Cut Detection in Wireless Sensor Networks

Prabir Barooah, *Member*, *IEEE*, Harshavardhan Chenji, *Student Member*, *IEEE*, Radu Stoleru, *Member*, *IEEE*, and Tamás Kalmár-Nagy

Abstract—A wireless sensor network can get separated into multiple connected components due to the failure of some of its nodes, which is called a "cut." In this paper, we consider the problem of detecting cuts by the remaining nodes of a wireless sensor network. We propose an algorithm that allows 1) every node to detect when the connectivity to a specially designated node has been lost, and 2) one or more nodes (that are connected to the special node after the cut) to detect the occurrence of the cut. The algorithm is distributed and asynchronous: every node needs to communicate with only those nodes that are within its communication range. The algorithm is based on the iterative computation of a fictitious "electrical potential" of the nodes. The convergence rate of the underlying iterative scheme is independent of the size and structure of the network. We demonstrate the effectiveness of the proposed algorithm through simulations and a real hardware implementation.

Index Terms—Wireless networks, sensor networks, network separation, detection and estimation, iterative computation.

1 Introduction

Wireless sensor networks (WSNs) are a promising technology for monitoring large regions at high spatial and temporal resolution. However, the small size and low cost of the nodes that makes them attractive for widespread deployment also causes the disadvantage of low-operational reliability. A node may fail due to various factors such as mechanical/electrical problems, environmental degradation, battery depletion, or hostile tampering. In fact, node failure is expected to be quite common due to the typically limited energy budget of the nodes that are powered by small batteries. Failure of a set of nodes will reduce the number of multihop paths in the network. Such failures can cause a subset of nodes—that have not failed—to become disconnected from the rest, resulting in a "cut." Two nodes are said to be disconnected if there is no path between them.

We consider the problem of detecting cuts by the nodes of a wireless network. We assume that there is a specially designated node in the network, which we call the *source node*. The source node may be a base station that serves as an interface between the network and its users; the reason for this particular name is the electrical analogy introduced in Section 2.2. Since a cut may or may not separate a node from the source node, we distinguish between two distinct outcomes of a cut for a particular node. When a node u is disconnected from the source, we say that a Disconnected frOm Source (DOS) event has occurred for u. When a cut occurs in the network that does not separate a node u from the source node, we say that Connected, but a Cut Occurred Somewhere (CCOS) event has occurred for u. By cut detection we mean 1) detection by each node of a DOS event when it occurs, and 2) detection of CCOS events by the nodes close to a cut, and the approximate location of the cut. By "approximate location" of a cut we mean the location of one or more active nodes that lie at the boundary of the cut and that are connected to the source. Nodes that detect the occurrence and approximate locations of the cuts can then alert the source node or the base station.

To see the benefits of a cut detection capability, imagine that a sensor that wants to send data to the source node has been disconnected from the source node. Without the knowledge of the network's disconnected state, it may simply forward the data to the next node in the routing tree, which will do the same to its next node, and so on. However, this message passing merely wastes precious energy of the nodes; the cut prevents the data from reaching the destination. Therefore, on one hand, if a node were able to detect the occurrence of a cut, it could simply wait for the network to be repaired and eventually reconnected, which saves on-board energy of multiple nodes and prolongs their lives. On the other hand, the ability of the source node to detect the occurrence and location of a cut will allow it to undertake network repair. Thus, the ability to detect cuts by both the disconnected nodes and the source node will lead to the increase in the operational lifetime of the network as a whole. A method of repairing a disconnected network by using mobile nodes has been proposed in [1]. Algorithms for detecting cuts, as the one proposed here, can serve as useful tools for such network repairing methods. A review of prior work on cut detection in sensor networks, e.g., [2], [3], [4] and others, is included in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

In this paper, we propose a distributed algorithm to detect cuts, named the *Distributed Cut Detection* (DCD) algorithm. The algorithm allows each node to detect DOS events and a subset of nodes to detect CCOS events. The

- *P. Barooah is with the Deptartment of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611. E-mail: pbarooah@ufl.edu.*
- *H. Chenji and R. Stoleru are with the Deptartment of Computer Science and Engineering, Texas A&M University, College Station, TX 77845. E-mail: {cjh, stoleru}@cse.tamu.edu.*
- *T. Kalmár-Nagy is with the Deptartment of Aerospace Engineering, Texas A&M University, College Station, TX 77845. E-mail: kalmarnagy@tamu.edu.*

Manuscript received 25 Jan. 2011; revised 26 Apr. 2011; accepted 10 May 2011; published online 13 June 2011.
Recommended for acceptance by D. Simplot-Ryl.
For information on obtaining reprints of this article, please send E-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-01-0040.
Digital Object Identifier no. 10.1109/TPDS.2011.178.

algorithm we propose is distributed and asynchronous: it involves only local communication between neighboring nodes, and is robust to temporary communication failure between node pairs. A key component of the DCD algorithm is a distributed iterative computational step through which the nodes compute their (fictitious) electrical potentials. The convergence rate of the computation is independent of the size and structure of the network.

The DOS detection part of the algorithm is applicable to arbitrary networks; a node only needs to communicate a scalar variable to its neighbors. The CCOS detection part of the algorithm is limited to networks that are deployed in 2D euclidean spaces, and nodes need to know their own positions. The position information need not be highly accurate. The proposed algorithm is an extension of our previous work [5], which partially examined the DOS detection problem.

