A Fault-Local Self-Stabilizing Clustering Service for Wireless Ad Hoc Networks

Murat Demirbas, *Member*, *IEEE*, Anish Arora, *Senior Member*, *IEEE*, Vineet Mittal, and Vinodkrishnan Kulathumani

Abstract—We present a fast, local clustering service, FLOC, that partitions a multihop wireless network into nonoverlapping and approximately equal-sized clusters. Each cluster has a clusterhead such that all nodes within unit distance and some nodes within distance m of the clusterhead belong to the cluster. We show that, by asserting a stretch factor $m \geq 2$, FLOC achieves locality of clustering and fault-local self-stabilization: The effects of cluster formation and faults/changes at any part of the network are contained within at most $m+1$ units. Through simulations and experiments with actual deployments, we analyze the trade-offs between clustering time and the quality of clustering and suggest suitable parameters for FLOC to achieve a fast completion time without compromising the quality of the resulting clustering.

Index Terms—Wireless sensor networks, fault tolerance, reliability, availability, and serviceability.

1 Introduction

Large-scale ad hoc wireless networks introduce challenges for self-configuration and maintenance. Centralized solutions that rely on predefined configurer or maintainer nodes are unsuitable: Requiring all the nodes in a large-scale network to communicate their data to a centralized base station depletes the energy of the nodes quickly due to the long distance and multihop nature of the communication and also results in network contention.

Clustering [1], [3], [9], [13], [4] is a standard approach for achieving efficient and scalable control in these networks, as it facilitates the distribution of control over the network. Clustering also saves energy and reduces network contention by enabling locality of communication: Nodes communicate their data over shorter distances to their respective clusterheads. The clusterheads aggregate these data into a smaller set of meaningful information. Not all nodes, but only the clusterheads, need to communicate far distances to the base station; this burden can be alleviated further by hierarchical clustering, i.e., by applying clustering recursively over the clusterheads of a lower level.

To enable efficient and scalable control of the network, a clustering service should combine several properties. The service should achieve clustering in a fast and local manner: Cluster formation and changes/failures in one part of the network should be insulated from other parts. Furthermore, the service should produce approximately equal-sized clusters with minimum overlap among clusters. Equal-sized clusters is a desirable property because it enables an even distribution of control (e.g., data processing, aggregation, storage load) over clusterheads; no clusterhead is overburdened or underutilized. Minimum overlap among clusters is desirable for energy efficiency because a node that participates in multiple clusters consumes more energy by having to transmit to multiple clusterheads.

In this paper, we are interested in a stronger property, namely, a solid-disc clustering property, that implies minimization of overlap. The solid-disc property requires that all nodes that are within a unit distance of a clusterhead belong only to its cluster. In other words, all clusters have a nonoverlapping unit radius solid-disc.

Solid-disc clustering is desirable since it reduces the intracluster signal contention: The clusterhead is shielded at all sides with nodes that belong to only its cluster, so the clusterhead receives messages from only those nodes that are in its cluster and does not have to endure receiving messages from nodes that are not in its cluster. Solid-disc clustering also results in a guaranteed upper bound on the number of clusters: In the context of hierarchical clustering, minimizing the number of clusters at a level leads to lower-cost clustering at the next level. Finally, solid-discs yield better spatial coverage with clusters: Aggregation at the clusterhead is more meaningful since the clusterhead is at the middle of the cluster and receives readings from all directions of the solid disc (i.e., is not biased to only one direction).

Equi-radius solid-disc clustering with bounded overlaps is, however, not achievable in a distributed and local manner. We illustrate this observation with an example for a 1D network (for the sake of simplicity).

Consider a clustering scheme that constructs clusters with a fixed radius, say $R = 1$, solid-disc. We show that, for fixed radius clustering schemes, a reclustering of the entire network may be unavoidable. In Fig. 1, consider the node j:

- M. Demirbas is with the Computer Science and Engineering Department, University at Buffalo, Buffalo, NY 14260. E-mail: demirbas@cse.buffalo.edu.
- A. Arora and V. Kulathumani are with the Computer Science and Engineering Department, The Ohio State University, Columbus, OH 43210. E-mail: {anish, vinodkri}@cse.ohio-state.edu.
- V. Mittal is with QUALCOMM Incorporated, 5775 Morehouse Dr., San Diego, CA 92121-1714. E-mail: vmittal@qualcomm.com.

Manuscript received 22 Feb. 2005; revised 29 May 2005; accepted 30 Aug. 2005; published online 26 July 2006.
Recommended for acceptance by I. Stojmenovic, S. Olariu, and D. Simplot-Ryl.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-0171-0205.

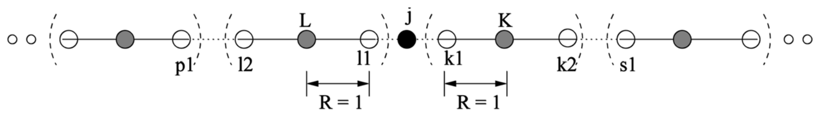

Fig. 1. Each pair of brackets constitutes one cluster of unit radius and colored nodes denote clusterheads.

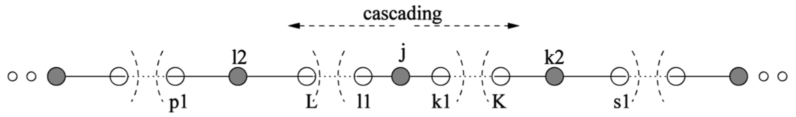

Fig. 2. Node j forms a new cluster and leads to reclustering of the entire network.

It cannot be subsumed into its neighboring clusters as j is not within unit distance of neighboring clusterheads L and K. j thus forms a new cluster with itself as the clusterhead. Since all nodes within unit radius of a clusterhead should belong to its cluster, j subsumes neighboring nodes l_1 and k_1 in its cluster. This leads to neighboring clusterheads L and K to relinquishing their clusters and election of l_2 and k_2 as the new clusterheads (Fig. 2). The cascading effect propagates further as the new clusterheads l_2 and k_2 subsume their neighboring nodes, leading to reclustering of the entire network.

1.1 Our Contributions

We show that solid-disc clustering with bounded overlaps is achievable in a distributed and local manner for approximately equal radii (instead of exactly equal radii). More specifically, we present FLOC, a fast local clustering service that produces nonoverlapping and approximately equal-sized clusters. The resultant clusters have at least a unit radius solid-disc around the clusterheads, but they may also include nodes that are up to m, where $m \geq 2$, units away from their respective clusterheads. By asserting a stretch factor of $m \geq 2$, FLOC achieves locality of clustering and fault-local self-stabilization: Effects of cluster formation and faults/changes at any part of the network are contained within at most m unit distance.

While presenting FLOC we take unit radius to be the reliable communication radius of a node and m to be the maximum communication radius. In so doing, we exploit the double-band nature of wireless radio-model and present a communication and, hence, energy-efficient clustering.

FLOC is suitable for clustering large-scale wireless ad hoc networks since it is fast and scalable. FLOC achieves clustering in O(1) time regardless of the size of the network. FLOC is also locally self-stabilizing in that, after faults (such as node failure and recovery, arbitrary state corruption) stop occurring, faults and changes are contained within the respective cluster or within the immediate neighboring clusters, and FLOC stabilizes within constant time.

