FLOC: A Fast Local Clustering Service for Wireless Sensor Networks

Murat Demirbas Anish Arora Vineet Mittal

Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio 43210 USA

Abstract

We present a fast local clustering service, FLOC, that produces nonoverlapping and approximately equal-sized clusters. The clustering is such that all nodes within unit distance of a clusterhead belongs to its cluster, and no node m units away from the clusterhead may belong to its cluster.

By asserting $m \geq 2$ FLOC achieves locality: effects of cluster formation and faults/changes at any part of the network are contained within at most 2 units.

By taking unit distance to be the reliable communication radius and m to be the maximum communication radius, FLOC exploits the double-band nature of wireless radio-model and achieves clustering in constant time regardless of the network size.

Keywords: *Clustering, locality, self-configuration, self-healing, local fault-tolerance.*

1 Introduction

Large-scale ad hoc wireless sensor networks introduce challenges for self-configuration and maintenance. Centralized solutions that rely on pre-defined 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 multi-hop nature of the communication and also results in network contention.

Clustering is a standard approach for achieving efficient and scalable control in these networks. Clustering facilitates the distribution of control over the network and, hence, enables locality of communication. Clustering nodes into groups saves energy and reduces network contention because 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 over-burdened or under-utilized. 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 to its cluster. In another words, all clusters have a nonoverlapping unit radius solid-disc.

Solid-disc clustering is desirable since it reduces the intra-cluster 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 clusterhead is at the median 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 1-D network (for the sake of simplicity).

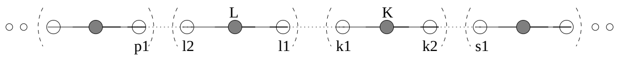

Figure 1. Each pair of brackets constitutes one cluster of unit radius, and colored nodes denote clusterheads.

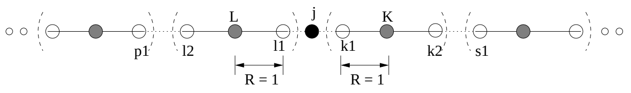

Figure 2. A new node j joins the network between clusters of clusterheads L and K.

Consider a clustering scheme that constructs clusters with a fixed radius, say $R = 1$, solid-disc. Figure 1 shows one such construction. 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 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 its cluster, j subsumes neighboring nodes l_1 and k_1 in its cluster. This leads to neighboring clusterheads L and K to relinquish their clusters and election of l_2 and k_2 as the new clusterheads (Figure 3). The cascading effect propagates further as the new clusterheads l_2 and k_2 subsume their neighboring nodes leading to re-clustering of the entire network.

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 $m \geq 2$ FLOC achieves locality: effects of cluster formation and faults/changes at any part of the network are contained within at most 2 units 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 program.

FLOC is fast and scalable: it achieves clustering in O(1) time regardless of the size of the network. FLOC is also locally self-healing in that after faults stop occurring, FLOC achieves re-clustering within constant time and in a local manner.

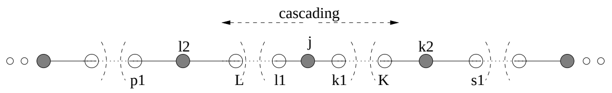

Figure 3. Node j forms a new cluster and leads to re-clustering of the entire network.

2 Model

We consider a wireless sensor network where nodes lie in a 2-D coordinate plane.

The wireless radio-model for the nodes is double-band: A node can communicate 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 [5, 13, 14]

We define the unit distance to be the i-band radius. We require that the o-band radius is m units where $m \geq 2$. This is a reasonable assumption for o-band radius [5, 13, 14]. In this paper, we take $m = 2$ for simplicity of presentation. Nodes can determine whether they fall within 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 [7]. This measurement may be used as an indication of distance from the sender. E.g., 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 the o-band neighbors receive the message with [0.2, 0.5] power.
- A sender may broadcast a special message with a low power and the nodes that receive this message conclude that they are in the i-band of the sender. E.g., in FLOC the candidacy message can be broadcast using a low power to serve this purpose.
- Nodes can maintain a record of percentage of received messages with respect to neighbors [5], and infer the i-band/o-band neighbors from the quality of the connections.
- An underlying localization service [8, 11] may provide the nodes with these distance information.