2 Distributed Cut Detection

2.1 Definitions and Problem Formulation

Time is measured with a discrete counter $k = -\infty, \ldots, -1, 0, 1, 2, \ldots$. We model a sensor network as a *time-varying graph* $\mathcal{G}(k) = (\mathcal{V}(k), \mathcal{E}(k))$, whose *node* set $\mathcal{V}(k)$ represents the sensor nodes active at time k and the *edge* set $\mathcal{E}(k)$ consists of pairs of nodes (u, v) such that nodes u and v can directly exchange messages between each other at time k. By an active node we mean a node that has not failed permanently. All graphs considered here are *undirected*, i.e., $(i, j) = (j, i)$. The *neighbors* of a node i is the set $\mathcal{N}_i$ of nodes connected to i, i.e., $\mathcal{N}_i = \{j | (i, j) \in \mathcal{E}\}$. The number of neighbors of i, $|\mathcal{N}_i(k)|$, is called its *degree*, which is denoted by $d_i(k)$. A path from i to j is a sequence of edges connecting i and j. A graph is called *connected* if there is a path between every pair of nodes. A *component* $\mathcal{G}_c$ of a graph $\mathcal{G}$ is a maximal connected subgraph of $\mathcal{G}$ (i.e., no other connected subgraph of $\mathcal{G}$ contains $\mathcal{G}_c$ as its subgraph).

In terms of these definitions, a *cut* event is formally defined as the increase of the number of components of a graph due to the failure of a subset of nodes (as depicted in Fig. 1). The *number of cuts* associated with a cut event is the increase in the number of components after the event.

The problem we seek to address is twofold. First, we want to enable every node to detect if it is disconnected from the source (i.e., if a DOS event has occurred). Second, we want to enable nodes that lie close to the cuts but are still connected to the source (i.e., those that experience CCOS events) to detect CCOS events and alert the source node.

There is an algorithm-independent limit to how accurately cuts can be detected by nodes still connected to the source, which are related to holes. Fig. 1 provides a motivating example. This is discussed in detail in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178, including formal definitions of "hole" etc. We therefore focus on developing methods to distinguish small holes from large holes/cuts. We allow the possibility that the algorithm may not be able to tell a *large* hole (one whose circumference is larger than $\ell_{\max}$) from a cut, since the examples of Figs. 1b and 1c show that it may be impossible to distinguish between them. Note that the discussion on hole detection part is limited to networks with nodes deployed in 2D.

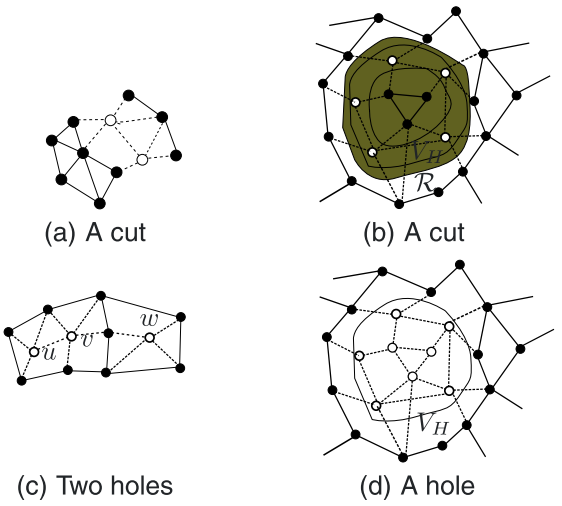


Fig. 1. Examples of cuts and holes. Filled circles represent active nodes and unfilled filled circles represent failed nodes. Solid lines represent edges, and dashed lines represent edges that existed before the failure of the nodes. The hole in (d) is indistinguishable from the cut in (b) to nodes that lie outside the region $\mathcal{R}$.

2.2 State Update Law and Electrical Analogy

The DCD algorithm is based on the following electrical analogy. Imagine the wireless sensor network as an electrical circuit where current is injected at the source node and extracted out of a common *fictitious* node that is connected to every node of the sensor network. Each edge is replaced by a $1\ \Omega$ resistor. When a cut separates certain nodes from the source node, the potential of each of those nodes becomes 0, since there is no current injection into their component. The potentials are computed by an iterative scheme (described in the sequel) which only requires periodic communication among neighboring nodes. The nodes use the computed potentials to detect if DOS events have occurred (i.e., if they are disconnected from the source node).

To detect CCOS events, the algorithm uses the fact that the potentials of the nodes that are connected to the source node also change after the cut. However, a change in a node's potential is not enough to detect CCOS events, since failure of nodes that do not cause a cut also leads to changes in the potentials of their neighbors. Therefore, CCOS detection proceeds by using probe messages that are initiated by certain nodes that encounter failed neighbors, and are forwarded from one node to another in a way that if a short path exists around a "hole" created by node failures, the message will reach the initiating node. The nodes that detect CCOS events then alert the source node about the cut.

Every node keeps a scalar variable, which is called its *state*. The state of node i at time k is denoted by $x_i(k)$. Every node i initializes its state to 0, i.e., $x_i(0) = 0, \forall i$. During the time interval between the kth and $k+1$th iterations, every node i broadcasts its current state $x_i(k)$ and listens for broadcasts from its current neighbors. Let $\mathcal{N}_i(k)$ be the set of neighbors of node i at time k. Assuming successful reception, i has access to the states of its neighbors, i.e., $x_j(k)$ for $j \in \mathcal{N}_i(k)$, at the end of this time period. The node then updates its state according to the following state

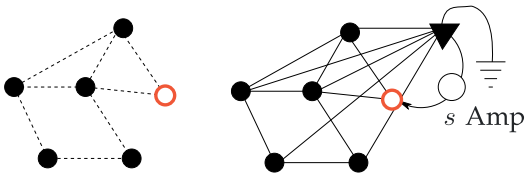


Fig. 2. A graph describing a sensor network $\mathcal{G}$ (left), and the associated fictitious electrical network $\mathcal{G}^{\text{elec}}$ (right). s Amp current is injected into the electrical network through the "source node" (unfilled circle), and extracted through the "ground" node (filled triangle). The line segments in the electrical network are $1\ \Omega$ resistors.

update law (the index $i = 1$ corresponds to the source node), where the *source strength* s (a positive number) is a design parameter

