Secure Distributed Cluster Formation in Wireless Sensor Networks*

Kun Sun
Intelligent Automation, Inc.
ksun@i-a-i.com

Pai Peng
Opsware Inc.
ppeng@opsware.com

Peng Ning
NC State University
pning@ncsu.edu

Cliff Wang
Army Research Office
cliff.wang@us.army.mil

Abstract

In wireless sensor networks, clustering sensor nodes into small groups is an effective technique to achieve scalability, self-organization, power saving, channel access, routing, etc. A number of cluster formation protocols have been proposed recently. However, most existing protocols assume benign environments, and are vulnerable to attacks from malicious nodes. In this paper, we propose a secure distributed cluster formation protocol to organize sensor networks into mutually disjoint cliques. Our protocol has the following properties: (1) normal nodes are divided into mutually disjoint cliques; (2) all the normal nodes in each clique agree on the same clique memberships; (3) while external attackers can be prevented from participating in the cluster formation process, inside attackers that do not follow the protocol semantics can be identified and removed from the network; (4) the communication overhead is moderate; (5) the protocol is fully distributed.

1 Introduction

A wireless sensor network typically consists of a potentially large number of resource constrained sensor nodes and a few relatively powerful control nodes such as mobile laptops. Each sensor node is usually battery powered, and has a low-end processor, a limited amount of memory, and a low-power communication module capable of short-range wireless communication. The sensor nodes form an ad-hoc network through the wireless links. Wireless sensor networks are ideal candidates for a wide range of applications, such as target tracking and monitoring of critical infrastructures.

In large sensor networks, the sensor nodes can be grouped into small *clusters* by their physical proximity to achieve better efficiency, and each cluster may elect a *cluster-head* to coordinate the nodes in the cluster. Many efficient cluster-based protocols have been developed for sensor networks to achieve scalability, power saving, channel access, routing, etc. For example, the cluster structure can prolong the lifetime of the sensor network by making the cluster-head aggregate data from the nodes in the cluster and reduce the data sent to the base station (e.g., [4, 14, 30]). As another example, a cluster-head can arrange a time-slotted scheduling for wireless channel access so that message collisions can be reduced by allowing only one node in the cluster to access the channel at any time (e.g., [20, 28]).

A randomly deployed sensor network requires a *cluster formation* protocol to partition the network into clusters. When cluster heads are required, nodes in each cluster may also perform a *leader election* protocol to determine their cluster head. Several cluster formation protocols have been proposed for wireless sensor networks (e.g., [2–7, 11, 14, 16–18, 27, 30]). Based on the order in which cluster formation and leader election are performed, we can divide the cluster formation protocols into two categories: *Leader-First (LF)* approaches and *Cluster-First (CF)* approaches. In Leader-First approaches (e.g., [2, 3, 14, 30], cluster-heads are first elected based on certain metrics (e.g., degree of connectivity, remaining energy), and then they agree on how to assign other nodes to different clusters. In Cluster-First approaches (e.g., [16, 18, 24, 27]), all the sensor nodes first form clusters, and each cluster then elects its cluster-head. Such approaches require all the nodes in one cluster agree on the same membership before electing their cluster-head, and sensor nodes are almost always divided into cliques so that nodes in each clique can directly communicate with each other.

Most existing cluster formation protocols assume benign environments, and cannot survive attacks from malicious participants in hostile environments. In the Leader-First approaches, malicious nodes may lie about their metrics (e.g., increase transmission power for cluster-head advertisement messages in LEACH [14]) to make themselves elected as cluster-heads. As a result, they can control all the nodes in their clusters. Similarly, none of the Cluster-First protocols can guarantee a consistent view on clique memberships when

*This work is partially supported by the National Science Foundation (NSF) under grant CAREER-0447761. Wang's work is supported by the US Army Research Office (ARO) under staff research grant W911NF-04-D-0003-0001. The work of Sun and Peng were performed when they were graduate students at NC State University.

malicious nodes send false information.

Vasudevan et al. proposed two secure leader election algorithms by using a trusted authority to certify each node's metrics used in the leader election process [26]. However, these algorithms assume all the participating nodes are reliable and no messages are lost or delayed, which cannot be guaranteed when there are malicious nodes.

In this paper, we propose a Cluster-First, secure and distributed cluster formation protocol. By exchanging information with 1-hop neighbors, normal sensor nodes are divided into mutually disjoint cliques, in which all the nodes can directly communicate with each other. Our protocol guarantees that all the normal nodes in each clique agree on the same clique membership even under the attacks from both external and internal malicious nodes. We use the protocol semantics to distinguish malicious behaviors from normal ones, identify and remove inside attackers that deviate from the protocol.

Our secure cluster formation protocol is different from the authenticated Byzantine Agreement algorithms (e.g., [8, 12, 25]), which can successfully solve the traditional Byzantine General problem [19]. These authenticated Byzantine Agreement algorithms can guarantee all the normal nodes in one group agree on a single or a set of value(s) by using the signature-based authentication. Our protocol aims to divide a sensor network (one large group) into multiple small groups (cliques) and guarantee all the normal nodes in each small group agree on the same group membership. All the normal nodes have to figure out consistently how to partition the network, and the normal nodes in different groups have different group membership.

Our secure distributed cluster formation protocol has the following properties even if there are external and insider attackers:

- The protocol is fully distributed. Each node computes its clique only using the information from its 1-hop neighbors.
- The protocol is guaranteed to terminate. Participating nodes that do not follow the protocol specification (e.g., send conflicting messages) will be identified and removed from all cliques.
- After the protocol terminates, all normal nodes are divided into mutually disjoint cliques. All normal nodes are guaranteed to have consistent views on their clique memberships even in hostile environments.

The rest of this paper is organized as follows. Section 2 presents the problem and the system model. Section 3 describes the secure distributed cluster formation protocol and proves its security. Section 4 evaluates the performance of the protocol through simulations. Section 5 discusses the related work. Section 6 concludes this paper.

