Proceedings of the
47th IEEE Conference on Decision and Control
Cancun, Mexico, Dec. 9-11, 2008

DISTRIBUTED CUT DETECTION IN SENSOR NETWORKS

Prabir Barooah

Abstract— **We propose a distributed algorithm to detect "cuts" in sensor networks, i.e., the failure of a set of nodes that separates the networks into two or more components. The algorithm consists of a simple iterative scheme in which every node updates a scalar state by communicating with its nearest neighbors. In the absence of cuts, the states converge to values that are equal to potentials in a fictitious electrical network. When a set of nodes gets separated from a special node, that we call a "source node", their states converge to 0 because "current is extracted" from the component but none is injected. These trends are used by every node to detect if a cut has occurred that has rendered it disconnected from the source. Although the algorithm is iterative and involves only local communication, its convergence rate is quite fast and is independent of the size of the network.**

I. INTRODUCTION

Wireless sensor networks (WSNs) have emerged as a promising new technology to monitor large regions at high spatial and temporal resolution. Virtually any physical variable of interest can be monitored by equipping a wireless device with a sensor and networking these sensors together with the help of their on-board wireless communication capability. WSNs can potentially have a large impact on diverse applications in the civil as well as defense arenas. However, several challenges have to be overcome to achieve the potential of WSNs. One of the challenges in the successful use of WSNs come from the limited energy of the individual sensor nodes. Significant current research has therefore been directed at reducing energy consumption at the sensor nodes. In the hardware front, energy efficient components have been developed, and in the software front, power aware routing, low complexity coding, and low power data processing algorithms have been examined.

Although these advances are expected to increase the lifetime of the wireless sensor nodes, due to their extremely limited energy budget and environmental degradation, node failure is expected to be quite common. This is especially true for sensor networks deployed in harsh and dangerous situations for critical applications, such as forest fire monitoring. In addition, the nodes of a sensor network deployed for defense applications may be subject to malicious tempering. When a number of sensors fail, whether due to running out of energy, environmental degradation, or malicious intervention, the resulting network topology may become disconnected. That is, as a result of failure of a set of nodes, a subset of nodes that have not failed become disconnected from the rest of the network.

Prabir Barooah is with the Dept. of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611. This research has been supported by the University of Florida.

In this paper we consider the problem of detecting *cuts* in wireless sensor networks. A sensor network is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ whose node set $\mathcal{V}$ correspond to the wireless sensors and whose edges $\mathcal{E}$ consists of pairs of nodes (u, v) that can communicate directly with each other. A *cut* is defined as the failure of a set of nodes so that the removal of those nodes and the edges incident on them from the original graph results in the separation of the graph into two or more components.

We propose a distributed algorithm that allows every node to monitor the topology of the (initially connected) graph and detect if a cut occurs. For reasons that will be clear soon, one node of the network is denoted as the "source node". The algorithm consists of every node updating a local state periodically by communicating with its nearest neighbors. The state of a node converges to a positive value in the absence of a cut. If a node is rendered disconnected from the source as a result of a cut, its state converges to 0. By monitoring its state, therefore, a node can determine if it has been separated from the source node. In addition, the nodes that are still connected to the source are able to detect that, one, a cut has occurred somewhere in the network, and two, they are still connected to the source node. We call it the *Distributed Source Separation Detection* (DSSD) algorithm.

Since the algorithm is iterative, a faster convergence rate is desirable for it to be effective. The convergence rate of the proposed algorithm is not only quite fast, but is independent of the size of the network. As a result, the delay between the occurrence of a cut and its detection by all the nodes can be made independent of the size of the network. This last feature makes the algorithm highly scalable to large sensor networks.

As noted by Shrivastava *et. al.* [1], the challenges posed by the possibility of network partitioning in WSNs has been recognized in several papers (see, e.g. [2], [3], [4]) but the problem of detecting when such partitioning occurs seems to have received little attention. Kleinberg *et. al.* have studied the problem of detecting network failures in *wired* networks, and proposed schemes for the case when k edges fail independently [5], [6].