We simulate FLOC using Prowler [18] and analyze the trade-offs between clustering time and the quality of the clustering. We observe that forcing a very short clustering time leads to network traffic congestion and message losses and, hence, degrades the quality of the resultant clustering. We suggest suitable parameters for FLOC to achieve a fast completion time without compromising the quality of the clustering. Furthermore, we implement FLOC on the Mica2 [19] sensor node platform and experiment with actual deployments on 25 nodes to corroborate our simulation results.

1.2 Outline

After presenting the network and fault model in the next section, we present the basic FLOC program in Section 3. We discuss the fault-local self-stabilization of FLOC in Section 4. In Section 5, we present additional actions that improve the convergence time of the clustering. We discuss our simulation and implementation results in Section 6. In Section 7, we present related work and we conclude the paper in Section 8.

2 Model

We consider a wireless ad hoc network where nodes lie on an undirected graph topology. The wireless radio-model for the nodes is double-band: A node can communicate (transmit and receive messages) reliably with the nodes that are in its inner-band (*i-band*) range and unreliably (i.e., only a percentage of messages go through) with the nodes in its outer-band (*o-band*) range. This double-band behavior of the wireless radio is observed in [20], [6], [21].

We define the unit distance to be the i-band range. We require that the o-band range be m units, where $m \geq 2$. This is a reasonable assumption for the o-band range [20], [21], [6]. Nodes can determine whether they fall within the i-band or o-band of a certain node by using any of the following methods:

- Nodes are capable of measuring the signal strength of a received message [19]. This measurement may be used as an indication of distance from the sender. For example, assuming a signal strength loss formula $(\frac{1}{1+d^2})$, where d denotes distance from the sender, the i-band neighbors receive the message with [0.5, 1] of the transmission power and, for $m = 2$, the o-band neighbors receive the message with [0.2, 0.5] power.
- Nodes may use time of flight of audio or ultrasound signals to calculate the distances to the nodes within single radio hop [7], [16].
- An underlying localization service [15], [14] may provide the nodes with these distance information.

In our paper, any of these three techniques could be applicable. We assume that all the nodes have the same i-band/o-band range. By using the ranging and localization methods, it is possible to get the i-band of all nodes in the

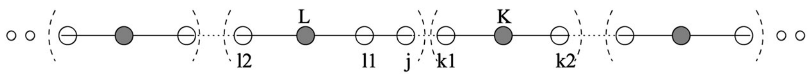

Fig. 3. Node j is subsumed by one of its neighboring clusters.

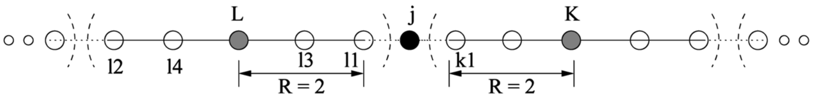

Fig. 4. $j's$ neighbors are l_1 and k_1.

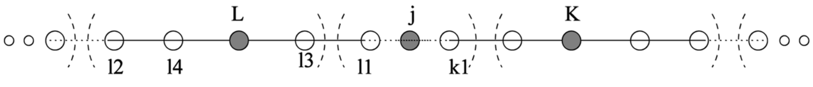

Fig. 5. j becomes the clusterhead.

network to have same "unit distance" modulo the measurement errors.

We say that a clustering of nodes satisfies *solid-disc clustering* if every node within i-band of any clusterhead j belongs to j's cluster. Locality of clustering states that every node is assigned to a cluster in a way that satisfies solid-disc clustering and also preserves the solid-disc property of the existing clusters.

We assume that nodes have timers, but we do not require time synchronization across the nodes. Timers are used for tasks such as sending of periodic heartbeats and timing out of a node when waiting on a condition. Nodes have unique ids. We use i, j, and k to denote the nodes and $j.var$ to denote a program variable residing at j. We denote a message broadcast by j as msg_j.

A *program* consists of a set of variables and actions at each node. 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 state is defined by a value for every variable in the program, 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 and is executed.

2.1 Fault Model

Nodes may fail-stop [17] and crash and new nodes may join the network. Moreover, the state of a node's program can be arbitrarily and transiently corrupted. Channels may suffer faults that corrupt, manufacture, duplicate, or lose (e.g., due to collision or fading) messages. These faults can occur in any finite number, at any time, and in any order.

A program is *self-stabilizing* iff, after faults stop occurring, the program eventually recovers to a state from where its specification is satisfied. A self-stabilizing program is fault-local self-stabilizing if the time and number of messages required for stabilization are bounded by functions of perturbation size rather than network size. The perturbation size for a given state is the minimum number of nodes whose state must change to achieve a consistent state of the network.

2.2 Problem Statement

Design a distributed, local, scalable, and self-stabilizing program that constructs a clustering of a network such that:

- a unique node is designated as a clusterhead of each cluster,
- every node in the inner-band of a clusterhead j belongs to j's cluster,
- no node outside the outer-band of a clusterhead j belongs to j's cluster,
- every node belongs to a cluster, and
- no node belongs to multiple clusters.

3 FLOC Program

In this section, we present the basic FLOC program in the absence of faults (i.e., no state corruptions, no node failures, or node additions occur during clustering). First, we show why a stretch factor $m \geq 2$ is sufficient for local clustering.

3.1 Justification for $m \geq 2$

As an illustration of local clustering of FLOC, consider Fig. 3. When $m \geq 2$, j is subsumed by one of its neighboring clusters as j is within two units of the clusterhead L, thus leading to local clustering.

Furthermore, Fig. 4 illustrates how FLOC constructs clusters locally when all clusters are of radius 2 and a node j is to be assigned a cluster. In this case, j elects itself as the clusterhead since it is not within two units of the clusterheads of its neighbors $l1$ and $k1$. Nodes $l1$ and $k1$ then join the cluster of j because they are not within one unit of their respective clusterheads but are within one unit of j. Thus, j forms a legitimate cluster as in Fig. 5.

Lemma 1. *A stretch factor $m \geq 2$ ensures locality of clustering.*

Proof. The proof is by induction. Locality of clustering states that each node is assigned to a cluster without affecting the solid-disc property of the existing clusters. In the base case, let CH be the set of existing clusterheads such that each cluster satisfies the solid-disc property and includes no node beyond distance m from the clusterhead. In the induction step, there are two cases for a node j not included in a cluster. If j is within distance m of a clusterhead in CH, then j is subsumed within that cluster (no existing cluster is affected). If j is outside of m of any clusterhead in CH, then stretch factor $m \geq 2$ ensures that j can become a clusterhead and construct its solid-disc (by including all nodes within unit distance in its cluster) without violating the solid-disc property of any existing clusterhead in CH. □

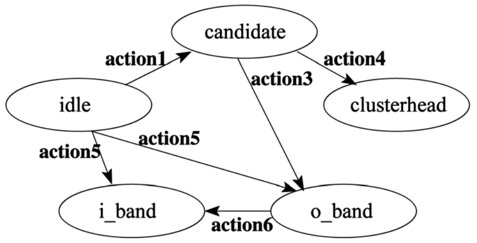

Fig. 6. The effect of actions on the $status$ variable.

