Ad Hoc Networks 9 (2011) 249–264

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Towards robustness and energy efficiency of cut detection in wireless sensor networks ☆

Myounggyu Won *, Stephen M. George, Radu Stoleru

Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77840, United States

ARTICLE INFO

Article history:
Received 8 May 2010
Received in revised form 23 June 2010
Accepted 29 June 2010
Available online 4 July 2010

Keywords:
Wireless sensor networks
Topology control
Network cut detection

ABSTRACT

Reliable, full network connectivity in wireless sensor networks (WSN) is difficult to maintain. Awareness of the state of network connectivity is similarly challenging. Harsh, unattended, low-security environments and resource-constrained nodes exacerbate these problems. An ability to detect connectivity disruptions, also known as cut detection, allows WSN to conserve power and memory while reducing network congestion. We propose ER-CD and LR-CD, protocols that detect cuts while providing energy-efficiency and robustness to attack. Using distributed, cluster-based algorithms, ER-CD recognizes and determines the scope of disrupted connectivity while examining available data for evidence of an attack. For more resource-constrained networks, LR-CD enhances security through the use of a robust outlier detection algorithm. Extensive simulations and a hardware implementation provide experimental validation across a range of network sizes and densities. Results indicate that energy-efficiency can be improved by an order of magnitude in denser networks while malicious nodes are detected at deviations of 1% from expected behavior.

1. Introduction

Wireless sensor networks (WSN), systems composed of numerous sensor nodes with small, low-power, inexpensive radios, have attracted a large amount of research leading to interesting and innovative applications in disaster response [2], military surveillance [3], and medical care [4], among others. However, difficult problems still exist. One of the most challenging problems in WSN is maintaining network connectivity to reliably communicate between peers or deliver data to a specified point, or sink, in an energy-efficient manner. Disrupted connectivity, known as a *cut*, can lead to skewed data, ill-informed decisions and even entire network outages. It can also lead to memory and power exhaustion in disconnected nodes and network congestion in disconnected segments. Such data loss and wasted resources can be avoided when nodes can independently determine if a cut exists.

Cut detection algorithms attempt to recognize and locate cuts. In [5–8], a subset of nodes are given the task of monitoring the connectivity of network. Of particular note is the work by Shrivastava et al. [8] that proposes deterministic and randomized algorithms to detect a linear cut using a set of specially designated entities called sentinel nodes. However, their algorithms are centralized and detect only a linear cut. The state-of-the-art cut detection algorithm, Distributed Source Separation Detection (DSSD) [9], offers a more flexible approach, reliably detecting arbitrarily-shaped cuts, and allowing individual nodes to perform cut detection autonomously by examining the convergence of a positive *state* scalar. However, DSSD suffers from a number of problems. First, the convergence of the state relies heavily on neighboring states. Thus, in a network with dynamically changing topology, convergence is hard to achieve due to the frequently changing neighbor

☆ A preliminary version of this article [1] was presented at the 2009 International Conference on Wireless Algorithms, Systems, and Applications (WASA), 2010. This work was supported in part by NSF Grant CNS-0923203.

* Corresponding author.
E-mail addresses: mgwon@cse.tamu.edu (M. Won), smgeorge@cse.tamu.edu (S.M. George), stoleru@cse.tamu.edu (R. Stoleru).

1570-8705/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2010.06.008

set. Second, DSSD fails to address security, a critical component of sensor deployments in unattended environments. The algorithm can erroneously converge when the network contains a malicious node that injects false state to influence the cut detection decision. Third, DSSD requires a lengthy, iterative convergence process. Since all nodes participate in frequent broadcasts required to achieve convergence, the algorithm is cost-prohibitive with regards to power, especially in denser networks.

In light of these problems, we propose two protocols, *Energy Efficient and Robust Cut Detection* (*ER-CD*) and *Lightweight and Robust Cut Detection Algorithm* (*LR-CD*).

ER-CD is an improved cut detection protocol that offers increased energy efficiency and robustness against masquerade or impersonation attacks. ER-CD divides the network into a grid of location-based clusters. Cluster leaders form a Virtual Grid Network. The cut detection algorithm runs on this high-level network that is significantly less affected by topological changes. As the algorithm executes, the states of leaders converge to some positive value if there is no cut in the network. The speed of convergence is faster thanks to the grid topology of the high level network, since the degree of a leader is typically at most 4. Furthermore, the simple grid topology enables leaders to maintain the global topological information in their adjacency matrices with a small computational cost. By exploiting the adjacency matrix, a leader can exactly compute the next states of its neighbors. Finding any inconsistency above a certain threshold in the states of adjacent leaders potentially indicates a masquerade or impersonation attack. In this attack, a malicious node injects erroneous state into the cut detection process.

LR-CD is a cut detection protocol designed for resource-constrained situations where ER-CD is too heavy. Built on top of the DSSD algorithm, LR-CD incorporates outlier detection, a statistical data analysis technique, to detect a masquerade or impersonation attack. Outlier detection enables the identification of statistically improbable data, a possible indicator of malicious activities, and provides a light-weight mechanism to validate neighbor data.

The contributions of this article are:

- A protocol, ER-CD, that provides reliable cut detection with fast convergence, good energy-efficiency, and robustness against a particular security threat, the masquerade attack.
- A lightweight protocol, LR-CD, that enhances the state of the art cut detection algorithm with robustness against the masquerade attack at low computation and memory cost.
- Extensive simulations that verify and validate performance of the protocols across a variety of network sizes and densities.
- An implementation on Epic wireless sensor motes [10] extending and confirming the simulation results.

This article is organized as follows. Section 2 discusses related work and is followed by the system model and problem formulation in Section 3. Section 4 discusses ER-CD, the proposed algorithm for improving the energy efficiency and robustness against the masquerade attack. LR-CD, a lightweight protocol focused on providing robustness, follows in Section 5. Implementation is addressed in Section 6 which is followed by experimental results in Section 7. Conclusions are presented in Section 8.

2. Related work

The challenges of the network partition monitoring problem have been emphasized in many papers [11–13]. Chong et al. [12] mentions the problem from a security perspective arguing that nodes deployed in a hostile environment must be able to detect tampering. In [13], Cerpa and Estrin stress the importance of the network cut detection problem in their self-configuring topology scheme but left it as a future work. Significant open questions remain in this area.

An early paper by Kleinberg et al. [5] considers the cut detection problem in a wired network. The authors define the (ϵ,k)-cut to be a network separation into $(1-\epsilon)n$ nodes and ϵn nodes when k independent edges are disabled. To detect the (ϵ,k)-cut, they place a set of *agents D* to monitor the connectivity of the network. Each agent periodically communicates with all other agents. Failed connections beyond some threshold are presumed to indicate the presence of a cut. The main result is that the size of the set D must be $O\left(k^3\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\frac{1}{\epsilon}\log\frac{1}{\delta}\right)$ to successfully detect any kind of (ϵ,k)-cut with probability $1-\delta$. However, in wireless sensor networks, due to their geometric structure, linear or other geometric shaped cuts are more likely than independent k disabled edges. Additionally, the number of agents required for this type of cut detection is very large.

Recently, Shrivastava et al. [8] proposed deterministic and randomized algorithms to detect network separation using a set of sentinel nodes to monitor for linear cuts in a network. The work is, in large part, based on [5]. Specifically, the authors defined the ϵ-cut where at least ϵ fraction of nodes are disconnected by the cut. However, Shrivastava minimized the number of required sentinels by reducing the problem to the linear cut, which is a more natural phenomenon for wireless sensor networks than independent k edge failures, and proved that there exist $O(\frac{1}{\epsilon})$ sentinels for any ϵ-cut with $\epsilon<1$. This is a relatively small number of sentinels when compared with the result of [5]. The authors proposed a deterministic algorithm to find the minimum number of sentinels and introduced a fast randomized algorithm to compute the sentinels of size $O(\frac{1}{\epsilon})$. However, Shrivastava's algorithm is limited to detecting linear cuts and fails to detect arbitrarily shaped cuts. Also, it is a centralized algorithm where information about a cut is only known to the base station.

In Ritter et al. [7], the authors select a source node and make it broadcast an *alive* message throughout the network. Border nodes detect a cut if they miss the *alive* message from the source node more than a given number of times.

The most recent cut detection algorithm is proposed by Barooah et al. [9] and overcomes several problems associated with previous solutions. Barooh's algorithm, DSSD, can not only detect an arbitrarily-shaped cut, but also

enables every node in the network to autonomously detect a cut in a distributed manner. Each node maintains a positive scalar called *state* and updates this scalar iteratively based on received states from one-hop neighbors. After a number of iterations, if a node is connected to the source that updates *state* for the network, its *state* converges to some positive value. Otherwise, if there exists a cut between the node and the source, its *state* rapidly decays to 0. Thus, a node can independently determine its connectivity status by monitoring its state.

The decision on cut detection is made based on available data sets, e.g., *state* is updated based on received states, and convergence is determined based on the history of *state* changes. Thus, ensuring the validity of data sets is an important process for accurate cut detection. Outlier detection is a statistical analysis tool often used to identify problems in data sets like measurement error or abnormal data distribution.

