The Sleep Deprivation Attack in Sensor Networks: Analysis and Methods of Defense

Matthew Pirretti, Sencun Zhu, Vijaykrishnan Narayanan,
Patrick McDaniel, and Mahmut Kandemir
The Pennsylvania State University
University Park, PA 16801
Email: pirretti@cse.edu

Richard Brooks
Clemson University
Clemson, SC 29634
Email: rrb@acm.org

Abstract— **The ability of sensor nodes to enter a low power sleep mode is very useful for extending network longevity. We show how adversary nodes can exploit clustering algorithms to ensure their selection as cluster heads for the purpose of launching attacks that prevent victim nodes from sleeping. We present two such attacks: the *barrage attack* and the *sleep deprivation attack*. The barrage attack bombards victim nodes with legitimate requests, whereas the sleep deprivation attack makes requests of victim nodes only as often as is necessary to keep the victims awake. We show that while the *barrage attack* causes its victims to spend slightly more energy, it is more easily detected and requires more effort on behalf of the attacker. Thus we have focused our research on the sleep deprivation attack. Our analysis indicates that this attack can nullify any energy savings obtained by allowing sensor nodes to enter sleep mode. We also analyze three separate methods for mitigating this attack: the *random vote* scheme, the *round robin* scheme, and the *hash-based* scheme. We have evaluated these schemes based upon their ability to reduce the adversary's attack, the amount of time required to select a cluster head, and the amount of energy required to perform each scheme. We have found that of the three clustering methods analyzed, the hash-based scheme is the best at mitigating the sleep deprivation attack.**

I. Introduction

Power management is a major research issue in sensor networks, given that sensor nodes are constrained to operate upon limited battery supplies. In general, the total energy consumption of a sensor node can be categorized into the following three groups: *1)* energy spent on computations, *2)* energy used to communicate, and *3)* energy spent as a result of the node's sleeping patterns. In this paper we focus on the ability of sensor nodes to enter a low power sleep mode for the purpose of extending the longevity of the network. It is widely accepted that this is an important issue as sensor nodes spend most of their time in sleep mode, allowing the overall lifetime of the node to be greatly extended [1]–[6].

The ability of sensor nodes to enter a sleep mode becomes a serious concern for sensor networks deployed in *unattended* and *hostile* environments. Limited tamper resistance inherent to a low cost sensor node leaves the node vulnerable to being compromised by an adversary [2], [7], [8]. Through compromised nodes, an adversary may launch security attacks against the sensor network ranging from the physical layer to the application layer. Due to the vast variety and novelty of attacks, we believe no single solution can address all the attacks. As such, we limit our work to addressing a specific type of attack aimed at disrupting the power management protocol in sensor applications such as target tracking.

We present two attacks which prevent victim nodes from sleeping: the *barrage attack* and the *sleep deprivation attack*. In the barrage attack the victim is barraged with seemingly legitimate requests; however, the purpose of these requests is to waste the victim's limited power supply by causing it to stay out of its sleep mode and perform energy intensive operations. In the sleep deprivation attack, the malicious node makes requests to victim nodes only as often as is necessary to keep the victims awake. Thus victim nodes are kept awake, but are not made to perform energy intensive operations as is the case in the barrage attack. From our analysis, we have found that while the barrage attack causes its victims to spend slightly more energy, it is more easily detected and requires much more effort on behalf of the attacker. For these reasons we have focused our research on the sleep deprivation attack.

To further analyze the sleep deprivation attack we have used a clustering based target tracking network as the basis for our analytical model. It was the decision of the authors to utilize a specific application rather than a more general model in order to grant the reader further insight into the mechanisms of the sleep deprivation attack. Note, that while our analysis was done based upon a particular implementation of target tracking, our results are readily applicable to any clustering based sensor network application. We show that for one set of network parameters, an adversary can double the power consumption of a 400 node network with as few as 20 compromised nodes. Further we show that a single adversary node can simultaneously attack as many as 150 victims and still outlive its victims.

Given the detrimental effect of the sleep deprivation attack, we have analyzed three separate defense mechanisms to mitigate this attack: the *random vote* scheme, the *round robin* scheme, and the *hash-based* scheme. We have evaluated these schemes based upon their ability to mitigate the adversary's attack and the amount of time required to select a cluster head. We have found that of the three clustering methods analyzed, the hash-based scheme is the best at mitigating the sleep deprivation attack. Given the viability of the hash-based scheme we have also analyzed the amount of energy it requires to select a cluster head.

The remainder of this paper is organized as follows. In Section II we discuss related works. In Section III we introduce the framework used to perform our analysis. In Section IV we compare the barrage attack and the sleep deprivation attack. In Section V we further analyze the sleep deprivation attack and in Section VI we discuss methods to mitigate this attack. In Section VII we make our conclusions.

II. Related Work

A. Sleep Deprivation Attack

The idea of the *sleep deprivation attack* was first proposed by Stajano [9], [10]. The victim of this attack is a battery powered computing device, such as a sensor node, which attempts to remain in a low power sleep mode for as long as would possible without adversely affecting the node's applications. The attacker launches a sleep deprivation attack by interacting with the victim in a manner that appears to be legitimate; however, the purpose of the interactions is to keep the victim node out of its power conserving sleep mode. Thus this attack can be used to dramatically reduce the lifetime of the victim. Further, this attack is difficult to detect given that it is carried out solely through the use of seemingly innocent interactions.

The work by Krishnaswami is one of few works that has attempted to quantify the impact of sleep deprivation attacks on energy-constrained devices [11]. The approach used by Krishnaswami consisted of measuring the difference between the average power consumption of laptops and

PDAs while the devices were idle (but not in sleep mode) and when the devices were under attack. We find Krishnaswami's approach to be overly simplistic in how devices were attacked. The adversary gives the victim a task that is specifically designed to burn inordinately large amounts of energy. Such an attack could be easily detected given that sensor nodes are by necessity, very diligent about their power management. We claim that a much more difficult to detect attack would be to have the adversary focus on keeping the victim out sleep mode, rather than try to have the victim actively processing.

