RE^2-CD: Robust and Energy Efficient Cut Detection in Wireless Sensor Networks

Myounggyu Won, Stephen M. George, and Radu Stoleru

Department of Computer Science and Engineering
Texas A&M University
{mgwon,mikegeorge,stoleru}@cse.tamu.edu

Abstract. Reliable network connectivity in wireless sensor networks (WSN) is difficult to achieve. Harsh, unattended, low-security environments and resource-constrained nodes exacerbate the problem. An ability to detect connectivity disruptions, due to either security or environmental problems, allows WSN to conserve power and memory while reducing network congestion. We propose RE^2-CD, an integrated solution incorporating both robustness to attack and energy-efficiency. To enhance security, a robust outlier detection algorithm assists nodes in detecting a specific threat in their environment. To improve energy-efficiency, a cluster-based cut detection algorithm recognizes and reacts to disrupted connectivity. Extensive simulations across a range of network sizes and densities 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), composed of numerous sensor nodes with small, low-power, inexpensive radios, have attracted a large amount of research that has led to interesting and innovative applications. However, challenging problems still exist. One of the most challenging problems in WSN is maintaining network connectivity to reliably 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 if a node can independently determine if a cut exists in the network.

Cut detection algorithms attempt to recognize and locate cuts. Using a state-based convergence mechanism, the current state-of-the-art cut detection algorithm, Distributed Source Separation Detection (DSSD) [1], reliably detects arbitrarily-shaped cuts and allows individual nodes to perform cut detection autonomously. However, the algorithm suffers from a number of problems. First, DSSD fails to address security, a critical component of sensor deployments in unattended environments. Second, the algorithm requires a lengthy, iterative convergence process. Finally, all nodes participate in the frequent broadcasts required to achieve convergence. This is cost-prohibitive with regards to power, especially in denser networks.

In light of these problems, we propose an algorithm with two principal components. Outlier detection, a statistical data analysis technique, resolves the security threat where a malicious node injects erroneous data into the cut detection process. Using data analysis, outlier detection identifies malicious source data and provides a light-weight, energy-efficient mechanism to validate neighbor data. Additionally, we propose an improved cut detection algorithm called *robust cluster-based cut detection*. This algorithm divides the network into a set of location-based clusters. Cluster leaders form a virtual grid network and the cut detection algorithm runs on this high-level network. As the algorithm executes, leaders converge to some state. A leader finding inconsistency in its expected state informs its neighbors and the sink that a cut has happened.

The contributions of this paper are:

– A method for identifying and recovering from changes caused by certain types of malicious nodes.
– An improved cut detection algorithm that converges faster while using less energy.
– Increased energy efficiency through more rapid detection of disrupted connectivity.

The paper is organized as follows. Section 2 discusses related work about cut detection and outlier detection algorithms in WSN. We introduce the robust cluster-based cut detection algorithm in Section 3. Our implementation is addressed in Section 4 which is followed by experimental results in Section 5. We offer conclusions and ideas for future work in Section 6.

2 Related Work

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 largely two mainstreams: 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 [2]. Non-parametric approaches using distance-based and density-based methods attempt to overcome this limitation. Knorr and Ng [3] proposed the first distance-based algorithm, where a point is regarded as an outlier if its distance to a k^{th} nearest neighbor point is greater than a certain threshold. One disadvantage is that the threshold must be defined. Ramaswamy et al. [4] studied distance-based detection, where a point is said to be an outlier if the distance to k^{th} nearest neighbor is greater than that of $n-1$ other points. Recently, Zhang et al [5] 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 [6][7] 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 [6], an outlier is determined by score. In [7], the bounding value MinPts is determined autonomously using statistical values such as inter-cluster distances and cluster diameters.