Outlier detection can be categorized into two main streams: a parametric approach, which assumes a priori known distribution of the data, and a non-parametric approach that does not rely on a specific distribution. With known data distribution, the parametric approach detects outliers with very high precision. However, in many cases, finding a matching distribution is very hard. Probabilistic models that infer distribution based on sample data compensate for this difficulty but often show high false positive rates [14]. Non-parametric approaches using distance-based and density-based methods attempt to overcome this limitation. Knorr and Ng [15] proposed the first distance-based algorithm, where a point is regarded as an outlier if its distance to a kth nearest neighbor point is greater than a certain threshold. One disadvantage is that the threshold must be defined. Ramaswamy et al. [16] studied distance-based detection, where a point is said to be an outlier if the distance to kth nearest neighbor is greater than that of $n-1$ other points. Recently, Zhang et al. [17] introduced an algorithm for finding an outlier based on the sum of distances to the point's k nearest neighbors. However, all distance-based solutions fail to detect outliers in clustered data. Density-based outlier detection schemes [18,19] gracefully solve this problem. Each data point is given a score called Local Outlier Factor (LOF) based on its local density, which is bounded by a specific value MinPts. In [18], an outlier is determined by score. In [19], the bounding value MinPts is determined autonomously using statistical values such as inter-cluster distances and cluster diameters.

In [1], we proposed algorithms to enhance the robustness and energy efficiency of DSSD, the existing state-of-the-art cut detection algorithm, and provided extensive simulation results validating the ideas. However, this article provides improved versions of the algorithms consolidated into protocols that do not rely on the assumption of uniform node distribution, cope well with the problem of local cut detection, and have higher precision in detecting malicious behavior. New experimental results obtained from a real-world setting provide a validation beyond the simplified radio and interference model from the simulator while new simulation results verify the improved versions of the algorithms.

3. Models and problem formulation

In this section we present the system and attacker models and explain the damaging impact of the attacker on a normal operation of a wireless sensor network system.

3.1. System model

The network is represented as a static, undirected, connected graph $G = (V,E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of n nodes in the network, and $E = \{(v_i, v_j) | v_i, v_j \in V\}$ represents radio connectivity between a pair of nodes v_i and v_j. Let $N_i = \{v_j | (v_i, v_j) \in E\}$ be the set of *neighbors* of node v_i, and let $|N_i|$ be the *degree* of node v_i.

The network is synchronous and time is modeled in discrete rounds $k = 0, 1, 2, \ldots$. Similar to [9], each node v_i maintains a positive real value $x_i(k)$. This value, called the *state* of node v_i at time k, is initially zero for each node, i.e., $x_i(0) = 0, 1 \leqslant i \leqslant n$. One arbitrarily selected node in the network is designated as the *source node*. For simplicity in notation, v_1 is designated to be the *source node*.[1]

At every round k, each node v_i updates its state $x_i(k)$ and sends the new state $x_i(k+1)$ to its neighbors. The states of nodes except the *source node* are updated using the following equation:

$$x_i(k+1) = \frac{1}{|N_i|+1} \sum_{v_j \in N_i(k)} x_j(k). \tag{1}$$

The *source node* v_1 uses a slightly different state update equation:

$$x_1(k+1) = \frac{1}{|N_1|+1} \left(\sum_{v_j \in N_1(k)} x_j(k) + s \right), \tag{2}$$

where s, called the *source strength*, is a user specified scalar. The state of each node converges after a number of iterations of the state update process [9].

A *cut* is defined as a network partition, in which the connected graph G is separated into n disjoint connected components $G_{src}, G_2, \ldots, G_n$, where $G_{src} = (V_{src}, E_{src})$ is a graph which contains the *source node*. When a *cut* occurs, the state of each node $v \notin V_{source}$ converges to 0 [9].

The convergence of a node's state is illustrated in Fig. 1. For the network depicted, around iteration 40, the scalar states of nodes in the network converge. Shortly after iteration 60, a cut occurs in the network when the four nodes in the middle fail. After this, the states of nodes in the segment disconnected from the source rapidly decay to 0 while the states of nodes still connected to the source converge to a new higher value. A critical observation is that the states of all nodes converge to new values after the cut. Thus, generally, nodes have the ability to autonomously detect a cut by monitoring their own states.

[1] Source node selection is highly dependent on network topology and purpose. Networks designed to aggregate data might reasonably place the source node at or near the sink. Surveillance networks might place the source near the center of the network. Other networks might choose different, more relevant placement criteria.

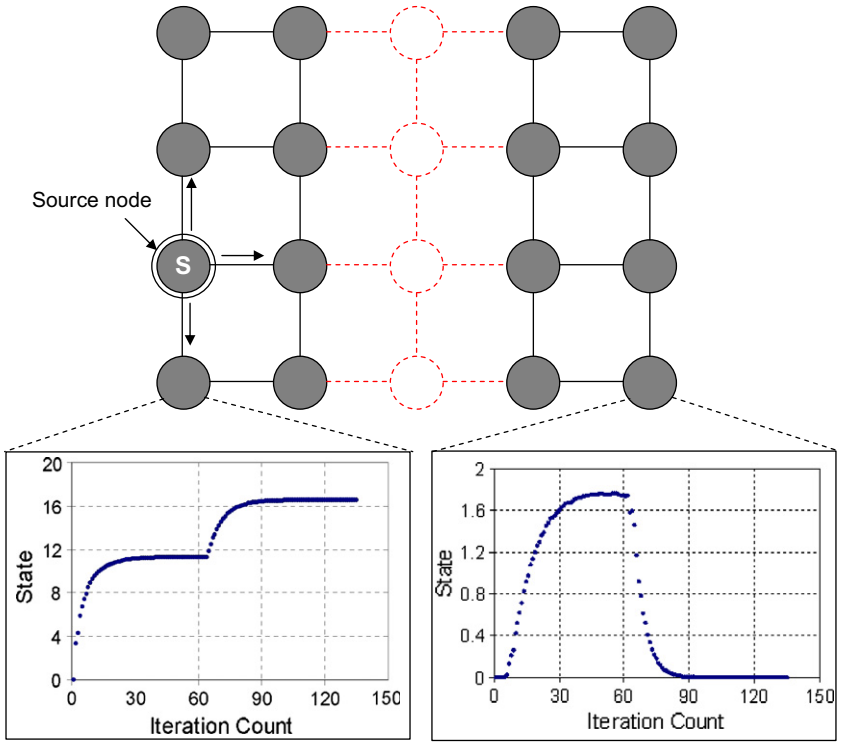


Fig. 1. A cut occurs in a connected network of 20 nodes. The graphs depict the scalar states of two nodes, one in G_{source} and one $\notin G_{source}$.

3.2. Attacker model

One notable vulnerability of cut detection using this distributed algorithm is that it is susceptible to a masquerade or impersonation attack where a malicious node located in the disconnected part of the network imitates behavior of the source node. This can affect the state value that other nodes compute.

Temporary deviations from state convergence, often caused by packet loss or small changes in neighbor set, can be tolerated by the system implementation. However, this is not true when a non-source node continuously injects a false state into the system in an attempt to impersonate a source node. This node may be defined as follows:

Definition 3.1. A node $v_m \in V$, $m \neq 1$ is a *malicious* node if it acts as a source node in the network, i.e., it updates its state with an arbitrary strength s', as given by:

$$x_m(k+1) = \frac{1}{|N_m|+1}\left(\sum_{v_j \in N_m(k)} x_j(k) + s'\right). \tag{3}$$

3.3. Defense against malicious nodes

Given that the purpose of cut detection is to avoid data loss and conserve resources, the impact of a malicious node can be severe. Algorithms designed to detect and protect from this attack are an important component of a cut detection protocol.

The following theorem shows the damaging impact of the malicious source node in the source-disconnected segment of the network:

Theorem 3.1. *If there exists a malicious node v_m, $(m \neq 1)$ in the network segment disconnected from the source node, the nodes in that region cannot detect a cut using only the state update Eq. (1).*

Proof. Eqs. (1) and (2) can be rewritten together in a matrix representation, as follows:

$$X(k+1) = (D+I)^{-1}(A \cdot X(k) + se_1), \tag{4}$$

where D is the diagonal matrix of node degrees and A is the adjacency matrix of G. Note that Eq. (4) is an iteration based on the Jacobi method to solve:

$$LX = se_1, \tag{5}$$

where $L := D - A + I$.

Now assume that a cut partitions network G into n connected components, $G_1, G_2, \ldots, G_n$, and let G_1 be the connected component having the source node, i.e, $G_1 = G_{src}$. Assume that there is a malicious source v_m with strength s' in a partitioned network G_j, where $2 \leqslant j \leqslant n$. Note that the matrices A, D, and I are also partitioned into $A = A_1 + A_2 + \ldots + A_n$, $D = D_1 + D_2 + \cdots + D_n$, and $I = I_1 + I_2 + \cdots + I_n$, where each A_i, D_i, and I_i are the adjacency matrix, diagonal matrix of node degrees, and identity matrix of the connected component G_i, respectively, where $1 \leqslant i \leqslant n$. Then,

$$L := \sum_{i=1}^{n} D_i - \sum_{i=1}^{n} A_i + \sum_{i=1}^{n} I_i$$
$$= \sum_{i=1}^{n} (D_i - A_i + I_i)$$
$$= L_1 + L_2 + \cdots + L_n,$$

which gives $(L_1 + L_2 + \cdots + L_n)X = se_1 + s'e_m$. Thus, the disconnected part of the network, which has a malicious node, is actually another distinct system with the source node v_m such that:

$$L_j X = s' e_m. \quad (6)$$

Note that $L_j := D_j - A_j + I_j$ is the Dirichlet Laplacians, and is invertible if its graph is connected [20]. Therefore, there exists a unique solution X where each node in the disconnected region G_j will converge to some positive value. □

The same logic applies to disconnected network segments with more than one malicious node as follows:

Corollary 3.2. *If there exists more than one malicious node in the source-disconnected region of the network, nodes in that region cannot detect a cut.*

Proof. Assume that there are k malicious nodes, $V_m = \{v_i : 2 \leqslant i \leqslant n\}$, $|V_m| = k$, with corresponding source strength $s'_1, s'_2, \ldots, s'_k$ in the disconnected region G_j of the network. Therefore, from Eq. (6),

$$L_j X = \sum_{v_i \in V_m} s'_i e_i. \quad (7)$$

So, a unique solution of X still exists. □

4. ER-CD: energy-efficient and robust cut detection

The current, state of the art cut detection algorithm [9] suffers from relatively high energy consumption, especially in a dense wireless sensor networks. This is largely due to the cost of communications because every node must broadcast at each iteration of the algorithm. In large or dense networks, it may take many iterations to reach convergence. Likewise, the time to convergence may be lengthy. This means that notification of significant changes may take a long time to propagate and may end up being significantly diluted at distant network extremes. In light of these problems, *Energy-efficient and Robust Cut Detection* (ER-CD) proposes to execute a robust cut detection algorithm on a subset of nodes. The predictable topology of this subset is leveraged to provide a malicious node detection scheme and probing routes are used to detect local cuts that might otherwise be missed.