B. Cluster Head Selection

In sensor networks clustering is used to organize sensor nodes into groups based in part on their physical proximity [2]. One of the nodes in the cluster is delegated the task of being the cluster head. The main benefit of clustering is that it enables *data fusion* [12], whereby redundant data is pruned from the network. In the case of target tracking, several nodes may detect the same target. It is wasteful for each of these sensor nodes to transmit identical tracking information to a centralized data processor. A more efficient solution is to have a group of nodes perform local processing of sensor data and transmit a single message.

In the clustering algorithm proposed in [13], clusters are formed by having each sensor node wait a random amount of time. If a node has not had the opportunity to join a cluster after this random amount of time, then it can declare itself to be a cluster head and subsequently start soliciting neighboring nodes to join its cluster. To maintain the cluster, the cluster head will select its own successor. We foresee two vulnerabilities with this approach. First, during cluster formation an adversary could ensure its selection as cluster head by immediately soliciting other nodes to join its cluster. Second, once an adversary node has been selected as cluster head it can remain cluster head indefinitely, by never selecting a successor. Consequently, this approach readily allows an adversary to launch a sleep deprivation attack.

In [14], each node declares itself a cluster head with a probability p. Each cluster head will then solicit any other node within k hops to join its cluster. Nodes receiving multiple solicitations join the closest cluster available. A node that has not received a cluster solicitation within a certain amount of time will then declare itself to be a cluster head, and will solicit other sensor nodes within k hops to join its cluster. Clearly this algorithm is vulnerable to a sleep deprivation attack by allowing a malicious node to simply declare itself a cluster head.

There are many other distributed clustering algorithms and sensor network applications which rely upon clustering (e.g. [14]–[20]), each of which assumes that participating nodes will act honestly. Thus an adversary can exploit each of these algorithms to ensure its selection as cluster head. Given that clustering is a widely used algorithm, it is crucial to make it secure.

C. Target Tracking

For didactic purposes we pose our discussion of the sleep deprivation attack within the context of a sensor network executing a distributed clustering target tracking application; however, our results are more generally applicable to any algorithm that relies upon clustering.

Target tracking research has focused on such issues as increasing the fidelity of sensor data, reducing redundant data, and reducing energy consumption. Since many of these algorithms [21]–[27] rely on a trusted cluster head, they are all susceptible to the sleep deprivation attack.

III. System Modelling

A. Model Assumptions

1) Node and Network Characteristics: We consider the placement of sensor nodes to be spread uniformly at random throughout a square region. The sensor nodes operate on non-renewable batteries; once a node exhausts its battery it is considered to be dead. To preserve their battery power sensor nodes will cycle in and out of a low-power sleep state.

We assume the presence of a publish/subscribe routing protocol [28], [29]. A publish/subscribe protocol has two network primitives, a *publish* method and a *subscribe* method. The subscribe method is used by individual nodes to indicate their desire to receive data fitting a specified description. In our model, sensor nodes subscribe to all track records within z meters of their position. When a local group of nodes detects a target they will share sensor readings. One of these nodes is then elected as cluster head. This node aggregates the sensor data on behalf of the cluster and forms a track record. The track record is sent to all subscribed nodes by repeatedly invoking the publish method. The publish method is invoked one time for each subscribed node. For simplicity we assume that track records are sent reliably(i.e. a track record will always reach its recipient node despite network errors or sleeping sensor nodes).

We assume a homogeneous network, where a cluster head is identical to any other node in the cluster in terms of capacity and resources. For simplicity we shall assume that nodes within the same cluster can communicate directly. This assumption could be relaxed by allowing multihop communication.

2) Security Assumption and Attack Model: We assume that sensor nodes do not have significant tamper resistance. Thus an adversary is capable of compromising sensor nodes [2], [7], [8]. That is, the adversary can obtain data, keys, and the code inside compromised nodes. To launch attacks, the adversary reprograms the compromised nodes with malicious code and then redeploys them into the network. Compromised sensor nodes may launch attacks in different layers in the protocol stack, e.g., channel jamming in the physical layers [7], disrupting the collaboration of sensor nodes in the MAC layer protocol [30]–[32], attacking the routing protocol [33], jeopardizing the localization service [34], [35] or sensor data [36]. Due to the diversity and novelty of attacks, no one-for-all solution exists. All of the aforementioned attacks have been addressed separately. As such, although compromised nodes (including cluster heads) may launch various attacks, in this work we limit our focus to identifying, analyzing, and defending against the attacks which specifically target the network's power management protocols.

We assume each sensor node is assigned a unique ID prior to deployment. Further, sensor nodes cannot impersonate uncompromised nodes. Preventing impersonation could be accomplished by requiring every node to authenticate its messages using pairwise keys shared with receiver nodes [37], [38]. This assumption ensures that a compromised node can only submit a single *vote* in a cluster head selection algorithm, such as the algorithm described in Section VI-C.

We consider the placement of adversary nodes to be uniformly random; this reflects the adversary's desire to spread its nodes in order to maximize their impact. In this paper the terms *malicious nodes* and *adversary nodes* are synonymous to such compromised nodes.

B. Sensor Node States

To help quantify the impact of an adversary attack we have partitioned the nonadversary sensor nodes into three separate states: *sleep*, *idle*, and *receive* (note that the lack of a transmit state reflects the fact that nonadversary nodes do not generate network traffic in our model; we have made this assumption to help illuminate the impact of an adversary attack better). As Table I indicates, the state of a sensor node is defined by the status of its CPU and wireless radio. Accordingly we consider the behavior of sensor nodes to cycle between the sleep state and the idle state, staying in each state for fixed time quantums. A sensor node in the idle state will enter the receive state upon receipt of a track record. This will only occur when the sensor node is subscribed to an adversary node. Once a sensor node has received a track record it will stay in idle mode for an extended period of time.

TABLE I
NONADVERSARY NODE STATE SPACE

Mode	CPU Status	Radio Status
Sleep	Sleep Mode	Off
Idle	On	Standby
Receive	On	Receiving

Similarly we have partitioned the adversary nodes into two states: *sleep*, and *transmit*. Table II indicates the status of the CPU and wireless radio for each state. In order for an adversary node to maximize its energy, it will transmit track records to each of its subscribed nodes in rapid succession and then go to sleep.