Research in the area of cut detection has emphasized the importance of the network partition monitoring problem [8]. For example, Chong et al [9] mentioned the problem from a security perspective arguing that nodes deployed in a hostile environment must be able to detect tampering. In [10], Cerpa and Estrin stressed the importance of the network cut detection problem in their self-configuring topology scheme but left it as a future work. However, little progress has been made to resolve the problem. An early paper by Kleinberg, et al. [11] considered the problem in a wired network. Their main argument is to select good "agents" to monitor the partition and accurately detect separation events. Much like the "agent" node, Ritter et al. [12] defined "border" nodes responsible for the detection of network partition. Recently, Shrivasta et al. [13] proposed a deterministic algorithm to detect network separation using a set of sentinel nodes to monitor the linear-cut of a network. The most recent cut detection algorithm is proposed by Barooah, et. al. [1]. Their algorithm can not only detect an arbitrary shape of cut, but also enables every node in the network to autonomously detect a cut by maintaining state.

3 Robust and Energy Efficient Cut Detection

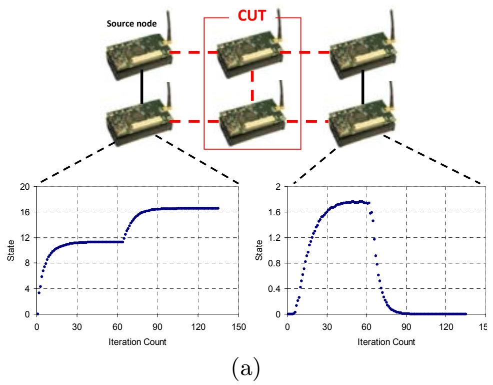
In this section, we present the theoretical foundations of cut detection and propose algorithms to enhance robustness and improve energy efficiency.

3.1 Preliminaries

We model our network as an undirected, connected graph $G = (V, E)$, where the set of vertices $V = \{v_1, v_2, ..., v_m\}$ is the set of m nodes in the network and the set of edges $E = \{(v_i, v_j)|v_i, v_j \in V\}$ represents radio connectivity among nodes in the network. We denote by $N_i = \{v_j|(v_i, v_j) \in E\}$ the set of *neighbors* of a node v_i, and by $|N_i|$ the *degree* of node v_i.

Time is denoted as a discrete counter $k = 0, 1, 2, ...$ Each node v_i maintains a positive real value $x_i(k)$ which is called the state. The state is initialized to zero, i.e., $x_i(0) = 0$ at time $k = 0$. One node in the network is designated as the *source node.* Although the *source node* may be selected arbitrarily, by convention we select the sink to be the source in WSN. For simplicity, we assume that v_1 is the *source node.*

At every iteration k, each node v_i updates its state $x_i(k)$ and broadcasts it. All nodes except the *source node* update their states using the following equation:



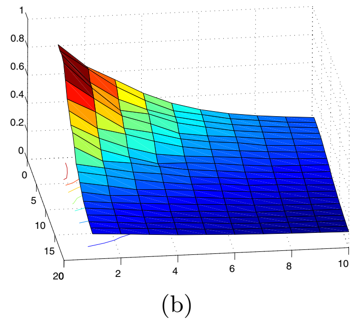


Fig. 1. (a) A cut occurs in a connected network of six nodes. The graph depicts the scalar states of two nodes, one in G_{source} and one $\notin G_{source}$. (b) The distribution of node states in a 20 grid network, with a source node at (0,0).

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

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

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

where s, called the *source strength*, is a user specified scalar. Previously it was proved that the state of each node converges, after a number of iterations, to a positive value [1].

We define a "cut" as a network partition, in which the graph G is separated into n disjoint connected components $G_{source}, G_2, ..., G_n$, where $G_{source} = (V_{source}, E_{source})$ is a graph which contains the *source node*. When a "cut" occurs, the state of each node $v \notin V_{source}$ converges to 0 [1].

The convergence of a node's state is illustrated in Figure 1(a). Around iteration 40, the scalar state of nodes in the network converges. Shortly after iteration 60, a cut occurs in the network when the two nodes in the middle fail. After the cut, the state of a node on the right side rapidly decays to 0 while the state of a node on the left side converges to a new higher state. A critical observation is that the states of all nodes converge to new values, hence *all nodes* have the ability to detect a cut in the network.