In ER-CD, the main idea is to partition the network into a grid of clusters as depicted in Fig. 2. Each cluster elects a leader who executes the cut detection algorithm by exchanging state values with leaders in adjacent clusters. Clustering provides significant benefits. Energy efficiency is improved by the small number of nodes that directly participate in the cut detection protocol. Also, the grid topology restricts the degree of the leaders in the clustered environment to normally no larger than 4, smaller than would be typical in a network where all nodes participate. Since the number of iterations required for convergence (convergence speed) is determined by the maximum degree of the network [9], convergence speed grows rapidly with increasing maximum degree of the network. Therefore, maintaining a small, constant degree by using a grid network allows the state to converge considerably faster. Fewer messages are sent, thereby increasing energy efficiency.

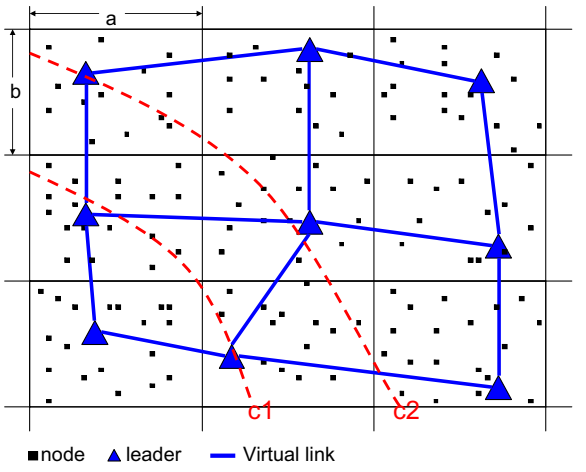


Fig. 2. The Virtual Grid Network formed by the leaders elected in their respective clusters, and the examples of network cuts, c1 and c2.

Despite the additional overhead caused by the introduction of leader election module, the ER-CD algorithm shows high energy savings in dense networks. The message complexity of a typical distributed leader election algorithm is known to be $O(n \log n)$ [21], where n is the total number of nodes. This cost is trivial compared with the total message overhead to reach convergence when all nodes participate, as in DSSD. Furthermore, since leader election is performed independently in each cluster that consists of a small subset of nodes, the message complexity for our leader election process is even smaller than $O(n \log n)$.

The algorithms implemented in the ER-CD protocol are presented in Algorithm 1 (executed by all nodes in the network), Algorithms 2 and 4 (executed by leader nodes) and Algorithm 3 (executed by non-leader nodes).

Further details of the ER-CD protocol are described below.

Algorithm 1. ER-CD: Initialization

```
1:  Initialization:
2:  determine a group id, G(i,j) s.t. x ∈ [i·a,(i + 1)·a], where 0 ⩽ i ⩽ ⌈A/a⌉, and y ∈ [j·b,(j + 1)·b], where 0 ⩽ j ⩽ ⌈B/b⌉.
3:  elect a leader L(i,j).
4:  if elected as a leader then
5:    run Algorithm 2 ER-CD: Leader.
6:  else
7:    run Algorithm 3 ER-CD: Non-leader.
8:  end if
```

4.1. Network initialization

The Network Initialization phase is described in Algorithm 1. The sensor network starts as a set of localized nodes distributed uniformly in a rectangular area of size $A \times B$. Nodes obtain their locations using existing node localization protocols [22]. We assume that location information is relatively precise. (Ref. [23] for the impact of localization accuracy on system performance.) The network is divided into a set of rectangular clusters of size $a \times b$ (where a and b are system parameters), as shown in Fig. 2. Based on location, a node becomes a member of a particular cluster (Algorithm 1, Line 2), e.g., a node located at (x,y) will become a member of cluster $G(i,j)$ if $x \in [i \cdot a,\ (i+1) \cdot a]$ and $y \in [j \cdot b, (j+1) \cdot b]$, where $0 \leqslant i \leqslant \lceil \frac{A}{a} \rceil$ and $0 \leqslant j \leqslant \lceil \frac{B}{b} \rceil$.

Next, in each cluster $G(i,j)$ a leader $L(i,j)$ is elected (Algorithm 1, Line 3) [24]. At the end of the leader election phase, all nodes in a cluster know the ID and location of their leader. Elected leaders in the network form a *Virtual Grid Network*, denoted by $G_{grid} = (V_{grid}, E_{grid})$, where V_{grid} is a set of all $L(i,j)$'s and E_{grid} is a set of all undirected virtual links connecting $L(i,j)$ and $L(i \pm 1,\ j \pm 1)$, where $0 \leqslant i - 1,\ i + 1 \leqslant \lceil \frac{A}{a} \rceil,\ 0 \leqslant j - 1,\ j + 1 \leqslant \lceil \frac{B}{b} \rceil$.

Algorithm 2. ER-CD: leader code

```
1:  determine a set of destinations for probing
    routes PR = {pr_1, pr_2, ..., pr_n}.
2:  run Algorithm 4 Update_Topology.
3:  On state-update-period expired:
4:  update the state of L(i,j) s.t. x_{i,j}(k+1) ←
    (x_{i+1,j}(k) + x_{i-1,j}(k) + x_{i,j+1}(k) + x_{i,j-1}(k)) / (|N_{i,j}| + 1).
5:  Dest ← pr_l, l is initially 1.
6:  if ++l = n then
7:    l ← 1.
8:  end if
9:  ω ← 1 // indicating a probing mode.
10: send the STATE_UPDATE message.
11: On receiving STATE_UPDATE message:
12: compute the expected state E_{i±1,j±1} by
    using Eq. (8), where X(k+1) =
    {E_{1,1}, E_{1,2}, ..., E_{MAX(i),MAX(j)}}.
13: if received state = E_{i±1,j±1} then
14:   x_{i±1,j±1} ← received state.
15: else
16:   report abnormal behavior.
17: end if
```

4.2. Robust cut detection in the Virtual Grid Network

Once the network initialization phase is complete, the robust cut detection algorithm begins to execute on G_{grid}, formed by the leader nodes. The pseudocode executed by a leader node is described in Algorithm 2.

A leader node first determines the best routing path it needs to use for communicating with adjacent leader nodes, if any. The pseudocode is depicted in Algorithm 2 (Lines 1–9). Due to the complexity of this routing logic, we defer its presentation until Sections 4.3 and 4.4.

Once a routing path is decided, each leader sends a STATE_UPDATE message containing the leader's current state value and its cluster coordinates to leaders of adjacent clusters (Algorithm 2, Line 10). Routing between leaders is handled with a variant of GPSR [25]. Since a leader, e.g., $L(i,j)$ does not know the ID or location of adjacent leaders, e.g., $L(i+1,j)$, it initially sends messages to a fictitious destination in the middle of cluster $G(i+1,j)$. When the messages reaches cluster $G(i+1,j)$, the first node that sees the STATE_UPDATE packet updates the destination with the correct location and ID of the cluster's leader.

Clustering enables the construction of a regular overlay on top of the true network topology. The resulting predictable subnetwork permits the execution of a strong, accurate malicious node detection scheme (Algorithm 2, Lines 11–17). Thanks to the simplified grid topology formed by leaders of each cluster, any leader easily computes expected convergence values for all other leaders in the network. This is possible because each leader can maintain an up-to-date global adjacency matrix and diagonal matrix of node degrees with low message overhead. More specifically, if the system parameters a and b, the cluster size, are known to leaders, they can easily construct the adjacency matrix A and the diagonal matrix D of node degrees. The state update Eqs. (1) and (2) then can be rewritten in matrix representation:

$$X(k+1) = (D+I)^{-1}(A \cdot X(k) + se_1). \tag{8}$$

Algorithm 3. ER-CD: Non-leader code

```
1:  On receiving STATE_UPDATE MSG:
2:  // if the message is in probing-mode,
3:  if ω = 1 then
4:    find the sub-cell id of Dest, sid_Dest.
5:    determine my sub-cell id sid_Mine.
6:    if sid_Dest ≠ sid_Mine then
7:      find the closest neighbor (x_1, y_1) to Dest.
8:      d ← min{distance((x_1, y_1), Src − Dest)}.
9:      if d < Th_d then
10:       forward the msg to (x_1, y_1).
11:     else
12:       drop the msg.
13:     end if
14:   else
15:     ω ← 0, Dest ← locs of adjacent leaders.
16:     relay STATE_UPDATE MSGs to adjacent
        leaders.
17:   end if
18: else
19:   relay the message using GPSR.
20: end if
```

Consequently, by simple matrix multiplication, each leader can exactly compute the next state of itself and all neighbors (note that since $D + I$ is a diagonal matrix, its inverse is simply a diagonal matrix containing the inverse

of each element of $D + I$). When there is a malicious node in the network, a leader can immediately and accurately detect it by checking if the state received from the malicious node, or from nodes affected by the malicious node, is different from the expected value.