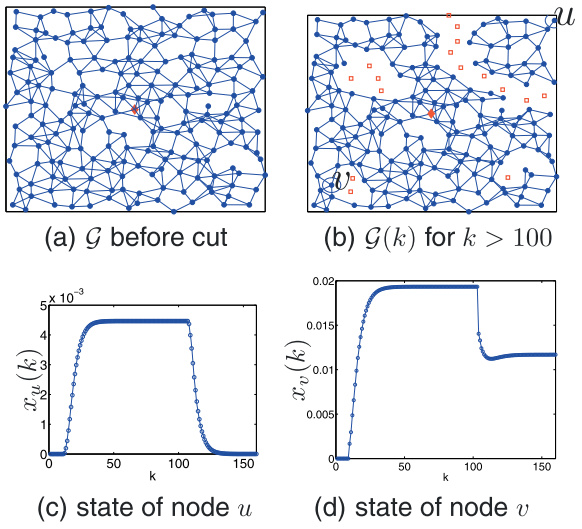
$$x_i(k+1) = \frac{1}{d_i(k)+1}\left(\sum_{j\in\mathcal{N}_i(k)} x_j(k) + s\mathbf{1}_{\{1\}}(i)\right), \qquad (1)$$

where $d_i(k) := |\mathcal{N}_i(k)|$ is the *degree* of node i at time k, and $\mathbf{1}_A(i)$ is the indicator function of the set A. That is, $\mathbf{1}_{\{1\}}(i) = 1$ if $i = 1$ (source node), and $\mathbf{1}_{\{1\}}(i) = 0$ if $i \neq 1$. After the state is updated, the next iteration starts. At deployment, nodes go through a neighbor discovery and every node i determines its initial neighbor set $\mathcal{N}_i(0)$. After that, i can update its neighbor list $\mathcal{N}_i(k)$ as follows: If no messages have been received from a neighboring node for the past τ_{drop} iterations, node i drops that node from its list of neighbors. The integer parameter τ_{drop} is a design choice.

To understand the state update law's relation to the electrical analogy described earlier, given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, imagine a fictitious graph $\mathcal{G}^{\text{elec}} = (\mathcal{V}^{\text{elec}}, \mathcal{E}^{\text{elec}})$ as follows: The node set of the fictitious graph is $\mathcal{V}^{\text{elec}} = \mathcal{V} \cup \{g\}$, where g is a fictitious *grounded* node; and every node in $\mathcal{V}$ is connected to the grounded node g with a single edge, which constitute the extra edges in $\mathcal{E}^{\text{elec}}$ that are not there in $\mathcal{E}$. Now an *electrical network* $(\mathcal{G}^{\text{elec}}, 1)$ is imagined by assigning to every edge of $\mathcal{G}^{\text{elec}}$ a resistance of $1\ \Omega$. Fig. 2 shows a graph $\mathcal{G}$ and the corresponding fictitious electrical network $(\mathcal{G}^{\text{elec}}, 1)$. It will be shown later in Theorem 1 (Section 2.4) that the state update law is simply an iterative procedure to compute the node potentials in the electrical network $(\mathcal{G}^{\text{elec}}, 1)$ in which s Ampere current is injected at the source node and extracted through the grounded node g. The potential of the grounded node g is held at 0.

When the sensor network $\mathcal{G}$ is connected, the state of a node converges to its potential in the electrical network $(\mathcal{G}^{\text{elec}}, 1)$, which is a positive number. If a cut occurs, the potential of a node that is disconnected from the source is 0; and this is the value its state converges to. If reconnection occurs after a cut, the states of reconnected nodes again converge to positive values. Therefore, a node can monitor whether it is connected or separated from the source by examining its state.

The above description assumes that all updates are done synchronously. In practice, especially with wireless communication, an asynchronous update is preferable. The algorithm can be easily extended to asynchronous setting by letting every node keep a buffer of the last received states of its neighbors. If a node does not receive messages from a neighbor during the interval between two iterations, it updates its state using the last successfully received state from that neighbor. In the asynchronous setting every node keeps a local iteration counter that may differ from those of other nodes by arbitrary amount.



(a) $\mathcal{G}$ before cut (b) $\mathcal{G}(k)$ for $k > 100$

(c) state of node u (d) state of node v

Fig. 3. (a)-(b): A sensor network with 200 nodes, shown before and after a cut. The cut occurs, at $k = 100$, due to the failure of the nodes shown as red squares. The source node is at the center. (c)-(d): The states of two nodes u and v as a function of iteration number.

Fig. 3 shows the evolution of the node states in a network of 200 nodes when the states are computed using the update law described above. The source node is at the center. The nodes shown as red squares in Fig. 3b fail at $k = 100$, and thereafter they do not participate in communication or computation. Figs. 3c and 3d show the time evolution of the states of the two nodes u and v, which are marked by circles in Fig. 3b. The state of node u (that is disconnected from the source due to the cut) decays to 0 after reaching a positive value, whereas the state of the node v (which is still connected after the cut) stays positive.

2.3 The Distributed Cut Detection Algorithm

2.3.1 DOS Detection

The approach here is to exploit the fact that if the state is close to 0 then the node is disconnected from the source, otherwise not (this is made precise in Theorem 1 of Section 2.4). In order to reduce sensitivity of the algorithm to variations in network size and structure, we use a normalized state. DOS detection part consists of steady-state detection, normalized state computation, and connection/separation detection. Every node i maintains a binary variable $\widehat{\texttt{DOS}}_i(k)$, which is set to 1 if the node believes it is disconnected from the source and 0 otherwise. This variable, which is called the *DOS status*, is initialized to 1 since there is no reason to believe a node is connected to the source initially.

A node keeps track of the positive steady states seen in the past using the following method. Each node i computes the *normalized state difference* $\delta x_i(k)$ as follows:

$$\delta x_i(k) = \begin{cases} \dfrac{x_i(k) - x_i(k-1)}{x_i(k-1)}, & \text{if } x_i(k-1) > \epsilon_{\text{zero}}, \\ \infty, & \text{otherwise,} \end{cases}$$

where ϵ_{zero} is a small positive number. A node i keeps a Boolean variable Positive Steady State Reached (PSSR) and updates $\texttt{PSSR}(k) \leftarrow 1$ if $|\delta x_i(\kappa)| < \epsilon_{\Delta x}$ for $\kappa = k - \tau_{\text{guard}}, k - \tau_{\text{guard}} + 1, \ldots, k$ (i.e., for τ_{guard} consecutive iterations), where

$\epsilon_{\Delta x}$ is a small positive number and τ_{guard} is a small integer. The initial 0 value of the state is not considered a steady state, so $\texttt{PSSR}(k) = 0$ for $k = 0, 1, \ldots, \tau_{\text{guard}}$.