One troublesome aspect of cut detection using this distributed algorithm is that it is susceptible to attacks. A malicious node located in the disconnected part of the network can imitate a source node, and hence affect the state value that each node computes. In the following section we analyze the impact of such malicious nodes and propose an algorithm to detect and recover from malicious behavior.

3.2 Robust Cut Detection Algorithm

Temporary variations of a node's state, often caused by packet loss, can be tolerated by a system implementing cut detection as described above. The states of nodes in the network will eventually converge. However, this is not true when a non-source node continuously injects a constant state to the system. This malicious source node is formally defined as:

Definition 1. *A node $v_i \in G$ is a* malicious *node M_i if it acts as a source node in the network, i.e., it updates its state according to equation 2 with an arbitrary strength s', as given by:*

$$x_i(k+1) = \frac{1}{|N_i|+1}\left(\sum_{j \in N_i(k)} x_j(k) + s'\right). \tag{3}$$

In the following theorem, we prove the damaging impact of a malicious source node in the source-disconnected segment of the network:

Theorem 1. *If there exists a malicious node M_i in the disconnected region of the network, the nodes in that region cannot detect a cut using the state update equation (1).*

Proof. We can rewrite equations 1 and 2 together in a matrix representation, as follows:

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

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

$$LX = se_1, \, where \, L = D - A + I. \tag{5}$$

Now assume that a cut partitions the network G into $G_{source}, G_2, \ldots, G_n$ and that there is a malicious source M_k with strength s' in a partitioned network G_j. It is clear that A, D, and I can also be partitioned such that $A = A_1 + A_2 + \ldots + A_n$, $D = D_1 + D_2 + \ldots + D_n$, and $I = I_1 + I_2 + \ldots + I_n$. Thus:

$$\begin{aligned} L &= \sum_{i=1}^{n} D_i - \sum_{i=1}^{n} A_i + \sum_{i=1}^{n} I_i \\ &= \sum_{i=1}^{n} (D_i - A_i + I_i) \\ &= L_1 + L_2 + \ldots + L_3, \end{aligned}$$

which gives $(L_1 + L_2 + \ldots + L_n)X = se_1 + s'e_k$. Accordingly, the disconnected part of the network, which has a malicious node, is actually another system with M_k such that:

$$L_j X = s'e_k \tag{6}$$

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

Corollary 1. *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 is more than one malicious node $M_1, M_2, \ldots, M_n$ with corresponding source strength $s'_1, s'_2, \ldots, s'_n$ in the disconnected region G_j of the network. From (6) we have,

$$L_j X = \sum_{i=1}^{n} s'_i e_i \tag{7}$$

It is clear that a unique solution of X still exists as the right side of the equation does not affect the invertibility of L_j. □

Our proposed robust cut detection algorithm is based on the observation that the states of nodes in close proximity are similar. As shown in Figure 1(b), which depicts the distribution of state values for a sample 10×20 network, state values are inversely related to distance from the source node. One might observe that there is no significant state difference between nearby nodes and be tempted to directly use the states of neighbor nodes as samples to construct a distribution for outlier detection. However, this naive approach is insufficient. In fact, it is hard to assume a certain distribution based on samples of received states from neighbors, because: i) the sample size of states is irregular and small for some nodes; ii) states in close proximity are not always similar, i.e., regional variations exist for some nodes, especially nodes close to the source; iii) the range of the state value is relatively large. Hence, a straightforward outlier rejection algorithm might fail.