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 *id*s. 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.

Fault model. Nodes may fail-stop and crash, but we assume that the network does not get partitioned. New nodes can join the network. These faults can occur in any finite number, at any time and in any order.

A program is *self-healing* iff after faults stop occurring the program eventually recovers to a state from where its specification is satisfied.

Problem statement. Design a distributed, local, scalable and self-healing 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

3.1 Justification for $m \geq 2$

As an illustration of local self-healing of FLOC, consider Figure 4. When j joins the network it is subsumed by one of its neighboring clusters as j is within 2 units of the clusterhead L, thus leading to local healing.

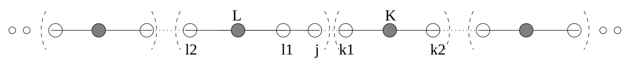

Figure 4. New node j joins one of its neighboring clusters.

Furthermore, Figure 5 illustrates how FLOC locally self-heals when all clusters are of radius 2 and a new node j joins the network. j elects itself as the clusterhead since it is not within 2 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 1 unit of their respective clusterheads but are within 1 unit of j. Thus j forms a legitimate cluster as in Figure 6.

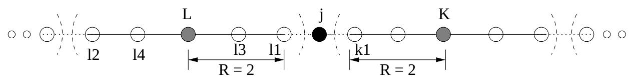

Figure 5. $j's$ neighbors are l_1 and k_1.

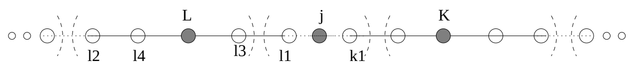

Figure 6. j becomes the clusterhead.

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 a 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 (resp. outer-band) member of a clusterhead; *j.cluster_id* denotes the cluster j belongs to. Initially for all j, $j.status = idle$ and $j.cluster\_id = \perp$.

FLOC program consists of 6 actions.

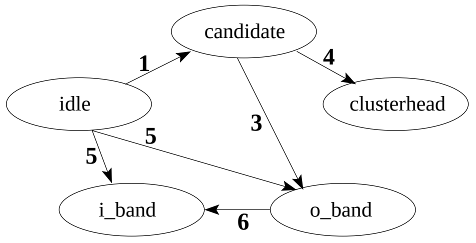

Figure 7. The effect of actions on the *status* variable.

Action 1 is enabled when a node j has been *idle* for some random wait-time chosen from the domain $[0 \ldots T]$. 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 $[0 \ldots t]$ 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 pre-defined period Δ. In this case j becomes a clusterhead, broadcasts 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 of another cluster. If j determines that j falls in the i-band of i, j joins i's cluster as an *i_band* member.

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

Figure 8. Program actions for j.

3.3 Analysis

The candidacy period for a node can last at most Δ time, and we require that the election of a clusterhead is completed in an atomic manner: If two nodes that are less than 2 units apart become candidates concurrently, both may succeed and as 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 no two nodes that are less than 2 units apart have idle-timers that expire within Δ time of each other.

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 2 unit square portion of a network to come up with a T that avoids collusions of clusterhead elections with a high probability. For the rare cases where the atomicity requirement for elections is violated, our additional actions presented in Section 4 reassert the solid-disc clustering property.

Theorem 1. Regardless of network size, FLOC produces a clustering of nodes 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 Figure 7 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 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 $T + \Delta$. Since $cluster\_id$ is set only once at any node, and no node belongs to multiple clusters.

Furthermore, when the atomicity of elections is satisfied, actions 2, 3, and 6 ensure that the clustering satisfies the solid-disc property: If there is a conflict with the i-band of a candidate j and that of a nearby existing 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. The o_band members of other clusters that fall in the i-band of j join j's cluster due to action 6. $\square$

Corollary 1. 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 $\sqrt{3}$). 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). In this worst case, the number of clusters constructed by FLOC is 3 times the minimum possible number. [1] $\square$

Optimization. Ideally, we want that a conflict is first reported by a node that is closest to the candidate, so that the candidate, upon aborting its candidacy, can join this closest cluster. Another advantage of selecting the notifier to be closest to the candidate is that, then the conflict message of the notifier is overheard by as many nodes within the i-band of the candidate, upon which these overhearing nodes can decide that there is no need to report a conflict again. This way communication- and, hence, energy-efficiency is achieved.