Each node keeps an estimate of the most recent "steady state" observed, which is denoted by $\hat{x}_i^{ss}(k)$. This estimate is updated at every time k according to the following rule: if $\texttt{PSSR}(k) = 1$, then $\hat{x}_i^{ss}(k) \leftarrow x_i(k)$, otherwise $\hat{x}_i^{ss}(k) \leftarrow \hat{x}_i^{ss}(k-1)$. It is initialized as $\hat{x}_i^{ss}(0) = \infty$. Every node i also keeps a list of steady states seen in the past, one value for each unpunctuated interval of time during which the state was detected to be steady. This information is kept in a vector $\hat{X}_i^{ss}(k)$, which is initialized to be empty and is updated as follows: If $\texttt{PSSR}(k) = 1$ but $\texttt{PSSR}(k-1) = 0$, then $\hat{x}_{ss}(k)$ is appended to $\hat{X}_i^{ss}(k)$ as a new entry. If steady state reached was detected in both k and $k-1$ (i.e., $\texttt{PSSR}(k) = \texttt{PSSR}(k-1) = 1$), then the last entry of $\hat{X}_i^{ss}(k)$ is updated to $\hat{x}_i^{ss}(k)$. For instance, for the node v in the network shown in Figs. 3a and 3b, $\hat{X}_v^{ss}(3) = \phi$ (empty), $\hat{X}_v^{ss}(60) = [0.0019]$ and $\hat{X}_v^{ss}(150) = [0.019, 0.012]^T$. For future use, we also define *an unsteady interval* for a node i, which is a set of two local time counters $[k_i^{(1)},\ k_i^{(2)}]$ such that the state $x_i(k_i^{(1)} - 1)$ is a steady-state (i.e., $\texttt{PSSR}(k_i^{(1)} - 1) = 1$) but $x_i(k_i^{(1)})$ is not, and $x_i(k_i^{(2)})$ is not steady but $x_i(k_i^{(2)} + 1)$ is. With reference to Fig. 3d, the last unsteady interval for node v at time 150 is $[81,\ 101]^T$.

Each node computes a *normalized state* $x_i^{\text{norm}}(k)$ as

$$x_i^{\text{norm}}(k) := \begin{cases} \dfrac{x_i(k)}{\hat{x}_i^{ss}(k)}, & \text{if } \hat{x}_i^{ss}(k) > 0, \\ \infty, & \text{otherwise,} \end{cases}$$

where $\hat{x}_i^{ss}(k)$ is the last steady state seen by i at k, i.e., the last entry of the vector $\hat{X}_i^{ss}(k)$. If the normalized state of i is less than ϵ_{DOS}, where ϵ_{DOS} is a small positive number, then the node declares a cut has taken place: $\widehat{\texttt{DOS}}_i \leftarrow 1$. If the normalized state is ∞, meaning no steady state was seen until k, then $\widehat{\texttt{DOS}}_i(k)$ is set to 0 if the state is positive (i.e., $x_i(k) > \epsilon_{\text{zero}}$) and 1 otherwise.

2.3.2 CCOS Detection

The algorithm for detecting CCOS events relies on finding a short path around a hole, if it exists, and is partially inspired by the jamming detection algorithm proposed in [6]. The method utilizes node states to assign the task of hole-detection to the most appropriate nodes. When a node detects a large change in its local state as well as failure of one or more of its neighbors, and both of these events occur within a (predetermined) small time interval, the node initiates a PROBE message. The pseudocode for the algorithm that decides when to initiate a probe is included in Section 2 of the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

Each PROBE message p contains the following information:

1. a unique probe ID,
2. probe centroid C_p (see Algorithm PROBE_INITIATION in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178),
3. destination node,
4. path traversed (in chronological order), and
5. the angle traversed by the probe around the centroid.

The probe is forwarded in a manner such that if the probe is triggered by the creation of a small hole or cut (with circumference less than $\ell_{\max}$), the probe traverses a path around the hole in a counter-clockwise (CCW) direction and reaches the node that initiated the probe. In that case, the net angle traversed by the probe is 360 degree. On the other hand, if the probe was initiated by the occurrence of a boundary cut, even if the probe eventually reaches its node of initiation, the net angle traversed by the probe is 0. Nodes forward a probe only if the distance traveled by the probe (the number of hops) is smaller than a threshold value $\ell_{\max}$. Therefore, if a probe is initiated due to a large internal cut/hole, then it will be absorbed by a node (i.e., not forwarded because it exceeded the distance threshold constraint), and the absorbing node declares that a CCOS event has taken place. Details on when the source node is alerted about the occurrence of a cut in the network is included in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

The information required to compute and update these probe variables necessitates the following assumption for CCOS detection:

Assumption 1. *1) The sensor network is a two-dimensional geometric graph, with $P_i \in \mathbb{R}^2$ denoting the location of the ith node in a common Cartesian reference frame; 2) Each node knows its own location as well as the locations of its neighbors.*

The location information needed by the nodes need not be precise, since it is only used to compute destinations of probe messages. The assumption of the network being 2D is needed to be able to define CW or CCW direction unambiguously, which is used in forwarding probes. At the beginning of an iteration, every node starts with a list of probes to process. The list of probes is the union of the probes it received from its neighbors and the probe it decided to initiate, if any. The manner in which the information in each of the probes in its list is updated by a node is described in Section 2 of the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

2.4 Performance Analysis

The evolution of the node states with and without the occurrence of cuts in the general asynchronous and time-varying setting is stated in the next theorem. In the statement of Theorem 1 and Assumption 2, k_i is the local iteration counter at node i, and k is a global time counter. The global counter is used solely for the ease of exposition; a node does not need to have access to it. The following assumptions are used:

Assumption 2.