To the best of our knowledge, the work by Shrivastava *et. al.* [1] is the only one that addresses the problem of detecting cuts in wireless sensor networks. They developed an algorithm for detecting ϵ linear cuts, which is a linear separation of ϵn nodes from the base station. The reason for the restriction to linear cuts is that their algorithm relies critically on a certain duality between straight line segments and points in 2D, which also restricts the algorithm in [1] to sensor

978-1-4244-3124-3/08/$25.00 ©2008 IEEE

networks deployed in the 2D plane. The algorithm developed in [1] needs a few nodes called *sentinels* that communicate with a base station either directly or through multi-hop paths. The base station detects ϵ-cuts by monitoring whether it can receive messages from the sentinels.

In contrast to the algorithm in [1], the DSSD algorithm proposed in this paper is not limited to ϵ-linear cuts; it can detect cuts that separate the network into multiple components of arbitrary shapes. Furthermore, the DSSD algorithm is not restricted to networks deployed in 2D, it does not require deploying sentinel nodes, and it allows every node to detect if a cut occurs.

The DSSD algorithm involves only nearest neighbor communication, which eliminates the need of routing messages to the source node. This feature makes the algorithm applicable to mobile nodes as well. Since the computation that a node has to carry out involves only averaging, it is particularly well suited to wireless sensor networks with nodes that have limited computational capability. Simulations are reported in [7] that illustrate the capability of the algorithm to detect cuts in mobile networks, and also its ability to detect if a "reconnection" occurs after a cut. The DSSD algorithm has been demonstrated in an wireless testbed with MicaZ motes [8].

Even though the proposed algorithm is iterative and involves only nearest neighbor communication, the convergence rate of the algorithm is quite fast and is independent of the size of the network. The assumptions are that the source node never fails, the sensor network is initially connected, and the communication between the sensor nodes is bidirectional.

II. Problem statement

Consider a sensor network modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, whose node set $\mathcal{V}$ represents the sensor nodes and the edge set $\mathcal{E}$ consists of pairs of nodes (u, v) such that nodes u and v can exchange messages between each other. Note that we assume inter-node communication is symmetric. An edge (u, v) is said to be incident on both the u and v. The nodes that share an edge with a particular node u are called the *neighbors* of u. A *cut* is the failure of a set of nodes $\mathcal{V}_{\text{cut}} \subset \mathcal{V}$ such that the removal of the nodes in $\mathcal{V}_{\text{cut}}$ and the edges that are incident on $\mathcal{V}_{\text{cut}}$ from $\mathcal{G}$ results in $\mathcal{G}$ being divided into multiple connected components. Recall that an undirected graph is said to be connected if there is a way to go from every node to every other node by traversing the edges, and that a *component* $\mathcal{G}_c$ of a graph $\mathcal{G}$ is a maximal connected subgraph of $\mathcal{G}$ (i.e., no other connected subgraph $\mathcal{G}'_c$ of $\mathcal{G}$ contains $\mathcal{G}_c$ as its subgraph). We are interested in devising a way to detect if a subset of the nodes has been disconnected from a distinguished node, which we call the *source node*, due to the occurrence of a cut.

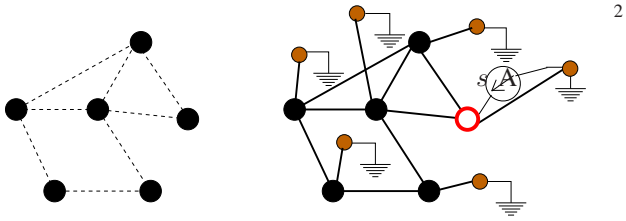


Fig. 1. A graph describing a sensor network (left), and the associated electrical network (right). In the electrical network, one node is chosen as the source that injects s Ampere current into the network, and additional nodes are introduced (fictitiously) that are grounded, through which the current flows out of the network. The thick line segments in the electrical network are resistors of 1Ω resistance.