4.3. Local cuts and their impact on ER-CD

The execution of the robust cut detection in the virtual grid is complicated by a scenario where a *local cut* occurs that does not include any leaders, e.g., cuts c1 and c3 in Fig. 3 (compare with the cut c2). Although periodic leader rotation can help address this problem, it does not solve it and ER-CD may fail to detect such cuts. In order to proactively detect a local cut, leaders adopt a two-phase routing protocol (as depicted in Fig. 3).

The first phase examines the cluster for any existence of a local cut. Motivated by Trajectory Based Forwarding (TBF) [26], a STATE_UPDATE packet is first sent along one of the *probing routes*, a predefined linear trajectory between a leader and a sub-cell near an edge of the cluster (Algorithm 2, Lines 1, 5–10). Upon receiving a probing packet, a relaying node (a non-leader) chooses the closest node to the destination as a next hop and forwards the packet only if the shortest distance between the chosen node and the linear trajectory is within the user specified threshold (Algorithm 3, Lines 4–10). If the distance is greater than the threshold the packet is dropped (Algorithm 3, Lines 11–13). This process is illustrated in Fig. 4. If a leader misses the packet more than k times, the adjacent leader will conclude that a local cut has occurred. The choice of the number of probing routes is a tradeoff. With a larger number of probing routes, more of the cluster is covered, increasing the accuracy of local cut detection. However, more paths with longer hop counts increase the overhead for sending a packet to adjacent leaders.

Once the packet has reached the subcell in the first phase, the second phase of the routing starts to execute.

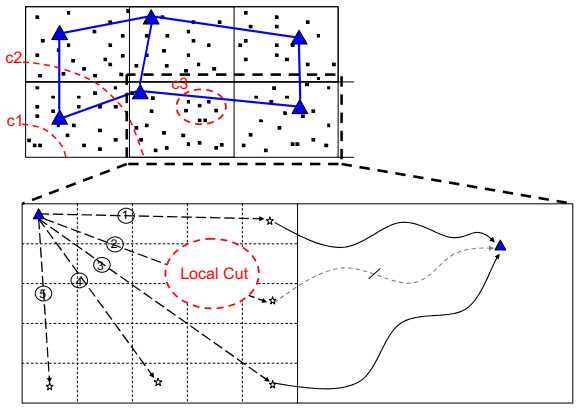


Fig. 3. In order to detect a local cut, leaders send state update messages along one of the *probing routes* (red dotted lines). At the probing destination, the packet is relayed to the adjacent leaders using the default routing protocol. Neighboring leaders detect a local cut when they miss a state update message for more than a certain number of rounds. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

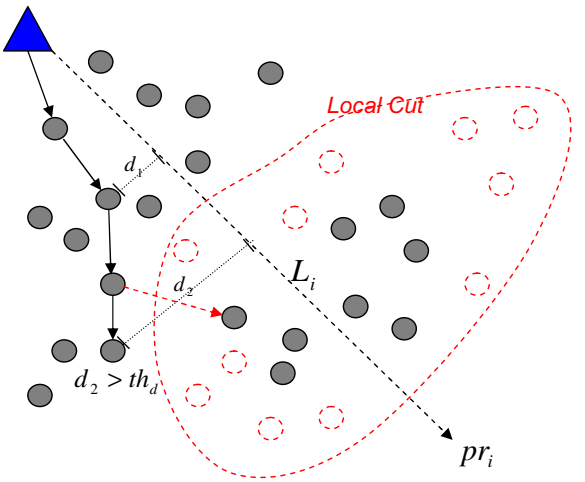


Fig. 4. A leader sends a packet along the predefined linear trajectory, L_i, between the leader and the location pr_i. A leader chooses the closest node to pr_i as the next hop and forwards the packet if the shortest distance between L_i and the selected node is smaller than the user specified threshold th_d.

In this phase, the first node in the subcell that sees the packet simply relays the packet to the leaders in the adjacent clusters using normal GPSR routing protocol (Algorithm 3, Lines 14–17). If there are no nodes in the subcell, that probing route is not used in the next iteration of the algorithm.

4.4. Partial grid topology

The system model's assumption of uniform distribution of nodes implies the formation of a complete rectangular grid topology of leaders. This allows leaders to maintain the adjacency matrix (A) of the Virtual Grid Network with little computational cost and provides an ability to use this matrix for computing the next states of their neighbors. However, such an assumption might be difficult in a real world deployment. For example, in many WSN applications, nodes are deployed randomly, e.g., from the aircraft, causing some portions of the target area to be unpopulated. Furthermore, obstacles in a target area, such as lakes, might prevent the uniform deployment of sensor nodes. As a result, some clusters might not have leaders and, therefore, only a partial grid is formed.

Consider two different partial grid cases, depicted in Fig. 5. Solid triangles represent clusters with leaders. In the first case, missing leaders due to non-uniform node distribution, represented as dotted triangles, can be detected by adjacent leaders and the adjacency matrix $A = [a_{mn}]$ for all leaders is updated. The second case is a cluster surrounded by empty clusters. In this case, the missing leader in the center, denoted with a question mark, has no adjacent clusters with leaders. Therefore, the network will never learn of this missing cluster and leader.

For the first case, where a missing leader is adjacent to leader, a simple distributed algorithm allows all leaders

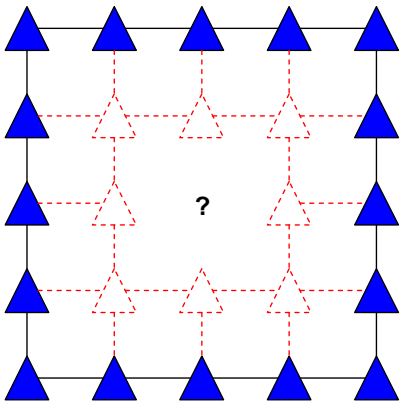


Fig. 5. Partial grid topology with some missing leaders represented as dotted triangles and quotation marks. Particularly, the missing leader '?' is not adjacent to any leaders.

update their topology information immediately with respect to missing leaders without causing additional message overhead. The pseudo code for it is presented in Algorithm 4.

Recall that $\mathscr{L} = \{L(i,j) | 0 \leqslant i \leqslant \lceil \frac{A}{a} \rceil, 0 \leqslant j \leqslant \lceil \frac{B}{b} \rceil\}$ is the set of leader ids in the complete grid topology. The topology is represented by an adjacency matrix $A = [a_{m,n}]$, where $m, n \in \mathscr{L}$ such that:

$$a_{m,n} = \begin{cases} 1, & \text{if } (m,n) \in E_{grid} \\ 0, & \text{otherwise} \end{cases}.$$

Algorithm 4. Update_Topology

```
Input: A = [a_{m,n}], L
On missing state msg #of probing routes + 1
    times from L(i,j):
On receiving M(i,j):
if L(i,j) ∈ L then
    L ← L \ {L(i,j)}.
    a_{L(i,j),n} ← 0, ∀n.
    a_{m,L(i,j)} ← 0, ∀m.
    // M(i,j) is piggybacked on the state msg.
    send M(i,j) to adjacent leaders.
else
    // already piggybacked the information of
    this missing node.
    stop propagating the msg.
end if
```

If a leader misses the STATE_UPDATE message (# of probing routes + 1) times from its adjacent leader, it regards the adjacent leader as missing. When a leader detects a missing leader, it updates its adjacency matrix and starts propagating the cluster coordinates (i,j) of the missing leader by piggybacking the information in the next STATE_UPDATE message. Upon reception of a STATE MESSAGE containing the information on the missing leader, other leaders, in turn, propagate the information to their adjacent neighbors. No duplicate messages are sent, since the leaders do not piggyback the information if it has already sent the message.

In ER-CD, this algorithm executes once, after the initialization phase (Algorithm 2, Line 2). The time complexity of the algorithm is $O(dist)$ iterations of state update process, where $dist$ is the distance of the grid topology. Since the algorithm piggybacks the message M on the state message, there is no additional message overhead.

The second case, that of a missing leader surrounded by clusters with missing leaders (the question mark at the center of Fig. 5), actually has no impact on ER-CD. The following theorem shows that such *isolated leaders* do not need to be taken into account when the adjacency matrix is updated, since they do not influence the convergence value of other leaders.

Theorem 4.1. *For any node v_i in $G = (V,E)$, only nodes that are connected to v_i will contribute to the convergence value of v_i.*

Proof. The claim is proven by using an induction on the number of hops k from the node v_i.

Basis step ($k = 1$): trivially holds by Eq. (1) inductive step ($k = n$, $n > 1$): assume that the convergence state of v_i is affected by the nodes in n hops away. If n is the maximum hop count, then done. Otherwise, the nodes in n hops away will use the states of their adjacent neighbors, which are $n + 1$ hops away from v_i. Thus, by inductive hypothesis, our claim holds. □

Algorithm 5. Compute distribution

```
Input: S_i = {s_{i1}, s_{i2}, ..., s_{in}}, p
Output: μ, σ
Ξ_i ← ∅
for each s_{ik}, 1 ⩽ k ⩽ n do
    pick p nearest state in S_i, {s^1_{ik}, s^2_{ik}, ..., s^p_{ik}}.
    compute d^j_{ik}, 1 ⩽ j ⩽ p s.t. d^j_{ik} = |s_{ik} − s^j_{ik}|.
    compute a corresponding new sample ξ_{ik} s.t.
    ξ_{ik} = Σ_{j=1}^{p} d^j_{ik}.
    Ξ_i ← Ξ_i ∪ ξ_{ik}.
end for
μ = (Σ_j ξ_{ij}) / |S_i|.
σ = sqrt( (Σ_j (ξ_{ij} − μ)^2) / (|S_i| − 1) ).
```

4.5. Discussion

In this section, we discuss several interesting issues with regard to the proposed protocol.

