an arc of 120 degrees, where θ is the angle between the assigned clusterhead and the clusterhead that assigned it. If a node does not exist at the required location then the node closest to the required location and farther than $2R$ from the parent clusterhead is assigned as a new clusterhead.

LC.j $\longrightarrow$ if $(p.j = j)$ then *assign.j(0, 360, mR)*
else if $(p.j \neq j)$
then $assign.j(\theta + 120,\ \theta + 240,\ mR)$

Theorem 6: The convergence time of fast LOCI from invariant state to fixpoint state is O(R^2).
Proof sketch: From Theorem 4 we know that in every circle of radius $O(R^2)$ there exists at least one legitimate cluster in $O(R^2)$ rounds of node execution. In fast LOCI, a legitimate cluster assigns clusterheads in its neighboring region through a one way diffusing computation. Since such computation will cover the circle with radius O(R^2) in time proportional to the radius of the circle, the convergence time of LOCI is O(R^2). □

A4. Proof of Correctness for LOCI.

Theorem 1: I is an invariant of LOCI.

Proof: We show that I is closed under program actions. When action 1 is executed, I trivially holds because the values assigned to the variables constitute a state in I. When any of the guards of action 2 are enabled then there must exist a valid cluster k such that $VC.k$ holds. It follows from the assignment values of the variables that when a node j with $d.j < 0$ or $d.j > R$ joins such a cluster k, it is still a valid cluster which also constitutes a state in I. When action 3 is executed then at least one of the valid clusters reduces its radius. Since all the nodes at the periphery of such a cluster relinquish the cluster, the resulting cluster is a valid cluster. If the cluster collapses this also constitutes a state in I because all the nodes in the collapsed cluster have $d.j$ value set to -1. □

Theorem 2: F is a fixpoint of LOCI.

Proof: We show that for any state in F, no action is enabled. Action 1 is disabled because all nodes j in F have $d.j \geq 0$. First disjunct of action 2's guard is not enabled because $d.j \geq 0$ and $LP.j$. Second disjunct of action 2's guard is not enabled because all clusters have $r.j = R$ which implies that no node j with $d.j > R$ can join another cluster such that distance of j from the clusterhead $\leq R$. Action 3 is not enabled because $(\forall j : |\{o.j\}| = 1)$. □

Lemma 1: Starting at a state in *I*, LOCI is guaranteed to form at least one legitimate cluster.
$(I \mapsto (\exists j :: LC.j)) \wedge stable(LC.j)$

Proof: From action 1, it follows that a node j will eventually timeout and set $d.j = 0$ and write the variables of its neighbors. If multiple nodes with $d.j = 0$ attempt to write the variables of the same nodes then the nodes with $d.j = 0$ set their neighbor's $d.k$ variables and their own variables to a value < 0, and the process is repeated after a random delay. As this process cannot be repeated infinitely, eventually there exists a node j such that $VC.j$ and $r.j \geq 1$. For node j if $Q1 \equiv (\forall l, k : l \in N_{r.j}(j) \wedge k \in nbr.l : d.k < 0 \vee (d.k \geq 0 \Rightarrow (c.k = j \vee (r.k < r.j \wedge c.k \neq j))))$ holds then cluster of node *j* increments its radius. Also $Q1 \Rightarrow (\forall l : l \in N_{r.j}(j) : |\{o.l\}| = 0)$. If *Q1* continues to hold for cluster of node *j* then eventually $r.j = R$ and hence *LC.j* is true.

If *Q1* does not hold for cluster of *j* ($\equiv c.j$) then cluster *c.j* overlaps with some other cluster *c.k* such that $r.j = r.k$. Each cluster has a unique priority given by the lexicographical ordering <Rad, O, ID>. When clusters *c.j* and *c.k* overlap then at least one cluster has higher priority than the other cluster. The number of overlaps of a cluster with other clusters of equal radius is upper bounded by a constant and hence a cluster dominates finite number of clusters before incrementing its radius. Since radius is bounded by *R* there is eventually a cluster of some node *j* such that *LC.j* holds at *j*, and from program actions it follows that *LC.j* is closed. □

Lemma 2: Let $LC = \{j | LC.j\}$ be the set of clusterheads of legitimate clusters. If there exists a node j that is outside of mR neighborhood of any node in LC, then LOCI guarantees to form a new legitimate cluster.
$|LC| = x \wedge (\exists j : (\forall k : k \in LC : dist.j.k > mR)) \hookrightarrow |LC| > x$

Proof: The proof is by contradiction. Assume that the number of legitimate clusters does not increase. Then all valid clusters die before becoming legitimate, and from action 2 it follows that eventually action 1 is not enabled for any node that is within mR of a legitimate cluster. That is, all nodes within mR of a legitimate clusterhead will join a legitimate cluster.