2 Problem Statement

Objective: The objective of our clique formation protocol is to divide the normal nodes in a sensor network into mutually disjoint cliques so that all the nodes in the same cliques can directly communicate with each other. Each node should individually compute its *view of clique* based on the information exchanged with its 1-hop neighbors. We denote the view of clique for node i as C_i. For brevity, we call C_i as the *clique* of node i. We call a node a *normal node* if it follows our protocol. Otherwise, it is a *malicious node*. We would like to guarantee that all normal nodes have consistent cliques, as reflected by the following clique agreement property. *Clique agreement* for a normal node i is defined as:

Definition 1 ***(Clique Agreement)*** *For each node $j \in C_i$, $C_j = C_i$.*

Definition 1 implies that for each normal node $j \notin C_i$, $i \notin C_j$ must hold. That is, each normal node belongs to only one clique. Clique agreement is broken if *Clique Inconsistency* is detected. For node i, clique inconsistency is defined as:

Definition 2 ***(Clique Inconsistency)*** *There exists a node $j \in C_i$ such that $C_j \neq C_i$.*

It is desirable that each node can find as large a clique as possible. We do not consider trivial solutions with which each node forms a clique that only includes itself.

Threat Model: We assume an adversary may launch arbitrary attacks against the cluster formation protocol except for completely jamming the communication channel. An external attacker may eavesdrop, inject, and replay packets to disrupt the cluster formation protocol. However, these attacks can be easily defeated with message authentication.

An attacker may generate more severe impact by participating in the clustering formation process using malicious nodes (e.g., those compromised by the adversary). The malicious nodes may arbitrarily deviate from the protocol in order to introduce clique inconsistency. In particular, a malicious node may use directional antenna to send different messages to different neighbor nodes. Moreover, it can communicate with some normal nodes while intentionally keep silence to others. (We call this *silence attack*.) The malicious nodes may launch Sybil attacks [9] or Wormhole attacks [15]. However, we assume these two kinds of attacks can be detected by using the techniques proposed in [22] and [15], respectively.

Assumptions: We assume each node knows its 1-hop neighbors. A message sent by a normal node can be received correctly by all its (1-hop) neighbors in a finite amount of time. We assume each sensor node has a unique ID, and each node can be uniquely identified due to its keying materials (e.g., unique pairwise keys shared with other nodes, private

keys used for digital signatures). All unicast messages exchanged between nodes are authenticated with the key shared between the two nodes.

We assume the sensor nodes can perform public key based digital signature operations. It has been shown in recent investigations [13, 21] that low-end sensor nodes (e.g., MICA2 motes with 8-bit processors) can perform public key cryptographic operations. Moreover, recent development of sensor platforms such as Intel motes[1] uses more advanced hardware, and can perform public key cryptographic operations efficiently.

We use a combination of μTESLA [23] and digital signature to authenticate broadcast messages. We use digital signatures when non-repudiation is necessary, and μTESLA for efficient broadcast authentication in other cases. We assume the clocks of the normal nodes are loosely synchronized, as required by μTESLA. We also assume the public keys used by the sensor nodes are properly authenticated. One approach to ensure this is to issue to each node a certificate for its public key so that other nodes can validate the node's public key by verifying the certificate.

3 The Secure Distributed Cluster Formation Protocol

In this section, we first present the details of our protocol, and then analyze its properties in normal situation and hostile environments, including clique consistency property and performance overheads.

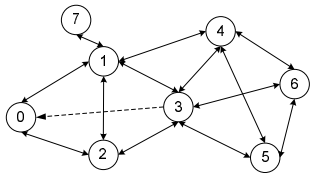
3.1 Protocol Specification

Our secure distributed cluster formation protocol consists of five steps. When all the nodes are normal, the cluster formation process terminates after the first four steps. In hostile environments, when clique inconsistency is detected, the protocol provides an extra Step 5 to remove the identified malicious nodes from the network and restart the protocol from Step 1.

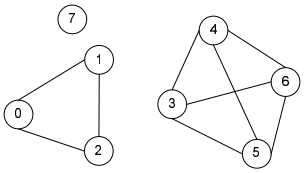
The protocol is summarized below:

- **Step 1:** Each node exchanges its neighbor lists with its neighbors, and computes its *local maximum clique*.
- **Step 2:** Each node exchanges its local maximum clique with its neighbors, and updates its maximum clique according to its neighbor nodes' local maximum cliques.
- **Step 3:** Each node exchanges the updated clique with its neighbors, and derives its final clique.
- **Step 4:** Each node exchanges the final clique with its neighbors. If no clique inconsistency is detected, it terminates successfully. Otherwise, it enters Step 5.
- **Step 5:** Each node performs conformity checking. If it identifies malicious (neighbor) nodes, it removes them from the network, and restarts the protocol from Step 1. Otherwise, it enforces the clique agreement and terminates.

In the following, we will explain these steps in detail. To facilitate the discussion, we will use the simple example shown in Figure 1. Figure 1(a) shows a sensor network consisting of 8 sensor nodes. A directional edge from node i to node j represents node j can receive messages from node i. Considering asymmetric communication, we assume node 0 can hear from node 3, while node 3 cannot hear from node 0. Figure 1(b) shows the results of our clique formation protocol when all the 8 nodes are normal.



(a) A network with 8 nodes



(b) Cluster formation

Figure 1. An Example of Cluster Formation

3.1.1 Step 1: Calculating Local Maximum Clique

Based on our assumptions, each node i can obtain a neighbor list L_i that contains the IDs of its 1-hop neighbor nodes. In the first step, all the nodes exchange their neighbor lists with all their neighbors. As discussed earlier, such messages should be authenticated with the pairwise key shared between neighbors.

After receiving its neighbors' neighbor lists, each node i can build a *neighbor matrix* M_i that records the connectivity between its neighbor nodes. Each element in a neighbor matrix is either 1 or 0. The element in the ith row and jth column of the neighbor matrix is 1 if node i contains node j in its neighbor list, or 0 otherwise. If node i fails to receive the neighbor list from a (previous) neighbor node j, it removes j from its neighbor list.

Each node then symmetrizes its neighbor matrix by considering unidirectional links as no links at all. For example, in Figure 1, node 1 considers that node 0 and node 3 are not connected, since node 0 is not in node 3's neighbor list. The neighbor matrix of node 1 in Figure 1(a) is shown in Table 1.