TABLE II
ADVERSARY NODE STATE SPACE

Mode	CPU Status	Radio Status
Sleep	Sleep Mode	Off
Transmit	On	Transmitting

To make our analysis realistic we have used values taken from actual sensor node specifications as seen in Table III. We were able to obtain actual values for power, data rate [2], and packet size [39]. Given symmetric upload and download speeds for each link, we derived t_{rs}, the time required to send or receive a packet. The timing values in Table III (t_a, t_{ar}, and t_{slp}) are based upon ongoing experiments being performed at our lab.

TABLE III
MODEL PARAMETERS

Param	Value	Description
P_{idle}	300 mW	Power in idle mode
P_{slp}	1 mW	Power in sleep mode
P_{rcv}	440 mW	Power in receive mode
P_{snd}	510 mW	Power in send mode
R	10 kbps	Data transmission rate
T_{size}	296 bytes	Track record size
t_{rs}	236.8 ms	Time to send/receive track record
t_{idle}	1 s	Time spent in idle mode
t_{ar}	60 s	Time spent waiting for a target
t_{slp}	1 s	Time spent in sleep mode

In Figure 1 we illustrate what we consider normal sensor node behavior when no tracking is occuring. A sensor node alternates between sleep mode for t_{slp} seconds and idle mode for t_{idle} seconds.

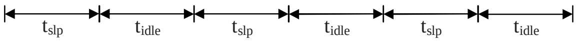

Fig. 1. Timing diagram of normal node behavior when no tracking is occurring

In Figure 2 we illustrate the behavior that a sensor node would exhibit upon receiving a single track record (represented pictorially by "TR" in the figure). Initially the sensor node is alternating between sleep and idle states, until its idle state is cut short by receipt of a track record. After t_{rs} seconds the node has received the entire track record and has determined that a possible target is coming its way. After waiting t_{ar} seconds for a target, the node determines it is safe to go back to sleep. At this point the node resumes the behavior exhibited by Figure 1. There are three important observations to note. First, only a cluster head will generate a track record. Second, a track record is the only message that will interfere with a sensor node's sleep cycle. Third, t_{ar} is much larger than is depicted in Figures 2 and 3.

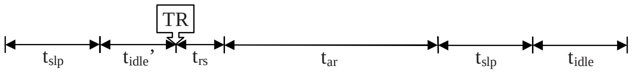

Fig. 2. Timing diagram of normal node behavior after receiving track record

Figure 3 illustrates a sensor node being victimized by a sleep deprivation attack. The attacker sends the victim a track record every $t_{ar} + t_{rs}$ seconds. Each time the victim is just about to decide that it can go to sleep mode, it receives another track record. Thus the victim is kept perpetually awake.

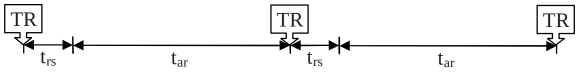

Fig. 3. Timing diagram of node under sleep deprivation attack

Figure 4 shows a sensor node that is under a barrage attack. Notice that the adversary sends the victim track records as rapidly as possible (i.e. every t_{rs} seconds).

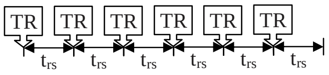

Fig. 4. Timing diagram of node under barrage attack

IV. COMPARISON OF SLEEP DEPRIVATION ATTACK AND BARRAGE ATTACK

In this section we explain the tradeoffs between the sleep deprivation attack and the barrage attack. Further, we explain why the sleep deprivation attack would be the preferable method of attack from the perspective of the attacker.

In both the sleep deprivation attack and the barrage attack the victim will never enter its low power sleep mode. The difference between the two attacks is that the victim of a barrage attack will be actively performing work, whereas the victim of a sleep deprivation attack will, for the most part, remain idle. In this section we investigate the difference between these two attacks based upon the following: *1)* the victim's power consumption, *2)* detectability of the attack, and *3)* the power required of the adversary.

A. Power Consumption of Victim

Consider the following scenario. A particular sensor node cycles in and out of its low power sleep mode. The proportion of time that the sensor node is asleep versus awake is denoted by the variable x. This sensor node has been given a small workload, thus it spends 300 mW in idle mode and 1 mW in sleep mode, giving the following equation denoting the sensor node's average power consumption in mW:

$$300(1 - x) + 1x \tag{1}$$

Thus this node spends 1 mw of power when $x = 0$ and it spends 300 mw of ow power when $x = 1$.

Now consider if this same node was victimized by a sleep deprivation attack or a barrage attack. In the sleep deprivation attack the adversary will interact with the sensor node intermittently so that it will never go to sleep, thus the victim will spend an average of 300 mW of power. Alternatively, if the adversary was to utilize a barrage attack, the victim will be constantly receiving messages, causing it to spend 440 mW of power. Clearly the victim of a sleep deprivation attack spends slightly less power than the victim of a barrage attack. It is also clear that the relative impact of either attack is directly proportional to x.

B. Detectability

An energy draining attack must remain undetected for as long as possible in order to waste a maximum amount of energy. An easily detected attack allows the victim to rapidly take remedial action, such as ignoring the requests of a detected attacker, before the attack can cause significant damage.

We speculate that the sleep deprivation attack would be much more difficult to detect than the barrage attack based on the observation that it generates messages at relatively infrequent rate. Using the parameters in Table III the barrage attack requires 254 messages to be sent every minute, whereas the sleep deprivation attack only requires a single message to be sent.

C. Energy Required of Attacker

Since the adversary nodes are compromised sensor nodes, they would be more likely to perform an attack which can cause a lot of damage without requiring a lot of energy to perform.

In a barrage attack the adversary has to transmit a network message every t_{rs} seconds, whereas in sleep deprivation attack the adversary only has to generate a network message every $t_{rs} + t_{ar}$ seconds. Thus the adversary has to spend a lot more power to launch the barrage attack. Considering that the adversary could simultaneously launch sleep deprivation attacks upon multiple sensor nodes for less power consumption than would be required to launch a barrage attack on a single node, it seems likely that the battery-constrained adversary nodes would prefer the sleep deprivation attack.

V. Sleep Deprivation Attack

In Section IV we showed that the sleep deprivation attack is a serious threat to a sensor network given that it nullifies any energy savings that would have been obtained by the victims' low power sleep mode. In this section we illustrate how an adversary can utilize the security vulnerabilities inherent to distributed cluster formation for the purpose of launching a sleep deprivation attack. We show that the sleep deprivation attack can as much as double the overall power consumption of a 400 node network. Further we find that this can be accomplished with as few as 20 adversary nodes.