Our main idea is to derive new samples from the states of neighbor nodes in the way that the new samples are not susceptible to the aforementioned problems. We denote the set of received states of a node v as $S = \{s_1, s_2, \ldots, s_n\}$ and also denote the newly derived set of samples as S^N. The algorithm for converting S to S^N and finding a distribution of S^N is presented in Algorithm 1. For each neighbor state of a node v, i.e., $s_i \in S$, the algorithm selects p nearest neighbor states $s_l \in S$, and computes the distances: $d_{il} = |s_i - s_l|$, for $i \neq l$. We denote these distances as $\{d_{i1}, d_{i2}, \ldots, d_{i(p-1)}\}$. Then S^N is the set described by:

$$S^N = \left\{ \sum_{j=1}^{p-1} d_{1j}, \sum_{j=1}^{p-1} d_{2j}, \ldots \sum_{j=1}^{p-1} d_{nj} \right\} \tag{8}$$

Once the mean μ and variance σ to approximate the distribution of S^N are computed, the node v invokes, as shown in Algorithm 2, the extreme studentized deviate test (ESD) which performs well in detecting outliers in a random normal sample. In the ESD test, maximum deviation from the mean is calculated and compared with a tabled value. If the maximum deviation is greater than the

Algorithm 1. Compute Distribution

```
Input: $S$, $p$
Output: $\mu$, $\sigma$
  $S_i^N \leftarrow \emptyset$, $min\_dist \leftarrow 0$, $min\_idx \leftarrow 0$
  for $i = 0$ to $|S|$ do
    for $k = 0$ to $p$ do
      for $j = 0$ to $|S|$ do
        if $i \neq j$ and $s_j \neq -1$ then
          $dist \leftarrow |s_i - s_j|$
          if $dist < min\_dist$ then
            $min\_dist \leftarrow dist$
            $min\_idx \leftarrow j$
          end if
        end if
      end for
      $s_{min\_idx} \leftarrow -1$
      $s_i^N \leftarrow s_i^N + min\_dist$
    end for
  end for
  $\mu \leftarrow \frac{\sum_i s_i^N}{|S|}$
  $\sigma \leftarrow \sqrt{\frac{\sum_i (s_i^N - \mu)^2}{|S|-1}}$
```

Algorithm 2. Outlier Detection

```
Input: $s_i^N$
Output: $True$, or $False$
  if $\frac{|s_i^N - \mu|}{\sigma} > t\_table[|S|]$ then
    return $True$
  end if
  return $False$
```

tabled value, then an outlier is identified. Since our algorithm is based on taking a difference between closest states, we can not only cancel out the regional variations in states, but also make the range of sample much smaller, which leads us to get a better sampling even for small data set.

3.3 Energy Efficient Cut Detection

The proposed robust cut detection algorithm (section 3.1), as well as other state of art algorithms [1], suffers from relatively high energy consumption, since all nodes participate in the execution of the algorithm. To address this problem, we propose to execute the robust cut detection algorithm on a small subset of nodes. The main idea is to partition the network into a grid of clusters as depicted in Figure 2(a). The nodes in each cluster elect a leader who executes the robust cut detection algorithm by exchanging state values only with leaders in adjacent clusters.

An interesting byproduct of clustering is that the grid topology formed by leaders in the network allows leaders to easily compute the expected convergence value of their states. This is possible because the grid structure bounds degree of each leader; therefore by knowing its position within the grid of leaders, an individual leader can easily compute the adjacency matrix and diagonal matrix of node degrees without additional message overheads. An additional benefit is that the degree of leaders in our clustered environment is much smaller, normally no larger than 4, than would be typical of a network where all nodes participate. Since the convergence is determined by the spectral gap (i.e., the smallest non-trivial eigenvalue of the graph Laplacian), convergence speed rapidly grows with increasing spectral gap [1]. Therefore, maintaining a small spectral gap is important for energy constrained WSN.

More formally, if the system parameters a and b are known to leaders (see upper left corner of Figure 2(a)), they can easily construct the adjacency matrix A and the diagonal matrix D of node degrees. The equations 1 and 2 can be rewritten in matrix representation:

$$X(k+1) = (D(k)+I)^{-1}(A(k)X(k)+se_1) \tag{9}$$

Consequently, by simple matrix multiplication, each leader can exactly compute the next state of itself and its 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 a cut occurs, a leader can accurately detect it by checking if its state is different from the expected value. Similarly, a leader can detect a malicious source node in the network by comparing the received state from the malicious node to the expected state.