Based on the neighbor matrix, each node i individually computes a local maximum clique that includes itself. Based on node i's neighbor matrix, we can construct a graph $G_i =$

[1] `http://www.intel.com/research/exploratory/motes.htm`

Table 1. Node 1's Neighbor Matrix

	0	1	2	3	4	7
0	1	1	1	$1 \Rightarrow 0$	0	0
1	1	1	1	1	1	1
2	1	1	1	1	0	0
3	0	1	1	1	1	0
4	0	1	0	1	1	0
7	0	1	0	0	0	0

$\{V_i, E_i\}$, where V_i consists of node i and its neighbors, and E_i consists of the bidirectional edges between nodes in V_i. It is well known that finding the maximum clique in a random graph is an NP-complete problem [10]. For node i, it is also NP-complete [29] to find the maximum clique containing node i in G_i. To reduce the computation complexity, we propose a heuristic algorithm for node i to compute its local maximum clique, as shown in Algorithm 1.

Algorithm 1 Heuristic Algorithm to Find the Local Maximum Clique

INPUT: $G_i = \{V_i, E_i\}, i \in V_i$
OUTPUT: C_i
STEPS:
$S_i = \{j | (i,j) \in E_i\}$; $C_i = \{i\}$;
while ($S_i \neq \emptyset$) **do**
  Find $k \in S_i$ with maximum $|L_i \cap L_k|$
  $L_i \leftarrow L_i \cap L_k$
  $C_i \leftarrow C_i \cup \{k\}$
  $S_i \leftarrow S_i - \{k\} - \{j | (j,k) \notin E_i, j \in S_i\}$
end while

The heuristic algorithm runs in rounds. L_i includes node i's 1-hop neighbor nodes that are eligible to be in the same clique as node i. In each round, node i chooses one neighbor node and adds it into its local maximum clique C_i. Node i maintains a set S_i containing its neighbor nodes that are eligible to be chosen in the next round. Initially, all the neighbors of node i are included in S_i, and C_i only contains node i itself. In the first round, node i computes the number of common neighbors between itself and each neighbor, and finds a neighbor k with the maximum common neighbors $|L_i \cap L_k|$. We use node ID to break the tie. Then node i removes node k from S_i and adds it into C_i. Node i also removes the nodes that are not directly connected with k from set S_i. In the second round, from the updated S_i, node i finds the neighbor node that has the maximum number of common neighbors with all the nodes in C_i (i.e., nodes i and k). Node i then removes this node from S_i and adds it into C_i. Those nodes that are not directly connected with this node will then be removed from set S_i. Node i continues doing so until the set S_i is empty.

After this algorithm finishes, node i sorts the nodes in C_i ascendingly by node IDs and gets its local maximum clique C_i^1. In our protocol, we use C_i^k to denote the clique derived by node i in the kth step ($1 \leq k \leq 4$). Our heuristic algorithm cannot guarantee to find the optimal clique; however, it provides a sub-optimal solution with less computation overhead. We show it through the simulation result in Section 4

Let us see how this algorithm works on node 1 in Figure 1. Initially, node 1 has $C_1 = \{1\}$, $L_1 = \{0, 2, 3, 4, 7\}$, and $S_1 = \{0, 2, 3, 4, 7\}$. In the first round, node 2 has 2 common neighbors $L_1 \cap L_2 = \{0, 3\}$ with node 1; node 3 also has 2 common neighbors $L_1 \cap L_3 = \{2, 4\}$ with node 1. Because node 2 and node 3 have the same maximum number of common neighbors with node 1, we prefer the smaller ID to break the tie. Thus, node 1 adds node 2 into C_1, and $C_1 = \{1, 2\}$. Then, node 1 removes node 2 from S_1, i.e., $S_1 = \{0, 3, 4, 7\}$. Because nodes 4 and 7 cannot directly communicate with node 2, node 1 also removes nodes 4 and 7 from S_1 and $S_1 = \{0, 3\}$. In the second round, node 0 and node 3 have the same number of common neighbors with both node 1 and node 2. Node 1 chooses node 0 that has a smaller ID into C_1. Then, $C_1^1 = \{0, 1, 2\}$, and $S_1 = \emptyset$ after removing node 0 and node 3. Node 3 is removed from S_1 since node 3 is not connected with node 0. Finally, node 1's local maximum clique is $C_1^1 = \{0, 1, 2\}$. Similarly, we have $C_0^1 = C_2^1 = \{0, 1, 2\}$, $C_3^1 = C_4^1 = C_5^1 = C_6^1 = \{3, 4, 5, 6\}$, and $C_7^1 = \{1, 7\}$.

3.1.2 Step 2: Ordering and Updating Maximum Cliques

The local maximum clique computed in step 1 at different nodes are likely to be different. In step 2, each node looks at the local maximum cliques derived by its neighbors, and updates its local maximum clique to prepare for final clique agreement.

In this step, each node i broadcasts its local maximum clique C_i^1 to all its neighbors. For efficiency, such broadcast messages can be authenticated with μTESLA. Because node i calculates its local maximum clique C_i^1 by a heuristic algorithm based on its local neighbor information, it is possible for node i to receive a larger local maximum clique C_j^1 that contains i from a neighbor j. Therefore, after receiving the local maximum cliques from its neighbors, node i checks if there exists any clique C_j^1 which is "better" than its clique C_i^1. To compare cliques computed by different nodes, we define a relation "$\overset{i}{\prec}$" on cliques as follows:

Definition 3 *$C_j \overset{i}{\prec} C_k$ if and only if*

1. *$i \in C_j$, $i \in C_k$, and*
2. *a). $|C_j| < |C_k|$, or*
   b). $|C_j| = |C_k|$, but $c_j < c_k$, where $c_j = min\{a_i | a_i \in C_j \wedge a_i \notin C_k\}$ and $c_k = min\{b_i | b_i \in C_k \wedge b_i \notin C_j\}$, or
   c). $C_j = C_k$, but $j < k$.