3.2 Program

Each node j maintains only two variables, $status$ and $cluster\_id$, for the FLOC program. $j.status$ has a domain of $\{idle, cand, c\_head, i\_band, o\_band\}$. As shorthand, we use $j.x$ to denote $j.status = x$. $j.idle$ is true when j is not part of any cluster. $j.cand$ means j wants to be a clusterhead and $j.c\_head$ means j is a clusterhead. $j.i\_band$ (respectively, $j.o\_band$) means j is an inner-band (respectively, outer-band) member of a clusterhead; $j.cluster\_id$ denotes the cluster j belongs to. Initially, for all j, $j.status = idle$ and $j.cluster\_id = \perp$.

The FLOC program consists of six actions, as seen in Fig. 7. Fig. 6 illustrates the effect of actions on the status variable of a node.

Action 1 is enabled when a node j has been $idle$ for some random wait-time chosen from the domain $[0 \dots T]$. Upon execution of action 1, j becomes a $candidate$ for becoming a clusterhead and broadcasts its candidacy.

Action 2 is enabled at an i-band node of an existing cluster when this node receives a candidacy message. If this recipient node determines that it is also in the i-band of the new candidate, it replies with a conflict message to the candidate and attaches its cluster-id to the message. We use a random wait-time from the domain $[0 \dots t]$ to prevent several nodes replying at the same time so as to avoid collisions.

Action 3 is enabled at j when j receives a conflict message in reply to its candidacy announcement. The conflict message indicates that if j forms a cluster, its i-band will overlap with the i-band of the sender's cluster. Thus, j gives up its candidacy and joins the cluster of the sender node as an o-band member.

Action 4 is enabled at j if j does not receive a conflict message to its candidacy within a predefined period Δ. In this case, j becomes a clusterhead, broadcasting this decision with $c\_head\_msg_j$.

Action 5 is enabled at all the idle nodes that receive a c_head_msg. These nodes determine whether they are in the i-band or o-band of the sender, adjust their status accordingly, and adopt the sender as their clusterhead.

Action 6 is enabled at an o_band node j when j receives a c_head_msg from a clusterhead i. If j determines that j falls in the i-band of i, j joins i's cluster as an i_band member.

3.3 Analysis

For the correctness of the FLOC program, we require that the election of a clusterhead be completed in an atomic manner: If two nodes that are less than two units apart become candidates concurrently, both may succeed and, as

```
(Action 1) timeout($j.idle$) $\longrightarrow$   $j.status := cand$;
                                                 bcast($cand\_msg_j$)
[]
(Action 2) timeout($j.i\_band \wedge$ rcv($cand\_msg_i$)) $\longrightarrow$
                              if($j \in$ i-band of $i$)
                                   bcast($conflict\_msg_j$)
[]
(Action 3) $j.cand \wedge$ rcv($conflict\_msg_i$) $\longrightarrow$
                              $j.status := o\_band$;
                              $j.cluster\_id := msg_i.cluster\_id$
[]
(Action 4) timeout($j.cand$) $\longrightarrow$   $j.status := c\_head$;
                                                 bcast($c\_head\_msg_j$)
[]
(Action 5) $j.idle \wedge$ rcv($c\_head\_msg_i$) $\longrightarrow$
                              $j.status := i\_band \mid o\_band$;
                              $j.cluster\_id := i$;
[]
(Action 6) $j.o\_band \wedge$ rcv($c\_head\_msg_i$) $\longrightarrow$
                              if($j \in$ i-band of $i$)
                                   $j.status := i\_band$;
                                   $j.cluster\_id := i$;
```

Fig. 7. Program actions for j.

a result, the i-bands of the resultant clusters could be overlapping. To avoid this case with a high probability, the domain T of the timeout period for action 1 should be large enough to ensure that, for a node j whose idle-timer expires, the idle-timers of none of the nodes within two units of j expire within Δ time of j's candidacy as the candidacy period for a node can last at most Δ time. Therefore, the probability of atomicity of elections can be calculated as $(1 - \frac{\Delta}{T})^{\omega}$, where ω denotes the maximum number of nodes within two units of a node.

Note that T depends only on the local density of nodes and is independent of the network size. Hence, it is sufficient to experiment with a representative small portion of a network to come up with a T that avoids collisions of clusterhead elections with a high probability. For the cases where the atomicity requirement for elections is violated, our additional actions presented in Section 5 reassert the solid-disc clustering property.

Lemma 2. *The following invariant holds for FLOC:*

$$\begin{array}{lll}
\forall j, k :: & & \\
\text{I1. } j.idle \vee j.cand & \equiv & j.cluster\_id = \perp \\
\text{I2. } j.c\_head & \equiv & j.cluster\_id = j \\
\text{I3. } j.i\_band \wedge j.cluster\_id = k & \Rightarrow & k.c\_head \wedge j \in \text{ i-band of } k \\
\text{I4. } j.o\_band \wedge j.cluster\_id = k & \Rightarrow & k.c\_head \wedge j \in \text{ o-band of } k \\
\text{I5. } k.c\_head \wedge j \in \text{ i-band of } k & \Rightarrow & j.i\_band \wedge j.cluster\_id = k
\end{array}$$

Proof. I1 and I2 follow trivially from the program. Also, I3 and I4 follow from actions 5 and 6. I5 holds when the

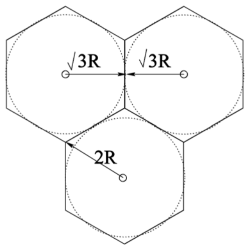

Fig. 8. The minimum number of clusters is achieved by tiling hexagonal clusters of radius 2.

atomicity of elections is satisfied. If there is a conflict with the i-band of a candidate j and that of a nearby cluster, then j is notified via action 2, upon which j becomes an o_band member of this nearby cluster via action 3. If there is no conflict, j becomes a clusterhead and achieves a solid-disc by dominating all the nodes in its i-band via action 5. The o_band members of other clusters that fall in the i-band of j join j's cluster due to action 6. □

Above, I5 is the dual of I3 and is satisfied when the atomicity of elections holds. Due to I5, the dual of I4 does not always hold. That is, a node j within the o-band of a clusterhead k and within the i-band of another clusterhead k' will choose to become a member of k' due to I5.

Note that the above invariant implies the solid-disc clustering property (due to $I5$) and satisfies the clustering requirement in the problem statement.

Theorem 1. *Regardless of network size, FLOC produces a clustering of nodes that satisfies the problem statement and terminates within constant time* $T + \Delta$.

Proof. An action is enabled at every node within at most T time: If no other action is enabled in the meanwhile, action 1 is enabled within T time.

From Fig. 6, it is easy to observe that once an action is enabled at a node j, j is assigned to a cluster within at most Δ time: If the enabled action is 5, then j is assigned to a cluster instantaneously. If the enabled action is 1, then, due to Lemma 1, one of actions 3 or 4 is enabled within at most Δ time, upon which j is assigned to a cluster immediately.

Also note that, once j is assigned to a cluster (i.e., $j.status \in \{c\_head, i\_band, o\_band\}$), no further action can violate this property. Only actions 2 and 6 can be enabled at j: Action 2 does not change $j.status$ and action 6 changes $j.status$ from o_band to i_band, but j is still a member of a cluster (in this case a closer cluster).