1. *Communication between nodes is symmetric;*
2. *If a node fails permanently, each of its neighbors can detect its failure within a fixed time period;*

3. *The source node never fails;*
4. *Every node takes part in the communication and state update infinitely often, i.e., as $k_1 \to \infty$, $k_i \to \infty$ for $\forall i$.*

Theorem 1. *Let the nodes of a sensor network $\mathcal{G}(k)$ execute the state update law in an asynchronous manner, subject to Assumption 2.*

1. *Let $\mathcal{G}_1(k) = (\mathcal{V}_1(k), \mathcal{E}_1(k))$ be the component of $\mathcal{G}(k)$ that contains the source node. If there exists k_0 such that $\mathcal{G}_1(k) = \mathcal{G}_1(k_0)$ for all $k \geq k_0$, then for every node $i \in \mathcal{V}_1(k)$ the state $x_i(k)$ converges to a positive number as $k \to \infty$ that is equal to the potential of the node i in the electrical network $(\mathcal{G}_1^{\text{elec}}(k_0), 1)$ with s Ampere flowing from the source node to the grounded node.*
2. *Let $\bar{\mathcal{G}}(k)$ be a component of $\mathcal{G}(k)$ that does not contain the source node for all $k \geq k_0$ for some positive integer k_0. Then, for every initial condition $\mathbf{x}(k_0) := [x_1(k_0), \ldots, x_n(k_0)]^T$, the state of every node in $\bar{\mathcal{G}}(k)$ converges to 0 as $k \to \infty$.*

The proof of this result is presented in Section 4 of the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178. It is important to notice that $\bar{\mathcal{G}}(k)$ is allowed to change with time in the second statement of the theorem; the only requirement is that the source node never be a part of it. Therefore, even if the graph keeps changing with time, e.g., due to node mobility, the states of the nodes that are disconnected from the source will converge to 0.

The DOS detection part of the proposed algorithm comes with a guarantee on the maximum delay incurred, which is stated in the following Lemma. The proof is provided in Section 4 of the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

Lemma 1. *Let the nodes of $\mathcal{G}(k)$ execute the DCD algorithm in a synchronous manner starting from $k = 0$, with $s \gg \epsilon_{\text{zero}}$ and ϵ_{zero} chosen such that $\epsilon_{\text{zero}} < \frac{1}{2}V_{\min}$, where $V_{\min}$ is the minimum node potential in the fictitious electrical network $\mathcal{G}^{\text{elec}}(0)$. Let the sequence $\mathcal{G}(k)$ be time invariant at all k except at one specific time instant $k^{\text{fail}} > 0$, at which time certain nodes fail leading to a cut in the network.*

1. *If $k^{\text{fail}} > K(\frac{1}{s}\epsilon_{\text{zero}}) + \tau_{\text{guard}}$, where $K(\cdot)$ is defined as*

$$K(x) := \frac{\log x}{\log\left(1 - \frac{1}{2+d_{max}}\right)}, \quad x > 0,$$

where d_{max} is the maximum node degree of the network $\mathcal{G}(0)$, then for every node i, we have $\widehat{\texttt{DOS}}_i(k) = 0$ for all $k \in [k_0\ k^{\text{fail}}]$ where k_0 is some integer that is less than k^{fail}.
2. *If $k^{\text{fail}} > K(\frac{1}{s}\epsilon_{\text{zero}}\epsilon_{\Delta x})$, then for each node i that is disconnected from the source after the cut, $\widehat{\texttt{DOS}}_i(k) = 1$ for all k that satisfies $k - k^{\text{fail}} > K(\frac{1}{s}\epsilon_{\text{zero}}\epsilon_{\text{DOS}})$.*

The first statement of the Lemma means that the nodes correctly determine that they are connected to the source at some time after deployment before the cut occurs. The second statement means that after some time after the cut, the nodes that are disconnected correctly determine the disconnection.

Lemma 1 follows from a number of technical results, which are stated and proved in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178. The key result among them is that the convergence rate of the state update law (1) does not depend on the size or topology of the network (see Proposition 1 in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178). The reason for this surprising attribute of the state update law is the following. Although communication takes place only among nearby neighbors in the physical network, every node can be thought of as communicating directly with the grounded node at every iteration in the fictitious electrical network. This is due to the $+1$ in the denominator in the update law (1), which averages the state of the grounded node (always 0) along with that of all other neighbors. Every node is one hop away from the grounded node in the fictitious electrical network, irrespective of the size and structure of the sensor network $\mathcal{G}$. As a result, the time it takes for each node's state to get arbitrarily close to its limiting value, is independent of the network's size and structure. This property makes the DCD algorithm scalable to large networks.

3 Performance Evaluation

Performance of the DCD algorithm was tested using MATLAB simulations (conducted in a synchronous manner) and then on a real WSN system consisting of micaZ motes [7]. Two important metrics of performance for the DCD algorithm are 1) detection accuracy, and 2) detection delay. Detection accuracy refers to the ability to detect a cut when it occurs and not declaring a cut when none has occurred. DOS detection delay for a node i that has undergone a DOS event is the minimum number of iterations (after the node has been disconnected) it takes before the node switches its $\widehat{\texttt{DOS}}_i$ flag from 0 to 1. CCOS detection delay is the minimum number of iterations it takes after the occurrence of a cut before a node detects it. A third metric, communication overhead, is discussed in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

In detecting disconnection from source (DOS) events, two kinds of inaccuracies are possible. A *DOS0/1 error* is said to occur if a node concludes it is connected to the source while it is in fact disconnected, i.e., node i declares $\widehat{\texttt{DOS}}_i$ to be 0 while it should be 1. A *DOS1/0 error* is said to occur if a node concludes that is disconnected from the source while in fact it is connected. In CCOS detection, again two kinds of inaccuracies are possible. A *CCOS0/1* error is said to occur when cut (or a large hole) has occurred but not a single node is able to detect it. A *CCOS1/0* error is said to occur when a node concludes that there has been a cut (or large hole) at a particular location while no cut has taken place near that location.