The relation $\overset{i}{\prec}$ gives a total order for the local maximum cliques received by node i. We can compare two cliques C_j and C_k by relation $\overset{i}{\prec}$ only if both cliques contain node i. We have $C_j \overset{i}{\prec} C_k$ if the number of nodes in C_k is greater than

that in C_j; or both cliques contain the same number of nodes, but for the first two different IDs $c_j \in C_j$ and $c_k \in C_k$ we have $c_j < c_k$; or C_j contains the same nodes as C_k, but $j < k$. In two ascendingly ordered local maximum cliques, the first two different IDs are also the smallest two different IDs. For example, if $C_j = \{1, 2, 3\}$ and $C_k = \{1, 3, 4\}$, then $c_j = 2$ and $c_k = 3$, and $C_j \overset{1}{\prec} C_k$.

Suppose node i receives n cliques that contain node i. Node i orders these cliques as $C^1_{\alpha_1} \overset{i}{\prec} \ldots \overset{i}{\prec} C^1_i \overset{i}{\prec} \ldots \overset{i}{\prec} C^1_{\alpha_n}$, and updates its clique to the "best" clique $C^1_{\alpha_n}$. After Step 2, node i has an updated clique $C^2_i = C^1_{\alpha_n}$. We call C^2_i as node i's *updated clique*.

Let us illustrate this step with the example in Figure 1. After receiving the local maximum cliques from neighbor nodes, node 1 has $C^1_0 = C^1_1 = C^1_2 = \{0, 1, 2\}$, $C^1_3 = C^1_4 = \{3, 4, 5, 6\}$, and $C^1_7 = \{1, 7\}$. Node 1 can immediately drop the cliques from nodes 3 and 4, since they do not contain node 1. Because $|C^1_7| < |C^1_0|$, node 1 has $C^1_7 \overset{1}{\prec} C^1_0$. Because $C^1_0 = C^1_1 = C^1_2$ but node IDs $0 < 1 < 2$, we have $C^1_0 \overset{1}{\prec} C^1_1 \overset{1}{\prec} C^1_2$. Therefore, node 1 orders the cliques from node 0, 1, 2 and 7 as $C^1_7 \overset{1}{\prec} C^1_0 \overset{1}{\prec} C^1_1 \overset{1}{\prec} C^1_2$, and updates its clique to $C^2_1 = C^1_2 = \{0, 1, 2\}$. Consider node 7. It will keep its clique unchanged since node 1's clique $C^1_1 = \{0, 1, 2\}$ does not contain node 7. After Step 2, we have $C^2_0 = C^2_1 = C^2_2 = \{0, 1, 2\}$, $C^2_3 = C^2_4 = C^2_5 = C^2_6 = \{3, 4, 5, 6\}$, and $C^2_7 = \{1, 7\}$. We can see that node 7 still has clique inconsistency with node 1.

3.1.3 Step 3: Obtaining Final Clique

In this step, each node i broadcasts its updated clique C^2_i to its neighbors. Similarly to the broadcast messages in step 2, these messages should also be authenticated with μTESLA. For every node j in C^2_i, node i checks if it is included in j's clique C^2_j. If not, node i removes j from its clique C^2_i. After this step, each node i obtains its final clique C^3_i. If node i does not receive node j's updated clique, node i simply keeps node j in its clique.

For our example in Figure 1, because $C^2_1 = \{0, 1, 2\}$ does not contain node 7, node 7 removes node 1 from $C^2_7 = \{1, 7\}$, and obtain its final clique $C^3_7 = \{7\}$. Finally, all the nodes are grouped into 3 cliques, which are $C^3_0 = C^3_1 = C^3_2 = \{0, 1, 2\}$, $C^3_3 = C^3_4 = C^3_5 = C^3_6 = \{3, 4, 5, 6\}$ and $C^3_7 = \{7\}$.

If all the nodes are normal, after the first three steps, we can guarantee the clique agreement. We prove this in Section 3.2. However, in hostile environments, since compromised nodes may deviate from the protocol, we need extra steps to detect the potential clique inconsistency and identify the malicious nodes.

3.1.4 Step 4: Checking Clique Agreement

All the nodes broadcast their final cliques to their neighbors. Each node i also calculates a secure hash over all the four messages sent in the first four steps, sign this hash value, and append it into the message that contains the final clique. When a normal node i receives the first copy of a final clique C^3_j from its neighbor j or forwarded by another neighbor, if $j \in C^3_i$, node i rebroadcasts the clique C^3_j. The goal of this rebroadcast is to prevent silence attacks.

Each node i verifies the clique agreement. That is, node i verifies for all $j \in C^3_i$, whether $C^3_j = C^3_i$ holds. When clique inconsistency is detected, node i enters Step 5; otherwise, it terminates the clique formation process.

3.1.5 Step 5: Identifying Insider or Enforcing Clique Agreement

This step consists of two stages. In Stage I, node i performs *conformity checking* to identify malicious nodes that send inconsistent messages in the previous four steps. The basic idea is to use the protocol semantics to distinguish malicious behaviors from normal ones. When malicious nodes are identified, node i sends an alert to other nodes, using the malicious nodes' signatures as proofs. After removing the malicious nodes from the network, all the remaining nodes restart the protocol from Step 1 again. The malicious nodes that have been identified will be removed from normal nodes' neighbor list and thus cannot launch further attacks.

A malicious node may send messages to some normal neighbor nodes, but keep silence to others. According our assumptions, the messages sent from normal nodes can be received in a finite amount of time. Thus, a normal node may detect a malicious node if certain messages are not received from the malicious node. However, the normal node does not have any proof to convince other normal nodes who do receive the messages from the malicious node. A normal node cannot distinguish a normal node who really detects a malicious node from a malicious node who forges a false alert on a normal node. In such cases, node i enters Stage II to enforce the clique agreement, and finish the clique formation protocol.

We describe these two stages in detail below.

Stage I: Conformity Checking.

Suppose a normal node i detects a clique inconsistency with node j. Node i requests node j to forward the messages that node j received in the first four steps. Because node j has received node i's authenticated final clique C^3_i in Step 4, only if $C^3_i \neq C^3_j$, node j will provide its previously received messages to node i. Node j need sign these messages to prove that these messages are forwarded by node j. For efficient signing, node j may calculate a secure hash over all the messages, and simply sign and send this hash value in one message. After verifying node j's signature, node i performs the following conformity checking for node j.