4.5.1. Reactions to the cut

Possible actions undertaken by the leaders and nodes when a cut is detected might vary depending on applications. One practical scenario is that the leader in the

disconnected area informs nodes in its cluster about the incidence of the cut. In the meantime, the leader in the connected segment notifies the source about the location of the cut for an immediate remedy. Nodes might take different reactions to a cut. They might simply defer transmission and temporarily save data in their local storage until connection is repaired, to avoid the loss of important data. A more proactive solution might be that a subset of nodes with strong storage capability are randomly deployed and act as temporary source nodes. Another possible scenario is that nodes located near the border of a cut might try using higher transmit power to increase communication range. Possible scenarios, however, are not limited to what we have discussed.

4.5.2. Mobility of nodes

The proposed algorithms, ER-CD and LR-CD, assume static network. Detecting a cut in the network with mobile nodes is a challenging problem, since convergence of the algorithm is significantly hampered by frequent topological changes. Furthermore, if leaders move to different clusters, new leaders need to be elected, complicating the problem. In such networks, a weight-based selective state update algorithm might be helpful. To be more specific, each node is assigned a weight based on its mobility pattern. Higher weights are assigned to more stable nodes, and their states are more likely to be used for the state update of neighboring nodes. Consequently, cut detection is performed in a high level network with more stable nodes. This kind of algorithm would particularly work well for partially mobile wireless networks.

4.5.3. Multiple source nodes

Corollary 3.2 implies that the convergence of state is still achieved in the presence of multiple source nodes (or multiple malicious source nodes). As long as a node is connected to at least one source node, convergence state will be maintained, although the convergence value will be smaller than when it is connected to multiple source nodes. Thus, the argument of determining connectivity by looking at convergence is still valid for the case of multiple sources. Multiple source nodes can be leveraged to increase the significantly diluted convergence value at distant network extremes.

5. LR-CD: lightweight and robust cut detection

The ER-CD protocol, presented in Section 4, provides good performance with regards to energy-efficiency and robustness. However, its use requires additional computation and memory to handle clustering, leader election and routing. For users with more constrained memory and computational profiles, we propose LR-CD, the *Lightweight and Robust Cut Detection protocol*. The protocol overlays DSSD with a mechanism to detect malicious nodes. It does this by using statistical measures to detect outliers in the states of neighboring nodes.

The ER-CD algorithm starts with the observation that states of nodes in close proximity are similar. So it may seem tempting to directly use the states of neighbors as samples for the construction of a distribution for outlier detection. However, this naive approach is insufficient. In fact, in some cases, it is hard to assume a bell-shaped distribution based on samples of received states from neighbors for the following reasons: (i) the sample size of the received states is too small for some nodes; (ii) states in close proximity are not always similar, i.e., regional variations in the sample set exist for nodes that are located close to the source node; and (iii) the range of the state value is relatively large (depending on the source strength). Hence, a straightforward outlier rejection algorithm based on the samples of neighbors' states is not sufficient.

To this end, an application-specific outlier detection algorithm is proposed that mitigates the aforementioned problems (Algorithm 5). The main idea is to derive a new sample set from the sample set of neighbor states by taking the sum of differences to p nearest states for each sample. By representing a new sample as a difference of only a few nearest states, regional variations in states can be canceled out, and the range of sample becomes much smaller, thereby leading to a better sampling even for small data set.

More formally, let the received states of node v_i be a set $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in}\}$ and let the newly derived set of samples as $\mathfrak{S}_i = \{\mathfrak{s}_{i1}, \mathfrak{s}_{i2}, \ldots, \mathfrak{s}_{in}\}$. Algorithm 5 converts S_i to $\mathfrak{S}_i$. For each neighbor state, $s_{ik} \in S$ $(1 \leqslant k \leqslant n)$, the algorithm selects p nearest neighbor states, say $\{s^1_{ik}, s^2_{ik}, \ldots, s^p_{ik}\}$, and computes the distances from s_i to each s^j_{ik}, $1 \leqslant j \leqslant p$, denoted by $d^j_{ik} = |s_{ik} - s^j_{ik}|$. Then $\mathfrak{S}_i$ is the set described as the following:

$$\mathfrak{S}_i = \left\{ \sum_{j=1}^{p} d^j_{i1}, \sum_{j=1}^{p} d^j_{i2}, \ldots, \sum_{j=1}^{p} d^j_{in} \right\}. \tag{9}$$

After computing the mean μ and variance σ of the distribution $\mathfrak{S}_i$, the node v_i invokes, as shown in Algorithm 6, the Extreme Studentized Deviate (ESD) test which performs well in detecting outliers in a random normal sample. In the ESD test, the maximum deviation from the mean is computed and it is compared with a tabled value. If it is larger than the tabled value, then an outlier is identified.

To show that $\mathfrak{S}_i$ is a better data set showing higher accuracy in detecting an outlier, a comparison of the relative standard deviation (RSD) of $\mathfrak{S}_i$ is made with that of S_i. The RSD measures how well the samples in a data set are related to each other and is expressed as:

$$RSD = \frac{\sigma}{\mu} \times 100,$$

where $\sigma = \sqrt{\frac{\sum_j (\mathfrak{s}_{ij} - \mu)^2}{|S_i| - 1}}$ and $\mu = \frac{\sum_j \mathfrak{s}_{ij}}{|S_i|}$.

As experimental validation, a TOSSIM simulation was run using 100 nodes deployed in a grid topology with internode distance of 2 m. The radio range was approximately 5 m and all links were symmetric. Samples were taken of $\mathfrak{S}_i$ and S_i from nodes having different distances to the source node. Fig. 6 shows the RSD for both data sets as a function of the distance to the source node. Observe that nodes closer to the source node have higher variation in their states. Also observe that, by using $\mathfrak{S}_i$, it is possible to reduce the impact of regional variation, thus yielding higher precision in the outlier detection process.

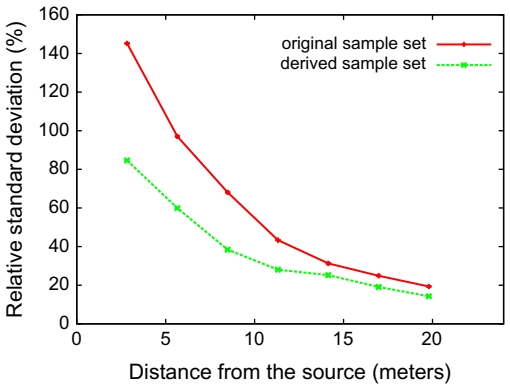


Fig. 6. The relative standard deviation of the derived sample set and the original sample set.

Algorithm 6. Outlier detection

Input: $\mathfrak{S}_i$
Output: *True*, *or False*
1: **if** $\frac{|s_{ik}-\mu|}{\sigma} > t\_table[|S_i|]$ **then**
2: return *True*
3: **end if**
4: return *False*

6. System implementation

The proposed algorithms for ER-CD and LR-CD were implemented in nesC for the TinyOS operating system [27]. Nodes in the network were provided with their locations during deployment and a loose time synchronization protocol was employed.

The complete ER-CD protocol executes in two phases. In the *Network Initialization* phase, each node builds neighbor tables and measures link quality by broadcasting a beacon every 5 s for 300 s and determining the ratio of successfully received beacons to the total number of beacons sent. The neighbor table and link measurements are used to find a GPSR routing path [25]. Each node joins a cluster based on its location and vertical/horizontal length of a cluster. The cluster elects a leader in a multihop, distributed manner [24]. If a node is elected as a leader, it randomly selects a user-specified number of probing locations. At the end of this phase, each node knows the locations of its neighbors and the cluster's leader.

In the second phase, the cut detection algorithm executes. Source strength is specified as $s = 200$ and the iteration period is set to 30 s. Each leader transmits its state to one of the probing locations using Geographic Forwarding. As long as the next hop does not deviate from the fictitious linear trajectory between the source and destination, the state message is forwarded to the probing location. Reaching the probing location, the state message is relayed to leaders in adjacent clusters using GPSR routing. After receiving state from an adjacent leader, a leader checks the sanity of the state by running the outlier detection algorithm. If the state is not an outlier, it is saved in the state table. The state is also stored in flash memory for future post-deployment analysis. When the iteration period expires, each node updates its state according to Eq. (1) and repeats the above procedure.

To ensure a lock-step execution of the algorithm, all motes are instructed to begin the first phase at roughly the same time via a "system start" message initiated by a designated node and forwarded by each node at most once.

In LR-CD, when a mote is turned on, it first goes into symmetry detection mode, and calculates the PRR. A node is considered to be a neighbor if its corresponding PRR is greater than 0.8 and is inserted in the neighbor table. The age threshold is also defined to monitor the link quality during the execution of the protocol. If a node misses the STATE_UPDATE message more than four times consecutively from a neighbor, then the neighbor is deleted from the neighbor table. Each node updates its state every 30 s using all neighbor states not considered to be outliers. At each update, it broadcasts the STATE_UPDATE message and also sends the message via the serial communication port for experimental data acquisition. The source strength s is set to 200.

7. Performance evaluation

To evaluate the performance of the two protocols, both simulations and experiments were conducted.