Thus, every node belongs to a cluster within $T + \Delta$ and no further action is enabled after $T + \Delta$. Furthermore, due to Lemma 2, the clustering satisfies the solid-disc property and the problem statement. □

Theorem 2. *The number of clusters constructed by FLOC is within 3-folds of the minimum possible number.*

Proof. A partitioning of the network with minimum number of clusters is achieved by tiling hexagonal clusters of radius 2 (and circular radius $\sqrt{3}$). This best-case construction is shown in Fig. 8. The worst-case construction, where FLOC partitions the network with maximum number of clusters, is achieved by tiling hexagonal clusters of radius $2/\sqrt{3}$ (and circular radius 1). Since the solid-disc clustering property asserts a circular radius of unit distance at least, it is not possible to do worse than this number. In this worst case, the number of clusters constructed by FLOC is 3 times the minimum possible number. □

3.4 Discussion

After clustering, a node can be in the i-band of at most one clusterhead. A clusterhead has all the nodes in its i-band as its members and some from its o-band. The o-band members do not need to hear the clusterhead every time, the i-band members may suffice for most operations. If the clusterhead is sending an important message that needs to reach all members, in order for the o-band members to also receive it reliably, the i-band members may relay this message when they detect missing acknowledgments from nearby o-band members—the i-band members can hear both the clusterhead and the o-band members reliably. During a convergecast (data aggregation) to the clusterhead, the messages from o-band members may or may not reach the clusterhead directly. If a message from an o-band member is tagged as important, it may be relayed by an i-band member upon detection of a missing acknowledgment from the clusterhead.

Ideally, first, we want a conflict to be reported by a node that is closest to the candidate, so that the candidate, upon aborting its candidacy, can join this closest cluster. One way to choose the closest notifier is to set t at a notifier node to be inversely proportional to the distance from the candidate. If an underlying localization service is not available, the same effect can be achieved by setting t inversely proportional with respect to the received signal strength of the candidacy message. The higher the received signal strength of the candidacy message at a notifier, the notifier sets t smaller.

4 Self-Stabilization

In this section, we present the stabilization actions to restore the system to its invariant and discuss the fault-local self-stabilization properties of our clustering service. These actions, S1 through S6, are presented in Fig. 9.

Actions S1 & S2. Violations of I1 and I2 (e.g., due to transient state corruption) are locally checkable and are locally corrected.

Action S3. For detecting any violation of I3 and I4 (e.g., due to failure of a clusterhead), we employ heartbeats. The clusterheads periodically broadcast a c_head_msg.

Action S4. For correcting I3 and I4, we employ leases. If the lease at a node j expires, i.e., j fails to receive a heartbeat from its clusterhead within the duration of a lease period, then j dissociates itself from the cluster by setting $j.status := idle$ and $j.cluster\_id := \perp$. The lease for o-band nodes should be kept high. Since they receive only a percentage of the heartbeats, they may make mistakes for small lease periods. Keeping the lease period high for the o-band nodes does not affect the performance significantly because the o-band nodes are moldable: Even if they have misinformation about the existence of a clusterhead, the o-band nodes do not hinder new cluster formation, and even join these clusters if they fall within the i-band of these clusterheads.

```
(Action S1) $(j.idle \vee j.cand) \wedge j.cluster\_id \neq \bot \longrightarrow$ $j.cluster\_id := \bot$
[]
(Action S2) $j.c\_head \wedge j.cluster\_id \neq j \longrightarrow$ $j.cluster\_id := j$
[]
(Action S3) timeout($j.c\_head$)) $\longrightarrow$ bcast($c\_head\_msg_j$)
[]
(Action S4) $(j.i\_band \vee j.o\_band) \wedge$
            timeout(rcv($c\_head\_msg$)) $\longrightarrow$
                $j.status{:=}idle$;
                $j.cluster\_id := \bot$
[]
(Action S5) $j.i\_band \wedge$ rcv($c\_head\_msg_i$) $\wedge j.cluster\_id \neq i \longrightarrow$
                if($j \in$ i-band of $i$)
                    bcast("demote($i$, $j.cluster\_id$)")
[]
(Action S6) $j.cluster\_id := j \wedge$ rcv("demote($j$)") $\longrightarrow$ $j.status{:=}idle$;
                $j.cluster\_id := \bot$
```

Fig. 9. Stabilization actions for j.

Action S5. Violation of I5 (e.g., due to the addition of a new node j) is detected by an i_band node j that receives a c_head_msg from a clusterhead i different from j's current clusterhead and j is also within the i-band of i. In this case, j sends a demote message to both clusterheads and both clusters are collapsed.

Action S6. For correcting I5, the clusters whose solid-discs overlap are collapsed. This is achieved by setting a clusterhead to be idle upon receiving a demote message.

Theorem 3. *FLOC is fault-locally self-stabilizing.*

Proof. Correction of I1 and I2 follow immediately from S1 and S2 and are local to the node. I3 and I4 are corrected due to action S4 and the correction is local to the node. I5 is detected by an i-band node j in S5 and is corrected due to S6. A violation of I5 is reduced to violation of I3 and I4 for the nodes that are at most $m+1$ distance from j (since j is an i-band node, it is at most unit distance away from the clusterheads and, since clusterheads do not have nodes beyond distance m, only $m+1$ distance is affected). Since the corrections of I3 and I4 are local, correction of I5 is contained in distance $m+1$. □

Note that, once the invariant is satisfied via the stabilization actions, due to locality of clustering (Lemma 1), reclustering is also achieved locally.

4.1 Node Failures and Additions

FLOC is inherently robust to failures of cluster members (nonclusterhead nodes) since such failures do not violate the invariant. The failure of a clusterhead leaves its cluster members orphaned and is dealt with via actions S3 and S4 above. The effects are contained within at most distance m. The addition of a new node to the system may violate I5. Due to periodic clusterhead messages (action S3), the new node may hear from an existing clusterhead and join a cluster via actions 5 and 6. If j is in the i-band of multiple clusterheads, this situation is handled via stabilization actions S5 and S6. On the other hand, if there are no clusterheads j can join, due to Lemma 1, j can construct a cluster via actions 1 and 4.

5 Extensions to the Basic FLOC Program

Choosing a sufficiently large T guarantees the atomicity of elections (and, hence, the solid-disc clustering) with a high probability. Here, we present some additional actions to ensure that the solid-disc property is satisfied even in the statistically rare cases where the atomicity of elections is violated.

Consider a candidate i and an idle node k that is within 2 units of i. If k's $idle$ timer expires before i's election is completed (i.e., within Δ time of i's candidacy announcement), then the atomicity of elections is violated. Even though there exists a node j that is within the i-bands of both i and k, both candidates may succeed in becoming clusterheads: Since k's candidacy announcement occurs before i's c_head_msg, action 2 is not enabled at j and j does not send a $conflict\_msg$ to k.

Our solution is based on the following observation: Since i broadcasts its $cand\_msg$ earlier than that of k and since a broadcast is an atomic operation in wireless networks: i's broadcast is received at the same instant by all the nodes within i's i-band. These i-band nodes can be employed for detecting a conflict if a nearby node announces candidacy within Δ of i's candidacy.

To implement our solution, we introduce a Boolean variable *lock* to capture the states where an idle node j is aware of a candidacy of a node that is within unit distance of itself. The value of $j.lock$ is material only when $j.status = idle$. Our solution consists of four actions, as seen in Fig. 10.

Action 7 is enabled when an idle node j receives a candidacy message. If j determines that j is in the i-band of the candidate, j sets *lock* as *true*.

Action 8 is enabled when an idle and locked node j receives a candidacy message. If j determines that it is also in the i-band of this new candidate, it replies with a "potential conflict" message to the candidate.