The algorithm's effectiveness is examined by evaluating the probabilities of the four types of possible errors enumerated above, as well as the detection delays. The probability of DOS0/1 error at time k is the ratio between

TABLE 1
List of Parameters that Have to Be Provided to the Nodes

Symbol	Name/description	Value
s	source strength	100
ϵ_{zero}	value below which the state is considered to be 0	10^{-10}
ϵ_{DOS}	value below which the normalized state is considered zero	10^{-3}
$\epsilon_{\Delta\text{x}}$	value below which the normalized state difference is considered zero	10^{-3}
τ_{guard}	time during which the normalized state difference has to be below $\epsilon_{\Delta\text{x}}$ for the state to be considered steady	3
τ_{drop}	number of failed consecutive transmissions before a neighbor is declared to have failed.	4
$\ell_{\max}$	maximum path length for a probe	15
$r^{\Delta ss}$	threshold ratio of change in the steady state for probe initiation	0.35

The numerical values shown here are used for all simulations and experimental evaluations reported in this document.

TABLE 2
DOS Detection Performance for the Networks Shown in Figs. 4

Network	(a)	(b)	(c)	(d)	(e)
Prob(DOS0/1 error)	0/0	0/0	0/0	0/0	0/0
Prob(DOS1/0 error)	0/0	0/0	0/0	0/0	0/0
DOS Delay (mean)	20	17	20	35	31
DOS Delay (std. dev.)	4.2	5.4	4.3	3.9	2

The two values of the probability shown in each cell correspond to $k = 60$ and $k = 160$, respectively.

the number of nodes that incur a DOS0/1 error (who believe they are connected but are not) at that time to the number of nodes that are disconnected from the source at that time. Probability of DOS1/0 error at k is the ratio between the number of nodes that incur a DOS1/0 error (who believe they are disconnected from the source but are in fact connected) to the number of nodes that are connected to the source at that time. The probability of CCOS0/1 error is the ratio between the number CCOS events (cuts or large holes) that are not detected by any nodes to the total number of such events in the network. The probability of CCOS1/0 error is the ratio between the number of nodes who declare that a CCOS event has taken place erroneously (i.e., due to absorbing a probe that was triggered by a small hole) to the number of nodes that initiate probe messages. Due to the fundamental difficulty in distinguishing cuts from holes discussed in Section 2.1, it is not considered an error if a node declares that a CCOS event has taken place in response to the creation of a large hole.

3.1 Choice of Parameters

The parameters $\epsilon_{\text{zero}}, \epsilon_{\text{DOS}}, \epsilon_{\Delta\text{x}}$, τ^{guard}, τ^{drop}, $\ell_{\max}$, and $r^{\Delta ss}$ have to be specified to all the nodes a priori. The parameter s has to be specified only to the source node. A detailed discussion on the choice of parameters and their effect on the DCD algorithm's performance is provided in Section 5 of the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178. The main conclusions are that 1) ϵ_{zero} should be chosen as small as possible and s should be chosen as large as possible to minimize detection error, 2) a smaller value of the parameter ϵ_{DOS} decreases probability of DOS1/0 error but increases DOS detection delay, and 3) the rest of the parameters do not seem to have a significant effect on the algorithm's performance. The values of the parameters used in all the simulations and experimental evaluations reported in this paper are shown in Table 1.

3.2 Evaluation through Simulations

Simulations are conducted on the five networks that are shown in Figs. 4a, 4b, 4c, 4d, and 4e.

3.2.1 DOS Detection Performance

In simulations with each of the five networks, the node failures occur at $k = 100$. Performance of the DOS detection part of the algorithm in terms of error probabilities and detection delays are summarized in Table 2. The error probabilities shown are the ones that are empirically computed at $k = 60$ and $k = 160$, i.e., 60 iterations after deployment and after the node failures occurred, respectively. The mean and standard deviation of DOS detection delay for a network are computed by averaging over the nodes that detected DOS events. We see from Table 2 that the algorithm is able to successfully detect initial connectivity to the source and then DOS events for all the five networks without requiring the parameters to be tuned for each network individually.

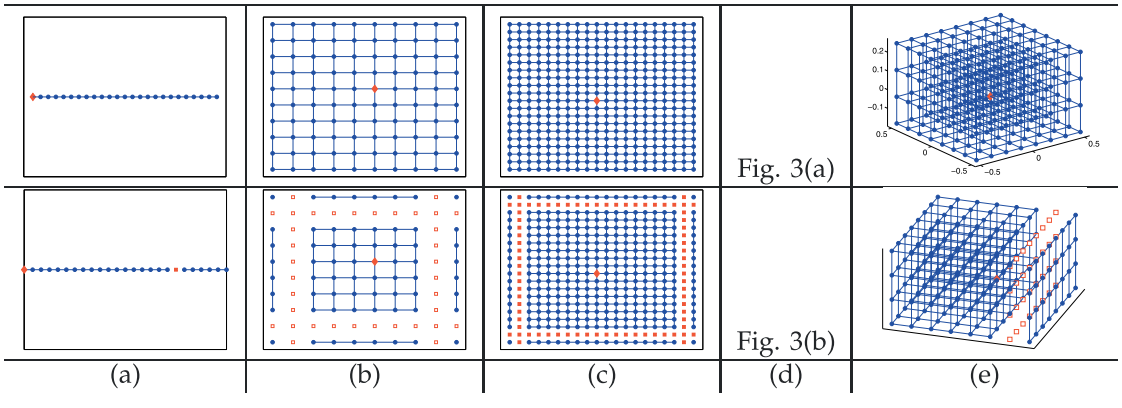


Fig. 4. Five networks before and after node failures: (a) 25-node 1D line network, (b) 100-node 2D grid, (c) 400-node 2D grid, (d) 200-node 2D random network, and (e) 256-node 3D grid ($8 \times 8 \times 4$).