Simulations were performed using TOSSIM [28] on sets of 64, 144, 256, and 400 uniformly deployed nodes in various network sizes. The network topology and link gains were generated using the TinyOS tool LinkLayerModel [29]. The parameters used are summarized in Table 1.

Experiments were conducted in a testbed consisting of 41 nodes laid out in a semiregular grid. The purpose of testbed experiments was to overcome possible inaccuracies of simulation results due to simplified radio and interference models, and to obtain more realistic metrics like current consumption and real clock time. Fig. 7 shows a map of the testbed which consists of 42 Epic motes [10] deployed on a ceiling in two adjacent office areas. The ceiling in both rooms is 2.7 m high. The smaller office measures 6 by 4.5 m. The larger office measures approximately 9 by 7.6 m. Epic motes have an MSP430 processor running at 25 MHz with 10 KB RAM, 48KB flash memory size, and CC2420 IEEE 802.15.4 Chipcon wireless transceiver. To prevent significant radio range reduction, nodes are detached from the

Table 1
Simulation parameters for LinkLayerModel.

Topology	Grid
Number of nodes	64, 144, 256, and 400
Inter-node distance	2 m
Symmetry level of node	High
Path loss exponent	4.7
Shadowing standard deviation	3.2 dB
Power decay for the reference distance (1 m)	55.4 dB
Radio noise floor	−105.0 dB m
White Gaussian noise	4 dB

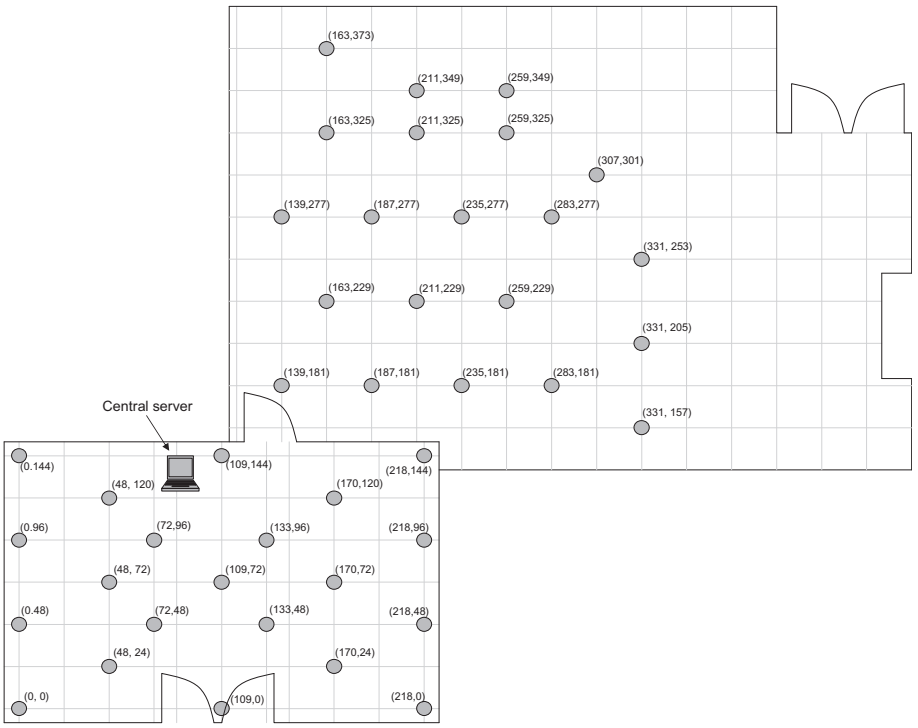


Fig. 7. The map of our testbed. (x,y) represents the relative coordinate of a node in inches with the node at the lower left corner as origin.

ceiling approximately 6 cm. Each mote is powered and programmed through a USB interface connected to the central server located in the middle of the deployment area.

To compare the ER-CD and LR-CD protocols with the state-of-the-art protocol, DSSD [9], several metrics were considered: convergence time, energy consumption, and energy distribution were assessed for ER-CD; malicious node detection accuracy (false positives, and negatives) and malicious node detection latency were assessed for LR-CD. The following parameters were varied: node density (*ND*), cluster size (*CS*), network size (*NS*), number of probing routes (*PR*), Ψ, and p values, where Ψ-value is a representation of how much the state of a malicious source deviates from the average state of its neighbors in percentage. To vary node density in the testbed, the transmission power was controlled by adjusting the TXCTRL.PA_LEVEL register of the CC2420 transceiver [30].

7.1. ER-CD: impact of node density and cluster size

Both node density and cluster size have an impact on the performance of ER-CD in terms of convergence rate and energy consumption. The convergence rate refers to the time required by nodes participating in the cut detection algorithm to achieve a steady state. This may be measured in iterations of the algorithm or in the amount of real clock time. Energy consumption is calculated either by measuring the number of algorithm-related packet transmissions in the entire network until convergence, or by gauging the current draw of real mote hardware. Experiments were performed both on the testbed and in the TOSSIM simulator. Data was obtained from the testbed to overcome the possible inaccuracies of the simulation result caused by the simplified radio and interference models in the TOSSIM simulator. Using this data as a baseline, additional simulations were performed using TOSSIM to extend the result to a large scale network and show the scalability of our algorithm.

Figs. 8 and 9 show the results obtained from the test-bed. For the test-bed experiment, CS was fixed to 10, and PR was fixed to 2. The convergence rate was measured in real time. Consumed energy (in Joules) required for con-

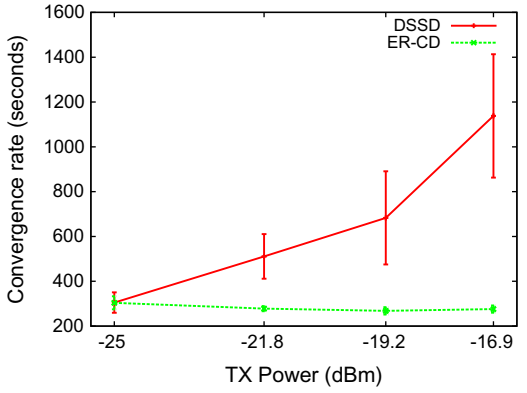


Fig. 8. Convergence rate in real clock time as a function of network density and cluster size (testbed).

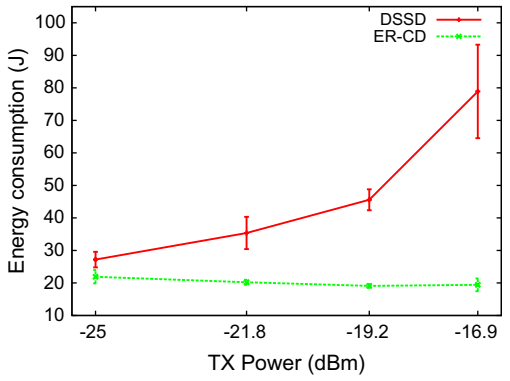


Fig. 9. Energy consumption in J as a function of different node density and different cluster size (testbed).

vergence was obtained by varying the transmission power. Transmit power and, as a consequence, network density were controlled by adjusting the TXCTRL.PA_LEVEL register [30]. As transmission power increased, convergence time for DSSD algorithm became greater, while convergence time for ER-CD stayed at an almost constant small value. This was because the convergence rate is determined mainly by the maximum degree of network [9]. For ER-CD, therefore, the convergence rate did not change much because the maximum degree of the virtual grid network was not affected by changing network density. However, at each iteration of the DSSD algorithm, all nodes in the network try to broadcast the STATE MESSAGE; so the higher the convergence rate is, the more algorithm-related packets are sent. Considering that the main contributor for energy consumption in wireless sensor networks is the radio transmission [31], total energy consumption for DSSD algorithm would also increase for higher network density. As expected, Fig. 9 shows that higher density caused higher energy consumption.

For the experiments in TOSSIM, network size (NS) was fixed to 256 nodes and the number of probing routes (PR) to 6. The number of iterations taken for the algorithm to reach convergence were measured across various network densities (average number of neighbors) and cluster sizes (the number of nodes in a cluster). Fig. 10 depicts the result. Observe that, similar to the results from the testbed, the convergence rate of ER-CD was almost constantly fast, while in DSSD the convergence rate rapidly increased as the network density increased. This result is, again, primarily due to the properties of the grid topology of the leaders, i.e., the maximum degree of the network of leaders is at most four regardless of network degree. The convergence rate for $CS = 16$ is slightly higher than that for $CS = 64$. The reason for the slight difference between the two cluster sizes is that, with smaller cluster size, there were more leaders, i.e., larger network size. The impact of network size on the performance of the algorithm will be further discussed in the next sub-section.

The total number of algorithm-related packets sent until the convergence is reached was measured as the metric for the energy consumption. Fig. 11 represents the result. As network density increased, the number of packet transmissions gradually decreased when using ER-CD. This is due to a reduction in the average number of hops in the GPSR routing path between pairs of leaders as the number of neighbors increased. For DSSD, the number of transmitted packets continuously increased because every node broadcasts once per iteration regardless of network density. Similarly, the number of iterations to reach convergence strictly increases as the network density increases.