To quantify the impact of this attack we measure the average node's power consumption. This is a useful metric because it measures the network-wide impact of the adversary's attack. As a point of clarification only the power consumption of non-adversary nodes will be measured in this metric.

A. Derivation of Model

We now derive the equations which we have used to analyze the impact of the sleep deprivation attack.

1) Probability of Subscribing to an Adversary Node: Using a geometric argument, the probability that a sensor node will be subscribed to an adversary cluster head, given that each node is deployed in a square region of length L and each node subscribes to all track records within a radius of size $z < L$, is:

$$(\pi z^2)/L^2 \tag{2}$$

The complement of this equation gives the probability that a sensor node is not subscribed to a particular adversary node. Given C adversary nodes with uniformly random placement, the probability that a given node will be subscribed to at least one malicious cluster head is:

$$p = 1 - (1 - (\pi z^2)/L^2)^C \tag{3}$$

Note that this analysis will be slightly skewed due to fringe effects. Nodes on the edges of the network are less likely to be in contact with a adversary cluster head. However, only for small networks will these fringe effects will be substantial, and thus we shall ignore their effects in our analysis.

2) Power Consumption of Nonadversary Nodes: To determine the average node's power consumption we have partitioned sensor nodes into two groups: those that are subscribed to adversary cluster heads and those that are not.

Nodes that are not subscribed to an adversary cluster head alternate between the idle state for t_{idle} seconds and the sleep state for t_{slp} seconds. Thus the rate that these nodes expend energy is:

$$P_{nrm} = \frac{t_{idle}P_{idle} + t_{slp}P_{slp}}{t_{idle} + t_{slp}} \tag{4}$$

Nodes that are subscribed to an adversary cluster head alternate between an extended idle state for t_{ar} seconds and the receive state for t_r seconds. Thus nodes subscribed to a malicious cluster head expend energy at the following rate:

$$P_{atk} = \frac{t_{ar}P_{idle} + t_{rs}P_{rcv}}{t_{ar} + t_{rs}} \tag{5}$$

Utilizing the equations for p, P_{nrm}, and P_{atk} it is straightforward to attain the average power consumption for non-adversary nodes:

$$P_{avg} = pP_{atk} + (1-p)P_{nrm} \tag{6}$$

3) Power Consumption of Adversary Nodes: Assume each adversary node goes through two states: first it sends out a track record to each of its subscribed nodes and then it sleeps. Adversary nodes send these messages as rapidly as possible in order to maximize sleep time. This is modeled by each adversary node sending x such messages to x victim nodes. As Figure 3 indicates, the victims have to receive track records every $t_{ar}+t_{rs}$ seconds to be coerced into staying out of sleep mode. Thus the adversary spends xt_{rs} seconds sending messages and $t_{ar}+t_{rs}-xt_{rs}$ seconds sleeping. Therefore the total power required of the adversary is:

$$P_{adv} = \frac{xt_{rs}P_{snd} + (t_{ar} + t_{rs} - xt_{rs})P_{slp}}{t_{ar} + t_{rs}} \tag{7}$$

4) Impact of Attack: Figure 5 illustrates the effect of malicious nodes on a network containing 400 legitimate nodes for different subscription radii (i.e. z). Our results indicate that as more malicious nodes are inserted, the average power consumption increases asymptotically to P_{idle}, meaning that at some point every sensor node will never go to sleep. We have also observed that it is desirable to keep z as small as possible to minimize the effect of malicious nodes. Unfortunately, this also reduces the ability of tracking nodes to give each other advance notice of a potential target. Thus a network designer should carefully consider the implications of selecting a particular subscription radius.

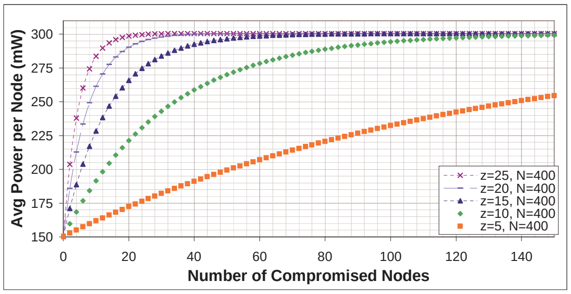

Fig. 5. Effect of Subscription Radius in Sleep Deprivation Attack

5) Feasibility of Attack: To show that the attack is viable for an adversary to perform, we utilize the equation for P_{adv} to show the energy required for the attacker to launch the attack. From our analysis an adversary node could simultaneously attack up to 150 nodes before its power consumption became higher than that of its victims, and thus from a power perspective this attack does not require the attacker to expend a great deal of energy. Therefore we conclude that the sleep deprivation attack enables the adversary to cause a lot of damage without expending a lot of effort.

VI. Secure Cluster Head Selection

We have shown that the sleep deprivation attack greatly increases the sensor network's power consumption without requiring excessive power to be consumed by the adversary. To launch this attack the adversary nodes must become cluster heads, which we have shown to be exceptionally easy. In this section we propose several algorithms which make it much more difficult for the adversary to become cluster head, and consequently these techniques greatly reduce the impact of the sleep deprivation attack. We assume adversary nodes are interested in increasing their chances at becoming cluster heads. Thus we do not consider an attack where the adversary intentionally delays cluster head selection.

A. Random Vote Cluster Head Selection

We have observed that clustering algorithms rely on the honesty of all participating nodes, allowing a malicious node to generate false information to ensure its selection as cluster head. The *random vote* scheme counteracts this characteristic by randomizing cluster head selection.

1) Overview of Random Vote Scheme: A group of local nodes using the random vote scheme form a cluster by performing the following steps:

1) Each node locally broadcasts its unique ID.
2) Each node uses a pseudorandom number generator to pick the ID of the local node it desires to become the next cluster head.
3) Nodes locally broadcast the ID of their desired cluster head.
4) Each node repeats step 2 until a single node attains a majority of votes. This node will become the next cluster head.

Each sensor node is only allowed to cast one vote at a time. However, in the case of a tie the nodes will restart the algorithm at step 2. We refer to the execution of steps 2 through 4 as a *round* or *iteration*. Thus the random vote algorithm is potentially composed of several rounds.