Action 9 is enabled when a node receives a "potential conflict" message as a reply to its candidacy announcement. In this case, the node gives up its candidacy and becomes idle again. This time, to avoid a lengthy waiting, the node selects the random wait-time from the domain $[0 \ldots T/2]$.

```
(Action 7)  $j.idle \wedge \text{rcv}(cand\_msg_i) \longrightarrow$
                if($j \in$ i-band of $i$)   $j.lock := true$
[]
(Action 8)  $\text{timeout}(j.lock \ \wedge j.idle \wedge \text{rcv}(cand\_msg_i)) \longrightarrow$
                if($j \in$ i-band of $i$)   $\text{bcast}(pot\_conf\_msg_j)$
[]
(Action 9)  $j.cand \wedge \text{rcv}(pot\_conf\_msg_i) \longrightarrow$
                $j.status := idle$
[]
(Action 10) $\text{timeout}(j.lock == true) \longrightarrow$   $j.lock := false$
```

Fig. 10. Additional actions for handling the violations of atomicity of elections.

Action 10 is enabled if an idle j remains locked for Δ time. Expiration of the Δ timer indicates that the candidate that locked j failed to become a leader since, otherwise, j would have received a *c_head_msg* and *j.status* would have been set to i_band. So as to not block future candidates, j removes the lock by setting $j.lock{:=}false$.

Note that these additional actions are applicable only in the statistically rare violations of atomicity of elections; they do not cure the problem for every case. If T is chosen too small, there may be some pathological cases where there is a chain of candidates whose i-bands overlap with each other that results in the deferring of all candidates in the chain. For example, assume that $x_1, x_2, x_3, \ldots x_n$ become candidates in that order in time and space (say, on a line). In this case, x_i suppresses x_{i+1} and is suppressed by x_{i-1}. In the end, only x_1 survives. So, even though each node's behavior is still local, the resulting pathological chain—due to high contention—is not local. These chains should be avoided by choosing a large enough T.

Lemma 3. *The additional actions preserve the invariant.*

Proof. The program with the additional actions is a *superimposition* [11] of the original program. In the absence of violations of the atomicity of elections, the additional actions do not modify any variables of the original program (since actions 8 and 9 are not enabled at all). In the presence of violations of the atomicity of elections, the program with the additional actions produces a subset of the behavior of the original program. In particular, in the amended program, the effects of the violations of the atomicity of elections is suppressed (i.e., $I5$ still holds). □

The safety part of Theorem 1 still holds for the amended program, however, in the presence of chain effects, it is not possible to put a constant upper bound on the termination time. Theorem 3, the fault-local stabilization proof, still holds for the amended program since the invariant of the program is unchanged. The *lock* variable itself is self-stabilizing since it is a soft-state variable that is cleaned when the lease period is over.

6 Simulation and Implementation Results

In this section, we analyze, through simulations and experiments, the trade-offs between smaller T and the quality of clustering, and determine a suitable value for T with a fast completion time without compromising the quality of the resulting clustering. We also analyze the scalability of FLOC with respect to network size.

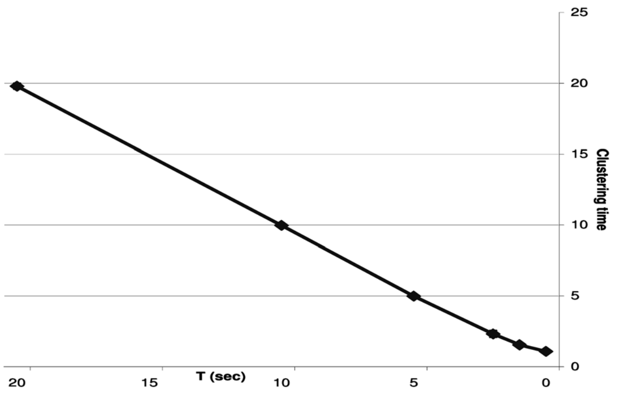

Fig. 11. Completion time versus T.

6.1 Simulation

For our simulations, we use Prowler [18], a MATLAB-based, event-driven simulator for wireless sensor networks. Prowler simulates the radio transmission/propagation/reception delays of Mica2 motes [19], including collisions in ad-hoc radio networks and the operation of the MAC-layer.

Our implementation of FLOC under Prowler is per node and is a message-passing distributed program. Our implementation includes actions 1 through 9 and leaves out the stabilization actions. Our code is available from http://www.cse.buffalo.edu/~demirbas/floc/. In our simulations, we use a grid topology for simplicity (note that FLOC is applicable for any kind of topology and does not require a uniform distribution of nodes). In the grid, each node is unit distance away from its immediate North, South, East, and West neighbors. We use a signal strength of 1 and $m = 2$; the i-band neighbors are the nodes with Received Signal Strength Indicator (RSSI) > 0.5 and the o-band neighbors have $RSSI > 0.2$. It follows that immediate N, S, E, W neighbors are i-band neighbors and immediate diagonal neighbors and 2-unit distance N, S, E, W neighbors are o-bound neighbors. Thus, the degree of a node in our network is between 4 and 12.

Below, we analyze the trade-offs involved in the selection of T; for this part, we use a 10-by-10 grid (as described above) for the simulations. Then, we consider larger networks (up to 25-by-25 grids) and investigate the scalability of the performance of FLOC with respect to network size. We repeat each simulation 10 times and take the average value from these runs. In all our graphs, the error bars denote the standard deviation in our data.

Due to MAC-layer delays, the average transmission time for a packet is around 25 msec. Thus, we fix $t{=}50$ msec and $\Delta{=}200$ msec for our simulations.

6.1.1 *Trade-Offs in the Selection of T*

Using a small value for T allows a shorter completion time for FLOC, as shown in Fig. 11. However, a small value for T also increases the probability of violation of atomicity of elections; Fig. 12 shows that, while T decreases, the number of violations of atomicity of elections increases. The experimental data in Fig. 12 are consistent with the formula we presented in Section 3 for estimating the probability of

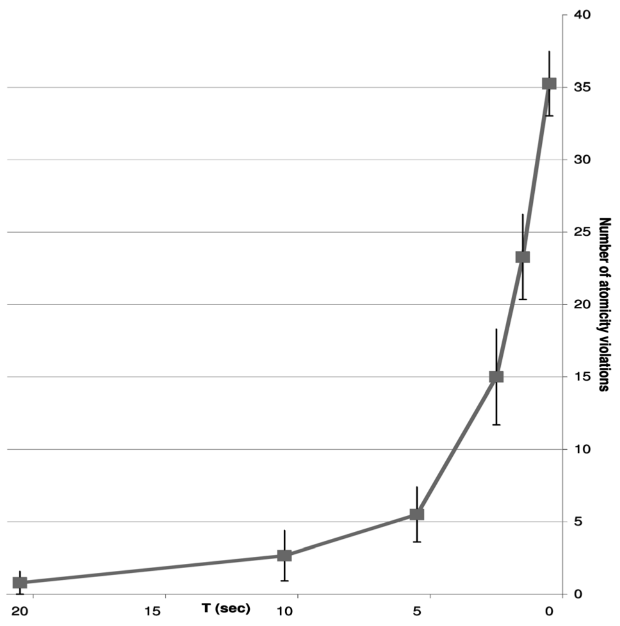

Fig. 12. Number of atomicity violations versus T.

satisfaction of atomicity of elections. In the formula $(1 - \frac{\Delta}{T})^{\omega}$, using $\Delta = 0.2$ sec, $\omega = 12$, $T = 20$ sec returns 0.89 as the probability for satisfaction of atomicity of elections. Using $T = 10$ sec returns 0.78, $T = 5$ sec returns 0.61, $T = 1$ sec returns 0.07, and $T = 0.5$ sec returns 0.02 for the atomicity of elections.