Then the timeout action of a node j such that $(\forall k : k \in LC : dist.j.k > mR)$ will fire and j will become a clusterhead. Note that nodes that are both in R-neighborhood of j and mR neighborhood of legitimate clusters can join j's cluster since $m \geq 2$ and because of the second component of action 2's guard. That is, from action 3, j can lose only to other valid clusterheads j' where $(\forall k : k \in LC : dist.j'.k > mR)$. Since the priority of the dominating clusterhead is higher and the number of overlaps of a cluster with other clusters of equal radius is upper bounded by a constant, eventually a clusterhead j' will reach radius R. However, this leads to a contradiction. □

Theorem 3: Starting from an invariant state *I*, the program is guaranteed to reach a state in *F*.

$I \hookrightarrow F$

Proof: We give a variant function on the number of legitimate clusters, i.e., $|LC|$. From Lemmas 1 and 2, it follows that $|LC|$ increases. However, there is an upperbound on $|LC|$ because a finite number of clusters of circular radius R can fit in a graph.

When $|LC|$ stops increasing, from action 2 it follows that within a finite number of steps, all nodes j will be part of a legitimate cluster, and hence, F holds. □

Theorem 4: The convergence time of LOCI from invariant state to fixpoint state is $O(R^4)$.

Proof: We assume that the maximum time it takes for a node to communicate with its neighbor is bounded by a constant. Thus, for a cluster to grow till radius r_i in i iterations, total time taken = O(r_i^2) ($\equiv$ O($r_1 + (r_1 + r_2) + \ldots + (r_1 + r_2 + \ldots + r_i)$)). Thus a legitimate cluster is formed in O(R^2) time.

Next, we show that in every circle of radius O(R^2) there exists a legitimate cluster. This directly follows from the fact that a valid cluster can only lose to a higher priority valid cluster. Since the radius is bounded, the farthest distance of a legitimate cluster from a cluster of unit radius is O($r_1 + r_2 + \ldots + r_R$) = O(R^2).

In the slowest convergence, first all the legitimate clusters are formed at the circumference of such circles. Since there can be O(R) such concentric circles within a circle of O(R^2) and O(R) legitimate clusters on the circumference of such circles, the convergence time is O(R^2)×O(R)×O(R) = O(R^4). □

Theorem 5: Starting at an arbitrary state, every system computation is guaranteed to reach a state in I in $O(R)$ rounds.

Proof: We give a variant function on the number of invalid clusters |IVC| where IVC = $\{j | (d.j = 0 \ \wedge \ \neg VC.j)\} \cup \{c.j | d.j > 0 \ \wedge \ \neg H.j\}$. We show that the number of invalid clusters never increase and eventually decrease.

Action 1 only produces valid clusters. When action 2 is enabled at some node j, it can only join an existing valid cluster or an invalid cluster. When action 3 is enabled, the number of invalid clusters can only decrease or remain constant.

Next, we show that the number of invalid clusters eventually decrease which follows from the stabilizing action. We again give a variant function for this proof. This time the variant function is on the number of nodes in an invalid cluster and show that they eventually decrease until no node remains in that invalid cluster. Let n_1 be the number of nodes in some invalid cluster and let n_2 be the number of neighbors of nodes in the invalid cluster which do not belong to the invalid cluster. Thus from action 2, maximum number of nodes that can join an invalid cluster is n_2. We show that the total number of nodes $(n_1 + n_2)$ in the invalid cluster decreases eventually.

Case 1: *An invalid cluster with no clusterhead.*
We know that there exists a node j in the invalid cluster with minimum $d.j$ value. Since the first part of stabilizing action's guard holds for such a node, it resets its $d.j$ value, and from first part of action 2's guard we know that such a node will not join that invalid cluster again. In the remaining invalid cluster there must exist a node with minimum $d.j$ value. Following the same argument as above, all the nodes in the invalid cluster relinquish the cluster eventually and the number of invalid clusters decrease.

Case 2: *An invalid cluster with a clusterhead.*
From part 3 and 4 of the stabilizing action's guard, if $r.j$ is corrupted to a large value then the boundary nodes whose $d.j < r.j$ leave the invalid cluster and from first part of action 2's guard such nodes do not join that invalid cluster again. Thus the number of nodes eventually decrease and the number of invalid clusters decrease. If $r.j$ is corrupted to a small value, then from third part of stabilizing action's guard, the nodes farther than $r.j + 1$ from the clusterhead relinquish the cluster thus resulting in a valid cluster or decrease in the number of invalid clusters. □

Theorem 6: The convergence time of fast LOCI from invariant state to fixpoint state is O(R^2).

Proof: From theorem 4 we know that it takes O(R^2) time to construct a legitimate cluster and in every circle of radius O(R^2) there exists a legitimate cluster. In fast LOCI, a legitimate cluster assigns clusterheads in its neighboring region through a one way diffusing computation. Since such computation will cover the circle with radius O(R^2) in time proportional to the radius of the circle, the convergence time of LOCI is O(R^2). □

A4. Implementation of LOCI

Please refer to the following webpages:
(1) Source code: `http://www.cis.ohio-state.edu/~mittalv/LociImpl.txt`,
(2) Simulation data: `http://www.cis.ohio-state.edu/~mittalv/LociData.txt`.