Details of the energy efficient cut detection protocol are further described below.

Network Initialization. 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 any existing node localization protocol [14]. The network is divided into a set of rectangular clusters of size $a \times b$ (a and b are system parameters), as shown in Figure 2(a). Based on location, a node becomes a member of a particular cluster, 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 $0 \le i \le \lceil \frac{A}{a} \rceil$ and $0 \le j \le \lceil \frac{B}{b} \rceil$.

Next, in each cluster $G(i,j)$ a leader $L(i,j)$ is elected [15]. At the end of the leader election phase, all nodes in a cluster know the ID and location of their leader. As a byproduct, the leader also knows the total numbers of nodes participating in the election. Elected leaders in the network form a *the 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 $0 \le i-1$, $i+1 \le \lceil \frac{A}{a} \rceil$, $0 \le j-1$, $j+1 \le \lceil \frac{B}{b} \rceil$.

Cut Detection in the Virtual Grid Network. Once the network initialization phase is complete, the robust cut detection algorithm begins to execute on

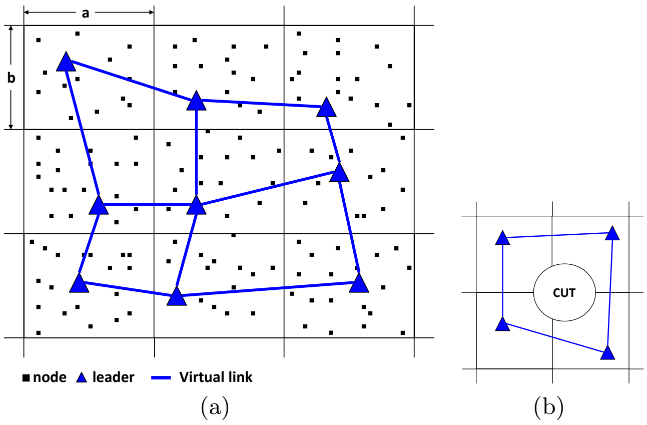


Fig. 2. (a) A Virtual Grid Network obtained from leaders elected in their respective clusters. (b) The occurrence of a "local cut" that does not trigger a network partition.

G_{grid}. Each leader sends a "STATE_UPDATE" message containing the leader's current state value and location to leaders of adjacent clusters. Routing between leaders is handled with a variant of GPSR [16]. 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 a "STATE_UPDATE" packet destined for a non-leader location updates the destination with the correct location and ID of the cluster's leader.

The execution of the cut detection in the virtual grid is complicated by a scenario in which a "local cut" does not include any leaders, as shown in Figure 2(b). This problem is overcome during a periodic leader rotation phase. Due to space constraints, we limit our description of the solution to the fact that, as part of leader rotation, the current leader node queries other nodes in the cluster about their energy level. If the leader detects that fewer than expected nodes responded, it may infer that a local cut has occurred. While leader rotation still occurs, the new leader is informed that a local cut may have occurred. The new leader executes a local cut detection algorithm in coordination with adjacent cluster leaders.

Since the cut detection algorithm iteratively executes in the the entire network, it is easy to observe that our proposed cluster-based cut detection algorithm consumes much less energy (since it performs only on the nodes in G_{grid}, which are significantly fewer). In addition, each leader in the virtual grid can detect erroneous state from a malicious attacker because it knows the expected states from its neighbors. Simple arithmetic comparison suffices as a defense algorithm against a malicious node.

4 Implementation

The proposed algorithms for robust and energy efficient cut detection were implemented in nesC for the TinyOS operating system [17] and executed in the

TOSSIM simulator. We assume that nodes in the network obtain their locations through other means and that a loose time synchronization protocol is present.