2) Analysis of Random Vote's Effectiveness at Increasing Attack Tolerance: For simplicity we have assumed that the N legitimate nodes and the C compromised nodes in the network have been evenly divided into G clusters. Thus we assume that there are n legitimate nodes per cluster and c malicious nodes per cluster.

To determine the effectiveness of the random vote scheme, we have derived an equation indicating the probability that any of the malicious nodes in the cluster will be selected as cluster head, given that i out of n nonmalicious nodes have voted for it. The adversarial nodes in the local cluster wait until all other nodes have voted so that they can all vote for the adversary node which currently has the most votes. The probability that this node attains a majority of the $n+c$ votes is modeled with a binomial distribution as follows:

$$p_{rand,c} = \sum_{i=\left\lfloor \frac{n+c}{2} \right\rfloor - c+1}^{n} \binom{n}{i} {\frac{1}{n+c}}^{i} \left(1-\frac{1}{n+c}\right)^{n-i} \tag{8}$$

Similarly the probability that any of the n nonadversary nodes are selected as a cluster head is:

$$p_{rand,n} = \binom{n}{1} \sum_{i=\left\lfloor \frac{n+c}{2} \right\rfloor +1}^{n} \binom{n}{i} {\frac{1}{n+c}}^{i} \left(1-\frac{1}{n+c}\right)^{n-i} \tag{9}$$

The equations for $p_{rand,c}$ and $p_{rand,n}$ are only for a single iteration of the algorithm. We utilize the three state Markov chain in Figure 6 to determine the probability of either an adversary node or a non-adversary node being elected once the algorithm has completed. In this figure the states s_c and s_n represent the selection of a malicious node as cluster head and selection of a legitimate node as cluster head respectively. State s_u represents any intermediate iterations where the algorithm fails to select a cluster head. Further, s_u is also the initial state of the algorithm. We denote the steady state probability that a legitimate node is selected as cluster head as $p_{rand,n'}$ and the steady state probability that an adversary node is selected as cluster head as $p_{rand,c'}$.

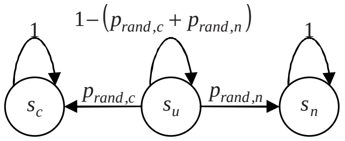

Fig. 6. Markov chain representation of random selection of cluster heads

In an argument similar to what was used to formulate p, the probability of a legitimate node being subscribed to at least one cluster containing adversary nodes is:

$$p' = 1 - \left(1 - \frac{\pi z^2}{L^2}\right)^{\frac{C}{c}} \tag{10}$$

The product of $p_{rand,c}$ and p' gives the probability that a legitimate node is subscribed to an adversary cluster head. Using this product, the network's average power consumption is:

$$P_{avg,rv} = p' p_{rand,c} P_{atk} + \left(1 - p' p_{rand,c}\right) P_{nrm} \tag{11}$$

Without any defense mechanism in place, a malicious node is selected as cluster head with a probability of 1. In Figure 7 we illustrate p' as a function of the number of compromised nodes in the cluster (c) and the total number of nodes in the cluster ($n+c$). This graph shows that given a single adversary node in a cluster, the random vote scheme nearly halves the probability that the adversary node will be selected as cluster head. However, this probability jumps to nearly 1 when multiple adversary nodes are located in the same cluster. It may startle the reader to see that when $n+c$ is even the adversary appears to have a better chance of becoming a cluster head. This is a direct result of the difference between attaining a majority for odd and even numbers.

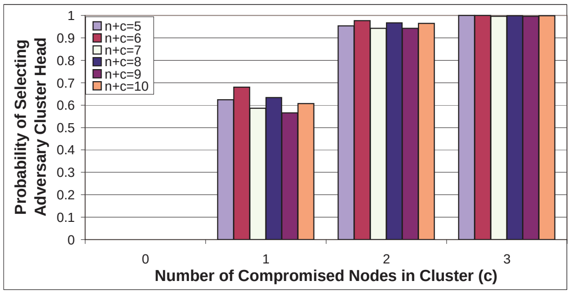

Fig. 7. Probability that the random vote scheme selects an adversary cluster head

In order for this algorithm to complete, any of the $n+c$ nodes in the local cluster must attain a majority of the votes, otherwise the nodes will keep voting. We can model this phenomenon using expected value as follows:

$$I = \sum_{x=1}^{\infty} x(p_{rand,n} + p_{rand,c})(1 - p_{rand,n} - p_{rand,c})^{x-1} = \frac{1}{p_{rand,n} + p_{rand,c}} \quad (12)$$

In Figure 8 we utilized equation I to investigate the amount of time the random vote algorithm requires to complete when we vary the parameters n and c. If a vote message is represented in 30 bytes, it would take about 24 ms to transmit a single vote. Since each round requires $n + c$ votes, this graph indicates that the random vote algorithm requires an acceptable amount of time to complete for moderate sized clusters (i.e. $n < 7$). However, for clusters where there are more than 7 nodes, the algorithm incurs too much latency. For example, when there are 10 nodes in the cluster the algorithm would require an average of 166 s to complete.

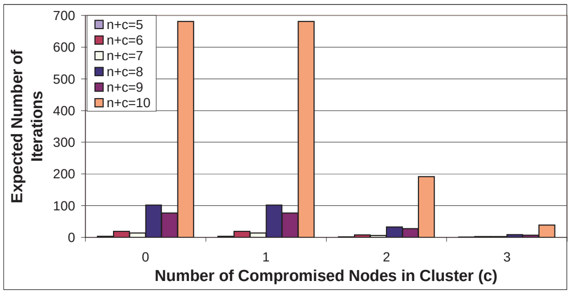

Fig. 8. Time required for random vote algorithm to complete

B. Round Robin Cluster Head Selection

The lack of scalability in the random vote clustering algorithm of Section VI-A caused us to consider another approach to secure cluster formation. The *round robin* scheme is based on the observation that if each node maintained more state, i.e. if clusters were maintained for long periods of time, a more scalable solution would be possible. With such a scheme cluster heads could be elected in a round robin fashion.

The round robin scheme operates in two phases. The first phase is a bootstrapping phase where the initial clusters are formed. The second phase is a maintenance phase, during which the precise membership of each cluster is updated due to node mobility, addition of new nodes to the network, and removal of nodes from the network.