III. Distributed Source Separation Detection (DSSD) Algorithm

The algorithm is based on an electrical analogy. Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with, say, n nodes and m edges that describes the sensor network, we first designate one of the nodes as the *source node*. The algorithm is designed to detect when nodes get disconnected from the source node. We now construct a fictitious graph $\mathcal{G}^{\text{elec}} = (\mathcal{V}^{\text{elec}}, \mathcal{E}^{\text{elec}})$ where $\mathcal{V}^{\text{elec}} = \mathcal{V} \cup \mathcal{V}^{\text{fict}}$, where $\mathcal{V}^{\text{fict}}$ consists of $n-1$ nodes, one node for every node in $\mathcal{V}$ except the source node, and every node in $\mathcal{V}$ is connected to its corresponding fictitious node in $\mathcal{V}^{\text{fict}}$ with a single edge. These edges constitute the extra edges in $\mathcal{E}^{\text{elec}}$ that were 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 $1\ \Omega$. Figure 1 shows a sensor network and the corresponding electrical network.

The DSSD algorithm consists of two phases. One is a state update law, which a simple iterative procedure to compute the node potentials in the electrical network $(\mathcal{G}^{\text{elec}}, 1)$ when s Ampere current is injected at the source node and extracted through the nodes $\mathcal{V}_{\text{fict}}$, with all the nodes in $\mathcal{V}_{\text{fict}}$ grounded. The *source strength* s is a design parameter. The other phase of the algorithm consists of monitoring the state of a node, which is used to detect if a cut has occurred. We now describe the two phases below. Note that the separation into two phases is merely for conceptual clarity, they are carried out simultaneously at every node.

A. State update law

Let $\mathcal{G}(k) = (\mathcal{V}(k), \mathcal{E}(k))$ denote the sensor network that consists of all the nodes and edges of $\mathcal{G}$ that are still active at time k, where $k = 0, 1, 2, \ldots$ is an iteration counter. For ease of description, we index the source node as 1. Every node u maintains a scalar state $x_u(k)$ that is iteratively updated. At every iteration k, nodes broadcast their current states. Let $\mathcal{N}_u(k) = \{v | (u, v) \in \mathcal{E}(k)\}$ denote the set of neighbors of u in the graph $\mathcal{G}(k)$. Every node in $\mathcal{V}$ except the source

updates its state as:

$$x_u(k+1) = \frac{1}{d_u(k)+1} \sum_{v \in \mathcal{N}_u(k)} x_v(k), \quad x_u(0) = 0,\ u \neq 1, \tag{1}$$

where $d_u(k) := |\mathcal{N}_u(k)|$ is the number of active neighbors of u at time k. If we count the fictitious node corresponding to u as one of u's neighbors whose state is held fixed at 0, then the above can be thought of as an average of the neighbors' states. The source node updates its state as:

$$x_1(k+1) = \frac{1}{d_1(k)+1} \left(\sum_{v \in \mathcal{N}_1(k)} x_v(k) + s \right) \quad x_1(0) = 0. \tag{2}$$

The description above assumes that all updates are done synchronously, or, in other words, every node shares the same iteration counter k. In practice, especially with wireless communication, an asynchronous update is preferable. To achieve this, every node keeps in its buffer a copy of the last received state of each of its neighbors. If in a particular iteration, a node does not receive messages from a neighbor during a time-out period, it updates its state using the last successfully received state from that neighbor. When a node fails, its neighbors will cease to receive messages from it permanently. When a node does not receive broadcasts from one of its neighbors for sufficiently long time, it removes that neighbor from its neighbor set. From then on, the node carries on the algorithm with the remaining neighbors.

We will need the following terminology. Given an electrical network $(\mathcal{G}, 1)$, two nodes u and o, and a set of nodes U, all in $\mathcal{G}$, suppose all of the nodes in U are shorted together and grounded. The potential at u (with respect to the ground) when a current source of s Ampere is connected between o and the ground is called *the potential difference between u and U with a current flow of s between o and U* in the network $(\mathcal{G}, 1)$.

The evolution of the node states with and without the occurrence of cuts is stated in the next theorem. Note that we assume that the source node never fails. The proof of the theorem is provided in the Appendix.

Theorem 1: Let the nodes of a sensor network modeled as an undirected graph $\mathcal{G}(t) = (\mathcal{V}, \mathcal{E}(t))$ that is initially connected (i.e., $\mathcal{G}(0)$ is connected) iteratively update their stated by (2) and (2) with an arbitrary initial condition $x_u(0)$, $u \in \mathcal{V}$.