The complete system executes in two phases. In the "Network Initialization" phase, each node in the network broadcasts a beacon message every 5 sec. Using the beacon messages from neighbors, nodes build neighbor tables and measure link quality by computing the Packet Reception Ratio (PRR), defined as the ratio of successfully received beacons to the total number of beacons sent by a neighbor. The neighbor table and the link measurement is used to find a GPSR routing path [16]. Each node joins a cluster and the cluster elects a leader in a multihop, distributed manner [15]. At the end of this phase, each node knows the locations of its neighbors and the cluster's leader.

In the second phase, the RE^2-CD algorithm executes. Source strength is specified as $s = 100$ and the iteration period is set to 5 seconds. Each leader transmits its state to leaders in adjacent clusters using GPSR routing and the leader location discovery mechanism explained previously. 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 equation 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.

5 Performance Evaluation

In order to evaluate the performance, we conducted simulations using TOSSIM [18] on a set of 264 uniformly deployed nodes in networks of sizes $320 \times 320\ m^2$, $560 \times 560\ m^2$ and $640 \times 640\ m^2$. The radio communication radius of a node was set to $50m$.

5.1 Convergence Speed

Convergence occurs when nodes participating in the cut detection algorithm achieve a steady state. This may be measured in iterations of the algorithm or in the amount of time required.

Cut-detection convergence latency measured in iterations is depicted in Figure 3(a). The number of iterations required for convergence depends on the average degree of a node [1]. Since the degree of a node is at most 4 in our virtual grid scheme, convergence is guaranteed to occur in a bounded number of iterations. Simulations show that convergence is obtained in RE^2-CD in an average of 34 iterations regardless of the number of clusters. In contrast, DSSD shows the number of iterations that rapidly increases as the degree of the network increases.

Figure 3(b) shows convergence latency measured in time, specifically in a "time unit" of packet transmission delay. To measure the convergence latency

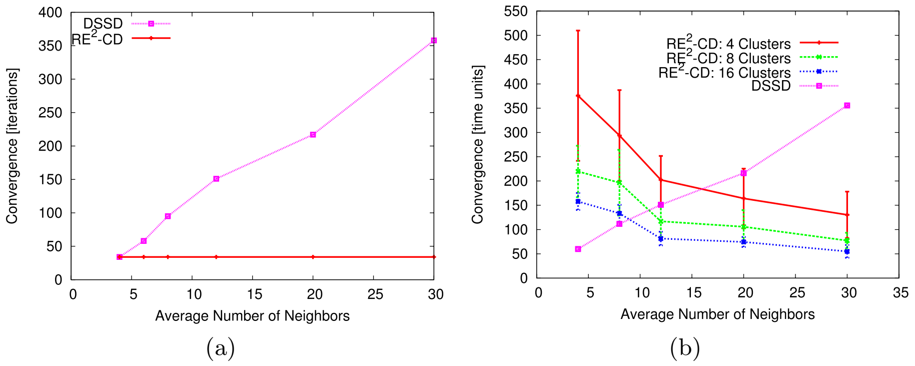


Fig. 3. (a) Convergence speed measured in number of iterations (b) Convergence speed measured in amount of time

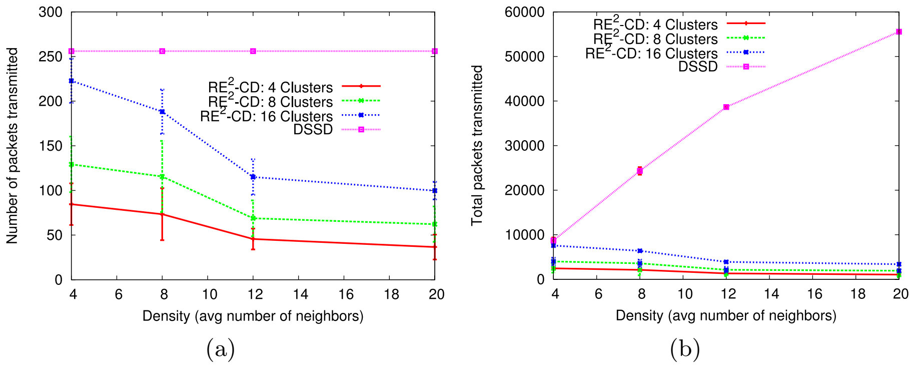


Fig. 4. (a) Number of packets transmitted per single iteration (b) Number of packets required to converge