Conformity Checking 1 *Node j follows the clique formation protocol correctly in the first four steps.*

In the above checking, node i re-computes the first three

steps of the cluster formation protocol for node j. If the derived final clique is not the same as what node i received from node j in Step 4, node j is a malicious node. Node i can use node j's signatures as a proof to notify other normal nodes in the network. If node j passes checking 1, node i performs the following checking on all the common neighbors of nodes i and j.

Conformity Checking 2 *For any node $k \in L_i \cap L_j$, k sends the same messages to i and j in every step.*

Because node i has messages directly received from node k and the message from node k received and forwarded by node j, if node k sends different messages to nodes i and j in any step, node i can detect the malicious node k and use the conflicting messages from node k as proofs to convince all the other nodes.

Conformity Checking 1 and 2 guarantee to detect the malicious nodes if clique inconsistency is caused by malicious nodes sending inconsistent messages. It is proved by Theorem 2 in Section 3.3.1. Node i enters Stage II when no malicious node is identified.

Stage II: Consistency Enforcement

When a malicious node launches silence attacks, a normal node may detect the malicious node if certain messages are not received from the malicious node. However, the normal node does not have any proof to convince other normal nodes who do receive the messages from the malicious node. Moreover, a normal node cannot distinguish a normal node who really detects a malicious node from a malicious node who forges a false alert on a normal node.

In such cases, our protocol can ensure that all the normal nodes achieve clique agreement by performing the following consistency enforcement. Suppose two normal nodes i and j find inconsistency, i.e., $j \in C_i^3$, $i \in C_j^3$ (which is proved in Lemma 3) and $C_i^3 \neq C_j^3$. Without loss of generality, we assume $k \in C_i^3$ and $k \notin C_j^3$.

Consistency Enforcement 1 *If $k \in C_i^2, k \notin C_j^2$, node i receives C_k^1, and node j does not receive C_k^1, then node i removes j from C_i^3, node j removes i from C_j^3.*

Consistency Enforcement 1 deals with the silence attack in Step 2, when a malicious node k sends its local maximum clique to node i and keep silence to node j. However, simply removing k from C_i^3 is not a good option, because node j may be malicious and lie about the receipt of C_k^1. As a result, a normal node k may become isolated. Thus, the safest way is to split nodes i and j into different cliques.

Consistency Enforcement 2 *If $k \in C_i^2 \cap C_j^2$, node j receives C_k^2 and $j \notin C_k^2$, node i does not receive C_k^2, then node i removes k from C_i^3.*

Consistency Enforcement 2 deals with the silence attack in Step 3, when a malicious node k sends its updated clique to node j, but does not send it to node i. Since node k is the only possible malicious node (among nodes i, j, and k), node i simply removes it from C_i^3.

After performing the above two enforcements, we name the new cliques as C_i^* and C_j^* for i and j, respectively. In Section 3.3.2, we prove that our protocol can guarantee clique agreement through these enforcements.

3.2 Effectiveness in Benign Environments

When all the nodes are normal, our protocol guarantees all the nodes in one clique agree on the same clique membership by following the first three steps.

Lemma 1 *For two nodes i and j, if $i \in C_j^2$ and $j \in C_i^2$, then $C_i^2 = C_j^2$.*

PROOF. In Step 2 of our protocol, after node i receives cliques from all its neighbors, it orders these cliques as $C_{\alpha_1}^1 \overset{i}{\prec} \ldots \overset{i}{\prec} C_i^1 \overset{i}{\prec} \ldots \overset{i}{\prec} C_{\alpha_n}^1$, and updates its clique to the "best" clique $C_i^2 = C_{\alpha_n}^1$. Similarly, node j can have an updated clique $C_j^2 = C_{\beta_n}^1$

From $i \in C_j^2 = C_{\beta_n}^1$, node i can compare $C_{\beta_n}^1$ with $C_{\alpha_n}^1$. Node i has $C_{\beta_n}^1 \overset{i}{\prec} C_{\alpha_n}^1$ since $C_{\alpha_n}^1$ is the best clique among from the cliques from all the neighbors. Because $j \in C_i^2 = C_{\alpha_n}^1$, node i can also derive $C_{\beta_n}^1 \overset{j}{\prec} C_{\alpha_n}^1$. However, from $j \in C_i^2$, node j has $C_{\alpha_n}^1 \overset{j}{\prec} C_{\beta_n}^1$. This can happens only if $\alpha_n = \beta_n$, so we can prove $C_i^2 = C_j^2$. □

Lemma 1 guarantees that if node i and node j contain each other in their updated cliques at the end of Step 2, then their updated cliques must contain the same clique membership.

Lemma 2 *Consider nodes i, j and k, where $k \in C_i^2 = C_j^2$. If $i \notin C_k^2$, then $j \notin C_k^2$.*

PROOF. We prove it by contradiction. Suppose $j \in C_k^2$. Because $i \notin C_k^2$ and $i \in C_i^2 = C_j^2$, we have $C_k^2 \neq C_j^2$. Because $k \in C_i^2 = C_j^2$, by Lemma 1, we have $C_j^2 = C_k^2$. Since $C_k^2 = C_j^2 = C_i^2$, it contradicts to $i \notin C_k^2$. □

From Lemma 1, when node k is included in both node i and node i's updated cliques at the end of Step 2, if node i is not included in node k's updated clique, node j will not be included either. Based on Lemmas 1 and 2, we have the following clique agreement theorem that guarantees all the normal nodes in each clique agree on the same clique membership.

Theorem 1 *For node i and any node $j \in C_i^3$, if all the nodes are normal, we must have $C_i^3 = C_j^3$.*

PROOF. For any node $j \in C_i^3$, $j \in C_i^2$ must hold. We also have $i \in C_j^2$, otherwise j should be removed from C_i^3. By

Lemma 1, we have $C_i^2 = C_j^2$. For any node k that $k \in C_i^2$ but $k \notin C_i^3$, we know $i \notin C_k^2$. Then by Lemma 2, we have $j \notin C_k^2$. Then k will not appear in C_j^3. It means for every node that is removed from C_i^3, it must also be removed from C_j^3. Therefore, we can prove that $C_i^3 = C_j^3$. □

3.3 Security Analysis in Hostile Environments

Malicious nodes may employ different methods to compromise clique agreement among normal nodes. Our protocol can prevent external attacks by using (unicast and broadcast) message authentication. Thus, a malicious node cannot use a fake identity in our protocol without grasping the keying materials. In the following, we focus on the insider attacks in which some participating nodes are malicious.