1) If no nodes or edges fail, so that $\mathcal{G}(t) = \mathcal{G}(0)$ for all t, the state of every node converges to the potential difference between itself and $\mathcal{V}^{\text{fict}}$ with a current flow of s between the source and $\mathcal{V}^{\text{fict}}$ in the electrical network $(\mathcal{G}^{\text{elec}}, 1)$. Furthermore, the steady state potential is positive for every node.
2) If a node u gets disconnected from the source at time $\tau > 0$ and remains disconnected from the source for all $k > \tau$, then its state converges to 0,i.e., $x_u(k) \to 0$ as $k - \tau \to \infty$. □

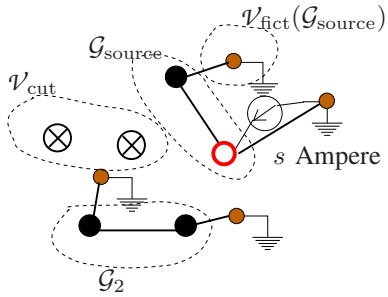


Fig. 2. The connected components of the electrical network after a cut occurs in the graph shown in Figure 1.

Figure 2 shows an example of a cut of the network that was shown in Figure 1.

B. State monitoring for cut detection

Theorem 1 shows how the occurrence of a cut in the network is manifested in the states of the nodes. By analyzing their own states, nodes can detect if a cut has occurred.

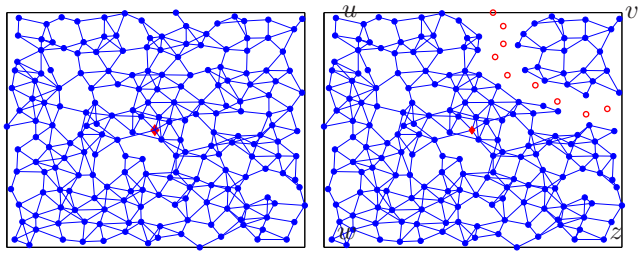
Suppose a cut occurs at some time $\tau > 0$ which separates the network into n components $\mathcal{G}_{\text{source}}, \mathcal{G}_2, \ldots, \mathcal{G}_n$, the component $\mathcal{G}_{\text{source}}$ containing the source node. Since there is no source (and therefore no current injection) in each of the components $\mathcal{G}_2, \ldots, \mathcal{G}_n$ disconnected from the source, it follows from Theorem 1 that the state of every node in each of these components will converge to zero. When the potential at a particular node drops below a particular threshold value, the node can declare itself cut from the source node. In fact, there may be additional node failures (and even increase in the number of components) after the cut appears. Since the state of a node converges to 0 if there is no path to the source, additional time variation in the network will not affect cut detection.

If additional failures do not occur after the cut occurs, it follows from Theorem 1 that the states of the nodes that are in the component $\mathcal{G}_{\text{source}}$ (which contains the source) will converge to new steady state values. So, if a node detects that its state has converged to a steady state, then changed, and then again converged to a new steady state value that is different from the initially seen steady state, it concludes that there has been a cut somewhere in the network.

A node detects when steady state is reached by comparing the derivative of its state (with respect to time) with a small number ϵ that is provided a-priori. The parameters s and ϵ are design variables.

IV. Simulations

The algorithm was tested in simulation on a graph shown in Figure 3(a) that consists of 200 nodes with an average degree 3.2. Figure 3(b) shows the graph after a cut appears.



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

Fig. 3. A sensor network with 200 nodes before and after a cut occurs. The source node is shown as a diamond shaped node at the center. Four nodes u, v, w, and z, at four corners of the network are shown for later reference.