1) Analytical Model: Assuming an average of c adversary nodes and n legitimate nodes in each cluster, the proportion of time that the c adversary nodes can launch a sleep deprivation attack from a particular cluster is:

$$p_{c,rr} = \frac{c}{c + n} \quad (13)$$

In Figure 9 we have used the equation for $p_{c,rr}$ to see how likely the round robin scheme is to elect an adversary cluster head. Comparison of Figures 9 and 7 illustrates that the round robin scheme makes it much more difficult for the adversary to become cluster head. In fact the adversary nodes are just as likely to become cluster head as is any other node.

A nice property of the round robin scheme is that it only requires a single iteration to select a cluster head. Each node must keep track of exactly which nodes are in the cluster and which node is the current cluster head. Thus, when a new cluster head is needed each node can independently determine who the new cluster head should be. However, this scheme requires that each sensor node must maintain a list indicating which nodes are in its cluster at all times. Such a list would require an unrealistic amount of per-node storage for larger clusters.

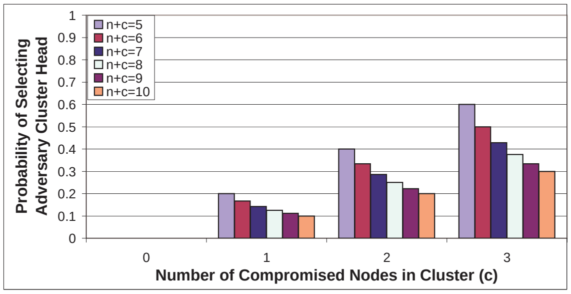

Fig. 9. Probability that round robin scheme selects an adversary cluster head

C. Hash-Based Cluster Head Selection

The lack of scalability that daunts the random vote scheme in Section VI-A and the excessive overhead inherent to the round robin scheme in Section VI-B motivated us to come up with the *hash-based cluster head selection* scheme, which we shall call the *hash-based* scheme for brevity. This scheme performs dynamic clustering in an attack and fault tolerant manner without excessive overhead.

1) Hashed Clustering Overview: In this scheme each node generates a random number and then broadcasts their number's hash. This allows each node to *commit* to their particular number without *revealing* it until all participating nodes have likewise committed. This process ensures that a cluster head is selected at random, so long as there is at least one "honest" node participating in the algorithm.

To prevent a malicious node from claiming to be multiple nodes, we assume that every message has been authenticated using an authenticated broadcast scheme such as μTESLA [38].

The hash algorithm operates by having each participating node execute the following steps:

1) Generate an integer, r_i using a pseudorandom number generator and locally broadcasts a *commit* message of the form $\langle ID, H(ID, r_i)\rangle$, where ID denotes the node's identifier and $H(\cdot)$ denotes a fixed length collision-resistant hash function.
2) Wait for enough time to pass, T_o, that it is sufficiently unlikely that any more *commit* messages will be received. Then locally broadcast a *list* message $\langle (ID_1, ..., ID_Y)\rangle$, where $(ID_1, ..., ID_Y)$ is a list of all IDs extracted from step 1, including the node's own ID as well.
3) For each *list* message received, verify that the local list of IDs is the same as the received list of IDs. Send a *request commit* message to any node whose ID is listed in another node's *list* message, but is not listed in the node's own list of IDs: $\langle ID_{dest}\rangle$
4) For each *request commit* message received whose ID_{dest} field matches the node's own ID, reply by repeating the original *commit* message.
5) Wait for T_o seconds to pass. Then locally broadcast a *reveal* message, to disclose r_i: $\langle ID, r_i\rangle$.
6) Verify each r_i with its associated *commit* message.
7) Wait for T_o seconds to pass. If any *reveal* messages have not been received, transmit a *request reveal* message: $\langle H(ID, r_i)\rangle$.
8) Any node that has a verified r_i can respond to a *request reveal* message.
9) Given Y participating nodes, the cluster head will be the node whose ID matches the following result:

$$Mod\left(\left(\sum_{i=1}^{n} r_i\right), Y\right) + 1.$$

The key observation regarding the security of this scheme is that every participating node must send out its *commit* message prior to receiving any *reveal* messages. This can be ensured with a high likelihood by selecting a sufficiently high value for T_o. Alternatively, this security could be guaranteed by building a communication scheme that is amenable to sensor networks and provides end-to-end reliability by utilizing current research efforts [1], [40].

2) Likelihood of an Adversary Cluster Head: In this section we analyze the likelihood of an adversary node being elected cluster head in the hash-based scheme.

Notice that the output of the hash-based scheme (i.e. step 9) will be a random integer so long as at least one of the participating nodes is not an adversary node. This is because a random number added to a nonrandom number yields a random number. Thus the malicious nodes will not affect the random output of the hash-based scheme by colluding. As the following equation indicates, the probability that a adversary node is selected to become a cluster head is the same as any node in the cluster:

$$p_{c,h} = \frac{c}{c+n} \tag{14}$$

Notice that the probability that an adversary becomes cluster head in the hash-based scheme is the same as in the round robin scheme (i.e. $p_{c,h} = p_{c,rr}$). Thus Figure 9 also applies to the hashed-based scheme.

3) Algorithm Latency: In this section we derive a timing model to show that the hash scheme is capable of forming clusters in a reasonable amount of time. Consider the hashing scheme within the context of a simple communication model, where R is the rate at which data can be transmitted and the total number of nodes which are participating in the algorithm is $n + c$. In the hash algorithm, each of the $n + c$ nodes must generate a *commit*, *list* and a *reveal* message. The total amount of time spent transmitting $n + c$ packets, each of which contains M_{size} bits, is:

$$T_{sndV} = (n+c) \cdot \frac{M_{size}}{R} \tag{15}$$

The probability that a given message is not received by at least one of the $n+c-1$ other nodes is p_e. In this case the node will have to retransmit the message. The average number of times a given message is transmitted until it is correctly received by all interested nodes is approximated as:

$$N_{rsnd1} = 1 + \sum_{i=1}^{\infty} i{p_e}^{i} = 1 + \frac{p_e}{(p_e - 1)^2} \tag{16}$$

Each of the $n + c$ nodes generates a single *commit* message and a single *reveal* message. To account for reliable transmission, each of these messages will be transmitted an average of N_{rsnd1} times, giving the total number of *commit* messages and *reveal* messages as:

$$N_{rComAll} = N_{rRevAll} = (n+c) \cdot N_{rsnd1} \tag{17}$$

There is sufficient redundancy in having each node generate a *list* message that we assume each unreceived *commit* and *reveal* message will be detected. Each node transmits its *list* message only once. Thus $N_{ListALL}$, the total number of *list* messages generated, is $n + c$.