7.2. ER-CD: impact of network size

Network size also influences the convergence rate and energy efficiency. For the experiments, the cluster size (CS) was fixed to 16, the network density (DS) to 14, and the number of probing routes (PR) to 6. The convergence rate of ER-CD and DSSD were plotted as a function of network size in Fig. 12. For both protocols, the convergence rate gradually increased as the network size increased. It is interesting to note that this result contradicts the argument in [9] that the convergence rate of DSSD algorithm does not depend on the network size

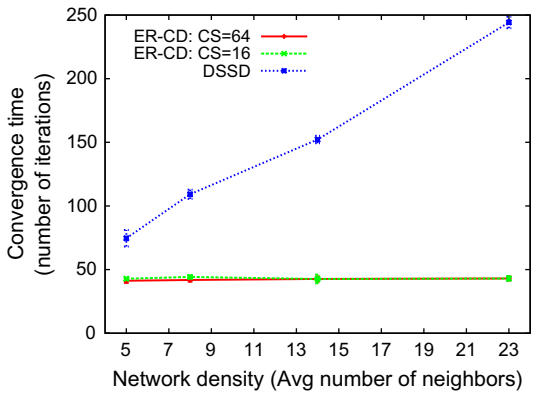


Fig. 10. Convergence time in number of iterations per different node density and different cluster size (TOSSIM).

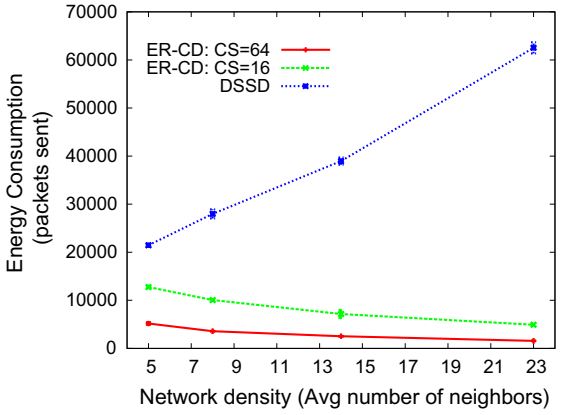


Fig. 11. Energy consumption in number of protocol-related packets sent as a function of different node density and different cluster size (TOSSIM).

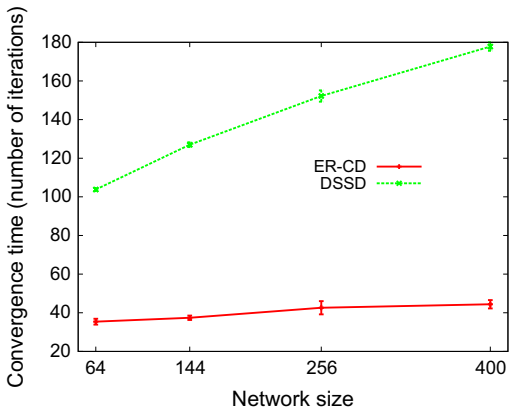


Fig. 12. Convergence time in number of iterations as a function of network size (TOSSIM).

and network topology (only the maximum degree of the network influences on the convergence rate). Conceptually, for each iteration, state is propagated one hop further from the source node until it reaches the farthest node in the network. Then the state is propagated back to entire network and influences other nodes' states. This procedure is repeated until all nodes converge. Thus, in a larger network of fixed density and maximum degree, the above procedure will require additional iterations due to the increased hop count.

Next, the relationship between energy consumption and network size was investigated. The result is depicted in Fig. 13. As expected, for both protocols, total energy consumption increased with increasing network size. A notable finding is that the energy consumption for DSSD increased more rapidly than that of ER-CD. The reason is straightforward: in DSSD, all nodes participate in the algorithm while in ER-CD only a subset of nodes, the leaders and relaying nodes, are engaged in the algorithm.

7.3. ER-CD: impact of number of probing routes

The number of probing routes has an effect on the total consumed energy and distribution of energy consumption throughout the network. First, experiments explored the relationship between the number of probing routes and the total consumed energy. For the experiments, the network size (NS) was fixed to 256 and the cluster size (CS) to 64. The total number of packets required to reach convergence were measured across various densities and using different numbers of probing routes. Fig. 14 depicts the result. Observe that as the number of probing routes increased, the amount of total energy consumption gradually increased. The reason for this is that, with more probing routes, it is more likely that there exists a longer path between the two adjacent leaders, which incurs additional communications cost. This additional cost becomes greater in lower density networks. This is because in these networks, the longer path contains larger number of additional hops than in the higher density networks.

The relationship between the number of probing routes and the distribution of the energy consumption throughout the network was also investigated. Calculations were based on the assumption that each node was equipped with a single AA battery as its power source which has 2000 mWh capacity, and that power consumption for transmission, reception, and staying awake are 3 W, 100 mW, and 10 mW per second, respectively. The residual energies of all nodes were measured when the first node depleted its power. Fig. 15 shows the result as a histogram of normalized residual energy as a function of node id. As the graph shows, the first depleted node appeared in the network with the smallest number of probing routes. For smaller PR, the same routes are used more often, thereby draining the power more quickly. On the other hand, for larger PR, the histogram is more evenly distributed, indicating a more equal distribution of energy consumption. However, in the network as a whole, more power is consumed for larger PR.

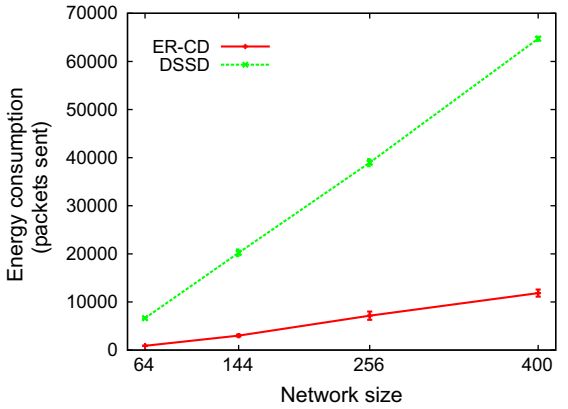


Fig. 13. Energy consumption in number of iterations as a function of network size (TOSSIM).

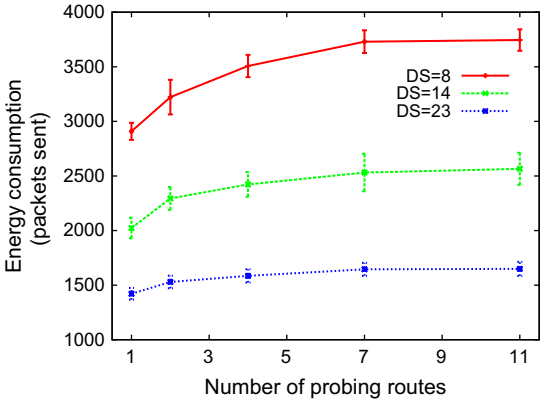


Fig. 14. Energy consumption in number of packets sent as a function of the number of probing routes (TOSSIM).

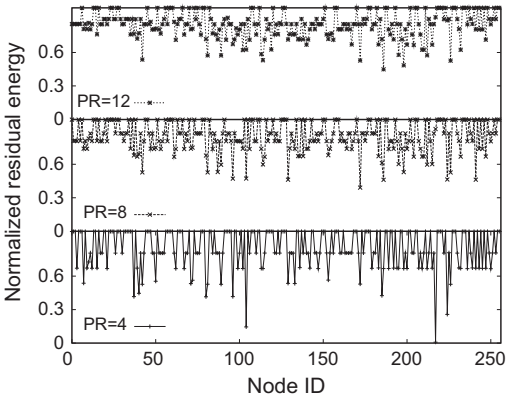


Fig. 15. Energy distribution as a function of the number of probing routes (TOSSIM).

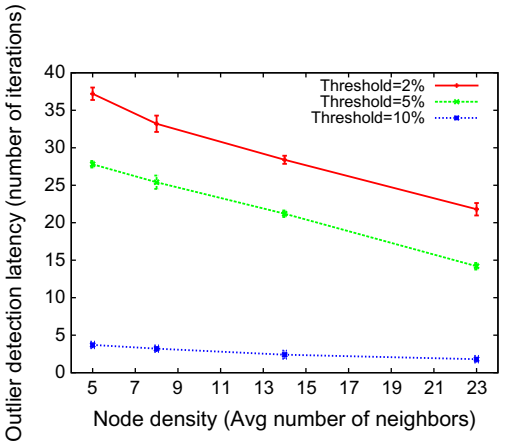


Fig. 17. Detection latency as a function of network density (TOSSIM).

7.4. LR-CD: malicious node detection latency

Since LR-CD builds on the DSSD protocol, only enhancements were investigated. In particular, experiments were conducted to measure malicious node detection latency, the number of iterations required to detect a malicious source. For this experiment, a cut was made by turning off some nodes at iteration k, after the network had converged. And then, at iteration $k+1$, a malicious source node began to inject false state into the network from a location in the disconnected segment. Elapsed time in the number of iterations between malicious source injection and detection by nodes in the disconnected segment was measured. For different thresholds (critical value in the Student t-table), the experiment was repeated using different Ψ values, a representation of how much the state of a malicious source deviates from the average state of its neighbors in percentage, i.e., $\Psi = (x_i(k) - \sum_{j \in N_i(k)} x_j(k)/|N_i(k)|)/100$, where $x_i(k)$ is the state of the malicious node (node i) and $N_i(k)$ is the set of its neighbors at iteration k.

Fig. 16 plots the detection latency as a function of Ψ. When a cut occurs, the states of nodes in the disconnected segment start to rapidly decrease towards zero, while the malicious source constantly injects a false state. As time elapses, with increasing number of iterations, the state difference between the malicious node and the other nodes becomes greater. If the difference exceeds the allowable limit, then the algorithm starts to detect the outlier. The graph shows that if the state of the malicious source is close to the average state of its neighbors, more iterations are needed to detect it. Especially, when the Ψ value was smaller than 1%, latency soared up making it hard to detect the malicious node. The impact of node density on malicious node detection latency was also examined. For the experiment, Ψ was fixed to 0.1. Fig. 17 depicts the result. Higher network density means larger sample size. With larger sample size, a better distribution can be determined which typically yields more accurate results. As the graph shows, with increased network density, better latency in detecting the malicious node was achieved.