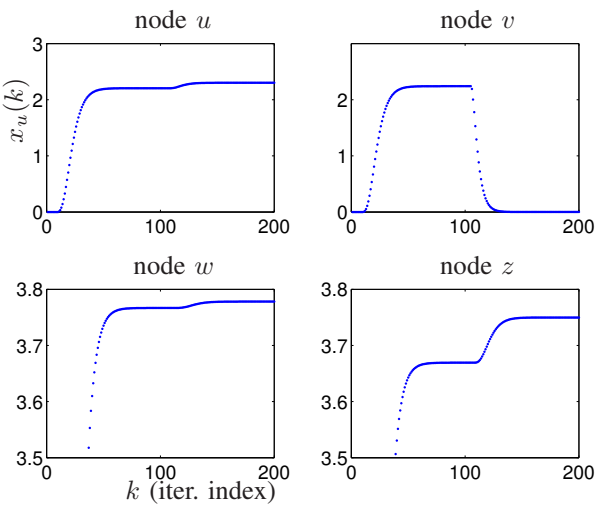


Fig. 4. The states of four nodes u, v, w and z (see Figure 3 for their locations) as a function of iteration number. The source strength was chosen arbitrarily at 5×10^4. A cut occurs at time $k = 100$. As the plots show, the states converge to their steady values quite fast.

For the sake of simplicity, simulations were run in a synchronous manner and a neighbor was assumed "dead" by a node the first time it failed to receive broadcasts from that neighbor (i.e., T_{th} in (6) was set to 1).

Figure 4 shows the time evolution of four nodes chosen from the four corners of the network, which are named u, v, w and z for easy reference (see Figure 3). Node v is the only one among the four that belongs to a component that is separated from the source after the cut occurs. Initially, the state of every node increases from 0 and then settles down to its steady state value. After the cut occurs, the state of the node v decreases to 0, and the states of the other three nodes u, w, and z converge to their new steady states.

Figure 5 shows the delay between the occurrence of a cut and its detection by the nodes, when cut detection is done by monitoring steady state values (as described in Section III-B). The average delay for the nodes that are disconnected is 49 iterations. Since no failures occur after the cut, even the nodes that remain connected to the source after the cut are able to detect correctly that there has been a cut somewhere but they are still connected to the source.

V. Analysis of the State Update Law

To analyze the behavior of the state update law described in Section III-A with and without cuts, we first express the iterations (1),(2) in a compact manner. Define $\mathbf{x}(k) := [x_1(k), \ldots, x_n(k)]^T$ as the vector of states of all the nodes in $\mathcal{G}$ at the k^{th} iteration. Now define $D = \text{diag}(d_1, \ldots, d_n)$ as the diagonal matrix of node degrees, where the degree of a node is the number of its neighbors. Let A be the adjacency matrix of the graph $\mathcal{G}$, i.e., $A_{u,v} = 1$ if $(u, v) \in \mathcal{E}$, and 0 otherwise. With these matrices, the iterations (1) and (2) can be compactly written as

$$\mathbf{x}(k+1) = (D + I)^{-1} \left(A\mathbf{x}(k) + s\, e_1 \right), \qquad (3)$$

where $e_1 = [1, 0, \ldots, 0]^T$. Eq. (3) is essentially the Jacobi method for solving the linear equation

$$\tilde{L}\mathbf{x} = s\, e_1 \qquad (4)$$

in an iterative manner [9], where

$$\tilde{L} := D - A + I. \qquad (5)$$

The matrix $\tilde{L}$ is the $n \times n$ principal sub-matrix of the Laplacian L^{elec} of the graph $\mathcal{G}^{\text{elec}}$, obtained by removing the rows and the columns corresponding to the nodes in $\mathcal{V}^{\text{fict}}$ from L^{elec}. Such matrices are known as Dirichlet Laplacians, since they appear in the numerical solution of the Laplace equation with Dirichlet boundary conditions [10]. As long as the graph $\mathcal{G}$ is connected, the matrix $\tilde{L}$ is invertible [10] and therefore, the solution to (4) exists and is unique.