Each *commit* and *reveal* causes a *request* message with probability p_e. Thus the average number of *request commit* messages and *request reveal* messages is:

$$N_{rRRAll} = p_e \cdot N_{rRevAll} = N_{rRCALL} = p_e \cdot N_{rComAll} \tag{18}$$

The sum of all *commit*, *reveal*, *list*, *request commit*, and *request reveal* messages gives $N_{rsndALL}$, the total number of packets transmitted by $n + c$ nodes on average. Given a transmission rate of R, the total time spent by $n + c$ nodes communicating a total of $N_{rsndAll}$ packets is:

$$T_{rsndAll} = \frac{N_{rsndAll} \cdot M_{size}}{R} \tag{19}$$

We estimate T_o with the amount of time required to reliably transmit $n + c$ messages:

$$T_o \approx \frac{(n+c) \cdot N_{rsnd1} \cdot M_{size}}{R} \tag{20}$$

While not included in our model, the actual value for T_o also depend upon the expected size of a cluster, which is dependent upon sensor range and the density of nodes in the network. Further the time spent waiting in step 2 would be slightly less than it would be in steps 5 and 7 as there are not retransmissions occurring during step 2. Since there are three points in which the nodes wait for a time period of T_o and we have the total time spend communicating, the total time spent by the algorithm is:

$$T_{total} = 3 \cdot T_o + T_{rsndAll} \tag{21}$$

In Figure 10 we used the equation for T_{total} to illustrate the time that could be expected for the hash-based algorithm to execute. Each grouping of six bars corresponds to a different choice of p_e and M_{size}. In order from left to right, we have used $(p_e = .1, M_{size} = 400)$, $(p_e = .01, M_{size} = 400)$, $(p_e = .1, M_{size} = 300)$, $(p_e = .01, M_{size} = 300)$. Within each grouping we varied the number of nodes within the cluster (i.e. $n + c$) from 5 to 10. We do not consider which nodes amongst these $n + c$ nodes are adversary nodes, as doing so does not change the amount of time the hash-based algorithm requires to complete. The most important feature that this figure indicates is that the amount of time required to select a cluster head in the hash-based scheme scales linearly with the number of nodes in the cluster. Thus this approach is scalable. We also note that even with a very high error probability (i.e. $p_e = .1$) our scheme completes in a reasonable amount of time.

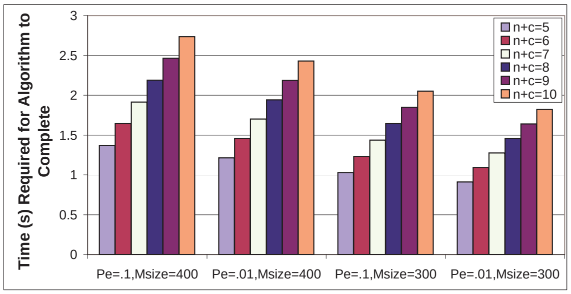

Fig. 10. Time required for hash-based scheme to select a cluster head

4) Determining Energy Consumption: To determine the energy consumed by the hash scheme we perform an average case analysis upon a single node. Note that we assume that communication energy dominates computational energy.

Given that a sensor node on average transmits a total of $N_{rsndAll}/(n+c)$ messages, the total energy that a node expends transmitting is:

$$E_{trans} = P_{snd} \cdot \frac{T_{rsndAll}}{n+c} \tag{22}$$

We assume that a node not sending a message is actively listening to the network: and the amount of energy spent receiving is:

$$E_{rcv} = (T_{total} - T_{rsndAll}) \cdot P_{rcv} \tag{23}$$

Thus the total energy spent by a single node in the hash algorithm is:

$$E_{total} = E_{rcv} + E_{trans} \tag{24}$$

In Figure 11 we have utilized equation E_{total} to determine the energy required for a node participating in the cluster head selection algorithm. This graph uses the same data points as were used in Figure 10, except the y-axis of this graph refers to the total energy expended by a single node participation in cluster head selection. We can see from this graph that the hash-based scheme does not require excessive energy consumption on behalf of participating nodes, and thus is amenable for use in sensor networks.

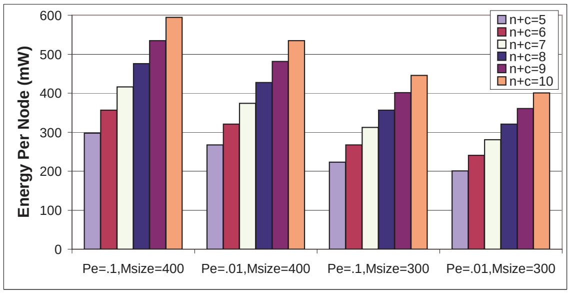

Fig. 11. Per-node energy required for hash-based scheme to select a cluster head

VII. Conclusions

In this paper we have shown a novel attack that an adversary can exploit upon sensor network applications which utilize distributed clustering. This attack, called the sleep deprivation attack, exploits the fact that conventional clustering algorithms rely upon the honesty of participating nodes. Thus a malicious node can ensure its selection as cluster head, and consequently it can launch a sleep deprivation attack against the network. In this attack a malicious cluster head sends victim nodes seemingly legitimate messages; however, the purpose of these messages is to keep its victims out of their low power sleep mode. The end result of this attack is greatly reduced node lifetime, potentially partitioning the network into disjoint pieces.

We also explain why an adversary would utilize a sleep deprivation attack, by comparing it to a more aggressive attack where the victim nodes are barraged with requests. We have found that the sleep deprivation attack is attractive (from the perspective of an attacker) due to its low cost in terms of energy and communication. Further, this attack is not readily detected.

Given the dire consequences of this attack, we have proposed three schemes by which its impact can be reduced: *1)* the random vote scheme, *2)* the round robin scheme, and *3)* the hashed-based scheme. We have found that the hashed-based scheme is superior to the other two methods in terms of resilience towards attack and required overhead.