Ideally, we want the elections to be completed in an atomic manner. For up to some number of atomicity violations, our extra actions in Section 5 enable successful solid-disc clustering. However, for small values of T ($T < 5$ sec), several nodes declare their candidacy around the same times and we encounter a sharp increase in the number of messages sent and the number of nodes sending messages, as shown in Fig. 13. This leads to network traffic congestion and loss of messages due to collisions. For $T = 2$ sec, the number of receptions of collided messages is 20 percent of the total messages received. This collision rate climbs to 30 percent for $T = 1$ sec and 55 percent for $T = 0.5$ sec. Due to these lost messages, for $T < 5$, we observe deformities in the shape of the clusters formed; the solid-disc clustering property is violated. For example, for $T = 0.5$ sec, half of the clusters formed are single node clusters. As a result, we observe an increase in the number of clusters formed as shown in Fig. 14.

To achieve a quick completion time while not compromising the quality of the resulting clustering, we choose $T = 5$ sec in our FLOC program—and for the rest of this section. Although $T = 5$ sec produces 0.61 probability for atomicity of elections according to the formula in Section 3 and $T = 5$ sec results in five atomicity violations in Fig. 12, the additional actions in Section 5 ensure that the solid-disc clustering property is satisfied by every run of the FLOC program for $T = 5$ sec. Fig. 15 shows a resulting partitioning on a 10-by-10 grid. The arrow at a node points to its respective clusterhead. There are 16 clusters; each clusterhead contains at least its i-band neighbors as its members, that is, solid-disc clustering is observed.

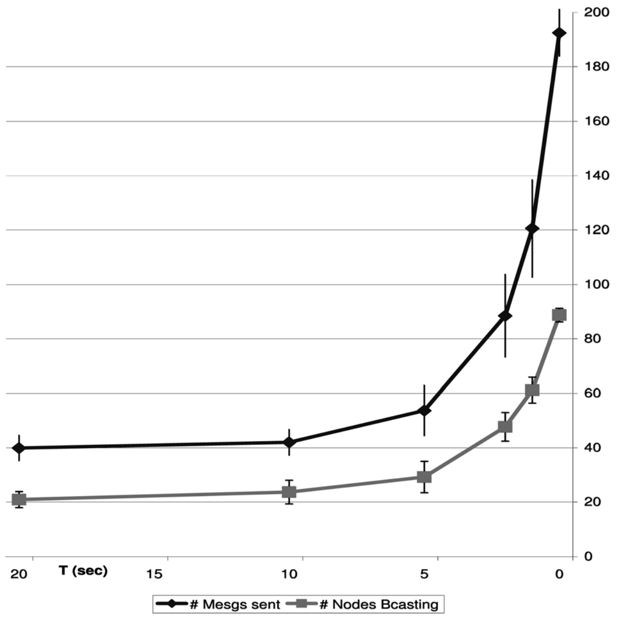

Fig. 13. Messages sent versus T.

6.1.2 Scalability with Respect to Network Size

In Theorem 1, we showed that the completion time of FLOC is unaffected by the network size. To corroborate this result empirically, we simulated FLOC with $T = 5$ sec for increasing network size of up to 25-by-25 nodes while preserving the node density. Fig. 16 shows that the clustering is achieved in 5 sec regardless of network size.

We also investigated the average number of clusters constructed (NCC) by FLOC with respect to increasing network size. An interesting observation is that, NCC for a given N is predictable; the variance is very small, as seen in Fig. 17. Since clusters have, on average, around six members, $N/6$ gives NCC for our grid topology networks.

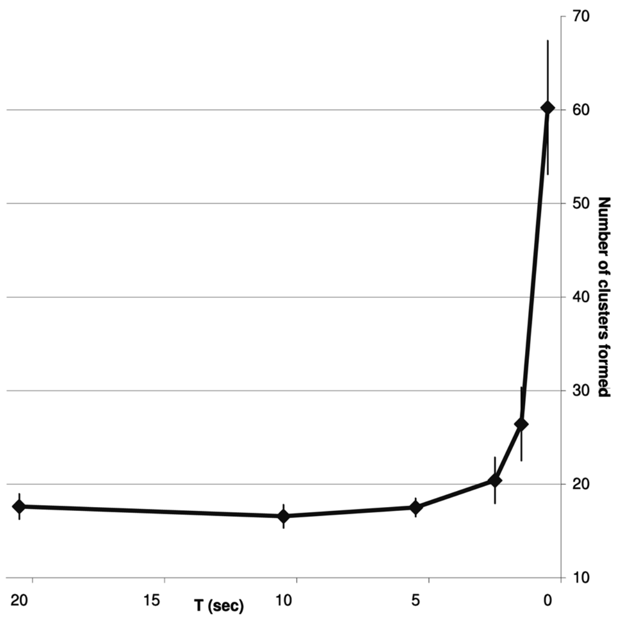

Fig. 14. Number of clusters formed versus T.

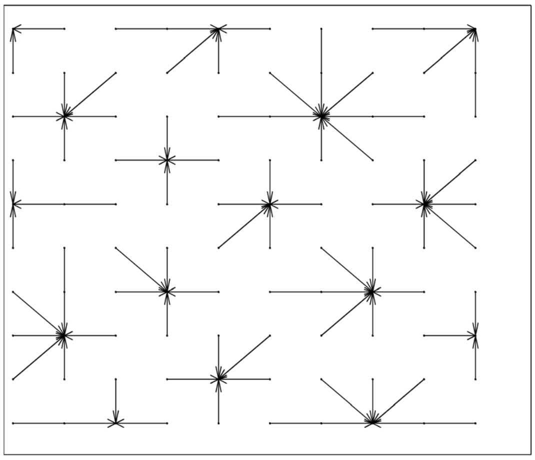
Fig. 15. Clusters formed by FLOC on a 10-by-10 grid.

For a grid of 25-by-25, FLOC constructs around 100 clusters. In the theoretical best case, an omniscient centralized partitioning scheme (see Theorem 2) could tile this grid with 60 hexagons (with circular radius of $\sqrt{3}$ and hexagonal radius of 2). That is, in practice, FLOC has an overhead of only 1.67 when compared with the best scheme. Note that, in Theorem 2, we have determined that NCC for FLOC is always within 3-folds of this best scheme.

6.2 Implementation

We implemented FLOC on the Mica2 [19] sensor node (mote) platform using the TinyOS [10] programming suite. Our implementation includes actions 1 through 9 and leaves out the stabilization actions. Our implementation is about 500 lines of code and available from http://www.cse.buffalo.edu/~demirbas/floc/.