When a cut occurs at a time $\tau > 0$, the nodes that were neighbors of the failed nodes will cease to receive broadcasts from the failed nodes. After a predetermined number of iterations in which messages are not received, the neighbors of a failed node recognizes its failure, and ceases to use the buffered state of the failed node to update their states. we use T_{th} to denote this predetermined number of iterations after which a node removes its failed neighbors from its list of neighbors. Since a component that has been disconnected from the source node after a cut contains no source node, the node belonging to it now execute a slightly different algorithm than what is described above. Consider the nodes of a component $\mathcal{G}_i$ that has been rendered disconnected from the source node after a cut. Since there is no current injection into that component, the iterations in that component can in fact be expressed as

$$\mathbf{x}^{(i)}(k+1) = (D^{(i)} + I)^{-1} \left(A^{(i)}\mathbf{x}^{(i)}(k) \right), \quad k > \tau + T_{\text{th}} \qquad (6)$$

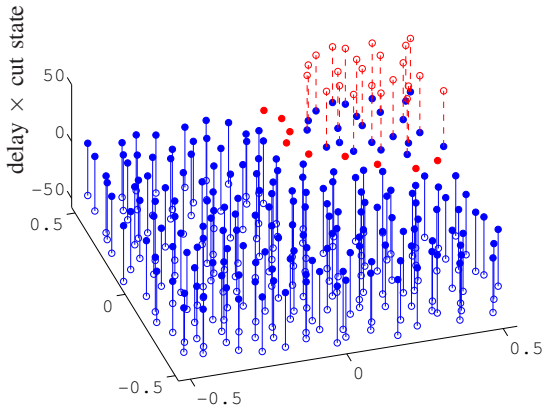


Fig. 5. The product of detection delay and the detected cut state of the nodes, when using steady state values to detect cuts, as described in Section III-B. Detection delay is the delay between the occurrence of the cut and its detection by a node. The "cut state" is: 0 if no cut is detected, 1 if the node detects that a cut has taken place and it is separated from the source, and -1 if a cut is detected but it is still connected to the source. The plot shows that every node detects its cut state correctly, with an average delay of 49 iterations for the nodes disconnected from the source, with a standard deviation of 0.8. Parameters: $s = 10^5$, $\epsilon = 10^{-3}$.

where the superscript (i) denotes that all quantities are defined for the component $\mathcal{G}_i$. This is precisely an average consensus algorithm [11].

On the other hand, the iterations carried out by the nodes in the component $\mathcal{G}_{\text{source}}$ that contains the source node still has the form (3). As stated in Theorem 1, the states of the nodes in $\mathcal{G}_{\text{source}}$ which now converge to positive values.

A. Convergence rate

Since the algorithm is iterative, a measure of its performance is its speed of convergence. Define the error $\mathbf{e}(k) := \mathbf{x}(k) - \mathbf{x}(k-1)$ so that (1),(2) reduce to

$$\mathbf{e}(k+1) = H\mathbf{e}(k), \qquad \text{where } H := (D+I)^{-1}A. \tag{7}$$

The error converges to 0 if and only if $\rho(H) < 1$, where $\rho(\cdot)$ denotes the spectral radius, since $\|\mathbf{e}(k)\| \approx \rho(H)^k \|\mathbf{e}(0)\|$ for large k. When $\rho(H)$ is much smaller than one, convergence is fast, while when it is close to 1, convergence is slow. Therefore the *spectral gap* $1 - \rho(H)$ can be taken as a measure of convergence speed. The larger the gap is, the faster the convergence, and faster the nodes can detect whether or not a cut has occurred, as described in Section III-B.

The next result, whose proof is provided in the Appendix, describes the convergence rate of the state-update law.

Lemma 1: For a connected graph $\mathcal{G}$, a lower bound on the convergence of the state update law (3) rate is given by

$$1 - \rho(H) \geq \frac{1}{2 + d_{\max}},$$

where H is defined in (7), and $d_{\max}$ is the maximum degree of a node in $\mathcal{G}$. □

Remark 1: A major strength of the algorithm is that its convergence rate is *independent of the number of nodes* in the graph. This is particularly remarkable in view of the fact that the algorithm is purely distributed and employs only nearest neighbor communication. The convergence rate of distributed algorithms that use nearest neighbor communication, such as average consensus, rendezvous, decentralized formation control, etc. typically depend on the algebraic connectivity of the graph (see [12], [13] and references therein). The algebraic connectivity tends to decrease as the size of the graph increases, slowing down the convergence rate [14]. In contrast, the DSSD algorithm's convergence rate is independent of the size of the network. The upshot of this property is that the delay between the occurrence of a cut and its detection can be bounded by a constant *irrespective of the size of the network*.