If malicious nodes broadcast the same false messages or keep silence to all the normal neighbors, they cannot introduce clique inconsistency. Malicious nodes may send inconsistent messages in different steps, so that the cliques are not correctly derived. However, since such attacks generate the same impact on all the normal neighbors, they cannot introduce clique inconsistency either. Therefore, clique inconsistency can only result from sending different messages to different normal nodes, or launching silence attacks from malicious nodes.

In Section 3.3.1, we prove that malicious nodes will be detected and identified if clique inconsistency is caused by sending inconsistent messages. In Section 3.3.2, we prove that our protocol can tolerate silence attacks and clique agreement can be enforced by removing the conflicting nodes.

3.3.1 Identifying Malicious Nodes

We first introduce Lemma 3, and then use it to prove Theorem 2.

Lemma 3 *For two normal nodes i and j, if $j \in C_i^3$, then we must have $i \in C_j^3$.*

PROOF. We prove it by contradiction. Suppose $i \notin C_j^3$. Since $j \in C_i^3$, we must have $j \in C_i^2$. We consider two cases. If $i \notin C_j^2$, j will send C_j^2 to i, then i should remove j from C_i^3 in Step 3. It is contrary to our condition that $j \in C_i^3$. Otherwise, if $i \in C_j^2$ but $i \notin C_j^3$, it means j has removed i from C_j^2. The only reason is that i's clique C_i^2 does not include j, i.e., $j \notin C_i^2$. It contradicts to $j \in C_i^2$. □

Lemma 3 guarantees that if node j is included in node i's final clique, then node j must include node i in its final clique, even in hostile environments.

Theorem 2 *If clique inconsistency is caused by malicious nodes sending inconsistent messages to different normal nodes, our protocol can identify the malicious nodes.*

PROOF. Suppose a normal node i detects clique inconsistency with node j in Step 4, i.e., $j \in C_i^3$ but $C_i^3 \neq C_j^3$. To detect the malicious nodes, node i asks node j to provide its previously received messages and performs Conformity Checking 1 on j. If j passes this checking, it means j follows the protocol correctly, and the inconsistency must come from other nodes. Otherwise, j is malicious.

Consider the case when j performs normally. By Lemma 3, if normal node $j \in C_i^3$, we must have $i \in C_j^3$. So any node k that is not a common neighbor of both node i and j cannot appear in either C_i^3 and C_j^3. Therefore the inconsistency must come from common neighbors of nodes i and j. By performing Conformity Checking 2 on all the common neighbors of i and j, we will find the different messages sent to i and j, and identify the malicious nodes. □

If node j is malicious, node i can detect the conflicts between the messages received from node j in Step 4 and the messages received from node j in Step 5. Because node j provides signatures on these messages, other nodes cannot impersonate it to send fake messages. Thus, node i can use these messages from node j as proofs to inform other nodes in the network. The malicious node j will be removed from the network. Similarly, if a common neighbor node k of node i and node j is malicious, node i can use the messages directly received from node k and node k's messages received and forwarded by node j as proof to remove node k from the network.

3.3.2 Enforcing Clique Agreement

We observe that silence attacks can introduce clique inconsistency only in Steps 2 and 3. In Step 1, a malicious node may send its neighbor list to some neighbor nodes, but withhold it from other neighbor nodes. However, in Step 2, our protocol allows a normal node i update its clique to a "better" clique, even if the better clique contains some nodes that did not send their neighbor lists to node i in Step 1. Thus, the silence attack in Step 1 will not cause clique inconsistency.

In Step 2, clique inconsistency can only come from the "better" cliques sent by malicious nodes, since a normal node will update its clique to a "better" clique. Suppose nodes i and j are normal. A malicious node k may send i a "better" clique C_k^1 that includes i and j, but withhold the message from node j. Then node i updates its clique to C_k^1. If node j receives the "better" clique from node i, it updates its clique to C_i^1. Therefore, node i and j include each other in their cliques that are inconsistent. However, Consistency Enforcement 1 can remove such clique inconsistency.

In Step 3, clique inconsistency can only be introduced by removing nodes from cliques. Suppose $k \in C_i^2 \cap C_j^2$. In Step 3, node k can send a clique to remove itself from i's clique, while keeping silence to j. Then the final clique of j contains k, which is not in node i's final clique.

In Step 4, after a normal node i receives a final clique C_k^3 from node k, node i rebroadcasts C_k^3 if $k \in C_i^3$. Because

we assume the messages from a normal node can be received correctly by its normal neighbors, this rebroadcast can guarantee that if one normal node receives C_k^3 from node k, all the other normal nodes in the same clique can receive C_k^3. Thus, it can prevent silence attacks in Step 4.

In the following Theorem 3, we prove that by removing the inconsistent nodes from cliques through the consistency enforcement, all the normal nodes can achieve clique agreement even if malicious nodes intentionally keep silence to certain normal nodes.

Theorem 3 *For any two normal nodes i and j, after Step 5, if $j \in C_i^*$, we have $C_i^* = C_j^*$.*

PROOF. We prove it by contradiction. Suppose $C_i^* \neq C_j^*$. Since our protocol can only remove nodes from cliques when inconsistency is detected, C_i^3 must contain all the nodes in C_i^*. Therefore $j \in C_i^3$. By lemma 3, we have $i \in C_j^3$. We consider two cases.

First, suppose $C_i^3 \neq C_j^3$ and $C_i^* \neq C_j^*$. Without loss of generality, we assume node $k \in C_i^3$ but $k \notin C_j^3$. Nodes i and j find inconsistency after exchanging C_i^3 and C_j^3. By Consistency Enforcement 1, node i removes j from its clique, and node j also removes i from its clique. Therefore we have $j \notin C_i^*$. It is contrary to the condition $j \in C_i^*$.