The Mica2 motes use Chipcon [5] radio CC1000 for transmission. RSSI at a mote can be obtained using the CC1000 radio interface in the TinyOS radio stack: RSSI varies from -53dB to -94dB, the radio interface encodes this into a 16 bit integer value—the lower the value the higher the signal strength. By experimenting with an outdoor environment and comparing power level and reliable range of reception we chose a transmission power of 7, from a range of 1 to 255. At a power level 7, we obtain reliable reception up to 5 meters with RSSI ranging from 0 to 140. By selecting appropriate thresholds for RSSI, we took $m = 2$ and divided this 5 meters distance into two equal halves as the i-band range and o-band range: We considered RSSI between 0-80 as i-band and 80-140 as o-band.

We performed our experiments at an outdoor parking lot; Fig. 18 shows a picture of our deployment. To mimic our simulation topology settings, we arranged 25 Mica2 motes in a 5-by-5 grid where each mote is 2 meters away from its immediate North, South, East, and West neighbors. From our signal strength settings, it follows that, ideally, immediate N, S, E, W neighbors are i-band neighbors and immediate diagonal neighbors and 2-unit distance N, S, E, W neighbors are o-bound neighbors. Based on our simulation results, to achieve a quick completion time while avoiding network contention, we chose $T = 5$ sec, $\Delta = 200$ msec.

In our set up, we placed a laptop in the center of the network to collect status reports from the motes: After the clustering is completed, every mote temporarily sets its transmission power to maximum level and broadcasts a status report. This report indicates the completion time of the clustering program at the respective mote and whether the mote is a clusterhead, i-band, or o-band member of a

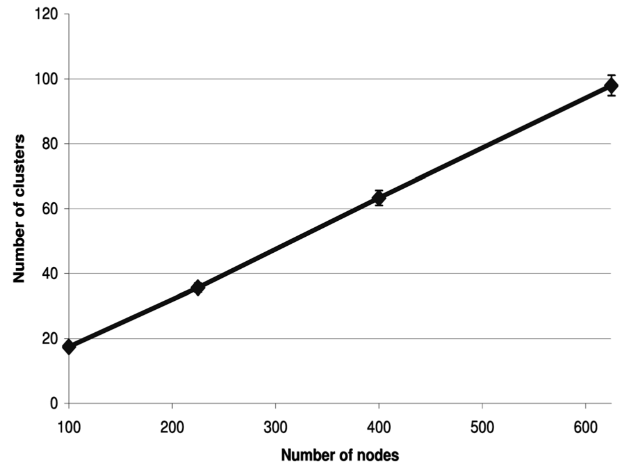

Fig. 17. Number of clusters formed versus network size.

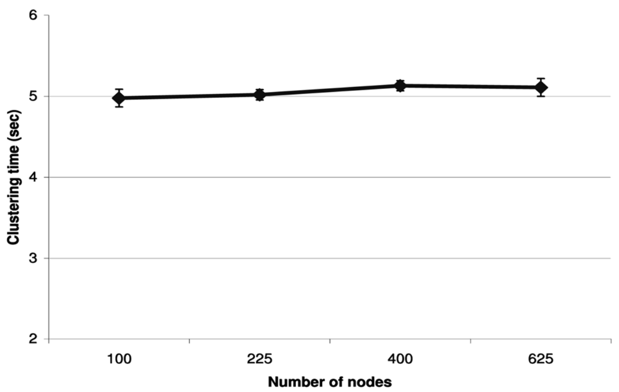

Fig. 16. Completion time versus network size.

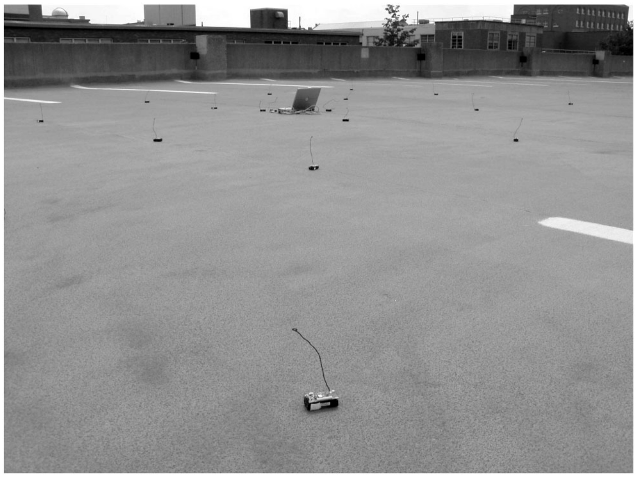
Fig. 18. 5-by-5 grid topology deployment.

cluster. In order to avoid collisions, these reports are spread in time.

We performed over 20 experiments with these settings. We observed the average number of clusters formed to be 4. The cluster sizes were reasonably uniform, the average number of motes per cluster was 6. The average completion time was 4.5 seconds.

When we increased the internode spacing to 2.5 meters, with the same settings for signal strength measurements, the number of clusters increased, as expected, to an average of 6. The average completion time was again 4.5 seconds.

We observed in our experiments that, due to the nondeterministic nature of wireless radio communication, the i-band/o-band membership determination using RSSI is not always robust. Transmitting candidacy and clusterhead messages 3 times and using the average RSSI from the corresponding 3 receptions would make the i-band/o-band determination more robust. Alternatively, as we discussed in Section 2, a connectivity service or an audio-based ranging service can be employed for i-band/o-band membership determination.

7 Related Work

Several protocols have been proposed recently for clustering in wireless networks [1], [3], [9], [13], [4].

The Max-Min D-cluster algorithm [1] partitions the network into d-hop clusters. It does not guarantee solid-disc clustering and, in the worst case, the number of clusters generated may be equal to the number of nodes in the network (for a connected network).

Clubs [13] forms 1-hop clusters: If two clusterheads are within 1-hop range of each other, then both the clusters are collapsed and the process of electing clusterheads via random timeouts is repeated. Clubs does not satisfy our unit distance solid-disc clustering property: Clusterheads can share their 1-hop members.

LEACH [9] also forms 1-hop clusters. 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. Nodes elect themselves as clusterheads based on this probabilistic rotation function and broadcast their decisions. Each nonclusterhead node determines its cluster by choosing the clusterhead that requires the minimum communication energy. LEACH does not satisfy our solid-disc property: Not all nodes within 1-hop of a clusterhead j belong to j. Hence, in LEACH, the clusterheads are susceptible to network contention induced by members of other clusters. The authors [9] suggest using different Code Division Multiple Access (CDMA) spreading codes for each cluster to solve this problem, however, for most sensor network platforms (including Mica2), the CDMA mechanism is not available. FLOC complements LEACH since it addresses the network contention problem at the clusterheads by constructing solid-disc clusters. Moreover, LEACH style load-balancing is readily applicable in FLOC by using the above-mentioned probabilistic rotation function for determining the waiting-times for the candidacy announcements at the nodes.

The algorithm in [3] first finds a rooted spanning tree of the network and then forms desired clusters from the subtrees. It gives a bound on the number of clusters constructed and the convergence time is on the order of the diameter of the network. It is locally fault-tolerant to node failures/joins but may lead to reclustering of the entire network for some pathological scenarios.

For a given value of R, the algorithm in [4] constructs clusters such that all the nodes within $R/2$ hops of a clusterhead belong to that clusterhead and the farthest distance of any node from its clusterhead is 3.5R hops. With high probability, a network cover is constructed in $O(R)$ rounds; the communication cost is $O(R^3)$.