VI. Comments

Although we only discussed cut detection in this paper, the proposed algorithm can also be used for detection of "reconnection". If a component that is disconnected due to a cut gets reconnected later (say, due to the repairing of some of the failed nodes), the nodes can detect such reconnection from their states. Simulations are reported in [7] that illustrate the capability of the algorithm to (i) detect cuts in mobile networks and (ii) detect re-connections after cuts.

There are several issues related to the DSSD algorithm that need to be examined. The first is the appropriate choice of the design parameter s (source strength) in the algorithm. The potentials of nodes far away from the source typically become smaller as the network size increases. To keep the state values become too small – which will affect cut detection – the parameter s has to be chosen as a large number for a large network. Guidelines for choosing s depending on the size of the network, and how much a-priori knowledge of the network structure is needed to make the appropriate choice, is being investigated.

The states of the nodes computed by the DSSD algorithm are affected by even those node failures that do not lead to cuts. This is intuitive, since the electrical potential of a node in a resistive electrical network is a function of the network structure [12], [15], [16], which changes due to node failures. This feature raises the possibility of designing algorithms to compute "electrical potentials" of nodes in a wireless network so as to detect structural changes that are more complex than simply cuts.

While a protocol that enables nodes to detect cuts is useful, there is also a need for protocols that allow a base station to detect when and where a cut has occurred. We envision a protocol that lies on top of the DSSD algorithm to determine the location of a cut when it occurs. This will the subject of future investigation. Another related issue that merits investigation is *secure* cut detection, when some of the nodes may "fail" in a malicious mode, such as when nodes are hacked by an adversary to send incorrect state data.

REFERENCES

[1] N. Shrivastava, S. Suri, and C. D. Tóth, "Detecting cuts in sensor networks," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005, pp. 210–217.

[2] A. Cerpa and D. Estrin, "ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies," in *IEEE Infocom*. New York, NY: IEEE, June 2002.

[3] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration in wireless sensor networks," in *SenSys03*, Los Angeles, California, USA, November 57 2003.

[4] X. J. Du, M. Zhang, K. E. Nygard, S. Guizani, and H.-H. Chen, "Self-healing sensor networks with distributed decision making," *International Journal of Sensor Networks (IJSNET)*, vol. 2, no. 5/6, 2007.

[5] J. Kleinberg, "Detecting a network failure," *Internet Mathematics*, vol. 1, pp. 37–56, 2003.

[6] J. Kleinberg, M. Sandler, and A. Slivkins, "Network failure detection and graph connectivity," in *the 15th ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[7] P. Barooah, H. Chenji, R. Stoleru, and T. Kalmár-Nagy, "Detecting separation in robotic sensor networks," 2008, submitted to the IEEE wireless communications magazine.

[8] H. Chenji, P. Barooah, R. Stoleru, and T. Kalmár-Nagy, "Demo abstract: Distributed cut detection in sensor networks," in *6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[9] G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. The John Hopkins University Press, 1996.

[10] F. Chung, "Spectral graph theory," Regional Conference Series in Mathematics, Providence, R.I., 1997.

[11] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, June 2003.

[12] P. Barooah and J. P. Hespanha, "Graph effective resistances and distributed control: Spectral properties and applications," in *Proc. of the 45th IEEE Conference on Decision and Control*, December 2006, pp. 3479–3485.

[13] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control Systems Magazine*, vol. 27, pp. 71–82, April 2007.

[14] A. Olshevsky and J. N. Tsitsiklis, "Convergence rates in distributed consensus and averaging," in *45th IEEE Conference on Decision and Control*, December 2006.

[15] P. Barooah, "Estimation and control with relative measurements: Algorithms and scaling laws," Ph.D. dissertation, University of California, Santa Barbara, July 2007.

[16] P. G. Doyle and J. L. Snell, "Random walks and electric networks," Math. Assoc. of America, 1984.

[17] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, ser. Computer Science and Applied Mathematics. Academic Press, 1979.