Second, we assume $C_i^3 = C_j^3$, but $C_i^* \neq C_j^*$. Without loss of generality, suppose node $k \in C_i^*$ but $k \notin C_j^*$. Because nodes can only be removed to enforce clique agreement in Step 5, k cannot be added to C_i^*, but removed from C_j^*. This means C_k^3 is inconsistent with C_j^3. Since $C_i^3 = C_j^3$, C_k^3 is also inconsistent with C_i^3. Because node j re-broadcasts the clique C_k^3 received from k, node i will receive C_k^3 even if node k keeps silence to i. Thus, i should remove k from C_i^*. We find contradiction. □

In our protocol, the clique consistency checking is only performed in Step 4, though it can be executed in each step. The reason is to reduce the computation overhead by decreasing the number of signature generation/verification. Each node need not verify the signatures from other nodes unless it detects clique inconsistency. Even if clique inconsistency is detected, each node only generates and verifies the signatures of the messages exchanged in Step 4 and Step 5. If the protocol checks the consistency in every step, malicious nodes may be detected in an earlier step. However, the computation overhead will be increased a lot.

3.4 Performance Analysis

Computation Overhead: We make several efforts to lower the computation overhead in our protocol. In all the five steps, each node i uses μTESLA to authenticate its broadcast messages. Because μTESLA uses secure key cryptography that has much less computation overhead than public key cryptography, we only analyze the computation overhead on public key operations.

In Step 4, each node i signs the secure hash of its local messages sent in the first four steps, instead of signing each message individually. Each node need not verify the signatures from other nodes unless it detects clique inconsistency with them. Therefore, in benign environments, no signature verification is necessary. In hostile environments, after detecting a clique inconsistency with node j, node i verifies the signature from node j. In Step 5, after receiving node i's request, node j generates a signature on the secure hash over the previous received messages from its neighbors. Then, node i needs to verify node j's signature on the forwarded messages. If node j passes Conformity Checking 1, node i needs to verify $|L_i \cap L_j|$ signatures from the common neighbors of i and j.

Because a node may verify more messages than those it signs, we propose to choose public key cryptosystems with a fast decryption speed, such as RSA, which can verify one signature in $0.43s$ on ATmega128 [13]. Since the clique formation process will not be performed frequently, the computation overhead is acceptable for sensor nodes.

Communication Overhead: Each node i broadcasts one message in each of the first three steps. In Step 4, besides broadcasting its final clique, node i also rebroadcasts the first copy of the final clique message about a neighbor in node i's final clique C_i^3. In total, node i sends $|C_i^3| + 3$ messages.

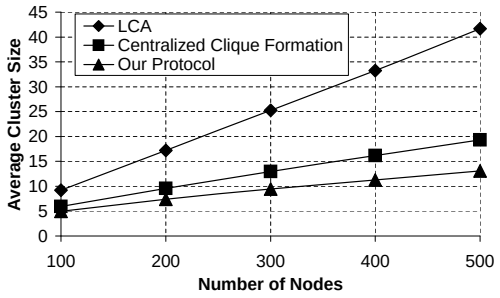
Suppose node j has $|L_j|$ neighbors. When node i detects a clique inconsistency and requests node j to forward its previously received messages in Step 5, node j needs to forward $4|L_j|$ messages received in the first four steps, plus one message including the signature for the secure hash over all the forwarded messages.

Storage Overhead: According to the analysis of the computation overhead, each node i should store all the $4|L_i|$ messages received in the four steps, where $|L_i|$ is the neighbor number of node i. When node i detects a clique inconsistency with node j, node i needs to store $4|L_j| + 1$ messages from node j. Node i can release the memory after verifying these messages.

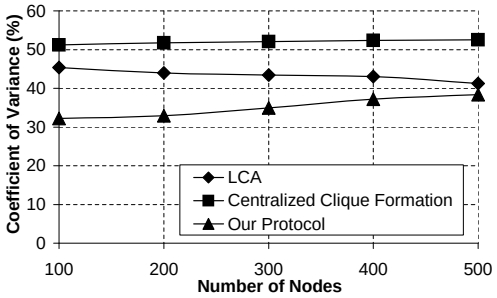
4 Experimental Results

Through simulation, we show that our protocol can provide secure cluster formation without sacrificing the performance of the clusters. We use the following metrics to evaluate the cluster characteristics: *average cluster size*, *maximum size of clusters*, *variance of the cluster size*, and *number of single-node clusters*.

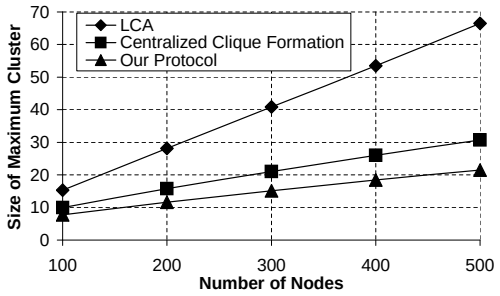
The average cluster size depends on the density of the networks and the transmission range of the sensor nodes. The average cluster size should not be too small. In sensor networks, it is not desirable to include too many nodes in a large cluster due to the increasing message collisions and transmission delay in a large cluster. We use Coefficient of Variance (CV) = 100*(Standard Deviation)/(mean value of set) to evaluate the variance of the cluster size. We expect to divide nodes into clusters with a low coefficient of variance. A



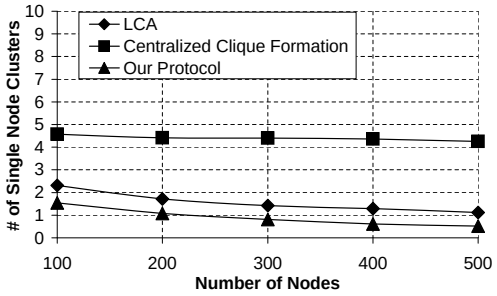
(a) Average Cluster Size



(b) CV (%) on Cluster Size



(c) Size of the Maximum Cluster



(d) # of Single-Node Clusters

Figure 2. Comparison of Cluster Metrics

cluster formation protocol should minimize the clusters with a single node.

In our simulation, we uniformly deploy 100, 200, 300, 400 and 500 sensor nodes in a 100×100 (m^2) simulation area, respectively. The transmission range of all the sensor nodes is fixed to 20 meters. Each point in the result figures is the average result of 1000 experiments.