References

[1] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan, “Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks,” in *MobiCom*, 2001, pp. 272–287.

[2] D. W. Carman, P. S. Kruus, and B. J. Matt, “Constraints and approaches for distributed sensor network security,” NAI Labs #00-010, Tech. Rep., 2000.

[3] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh, “Energy-efficient surveillance system using wireless sensor networks,” in *MobiSys*, 2004, pp. 270–283.

[4] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, “Energy-aware wireless microsensor networks,” *IEEE Sig. Proc. Magazine*, vol. 19, no. 2, 2002.

[5] A. Boukerche, X. Cheng, and J. Linus, “Energy-aware data-centric routing in microsensor networks,” in *MSWIM*, 2003, pp. 42–49.

[6] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, “Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks,” *Wireless Networks*, vol. 8, no. 5, pp. 481–494, 2002.

[7] A. D. Wood and J. A. Stankovic, “Denial of service in sensor networks,” *Computer*, vol. 35, no. 10, pp. 54–62, 2002.

[8] H. Chan and A. Perrig, “Security and privacy in sensor networks,” *IEEE Computer Magazine*, pp. 103–105, 2003.

[9] F. Stajano, *Security for Ubiquitous Computing*. John Wiley & Sons, Ltd., 2002.

[10] F. Stajano and R. Anderson, “The resurrecting duckling: security issues in ad-hoc wireless networks,” in *Proc. of the Third AT&T software symposium*, 1999.

[11] J. Krishnaswami, “Denial-of-service attacks on battery-powered mobile computers,” Master’s thesis, Virginia Polytechnic Institute and State University, 2003.

[12] R. R. Brooks and S. S. Iyengar, *Multi-Sensor Fusion: Fundamentals and Applications with Software*. Prentice Hall, 1998.

[13] H. Chan and A. Perrig, “ACE: An emergent algorithm for highly uniform cluster formation,” in *EWSN*, 2004.

[14] S. Bandyopadhyay and E. J. Coyle, “An energy-efficient hierarchical clustering algorithm for wireless sensor networks,” in *INFOCOM*, vol. 3, 2003, pp. 1713–1723.

[15] E. J. Duarte-Melo and M. Liu, “Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks,” in *Globecom*, 2002, pp. 21–25.

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

[17] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu, “Discrete mobile centers,” in *Computational Geometry*, 2001, pp. 188–196.

[18] S. Basagni, “Distributed clustering for ad hoc networks,” in *ISPAN*, 1999, pp. 310–315.

[19] M. Gerla and J. T.-C. Tsai, “Multicluster, mobile, multimedia radio network,” *Wireless Networks*, vol. 1, no. 3, pp. 255–265, 1995.

[20] C. Chiang, H. Wu, W. Liu, and M. Gerla, “Routing in clustered multihop, mobile wireless networks with fading chanel,” in *SICON*, 1997, pp. 197–211.

[21] S. Pattem, S. Poduri, and B. Krishnamachari, “Energy-quality tradeoffs for target tracking in wireless sensor networks,” in *IPSN*, 2003, pp. 32–46.

[22] J. Shin, L. J. Guibas, and F. Zhao, “A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks,” in *IPSN*, 2003, pp. 223–238.

[23] F. Zhao, J. Shin, and J. Reich, “Information-driven dynamic sensor collaboration,” *IEEE Sig. Proc. Magazine*, vol. 19, no. 2, pp. 61–72, 2002.

[24] M. Chu, H. Haussecker, and F. Zhao, “Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks,” *The Int’t J. of High Performance Computing Applications*, vol. 16, no. 3, pp. 293–313, 2002.

[25] S. Phoha, N. Jacobson, D. Friedlander, and R. R. Brooks, “Sensor network based localization and target tracking through hybridization and dynamic space-time clustering,” in *Globecom*, 2003, pp. 1555–1567.

[26] W. Zhang and G. Cao, “DCTC: Dynamic convoy tree-based collaboration for target tracking in sensor networks,” *IEEE Trans. on Wireless Communication*, vol. 3, no. 5, pp. 1689–1701, 2004.

[27] ——, “Optimizing tree reconfiguration for mobile target tracking in sensor networks,” vol. 4, 2004, pp. 2434–2445.

[28] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, “Building efficient wireless sensor networks with low-level naming,” in *SOSP*, 2001, pp. 146–159.

[29] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed diffusion: a scalable and robust communication paradigm for sensor networks,” in *MobiCom*, 2000, pp. 56–67.

[30] A. Cardenas, S. Radosavac, and J. Baras, “Detection and prevention of mac layer misbehavior for ad hoc networks,” in *SASN*, 2004.

[31] P. Kyasanur and N. Vaidya, “Detection and handling of mac layer misbehavior in wireless networks,” in *DSN*, 2003.

[32] M. Raya, J. Hubaux, and I. Aad, “Domino: A system to detect greedy behavior in ieee 802.11 hotspots,” in *MobiSys*, 2004.

[33] C. Karlof and D. Wagner, “Secure routing in sensor networks: Attacks and countermeasures,” in *Proc. of the IEEE SNPA Workshop*, 2003.

[34] S. Capkun and J. Hubaux, “Secure positioning in sensor networks,” EPFL/IC/200444, Tech. Rep., 2004. [Online]. Available: http://www.terminodes.org/micsPublications.php

[35] L. Lazos and R. Pooverdran, “Serloc: Secure range-independent localization for wireless sensor networks,” in *WiSe*, 2004.

[36] D. Wagner, “Resilient aggregation in sensor networks,” in *SASN*, 2004.

[37] S. Zhu, S. Setia, and S. Jajodia, “LEAP: efficient security mechanisms for large-scale distributed sensor networks,” in *CCS*, 2003, pp. 62–72.

[38] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, “SPINS: security protocols for sensor networks,” *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[39] R. R. Brooks, P. Ramanathan, and A. M. Sayeed, “Distributed target classification and tracking in sensor networks,” in *Proceedings of the IEEE*, vol. 91, 2003, pp. 1163–1171.

[40] A. Woo, T. Tong, and D. Culler, “Taming the underlying challenges of reliable multihop routing in sensor networks,” in *SenSys*, 2003, pp. 14–27.