[18] A. Frommer and D. Szyld, "On asynchronous iterations," *Journal of Comp. Appl. Math., 123*, pp. 201–216, 2000.

[19] W.-K. Chen, *Applied Graph Theory*, H. A. Lauwerier and W. T. Koiter, Eds. North Holland Publishing Company, 1971.

APPENDIX

In the following, we will use $A \succeq 0 (> 0)$ for a matrix A to denote entry-wise non-negativity (positivity).

Proof of Theorem 1. When $\tilde{L}$ is a Dirichlet Laplacian of a connected graph, it follows from straightforward application of Kirchhoff's current and voltage laws that the unique solution of $\tilde{L}x = e_1$ is a vector of node potentials with 1 Ampere current injected at node 1 and extracted from the nodes that are grounded; see [15] for a proof.

Since $\tilde{L}$ is a Dirichlet Laplacian of the connected undirected graph $\mathcal{G}^{\text{elec}}$ (that is, it is a principal submatrix of the graph Laplacian of $\mathcal{G}^{\text{elec}}$), it is a non-singular M-matrix, so that $\tilde{L}^{-1} \succ 0$ [17]. As a result, for a connected graph, $x = \tilde{L}^{-1}e_1$[6] is entry-wise positive.

Since $A \succeq 0$, we get $\tilde{L}^{-1}A \succeq 0$ [17]. Since $\tilde{L} = (D+I) - A$ with $D + I$ and $\tilde{L}$ invertible and $H \succeq 0$ where $H := (D + I)^{-1}A$, from Theorem 5.2 of [17, Chapter 7] it follows that $\rho(H) < 1$ and also

$$\rho(H) = 1 - \frac{1}{1 + \rho(\tilde{L}^{-1}A)} \tag{8}$$

Moreover, since H is entry-wise non-negative, it also follows from the above that $\rho(|H|) < 1$, where $|H|$ is a matrix obtained by replacing every entry of H with its absolute value. It follows from standard results that the synchronous iterations $x(i + 1) = Hx(i)$ as well as the corresponding asynchronous iterations converges [18]. The first statement of the theorem follows.

After a cut occurs, in every component $\mathcal{G}_i$ of the graph that does not contain the source, the nodes of $\mathcal{G}_i^{\text{elec}}$ run an average consensus algorithm, but with the fictitious nodes in $\mathcal{G}_i^{\text{elec}}$ holding their states at 0 at al times. The second statement follows now from standard results in consensus algorithms [13]. ∎

Proof of Lemma 1. Since $A = D + I - \tilde{L}$, we have that $\rho(\tilde{L}^{-1}A) = \rho(\tilde{L}^{-1}(D + I)) - 1$. Define $M := D + I$ and note that the spectrum of $\tilde{L}^{-1}M$ is the same as the spectrum of $M^{\frac{1}{2}}\tilde{L}^{-1}M^{\frac{1}{2}}$, and the latter being a symmetric positive definite matrix, its spectral radius is equal to its 2-norm. Therefore,

$$\begin{aligned}
\rho(\tilde{L}^{-1}M) = \|M^{\frac{1}{2}}\tilde{L}^{-1}M^{\frac{1}{2}}\| &= \max_{x \neq 0} \frac{x^T M^{\frac{1}{2}}\tilde{L}^{-1}M^{\frac{1}{2}}x}{x^T x} \\
&= \max \frac{y^T \tilde{L}^{-1} y}{y^T M^{-1} y} \quad (y := M^{\frac{1}{2}}x) \\
&\leq \max(M_{ii}) \max_y \frac{y^T \tilde{L}^{-1} y}{y^T y} = (d_{\max} + 1)\frac{1}{\lambda_{\min}(\tilde{L})}
\end{aligned}$$

Since $\tilde{L} = D - A + I = L + I$ where L is the Laplacian matrix of $\mathcal{G}$, the smallest eigenvalue of L is 0 since $\mathcal{G}$ is connected [19]. So $\lambda_{\min}(\tilde{L}) = 1$. Combining the above with (8), we get

$$\rho(H) \leq 1 - \frac{1}{2 + d_{\max}},$$

from which the result follows immediately. ∎