We compare the cluster characteristics of our distributed protocol to LCA [3], one typical Leader-First based cluster formation protocol, and a centralized clique formation protocol. In LCA, from the lowest ID node to the highest ID node, a node declares itself to be a cluster-head if it has the lowest ID among the non-covered neighbor nodes. A node is covered if it is in the 1-hop neighborhood of a node who has declared itself to be a cluster-head. In the centralized clique formation protocol, we assume a sink node has obtained the topology graph G of the whole network. The sink node first finds the maximum clique C_1 in G, and updates G by removing C_1 from G. Then, it finds the maximum clique C_2 in the remaining G, and then removes C_2 from current G. The algorithm completes when G becomes empty. We borrow the C implementation (dfmax) from [1] to find a maximum clique in a random graph.

Figure 2 compares the cluster characteristics of three protocols. As Figure 2(a) shows, the average cluster sizes of the three protocols increase with the node density of the network. Our protocol has a smaller average cluster size than the other two protocols. The reason is that our protocol requires all the nodes in a clique be able to directly communicate with each other. While, in LCA, the maximum distance between any two nodes in one cluster is two hops. Compared to the centralized clique formation protocol, our heuristic protocol in Step 1 may not find the maximum local clique. Thus, the average cluster number is a little smaller.

Figure 2(b) shows the variance of the cluster sizes. Our protocol has a smaller coefficient of variance than the other two protocols, which means our protocol generates more uniform clusters. Figure 2(c) presents the maximum cluster sizes in three protocols. Our protocol has a moderate maximum cluster size. As Figure 2(d) shows, our protocol has fewer single-node clusters than the other two protocols. The reason is that LCA and the centralized clique formation protocol attempt to form the largest cluster first, and thus leave some nodes into small clusters. While in our protocol, because all the nodes choose their clusters in a distributed and parallel way, it decreases the chances to form large clusters and single-node clusters.

5 Related Work

The cluster structure in sensor networks can help to achieve scalability, power saving, channel access, routing, etc. In recent years, many Leader-First cluster formation protocols have been proposed by selecting the cluster heads with respect to one or multiple metrics, such as node IDs (e.g., [3]), node connectivity (e.g., [6, 11]), node mobility (e.g., [5,7]), residual energy (e.g., [4,7,14,30]). Several cluster formation protocols (e.g., [2, 17]) have been proposed by considering the cluster heads selection problem as a special case of finding the minimum dominating set (MDS) problem. Several Cluster-First clique formation protocols (e.g., [16, 18, 24, 27]) have been proposed for sensor networks.

All the above cluster formation protocols assume benign

environments, but cannot resist attacks in hostile environments. In [26], two secure clustering formation algorithms are proposed for wireless ad hoc networks. It depends on a trusted authority to certify each node's metrics used in the leader election process. However, these algorithms are not fault tolerant, since they assume all the participating nodes are reliable and no messages are lost or delayed, which cannot be guaranteed when there exist malicious nodes. Moreover, a centralized trusted authority may not be always available.

In malicious environments, our secure cluster formation protocol guarantees that all the normal nodes in each group (clique) agree on the same group membership. The cluster formation problem is different from the traditional Byzantine Agreement problem [19], which is to guarantee all the correct nodes in a group agree on a single value sent from a single (possible malicious) node. Traditional authenticated Byzantine Agreement algorithms (e.g., [8, 12, 25]) cannot be directly applied to achieve secure cluster formation.

6 Conclusion and Future Work

We proposed a secure and distributed clique formation protocol for sensor networks to divide sensor nodes into mutually disjoint cliques. The clique structures built by our protocol can be widely used in sensor network applications, such as routing, data fusion, time-slotted scheduling, etc. Currently, our protocol is suitable for static sensor networks, in which nodes do not move frequently. We plan to investigate how to extend our protocol into mobile sensor networks.

References

[1] dfmax.c. `ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/`.

[2] A. Amis, R. Prakash, T. Vuong, and D. Huynh. Max-Min D-cluster formation in wireless ad hoc networks. In *INFOCOM*, 1999.

[3] D. Baker, A. Ephremides, and J. Flynn. The design and simulation of a mobile radio networkwith distributed control. *IEEE Journal on Selected Areas in Communications*, SAC-2(1):226–237, 1984.

[4] S. Bandyopadhyay and E. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM*, 2003.

[5] S. Basagni. Distributed clustering for ad hoc networks. In *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99)*, 1999.

[6] H. Chan and A. Perrig. ACE: An emergent algorithm for highly uniform cluster formation. In *European Workshop on Wireless Sensor Networks (EWSN 2004)*, Jan 2004.

[7] M. Chatterjee, S. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5(2):193–204, 2002.

[8] D. Dolev and H. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal of Computing*, 12(4):656–665, 1983.

[9] J. R. Douceur. The sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Mar 2002.

[10] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman And Company, 1979.

[11] M. Gerla and J. T. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[12] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Dependable Computing for Critical Applications–5*, volume 10, pages 139–157, sep 1995.

[13] N. Gura, A. Patel, and A. Wander. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2004.

[14] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.

[15] Y. Hu, A. Perrig, and D. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *INFOCOM*, April 2003.

[16] H. Ishii and H. Kakugawan. A self-stabilizing algorithm for finding cliques in distributed systems. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, Oct 2002.

[17] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 2001.

[18] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *SIGCOMM Computer Communication Review*, 27(2), 1997.

[19] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[20] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[21] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In *SECON*, October 2004.

[22] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In *IEEE Symposium on Security and Privacy*, May 2005.

[23] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks*, July 2001.

[24] T. Predrag and G. Agha. Maximal clique based distributed group formation for autonomous agent coalitions. In *Coalitions and Teams Workshop (W10), 3rd Int'l Joint Conf. on Agents and Multi Agent Systems*, 2004.

[25] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[26] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. In *DARPA Information Survivability Conference and Exposition DISCEX*, 2003.

[27] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom*, 2001.

[28] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM*, June 2002.

[29] C. Young and J. A. Stevens. Clique activation multiple access (cama): A distributed heuristic for building wireless datagram networks. In *MILCOM*, 1998.

[30] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *INFOCOM*, 2004.