in time unit, average hop count between leaders are considered. In DSSD, since a node communicates only with the neighbors within one hop distance, the hop count at each iteration is always 1, which means that convergence time depends only on the number of iterations. In RE^2-CD, however, increased density leads to decreased average hop count between leaders, because packets can be routed more directly. Since the number of iterations for convergence is almost constant in RE^2-CD, the convergence time depends on the average hop count; therefore, as the network density increases, the convergence latency in time unit decreases.

5.2 Protocol Overhead

Protocol overhead was calculated by measuring the number of algorithm-related packet transmissions in the entire network for single iteration of both DSSD and RE^2-CD cut detection algorithms using varied network density and cluster size.

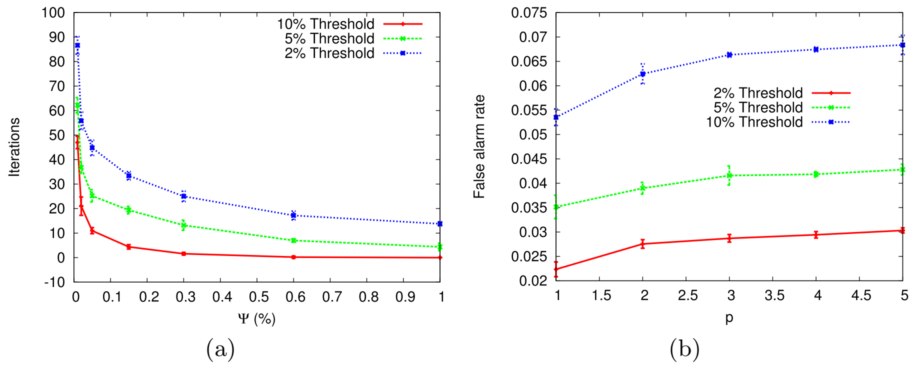


Fig. 5. (a) Number of iterations required to detect a malicious source (b) False alarm rate

The first step of this experiment was to randomly select a leader from each cluster. This was followed by the execution of one iteration of the cut detection algorithm on the virtual grid network of leaders (RE^2-CD) and, separately, on the entire network (DSSD). Each data point represents 50 iterations of this experiment.

Figure 4(a) depicts the results. As network density increased, the number of packet transmissions decreased when using RE^2-CD. This is primarily 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 for each iteration remained constant at 264 because every node broadcasts once per iteration regardless of network density. Also note that larger cluster sizes yield lower packet overhead because leaders in larger clusters have more neighbors that can route packets more efficiently.

Figure 4(b) shows the total number of packet transmissions required for convergence. This is based on multiplication of total iterations to convergence by the number of packet transmissions for a single iteration. The two algorithms display significant differences in message overhead.

5.3 Robustness

Detection latency describes 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, e.g., $k = 34$ for RE^2-CD. 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 between malicious source injection and detection by a leader node was measured. For different thresholds, 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.

Fig. 5(a) plots the result. In terms of detection accuracy, RE^2-CD detects a malicious source that deviates only 1% from the average state of neighbors. Higher thresholds allow faster detection but, as Fig 5(b) notes, they cause false alarms and cause correct states to be dropped. Although high false alarm rates cause unstable convergence with some fluctuations, rates less than 30% still allow the system to converge [19].

Figure 5(b) explores the problem of selecting *p* parameter. This parameter indicates the number of nearest neighbor states used in the outlier detection algorithm and its selection represents a tradeoff. Larger values of *p* tend to increase the false alarm rate but enhance detection of malicious source node clusters, which 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 the cluster of malicious sources injecting similar states.