7.5. LR-CD: malicious node detection accuracy

Fig. 18 explores the problem of selecting p parameter. This parameter indicates the number of nearest neighbor

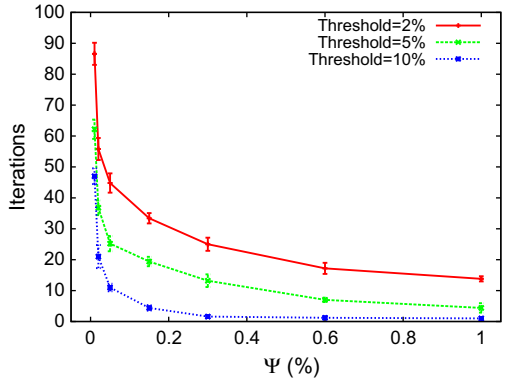


Fig. 16. Detection latency as a function of Ψ (TOSSIM).

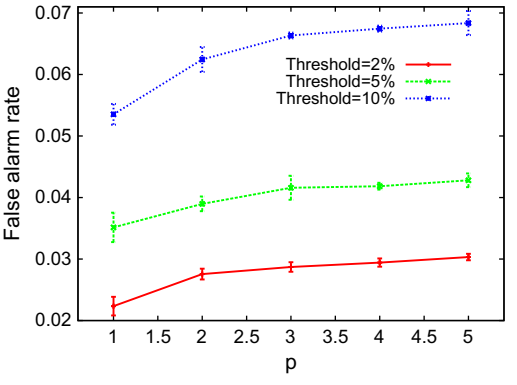


Fig. 18. False alarm rate in detecting malicious nodes (TOSSIM).

states used in the outlier detection algorithm. Its selection represents a tradeoff, i.e., larger values of p tend to increase the false alarm rate but enhance the detection of malicious nodes that collaborate to inject similar malicious states. On the other hand, smaller p-values yield lower false alarm rates but may impact the algorithm's ability to detect a group of malicious nodes injecting similar states. To be more specific, the use of a smaller number of nearest neighbors in computing the new sample and using distances to only a small number of top p closest neighbor states, causes the variance of the new sample to be smaller. However, if p is large, then it is more likely that the sample taken by a node might include states which are far from its own. Thus, the variance of the new sample set would increase, thereby giving lower detection accuracy with higher false alarm rate.

Higher thresholds mean that the system is more strict in detecting outliers, i.e., even a small deviation from normal behavior might be regarded as an outlier. On the other hand, smaller thresholds indicate the system is more flexible in determining the existence of an outlier. Only an outlier with relatively large deviation from normal behavior would be detected. Thus, as Fig. 18 shows, higher thresholds have a higher false alarm rate.

8. Conclusions

This article proposes robust, energy-efficient algorithms to enhance the detection of disrupted network connectivity in harsh, unattended low-security environments using a network composed of resource-constrained nodes. Through adoption of a clustered, leader-based convergence algorithm, it is shown that it is possible to greatly reduce the energy required to detect a cut. The algorithm also enhances security by enabling detection of malicious source nodes, even at very low thresholds. A simpler algorithm is offered that provides security without incurring the additional overhead needed for clustering and leader election. Parameters examined include cluster size, node density, and deviation thresholds. Each of these offer opportunities to trade off energy use and malicious source detection speed for optimal results in arbitrary networks.

References

[1] M. Won, M. George, R. Stoleru, RE^2-CD: robust and energy efficient cut detection in wireless sensor networks, in: Proceedings of International Conference on Wireless Algorithms, Systems, and Applications, 2009, pp. 80–93.

[2] S.M. George, W. Zhou, H. Chenji, M. Won, Y. Lee, A. Pazarloglou, R. Stoleru, P. Barooah, DistressNet: a wireless AdHoc and sensor network architecture for situation management in disaster response, IEEE Communications Magazine 48 (3).

[3] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J.A. Stankovic, T.F. Abdelzaher, J. Hui, B. Krogh, VigilNet: an integrated sensor network system for energy-efficient surveillance, ACM Transactions on Sensor Networking 2 (1) (2006) 1–38.

[4] A. Wood, J. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, R. Stoleru, Context-aware wireless sensor networks for assisted living and residential monitoring, Network IEEE 22 (4) (2008) 26–33.

[5] J. Kleinberg, Detecting a network failure, in: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 2000, 231.

[6] J. Kleinberg, M. Sandler, A. Slivkins, Network failure detection and graph connectivity, in: SODA '04: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004, pp. 76–85.

[7] H. Ritter, R. Winter, J. Schiller, A partition detection system for mobile ad-hoc networks, in: Proceedings of IEEE Conference on Sensor and Ad Hoc Communications and Networks (SECON), 2004, pp. 489–497.

[8] N. Shrivastava, S. Suri, C. Tóth, Detecting cuts in sensor networks, ACM Transactions on Sensor Networks 4 (2) (2008) 1–25.

[9] P. Barooah, Distributed cut detection in sensor networks, in: Proceedings of Conference on Decision and Control (CDC), 2008, pp. 1097–1102.

[10] P. Dutta, J. Taneja, J. Jeong, X. Jiang, D. Culler, A building block approach to sensornet systems, in: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys), 2008, pp. 267–280.

[11] V. Park, M. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, in: INFOCOM '97, vol. 3, 1997, pp. 1405–1413.

[12] C.-Y. Chong, S. Kumar, Sensor networks: evolution, opportunities, and challenges, Proceedings of the IEEE 91 (8) (2003) 1247–1256.

[13] A. Cerpa, D. Estrin, ASCENT: adaptive self-configuring sensor networks topologies, IEEE Transactions on Mobile Computing 3 (3) (2004) 272–285.

[14] M.V. Mahoney, P.K. Chan, Learning nonstationary models of normal network traffic for detecting novel attacks, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 376–385.

[15] E. Knorr, R. Ng, V. Tucakov, Distance-based outliers: algorithms and applications, The VLDB Journal 8 (3–4) (2000) 237–253.

[16] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, SIGMOD Record 29 (2) (2000) 427–438.

[17] J. Zhang, H. Wang, Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance, Knowledge and Information Systems 10 (3) (2006) 333–355.

[18] M. Breunig, H. Kriegel, R. Ng, J. Sander, LOF: identifying density-based local outliers, SIGMOD Record 29 (2) (2000) 93–104.

[19] S. Papadimitriou, H. Kitagawa, P. Gibbons, C. Faloutsos, LOCI: fast outlier detection using the local correlation integral, in: International Conference on Data Engineering, 2003, p. 315.

[20] F. Chung, Spectral graph theory, Regional Conference Series in Mathematics, Providence, RI, 1997.

[21] H. Attiya, J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, John Wiley & Sons, 2004.

[22] R. Stoleru, T. He, J.A. Stankovic, Range-free localization, Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks 30.

[23] T. He, C. Huang, B.M. Blum, J.A. Stankovic, T.F. Abdelzaher, Range-free localization and its impact on large scale sensor networks, ACM Transactions on Embedded Computing Systems 4 (4) (2005) 877–906.

[24] N. Malpani, J.L. Welch, N. Vaidya, Leader election algorithms for mobile ad hoc networks, in: Proceedings of Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 2000, pp. 96–103.

[25] B. Karp, H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: Proceedings of Conference on Mobile Computing and Networking (MOBICOM), 2000, pp. 243–254.

[26] D. Niculescu, B. Nath, Trajectory based forwarding and its applications, in: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom), 2003, pp. 260–272.

[27] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, in: Proceedings of Conference on Architectural Support for Programming Languages and Operating Systems, 2000, pp. 93–104.

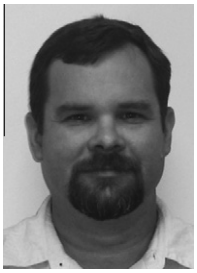
[28] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: accurate and scalable simulation of entire tinyos applications, in: SenSys '03, USA, 2003, pp. 126–137.

[29] M. Zuniga, B. Krishnamachari, Link layer models for wireless sensor networks, USC Technical Report, 2005.

[30] CC2420 Data Sheet. <http://www.chipcon.com>.

[31] V. Raghunathan, C. Schurgers, S. Park, M.B. Srivastava, Energy aware wireless sensor networks, IEEE Signal Processing Magazine 19 (2002) 40–50.

MyoungGyu Won (mgwon@cse.tamu.edu) received his B.E. degree with honors from Sogang University, Seoul, Korea. He is currently pursuing his Ph.D. degree in the Department of Computer Science and Engineering at Texas A&M University. His research interests include topology control, energy efficient routing protocols, and distributed computing in wireless ad-hoc and sensor networks.



Stephen M. George (smgeorge@cse.tamu.edu) is pursuing a Ph.D. in the Department of Computer Science and Engineering at Texas A&M University. His research interests include security and adaptive behavior in large-scale wireless sensor networks, particularly in disaster response and military applications. He received an M.S. in software engineering from Southern Methodist University.

Radu Stoleru (stoleru@cse.tamu.edu) is an assistant professor in the Department of Computer Science and Engineering at Texas A&M University. His research interests are in deeply embedded wireless sensor systems, distributed systems, embedded computing, and computer networking. He has authored over 35 papers and won the Outstanding Graduate Student Research Award from the Department of Computer Science, University of Virginia in 2007. He received a Ph.D. in computer science from the University of Virginia in 2007.