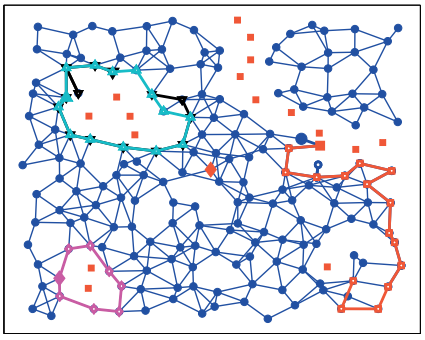


Fig. 5. The path of the probe messages in the network of Fig. 4d. Each probe path is marked with a distinct legend (circle, triangle, square, etc.), and the node that initiated the probe is shown as the one with the larger legend.

3.2.2 CCOS Detection Performance

Recall that the CCOS detection part of the algorithm is not applicable to 3D networks, so it was only tested on networks Figs. 4a, 4b, 4c, and 4d. As a specific example, Fig. 5 shows the path of the probes and their originating nodes in the network of Fig. 4d. Two probes were triggered by nodes close to the cut on the upper right corner, both of them were absorbed when the length of their path traversed exceeded $\ell_{\max}$ hops, which led to correctly detecting CCOS events. Among three probes that were triggered by nodes near small holes in this network, one of them—near the hole in the upper left corner—failed to find a path back to its originating node, leading to an erroneous declaration of an CCOS event by the absorbing node. The probability of a CCOS1/0 error in this case is therefore 0.33.

Table 3 summarizes the performance of the CCOS detection part of algorithm (executed with parameter values shown in Table 1). The CCOS detection error probabilities are 0 except in case of the network in Fig. 4d as described above.

Simulation studies reported in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178, (Section 5) shows that imprecise position information has little effect on the performance of the CCOS detection part of the algorithm. Analysis of communication cost of the algorithm is also reported in Section 5 of the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.178.

3.3 System Implementation and Evaluation

In this section, we describe the hardware/software implementation, outdoor deployment, and evaluation of the DCD algorithm. A network of 24 motes was deployed outdoors in a grassy field at Texas A&M University for a total deployment area of approximately 13×5 m^2. A partial

TABLE 3
CCOS Detection Performance for Four Networks in Figs. 4a and 4d

Network	(a)	(b)	(c)	(d)
Prob(CCOS1/0 error)	0	0	0	0.33
Prob(CCOS0/1 error)	0	0	0	0
CCOS Delay	33	40	37	40

The Error Probabilities Are at $k = 160$.

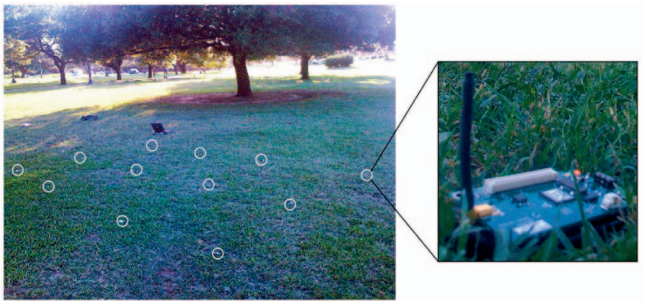


Fig. 6. Partial view of the 24 node outdoor deployment.

view of the outdoor deployment is shown in Fig. 6. The network connectivity is depicted in Fig. 7a.

The algorithm was implemented using the nesC language on micaZ motes [7] running the TinyOS operating system [8]. The code uses 16 KB of program memory and 719 B of RAM. The system executes in two phases: the Reliable Neighbor Discovery (RND) phase and the DCD Algorithm phase. In the RND phase each mote broadcasts a beacon within a fixed time interval of 5 s for 15 such intervals. Upon receiving a beacon, the mote updates the number of beacons received from that particular sender. To determine whether a communication link is established, each mote first computes for each of its neighbors the Packet Reception Ratio (PRR), defined as the ratio of the number of successfully received beacons and the total number of beacons sent by a neighbor. A neighbor is deemed reliable if the PRR > 0.8. Next, the DCD algorithm executes. After receiving state information from neighbors, a node updates its state according to (1) in an asynchronous manner and broadcasts its new state. The state is stored in the 512 KB on board flash memory at each iteration (for a total of about 1.6 KB for 200 iterations) for postdeployment analysis.

Experimental results for two of the sensor nodes deployed are shown in Fig. 7. The states of all nodes converged after about 30 iterations. At iteration $k = 83$ a cut

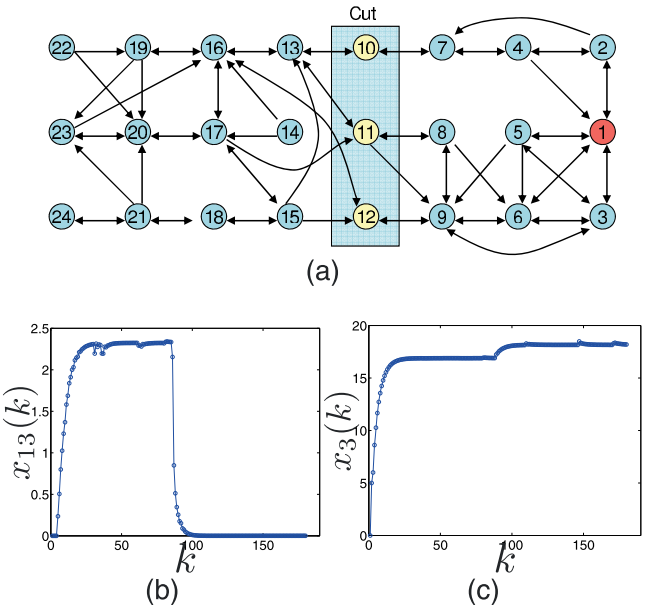


Fig. 7. (a) The network for the outdoor deployment. (b)-(c) The states of nodes 13 and 3, which are disconnected from and connected to, respectively, the source after the cut has occurred.

is created by turning off motes inside the rectangle labeled "*Cut*" in Fig. 7a. The states for this network approach their new steady state values around iteration $k = 117$. Figs. 7b and 7c show the states for nodes u and v, as depicted in Fig. 7a, which were connected and disconnected, respectively, from the source node after the cut.