6 Conclusions

We proposed a robust, energy-efficient algorithm to enhance the detection of disrupted network connectivity in harsh, unattended low-security environments using network of resource-constrained nodes. Our algorithm enhances security by enabling detection of malicious source nodes, even at very low thresholds. Simultaneously, through adoption of a clustered, leader-based convergence algorithm, we greatly reduced the energy required to detect a cut. Parameters including cluster size, node density, and deviation thresholds offer opportunities to trade off energy use and malicious source detection speed for optimal results in arbitrary networks.

Current work in progress includes deployment of a cut-detection enabled sensor network in an outdoor setting. Future work will address tradeoffs between security and energy efficiency and investigate the impact of modifying iteration length in response to changes in the local threat level. Longer breaks between iterations are likely to improve energy efficiency. Independent leader elections are another enhancement that promises to increase network lifespan. Instead of having leaders rotate in lock step, clusters internally determine when to elect a new leader based on local network activity and conditions. Other open questions include finding out other types of possible malicious behaviors critical to the operation of our cut detection algorithm and developing mechanisms to defend against the attacks.

References

1. Barooah, P.: Distributed cut detection in sensor networks. In: Proc of IEEE Conf. on Decision and Control (2008)
2. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proc. of Conf. on Knowledge discovery and data mining (2002)
3. Knorr, E., Ng, R., Tucakov, V.: Distance-based outliers: algorithms and applications. VLDB Jrnl (2000)
4. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: Proc. of Conf. on Management of data (2000)

5. Zhang, J., Wang, H.: Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. Knowledge and Information Systems (2006)
6. Breunig, M., Kriegel, H., Ng, R., Sander, J.: LOF: identifying density-based local outliers. In: SIGMOD (2000)
7. Papadimitriou, S., Kitagawa, H., Gibbons, P., Faloutsos, C.: LOCI: Fast outlier detection using the local correlation integral. In: Proc. of Conf. on Data Engineering (2003)
8. Park, V., Corson, M.: A highly adaptive distributed routing algorithm for mobile wireless networks. In: INFOCOM 1997. Sixteenth Annual Joint Conf. of the IEEE Computer and Communications Societies. Proceedings IEEE, vol. 3, pp. 1405–1413 (1997)
9. Chong, C.Y., Kumar, S.: Sensor networks: evolution, opportunities, and challenges. Proc. of the IEEE (2003)
10. Cerpa, A., Estrin, D.: ASCENT: Adaptive self-configuring sensor networks topologies. IEEE Trans. on Mobile Computing (2004)
11. Kleinberg, J.: Detecting a network failure. Internet Mathematics (2004)
12. Ritter, H., Winter, R., Schiller, J.: A partition detection system for mobile ad-hoc networks. In: Proc. of IEEE Conf. on Sensor and Ad Hoc Communications and Networks (SECON) (2004)
13. Shrivastava, N., Suri, S., Tóth, C.: Detecting cuts in sensor networks. ACM Trans. on Sensor Netwks (2008)
14. Stoleru, R., He, T., Stankovic, J.A.: Range-free localization. Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks 30 (2007)
15. Malpani, N., Welch, J.L., Vaidya, N.: Leader election algorithms for mobile ad hoc networks. In: Proc. of Workshop on Discrete algorithms and methods for mobile computing and communications (2000)
16. Karp, B., Kung, H.T.: GPSR: Greedy perimeter stateless routing for wireless networks. In: Proc. of Conf. on Mobile Computing and Networking (MobiCom) (2000)
17. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. In: Proc. of Conf. on Architectural support for programming languages and operating systems (2000)
18. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: SenSys 2003: Proc. of the 1st Int. Conf. on Embedded networked sensor systems, pp. 126–137. ACM, New York (2003)
19. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4, 382–401 (1982)