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. A notifier sets t smaller the higher the received signal strength of the candidacy message at that notifier.

[1] In practice, this number is less than 3, since clusterheads recruit o-band numbers and have an average circular radius greater than 1.

3.4 Self-healing

FLOC is inherently robust to failures of cluster members (non-clusterhead nodes), since such failures do not violate the clustering specification in Section 2.

Failure of a clusterhead leaves its cluster members orphaned. In order to enable the members to detect the failure of the clusterhead, we employ heartbeats. The clusterhead periodically broadcasts a *c_head_msg*. 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, L, then j dissociates itself from the cluster by setting $j.status := idle$ and $j.cluster\_id := \perp$. While setting the idle-timer, j adds L to the selected random wait time so as not to become a candidate before all the members can detect the failure of the clusterhead.

After a clusterhead failure, all the cluster members become *idle* within at most L time. After this point, the dissolved members either join neighboring clusters as o-band members, or an eligible candidate unites these nodes in a new cluster within $T + \Delta$ time. Due to our selection of $m = 2$, this is achieved in a local manner.

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 values of L. 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.

Addition of new nodes is handled via the heartbeat messages of the clusterhead. FLOC requires that nodes wait for some random time (chosen from $[0 \ldots T]$) before they can become a candidate. Some of the newly added nodes receive a heartbeat (*c_head_msg*) from a nearby clusterhead within their initial waiting period and join the corresponding cluster as an *i_band* or *o_band* member. Those nodes that fail to receive a heartbeat message within their determined waiting times become candidates, and either form their own clusters (via action 2), or receive a conflict message from an *i_band* member of a nearby cluster and join that cluster (via action 3).

4 Extensions

Choosing a sufficiently large T guarantees the atomicity of elections and, hence, the solid-disc clustering. Here we present some additional actions to ensure that the solid-disc property is satisfied even in the statistically rare cases where atomicity of elections are 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 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 sensor 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 to itself. The value of *j.lock* is material only when $j.status = idle$. Our solution consists of 4 actions.

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.

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 not to block future candidates j removes the lock by setting $j.lock := false$.

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

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. These chains should be avoided by choosing a large enough T.

5 Related work

Several protocols have been proposed recently for clustering in wireless networks [1, 3, 4, 6, 10].

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).

LEACH [6] 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. LEACH does not satisfy our solid-disc property: Not all nodes within 1-hop of a clusterhead j belongs to j. Hence, LEACH also does not guarantee a tight upper-bound on the number of clusters formed.

Clubs [10] also 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. Also, in contrast to Clubs, FLOC does not collapse any cluster once it is formed. FLOC resolves contentions by delaying the latter candidates from becoming clusterheads.

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 of the order of the diameter of the network. It is locally fault-tolerant to node failures/joins but may lead to re-clustering 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 [9], 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.

6 Concluding remarks

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

Currently we are working on implementing FLOC under Prowler [12], a MATLAB based, event-driven simulator for wireless sensor networks, in order to determine suitable values for the size of timeout domains t and T under varying node densities. As part of future work, we are planning to implement FLOC to achieve scalable and fault-local clustering in our "Line in the Sand" (LITeS) tracking service [2].

References

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

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Gouda, and Y. Choi. A line in the sand: A wireless sensor network for target detection, classification, and tracking. Technical Report OSU-CISRC-12/03-TR71, The Ohio State University, 2003.

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

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

[5] Y. Choi, M. Gouda, M. C. Kim, and A. Arora. The mote connectivity protocol. *Submitted for publication*, August 2002.

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

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

[8] K. Mechitov, S. Sundresh, Y-M. Kwon, and G. Agha. Cooperative tracking with binary-detection sensor networks. Technical Report UIUCDCS-R-2003-2379, University of Illinois at Urbana-Champaign, 2003.

[9] 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.

[10] 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.

[11] D. Niculescu and B. Nath. Dv based positioning in ad hoc networks. *Kluwer journal of Telecommunication Systems*, 2003.

[12] 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.

[13] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 14–27, 2003.

[14] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 1–13, 2003.