The values of the parameters used by the DCD algorithm in the experimental evaluation are the same as those used in the MATLAB simulations, which are shown in Table 1. All nodes disconnected from the source detected the DOS event correctly; the mean DOS detection delay is 19 iterations, with a standard deviation of 4. The DOS detection delays can be substantially reduced by choosing a larger value for ϵ_{zero}. The CCOS detection part was executed offline, after the state data was collected from the nodes. Node 7 was the only node that initiated a probe, which reached node 7 again by traveling through the edges (7, 4), (4, 2), (2, 7), with a net angle of 0 around the probe centroid. Thus, 7 detected a CCOS event, with its former neighbor 10 as a boundary of the cut (or large hole).

4 Conclusions

The DCD algorithm we propose here enables every node of a wireless sensor network to detect Disconnected frOm Source events if they occur. Second, it enables a subset of nodes that experience CCOS events to detect them and estimate the approximate location of the cut in the form of a list of active nodes that lie at the boundary of the cut/hole. The DOS and CCOS events are defined with respect to a specially designated source node. The algorithm is based on ideas from electrical network theory and parallel iterative solution of linear equations.

Numerical simulations, as well as experimental evaluation on a real WSN system consisting of micaZ motes, show that the algorithm works effectively with a large classes of graphs of varying size and structure, without requiring changes in the parameters. For certain scenarios, the algorithm is assured to detect connection and disconnection to the source node without error. A key strength of the DCD algorithm is that the convergence rate of the underlying iterative scheme is quite fast and independent of the size and structure of the network, which makes detection using this algorithm quite fast. Application of the DCD algorithm to detect node separation and reconnection to the source in mobile networks is a topic of ongoing research.

References

[1] G. Dini, M. Pelagatti, and I.M. Savino, "An Algorithm for Reconnecting Wireless Sensor Network Partitions," *Proc. European Conf. Wireless Sensor Networks,* pp. 253-267, 2008.

[2] N. Shrivastava, S. Suri, and C.D. Tóth, "Detecting Cuts in Sensor Networks," *ACM Trans. Sensor Networks,* vol. 4, no. 2, pp. 1-25, 2008.

[3] H. Ritter, R. Winter, and J. Schiller, "A Partition Detection System for Mobile Ad-hoc Networks," *Proc. First Ann. IEEE Comm. Soc. Conf. Sensor and Ad Hoc Comm. and Networks (IEEE SECON '04),* pp. 489-497, Oct. 2004.

[4] M. Hauspie, J. Carle, and D. Simplot, "Partition Detection in Mobile Ad-Hoc Networks," *Proc. Second Mediterranean Workshop Ad-Hoc Networks,* pp. 25-27, 2003.

[5] P. Barooah, "Distributed Cut Detection in Sensor Networks," *Proc. 47th IEEE Conf. Decision and Control,* pp. 1097-1102, Dec. 2008.

[6] A.D. Wood, J.A. Stankovic, and S.H. Son, "Jam: A Jammed-Area Mapping Service for Sensor Networks," *Proc. IEEE Real Time Systems Symp.,* 2003.

[7] http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAZ_Datasheet.pdf, 2011.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS),* 2000.

Prabir Barooah was born in Jorhat, Assam, India. He received the BTech degree in mechanical engineering from the Indian Institute of Technology, Kanpur, in 1996, the MS degree in mechanical engineering from the University of Delaware in 1999, PhD degree in electrical and computer engineering in 2007 from the University of California, Santa Barbara. He is currently an assistant professor in the Department of Mechanical and Aerospace Engineering at University of Florida. From 1999 to 2002, he was a research engineer at United Technologies Research Center, East Hartford, CT. He is the winner of the US National Science Foundation (NSF) CAREER award (2010), General Chairs' Recognition Award for Interactive papers at the 48th IEEE Conference on Decision and Control (2009), Best Paper Award at the second International Conference on Intelligent Sensing and Information Processing (2005), and NASA group achievement award (2003). He serves on the editorial board of International Journal of Distributed Sensor Networks. He is a member of the IEEE.

Harshavardhan Chenji received the Bachelor of Technology degree in electrical and electronics engineering from the National Institute of Technology Karnataka, Surathkal, India in May 2007 and MS (computer engineering) degree in December 2009. He is currently working toward the PhD degree in the embedded & networked sensor systems (LENSS) Laboratory under the guidance of Dr. Radu Stoleru. He joined the Department of Computer Science and Engineering at Texas A&M University in August 2007. He is a student member of the IEEE and the IEEE Computer Society.

Radu Stoleru received the PhD degree in computer science from the University of Virginia in 2007, under professor John A. Stankovic. He is currently an assistant professor in the Department of Computer Science and Engineering at Texas A&M University. While at the University of Virginia, he received from the Department of Computer Science the Outstanding Graduate Student Research Award for 2007. His research interests are in deeply embedded wireless sensor systems, distributed systems, embedded computing, and computer networking. He has authored or co-authored more than 50 conferences and journal papers with over 1,000 citations. He is currently serving as an editorial board member for 3 international journal and has served as technical program committee member on numerous international conferences. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Tamás Kalmár-Nagy received the MS degree in engineering mathematics from the Technical University of Budapest and the PhD degree in theoretical and applied mechanics from Cornell University in 1995 and 2002, respectively. During 2002-2005, he was a research engineer at the United Technologies Research Center and he is now an assistant professor in the Department of Aerospace Engineering at Texas A&M University. He is the winner of the US National Science Foundation (NSF) CAREER award (2009), serves on the editorial board of two international journals and is a member of two ASME committees.