In an earlier technical report [12], we have presented —under a shared memory model—a self-stabilizing clustering protocol, LOCI, that partitions a network into clusters of bounded physical radius $[R, mR]$ for $m \geq 2$. LOCI achieves a solid-disc clustering with radius R. Clustering is completed iteratively within $O(R^4)$ rounds. In a recent paper [8], we outlined the basic FLOC algorithm. However, [8] does not include self-stabilization results with respect to transient state corruption. Also, neither [12] nor [8] contain the extension to the undirected graph topology.

8 Concluding Remarks

The properties of FLOC that make it suitable for large scale wireless ad hoc networks are its: 1) locality, in that each node is affected only by nodes within m units, 2) scalability, in that clustering is achieved in constant time independent of network size and, finally, 3) self-stabilization capability, in that it tolerates faults and changes locally within m units.

Through simulations and experiments with actual deployments of Mica2 nodes in a 5-by-5 grid topology, we analyzed the trade-offs between completion time and the quality of the resulting clustering and suggested suitable values for the domain, T, of the randomized candidacy timer to achieve a fast completion time without compromising the quality of the clustering. Since, in FLOC, each node is affected only by nodes within m units, it is sufficient to experiment with a representative small portion of a network to determine suitable values for T.

As part of future work, we are planning on integrating FLOC into our "Line in the Sand" (LITeS) tracking service [2] to achieve scalable and fault-local clustering. As part of the DARPA/Network Embedded Systems Technology project, our research group has already deployed LITeS over a 100-node sensor network across a large terrain and achieved detection, classification, and tracking of various types of intruders (e.g., people, cars) as they moved through the network. We are also investigating the role of geometric, local clustering in designing efficient data structures for evaluation of spatial queries in the context of ad hoc networks. An interesting direction for future work is to study our clustering program under the mobile ad hoc networks model.

Acknowledgments

This material is based upon work supported by the US National Science Foundation under Grant No. 0341703 and The US Defense Advanced Research Projects Agency (DARPA) NEST contract OSU-RF program F33615-01-C-1901.

REFERENCES

[1] A. Amis, R. Prakash, T. Vuong, and D. Huynh, "Max-Min d-Cluster Formation in Wireless Networks," *Proc. IEEE INFOCOM,* pp. 32-41, Mar. 2000.

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y.-R. Choi, T. Herman, S.S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita, "A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking," *Computer Networks J.,* vol. 46, no. 5, pp. 605-634, Dec. 2004.

[3] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Multi-Hop Wireless Networks," *Proc. IEEE INFOCOM,* pp. 1028-1037, Apr. 2001.

[4] J. Beal, "A Robust Amorphous Hierarchy from Persistent Nodes," AI Memo 2003-011, MIT, 2003.

[5] Chipcon, Cc1000 Radio Datasheet, www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf, 2004.

[6] Y. Choi, M. Gouda, M.C. Kim, and A. Arora, "The Mote Connectivity Protocol," *Proc. Int'l Conf. Computer Comm. and Networks (ICCCN '03),* pp. 533-538, Oct. 2003.

[7] B. Dalton and M. Bov, "Audio-Based Self-Localization for Ubiquitous Sensor Networks," *Proc. Audio Eng. Soc. (AES) 118th Convention,* 2005.

[8] M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani, "Design and Analysis of a Fast Local Clustering Service for Wireless Sensor Networks," *Proc. Broadband Wireless Networking Symp. (BroadNets),* pp. 700-709, Oct. 2004.

[9] W.B. Heinzelman, A.P. Chandrakasan, and H. Balakrishnan, "Application Specific Protocol Architecture for Wireless Micro-sensor Networks," *IEEE Trans. Wireless Networking,* vol. 1, no. 4, pp. 660-670, Oct. 2002.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Network Sensors," *Proc. Int'l Conf. Architectural Support for Programming Languages (ASPLOS),* pp. 93-104, 2000.

[11] S. Katz, "A Superimposition Control Construct for Distributed Systems," *ACM Trans. Programming Language Systems,* vol. 15, no. 2, pp. 337-356, 1993.

[12] 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 Univ., Feb. 2003.

[13] R. Nagpal and D. Coore, "An Algorithm for Group Formation in an Amorphous Computer," *Proc. 10th Int'l Conf. Parallel and Distributed Computing Systems (PDCS),* pp. 28-31, Oct. 1998.

[14] D. Niculescu and B. Nath, "DV Based Positioning in Ad Hoc Networks," *J. Telecomm. Systems,* vol. 22, pp. 267-280, 2003.

[15] B. Parkinson and J. Spilker, "Global Positioning System: Theory and Application," *Am. Inst. of Aeronautics and Astronautics,* 1996.

[16] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System," *Proc. MOBICOM,* pp. 32-43, Aug. 2000.

[17] F.B. Schneider, "Byzantine Generals in Action: Implementing Fail-Stop Processors," *ACM Trans. Computer Systems,* vol. 2, no. 2, 1984.

[18] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi, "Simulation-Based Optimization of Communication Protocols for Large-Scale Wireless Sensor Networks," *Proc. IEEE Aerospace Conf.,* Mar. 2003.

[19] Crossbow Technology, Mica2, 2004, www.xbow.com/Products/Wireless_Sensor_Networks.htm.

[20] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," *Proc. First ACM Conf. Embedded Networked Sensor Systems,* pp. 14-27, 2003.

[21] J. Zhao and R. Govindan, "Understanding Packet Delivery Performance in Dense Wireless Sensor Networks," *Proc. First ACM Conf. Embedded Networked Sensor Systems,* pp. 1-13, 2003.

Murat Demirbas received the BTech degree from the Middle East Technical University, Ankara, Turkey, in 1997. He received the Master's and PhD degrees from The Ohio State University in 2000 and 2004, respectively. After a one-year postdoctorate with the Theory of Computing Group at the Massachusetts Institute of Technology, he is currently an assistant professor in the Computer Science and Engineering Department at the University at Buffalo (SUNY Buffalo). His research interests are in the area of distributed systems, sensor networks, and fault tolerance. He is a member of the IEEE.

Anish Arora received the BTech degree from the Indian Institute of Technology at New Delhi and the Master's and PhD degrees from the University of Texas at Austin, all in computer science. He is a professor of computer science at The Ohio State University. His research is on fault tolerance, security, and timeliness properties of systems, especially distributed and networked systems of large scale. Recent case studies in his research have centered on sensor networking and home networking, with support from the US Defense Advanced Research Projects Agency (DARPA), the US National Science Foundation (NSF), and Microsoft Research. He is a leading expert in self-stabilization and has chaired or cochaired seminars and symposia in this area in 1998, 1999, 2000, and 2002. He is program cochair of the 25th International Conference on Distributed Computer Systems. From 1989 to 1992, he worked at Microelectronics and Computer Technology Corporation (MCC) in Austin, Texas. He is a senior member of the IEEE.

Vineet Mittal received the Bachelor's degree in computer science and technology from the University of Roorkee, India. He received the Master's degree in computer and information science from The Ohio State University. His research interests are in wireless networks and distributed systems. He is currently working at QUALCOMM Incorporated in the WCDMA/HSDPA Technologies Group.

Vinodkrishnan Kulathumani is a PhD student in the Department of Computer Science and Engineering at The Ohio State University. His research interests are in distributed systems and sensor networks. Specifically, his research focuses on 1) designing robust network tracking services for distributed pursuer evader applications and 2) designing reliable distributed control application using sensors and actuators despite faults in the underlying network components and services.